TAMPEREEN TEKNILLINEN YLIOPISTO

KIMMO RANTALA

SOVELLUSPALVELIMIEN KONFIGURAATIONHALLINNAN AUTOMATISOINTI

Diplomityö

Tarkastaja: Prof. Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunnan tiedekuntaneuvoston
kokouksessa 4. kesäkuuta 2014

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
Rantala, Kimmo: Sovelluspalvelimien konfiguraationhallinnan automatisointi
Diplomityö, 62 sivua
Tammikuu 2015
Pääaine: Ohjelmistotuotanto
Tarkastajat: Prof. Tommi Mikkonen
Avainsanat: Automatisointi, Jatkuva toimitus, Konfiguraationhallinta, Konfigurointi

Ympäristöjen konfiguraatioilla ja niiden hallinnalla on oleellinen merkitys tietojärjestelmien toimivuudelle. Laadukkaasti toteutetut ja vaatimuksia vastaavat sovellukset eivät yksinään riitä takaamaan järjestelmien toimivuutta. Sovellusten suoritusympäristöihin kohdistuu vaatimuksia, joilla on suuri merkitys muun muassa tietoturvaan, saatavuuteen ja järjestelmien välisiin integraatioihin.

Suuret, monimutkaiset ja toiminnaltaan kriittiset järjestelmäkokonaisuudet asettavat konfiguraationhallinnalle merkittäviä haasteita, joihin manuaalisella konfiguraationhallinnalla vastaaminen ei ole aina tehokas ja toimiva ratkaisumalli. Konfiguraationhallinnan tehtävien ratkaiseminen automatisoinnilla tarjoaa useita etuja muun muassa tehtävien toistettavuuden, ketteryyden sekä laadunvarmistuksen kannalta. Tehtävien automatisointi vaatii tarkkaa tietoa ympäristöjen ja prosessien rakenteesta. Automatisoinnilla saavutettavat edut ovat kuitenkin usein siihen vaaditun työmäärän arvoisia.

Tässä diplomityössä selvitetään erilaisia konfiguraationhallinnallisia haasteita, joita voidaan ratkaista automatisoinnilla, sekä esitellään automatisoinnilla saavutettavia hyötyjä. Työssä automatisoidaan Solitan toteuttamien Maanmittauslaitoksen UKIR-hankkeen sovellusten konfiguraationhallintaa Solitan ympäristötarpeiden osalta.

Toteutuksen perusteella voidaan todeta automatisoinnin ratkaisevan monia konfiguraationhallintaan liittyviä ongelmia, ja tarjoavan useita etuja, joita manuaalisella tehtävien suorittamisella ei voida tehokkaasti saavuttaa. Tekniset automatisointiratkaisut eivät silti yksinään pysty ratkaisemaan kaikkia konfigurointiin ja ympäristöihin liittyviä ongelmia. Konfiguraationhallinta vaatii suunnitelmallisuutta ja osapuolten välistä kommunikointia. Automatisointi edellyttää osittain erilaista suhtautumistapaa konfiguraatioihin ja toimitusprosesseihin verrattuna manuaaliseen hallintamalliin. Vastineena voidaan kuitenkin saavuttaa merkittäviä etuja muun muassa toimintavarmuudessa sekä prosessien nopeudessa.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
Rantala, Kimmo: Automatic configuration management for application servers
Master of Science Thesis, 62 pages
January 2015
Major: Software Engineering
Examiner: Prof. Tommi Mikkonen
Keywords: Automation, Continuous Delivery, Configuration, Configuration Management

Software environments and their configuration management have essential significance on the functionality of information systems. High quality applications that meet their demands are not enough to guarantee the functioning of the systems alone. There are requirements for the execution environments that have considerable effect, for example on information security, availability and integrations between different systems.

Large, complex and operationally critical systems create significant challenges for the configuration management, that can not be efficiently solved with manually performed configuration tasks. Through task automation, many configuration management challenges can be solved by achieving multiple benefits on task repeatability, agility and quality assurance. Task automation presumes exact knowledge of the environments and the structures of different processes, but gained benefits through automation often outweigh the amount of required work.

This master's thesis examines different challenges on configuration management that can be resolved through automation. This thesis also presents some of the advantages that automation provides. As a part of this thesis, configuration automation is applied to different applications that Solita has implemented for the UKIR project of the National Land Survey of Finland. The automation focuses on the development, testing and maintenance needs in Solita's environments.

Based on the implementation, automatic configuration management can resolve many problems and provide multiple benefits that can not be efficiently achieved with manually executed tasks. However, different technical solutions per se are incapable resolution to all the configurational and environmental issues. Configuration management requires planning and communication between people. To some extent, automation requires different kind of attitude towards configurations and deployment processes compared to the manual configuration management. However, in return, many significant benefits can be achieved, for example in functional reliability and cycle time of processes.

ALKUSANAT

Suuri kiitos Solita Oy:lle rahallisesta tuesta diplomityön tekemiseen. Kiitokset myös Maanmittauslaitokselle ja Jorma Turuselle, joiden hyväksynnällä työssä voitiin käyttää sovelluskohteena UKIR-hankkeeseen kuuluvia sovelluksia. Kiitokset ansaitsevat myös kollegani Solitalla sekä kaikki ystävät ja läheiset, joiden tuen ja kannustuksen ansiosta työ valmistui. Erityiskiitos myös diplomityön tarkastajalle Tommi Mikkoselle ja ohjaajalle Petri Sirkkalalle kaikista neuvoista ja palautteesta.

Tampereella 1.12.2014

Kimmo Rantala

SISÄLLYS

TERMIT JA NIIDEN MÄÄRITELMÄT

APA	Ajastuspalvelu, jolla voidaan luoda ajastettuja eräajoja muun muassa KIRREen ja välittää tuloksia ulkopuolisiin järjestelmiin.
ASSI	Kiinteistötietojärjestelmän asiakkuudenhallintasovellus. Tarjoaa muun muassa toiminnallisuuden asiakkaiden yhteyshenkilöiden sekä osoitteiden hallintaan.
CI	Continuous Integration. Ohjelmistotuotannon käytäntö, jossa kaikki kehittäjien eri komponentteihin tekemät muutokset yhdistetään osaksi sovelluskokonaisuutta useita kertoja päivässä.
CD	Continuous Delivery. Ohjelmistotuotannon käytäntö, joka pyrkii ohjemistojen julkaisuprosessin kehittämiseen ja automatisointiin.
DNS	Domain Name System. Nimipalvelujärjestelmä, jolla verkkotunnukset muutetaan IP-osoitteiksi.
DSL	Domain-Specific Language. Tietyn sovellusalueen kuvaamiseen tarkoitettu kieli.
EC2	Amazon Elastic Compute Cloud. Amazon.comin pilvipalvelualusta, joka tarjoaa käyttäjille virtuaalipalvelimia omien sovellusten suorittamiseen.
ESB	Enterprise Service Bus. Palveluväylä, jonka avulla sovellukset kommunikoivat keskenään.
GNU	GNU's Not Unix. Richard Stallmanin käynnistämä Free Software Foundationin hallinnoima projekti, jonka tavoittena on kehittää täysin vapaa käyttöjärjestelmä.
GPL	GNU General Public License. Vapaiden ohjelmistojen julkaisuun tarkoitettu lisenssi.
HTTP	Hypertext Transfer Protocol. Protokolla, jota käytetään web-selaimissa ja -palvelimissa tiedonsiirtoon.
Infrastruktuuri	Perusta, jolla palvelut toimivat. Sisältää yleensä palvelimet, niiden konfiguraatiot ja ylläpidon.
IT	Informaatioteknologia. Tietokoneiden ja digitaalisen tietoliikenteen avulla tehtävää tietojen käsittelyä.

JDBC	Java Database Connectivity. Rajapinta, jolla Java-sovellukset voivat käyttää tietokantaa.
JSON	JavaScript Object Notation. Tiedon välitykseen käytetty avoimen standardin tiedostomuoto.
KIOS	Solitan nimi kehitysprojektille, jossa toteutettiin KIRRE, MUTI, ASSI, APA ja järjestelmäintegraatiot. Myös KIRREn alkuperäinen nimi.
KIRRE	Maanmittauslaitoksen kiinteistötietojärjestelmän kirjaamisosa, eli lainhuuto- ja kiinnitysrekisteri.
Konfiguraatio	Ryhmä palvelun tuottamiseen tarvittavia yhdessä toimivia rakenneosia (configuration item). Voidaan käyttää myös kuvaamaan yhden tai useamman rakenneosan parametriasetuksia.
MUTI	Muutostietopalvelu. Toimii välivarastona Väestörekisterikeskuksen tarjoamille henkilöiden yhteystiedoille.
NTP	Network Time Protocol. Protokolla, jonka avulla voidaan synkronoida palvelimien kellot oikeaan aikaan.
REST	Representational State Transfer. Arkkitehtuurityyli, jota käytetään useissa web-palveluiden ohjelmointirajapinnoissa.
SFTP	SSH File Transfer Protocol. Verkkoprotokolla, jolla välitetään tiedostoja salatun yhteyden yli.
Solmu	Node. Yksittäinen verkossa oleva konfiguroitava laite, esimerkiksi tietokone. Voi olla myös virtuaalinen.
SSH	Secure Shell. Protokolla salattua tietoliikennettä varten.
Tehtäväputki	Peräkkäin suoritettavista tehtävistä koostuvan ketjun yleisnimitys.
Toimitusputki	Deployment pipeline. Eri tehtävistä ja vaiheista koostuva tehtäväputki, joka lähtee yksittäisestä kehittäjän tekemästä muutoksesta ja päättyy tuotantovalmiiseen sovellukseen. Sisältää joissakin yhteyksissä myös tuotantoasennuksen.
Tynkä	Stub. Yksinkertaistettu testausta varten tehty toteutus, joka simuloi rajoitetusti varsinaisen ohjelman toimintaa.
UKIR	Maanmittauslaitoksen uuden kiinteistöjen kirjaamisosan toteutushanke. KIRRE, ASSI, APA, MUTI ovat osa tätä hanketta.

URL	Uniform Resource Locator. Merkkijono, jolla kuvataan jonkin verkossa olevan resurssin sijainti ja protokolla sen noutamiseen.
XML	Extensible Markup Language. Tiedonvälitykseen käytetty rakenteellinen kuvauskieli.
YAML	YAML Ain't Markup Language. Datan välitykseen tarkoitettu merkintäkieli.

1. JOHDANTO

Erilaiset konfiguraatiot ja niiden hallinta ovat oleellinen tekijä tietojärjestelmien toimivuudelle. Sovellukset asettavat toimiakseen vaatimuksia palvelimille ja ympäristöille, joissa sovelluksia suoritetaan. Ohjelmistojen asettamat vaatimukset voivat olla hyvin erilaisia eri tilanteissa. Monimutkaisten useista komponenteista koostuvien järjestelmäkokonaisuuksien konfigurointi vaatii organisoidun konfiguraationhallinnan, jonka on pystyttävä takaamaan järjestelmien toimivuus myös komponenttien ja rakenteiden muuttuessa.

Muun muassa suuret järjestelmäkokonaisuudet, useat hallittavat ympäristöt sekä haasteelliset vasteaikatavoitteet luovat konfiguraationhallinnalle vaatimuksia, joihin manuaalisella konfiguraationhallinnalla ei voida tehokkaasti vastata. Automatisoimalla konfiguraationhallinnan tehtäviä voidaan ratkaista useita konfiguraationhallintaan liittyviä ongelmia. Tehtävien automatisointi vaatii tarkkaa tietoa ympäristöjen ja prosessien rakenteesta. Automatisoinnilla saavutettavat edut ovat kuitenkin usein siihen vaaditun työmäärän arvoisia.

Tässä diplomityössä selvitetään erilaisia konfiguraationhallinnallisia haasteita, joita voidaan ratkaista automatisoinnilla, sekä esitellään automatisoinnilla saavutettavia hyötyjä. Työssä automatisoidaan Solitan toteuttamien Maanmittauslaitoksen UKIR-hankkeen sovellusten konfiguraationhallintaa Solitan ympäristötarpeiden osalta.

Tämän diplomityön luvussa 2 määritellään mitä konfiguraationhallinnalla tarkoitetaan ja esitellään siihen liittyviä erilaisia haasteita. Luvussa esitellään jatkuvan integroinnin ja jatkuvan toimituksen mallien käsitteet sekä kuvataan automatisoinnilla saavutettavia etuja ja eri työkaluja, joilla automatisointia voidaan toteuttaa. Luvussa 3 esitellään tarkemmin eräs automatisoituun konfiguraationhallintaan käytettävä työkalu nimeltä Chef. Luvussa 4 esitellään Maanmittauslaitoksen UKIR-hanke ja siihen liittyvät sovellukset, joille tässä diplomityössä sovelletaan automatisoidun konfiguraationhallinnan mallia. Luku 5 sisältää kuvauksen sovelluskohteeseen liittyvistä konfiguraationhallinnan haasteista sekä esittelee diplomityön osana Chefillä toteuteutun sovelluskohteen konfiguraationhallinnan automatisoinnin. Luvussa 6 arvioidaan tehtyä toteutusta, sen ratkaisuja ja vastaavuutta asetettuihin vaatimuksiin. Luku 7 sisältää työn yhteenvedon.

2. KONFIGURAATIONHALLINTA

Konfiguraationhallinta on keskeinen osa ohjelmistotuotantoa. Termillä konfiguraationhallinta on useita merkityksiä, joten on tärkeää määritellä mitä sanalla tarkoitetaan. Konfiguraationhallintaan liittyy useita haasteita, joihin voidaan vastata erilaisilla tavoilla tilanteesta riippuen. Joihinkin haasteista voidaan vastata automatisoinnilla. Se edellyttää kuitenkin prosessien tarkkaa kuvaamista ja vakiointia. Ainakin osittainen testauksen automatisointi on usein edellytyksenä laadukkaalle automatisoinnille. Jatkuvalla integroinnilla voidaan vähentää eri sovelluskomponenttien yhteentoimivuuteen liittyviä riskejä ja automatisoida testien suorittamista. Kattavalla automaattitestauksella ja automatisoidulla konfiguraationhallinnalla toimitusputken kestoa voidaan lyhentää merkittävästi. Automatisoinnilla voidaan saavuttaa paljon hyötyä, mutta siihen liittyy myös ongelmia, jotka tulee tiedostaa.

Tässä luvussa määritellään mitä termillä konfiguraationhallinta tarkoitetaan ja kuvataan joitakin yleisimpiä konfiguraationhallinnan haasteita. Lisäksi esitellään jatkuvan integroinnin ja jatkuvan toimituksen käsitteet sekä mitä automatisoidulla konfiguraationhallinnalla voidaan saavuttaa. Lopuksi esitellään työkaluja automatisoidun konfiguraationhallinnan toteuttamiseen.

2.1 Määritelmä

Konfiguraatio ja konfiguraationhallinta ovat termejä, joita käytetään useissa asiayhteyksissä, ja joiden määritelmä usein vaihtelee käyttötilanteesta riippuen. Information Technology Infrastructure Libraryn versiossa kolme (ITILv3) määritellään termin konfiguraationhallinta tarkoittavan prosessia, jolla ylläpidetään palvelua tukevia rakenneosia (configuration item). Rakenneosia ovat kaikki palvelun tuottamiseen kuuluvat hallittavat komponentit, kuten laitteet, sovellukset, ihmiset ja dokumentaatio. Konfiguraatio on ryhmä palvelun tuottamiseen tarvittavia yhdessä toimivia rakenneosia. Termiä konfiguraatio voidaan käyttää myös kuvaamaan yhden tai useamman rakenneosan parametriasetuksia. [1, s. 12–13.]

Termiä konfiguraationhallinta käytetään joskus myös synonyyminä versionhallinnalle ja versionhallintaan käytettäville työkaluille, joilla ohjelmakoodin muutoksia pyritään hallitsemaan. Konfiguraationhallinnan määritelmiä on myös muita, ja useissa lähteissä konfiguraationhallinnalla tarkoitetaankin erityisesti ohjelmallisia työkaluja, joiden avulla infrastruktuurin luominen ja muutokset pyritään automatisoi-

maan.

Konfiguraationhallinnan voidaan nähdä käsittävän myös vielä laajemman kokonaisuuden, johon Bob Aiello ja Leslie Sachs ovat kirjassa Configuration Management Best Practices listanneet kuusi pääkohtaa. Näihin kuuluu lähdekoodin hallinta (source code management), käännösten luonti (build engineering), ympäristökonfiguraatiot (environment configuration), muutosten hallinta (change control), julkaisutekniikka (release engineering) ja käyttöönotto (deployment). [2, s. 33.]

Jez Humblen ja David Farleyn kirjassa Continuous Delivery määritellään konfiguraationhallinnan olevan prosessi, joka käsittää kaikki projektiin liittyvät artefaktit ja niiden väliset suhteet, sekä kuinka niitä tallennetaan, käytetään, identifioidaan ja muutetaan [3, s. 31]. Tässä diplomityössä konfiguraationhallinnalla tarkoitetaan edellä kuvattua määritelmää. Konfiguraationhallinta käsittää infrastruktuurin ja palvelimilla olevien sovellusten konfiguraatiot sekä niiden hallinnan. Apuna voidaan käyttää versionhallintatyökaluja, mutta ne ovat kuitenkin vain yksi osa konfiguraationhallinnan kokonaisuutta.

2.2 Haasteet

Konfiguraationhallintaan liittyy useita haasteita, joita pyritään ratkaisemaan erilaisilla menetelmillä. Samat haasteet eivät välttämättä ole yhtä merkityksellisiä kaikissa ympäristöissä ja ohjelmistohankkeissa. Eri tilanteissa ympäristöt ja vaatimukset saattavat erota paljonkin toisistaan, esimerkiksi hallittavien resurssien määrän suhteen. Erilaisten haasteiden ratkaiseminen vaatii huolellista suunnittelua ja eri tilanteisiin varautumista.

Tässä kohdassa kuvattuja yleisiä haasteita on pyritty ryhmittelemään niiden luonteen mukaisesti eri alakohtiin. Ryhmittely on kuitenkin viitteellinen eikä pyri kattamaan kaikkia eri ohjelmistohankkeiden ja -ympäristöjen konfigurointiin liittyviä ongelmallisuuksia.

2.2.1 Muutosten hallinta

Konfiguraationhallinnalle luo haasteita tilanteiden ja komponenttien muuttuminen. Varsinkin kehitysaikana ympäristöjen konfiguraatiot saattavat muuttua paljon lyhyessä ajassa. Ohjelmistojen muuttuvat rajapinnat ja uusien komponenttien liittäminen mukaan järjestelmään vaativat muutoksia konfiguraatioihin vähintään uusien sovellusversioiden asentamisen muodossa.

Konfiguraationhallinta voi yksinkertaisimmissa tapauksissa tapahtua vain tilannekohtaisina manuaalisina muutoksina. Mahdollisesti ongelmia selvitetään ja yritetään korjata tuotantoympäristössä kokeilemalla erilaisia vaihtoehtoja, joita ei dokumentoida toistettavaan muotoon. Asiantuntevalta henkilöltä korjaus saattaa onnis-

tua nopeasti ja pienellä työmäärällä. Tällainen toimintamalli voidaan nähdä kuitenkin riittämättömänä toimintavarmuuden ja laadun kannalta, kuten on todettu muun muassa Jez Humblen ja David Farleyn Continuous Delivery -kirjassa [3, s. 5–6].

Kaikki ongelmat eivät välttämättä ole helposti korjattavissa ja vaativat syvällisempää selvittelyä mieluiten ympäristössä, joka ei ole yhtä krittinen kuin useille käyttäjille käytössä oleva tuotantoympäristö. Selvittely ja kokeilu saattavat vaikeuttaa tilannetta entisestään ja erilaiset yritykset jäävät helposti dokumentoimatta. Välttämättä ratkaisukaan ei ole dokumentoituna riittävällä tasolla ja saattaa olla vain kyseisen tekijän itsensä ymmärtämässä muodossa [3, s. 6].

Konfiguraatioiden ja muutosten dokumentointi on tärkeä osa konfiguraationhallintaa. Hyvä dokumentaatio parantaa muun muassa konfiguroinnin toistettavuutta ja testattavuutta sekä vähentää niin sanottua siiloutumista, eli tiedon ja osaamisen kertymistä pelkästään tietyille henkilöille. Kaikissa tilanteissa ei välttämättä koeta tarpeelliseksi konfiguraatioihin kohdistuneiden muutosten dokumentointia, vaan ylläpidetään vain kuvausta konfiguraatioiden nykytilasta. Muutoshistorian säilyttäminen voi helpottaa ongelmatilanteiden selvittämistä ja selittää minkä vuoksi tiettyihin ratkaisuihin on päädytty. Muutoshistorian ylläpitäminen parantaa konfiguraatioiden toistettavuutta dokumentoimalla myös muutosten tekojärjestyksen.

2.2.2 Ympäristöt

Usein ohjelmistohankkeissa on käytössä monia ympäristöjä eri tarkoituksia varten. Ympäristöissä voi olla joitakin eroavaisuuksia, koska niiden tarkoituskin saattaa poiketa toisistaan. Esimerkiksi jonkin sovelluksen kehitysympäristö voi poiketa tuotantoympäristöstä sillä, että joitakin palveluita on korvattu yksinkertaistetuilla tynkäpalveluilla, jotka soveltuvat paremmin kehitystyöhön. Ohjelmistohankkeessa käytettävien eri ympäristöjen konfiguraatioissa on kuitenkin yhteisiä osia, joiden halutaan toimivan samalla tavalla. Konfiguraatioiden muuttuessa halutaan myös mahdolliset konfiguraatiomuutokset tuoda hallitusti eri ympäristöihin. Continuous Delivery -kirjassa korostetaan, että on tärkeää tunnistaa konfiguroinnin kannalta kriittiset tekijät, ja määritellä miltä osin eri tarkoituksiin olevien ympäristöjen tulee vastata tuotantoympäristöä [3, s. 253].

Useiden ympäristöjen kanssa toimimiseen liittyy monenlaisia haasteita. On määriteltävä ovatko ympäristöt täysin itsenäisiä kokonaisuuksia vai voiko niillä olla joitakin yhteisiä osia. Aina ei haluta tai ei ole mahdollista saada täysin itsenäisiä ympäristöjä, jolloin yhteisiin osiin kohdistuvat muutokset vaikuttavat laajemmin useampaan ympäristöön. Mahdollisesti jokin jaettu palvelin täytyy uudelleenkäynnistää tai sen tietoja muokata toisen järjestelmän tarpeisiin, jolloin saattaa jäädä huomaamatta vaikutukset muihin ympäristöihin. Epätoivottuja vaikutuksia saattaa aiheuttaa myös jonkin ympäristön kuormitus, joka ilmenee muissa ympäristöissä

jaetun resurssin hitautena.

Monitoimittajaympäristössä kaikki toimittajat eivät välttämättä toimi samoilla tavoilla eivätkä koe samoja asioita yhtä merkityksellisiksi kuin toiset. Tämä saattaa johtaa tilanteeseen, jossa jotkin palvelut toimivat eri tavalla eri ympäristöissä. Välttämättä kaikkia järjestelmiä ja palveluita ei ole mahdollista saada useisiin ympäristöihin. Syinä voi olla esimerkiksi lisenssikustannukset, tai että järjestelmätoimittaja ei pysty toimittamaan tarvittavaa toista järjestelmää. Tällöin esimerkiksi järjestelmien välinen integraatiotestaus saadaan suoritettua vain tuotantoympäristössä.

2.2.3 Laajuus

Suurilla käyttäjämäärillä ja suorituskykyvaatimuksilla palveluiden skaalautuminen luo haasteita ympäristöjen ja konfiguraatioiden hallinnalle. Palveluiden hajauttaminen useammalle suorittavalle sovelluspalvelimelle on yksi tapa ratkaista skaalautuvuuden haasteita. Useamman sovelluspalvelimen hallinta luo erilaisia vaatimuksia konfiguraationhallinnalle. Käyttöasteen mukaan skaalautuvissa palveluissa sovelluspalvelinten määrä vaihtelee kuormituksen mukaan. Uusia palvelimia pitää saada luotua nopeasti kuormituksen kasvaessa. Kun kuormitus on vähäisempää, voidaan sovelluspalvelinten määrää vähentää.

Haasteita aiheutuu myös korkeista palveluiden saatavuusvaatimuksista eli siitä, että palveluiden on oltava toiminnassa jatkuvasti. Kaikissa tilanteissa konfiguraatiomuutokset ja sovellusversioiden päivittäminen ei saa aiheuttaa lyhyttäkään käyttökatkoa. Erityisesti tällaisissa tapauksissa suoritettavien muutosten toimivuus tulee olla varmistettu erilaisilla testeillä ennen tuotantoon vientiä.

Sovelluspalvelinten suuri määrä luo haasteita konfiguraationhallinnalle, vaikka palvelinten määrää ei skaalattaisikaan kuormituksen mukaan. Mikäli konfiguraatiomuutokset suoritetaan manuaalisesti palvelin kerrallaan, muutoksien toteuttamiseen tarvittava työmäärä on lähes suoraan verrannollinen sovelluspalvelinten lukumäärään.

2.3 Jatkuva integrointi

Artikkelissa Continuous Integration Martin Fowler määrittelee jatkuvan integroinnin (continuous integration, CI) ohjelmistotuotannossa tarkoittavan käytäntöä, jossa kehittäjien yksittäisiin sovelluskomponentteihin tehdyt muutokset yhdistetään heti osaksi sovelluskokonaisuutta [4]. Yleensä yksittäiset kehittäjät tekevät tämän useita kertoja päivässä, mikä johtaa päivittäin useisiin integrointeihin. Jokainen integraatio varmistetaan toimivaksi automaattisella käännöksellä (build), jolloin mahdolliset virheet voidaan havaita nopeasti. Tämän on havaittu vähentävän huomattavasti integroinnista aiheutuneita ongelmia ja nopeuttavan yleiskäyttöisemmän sovellus-

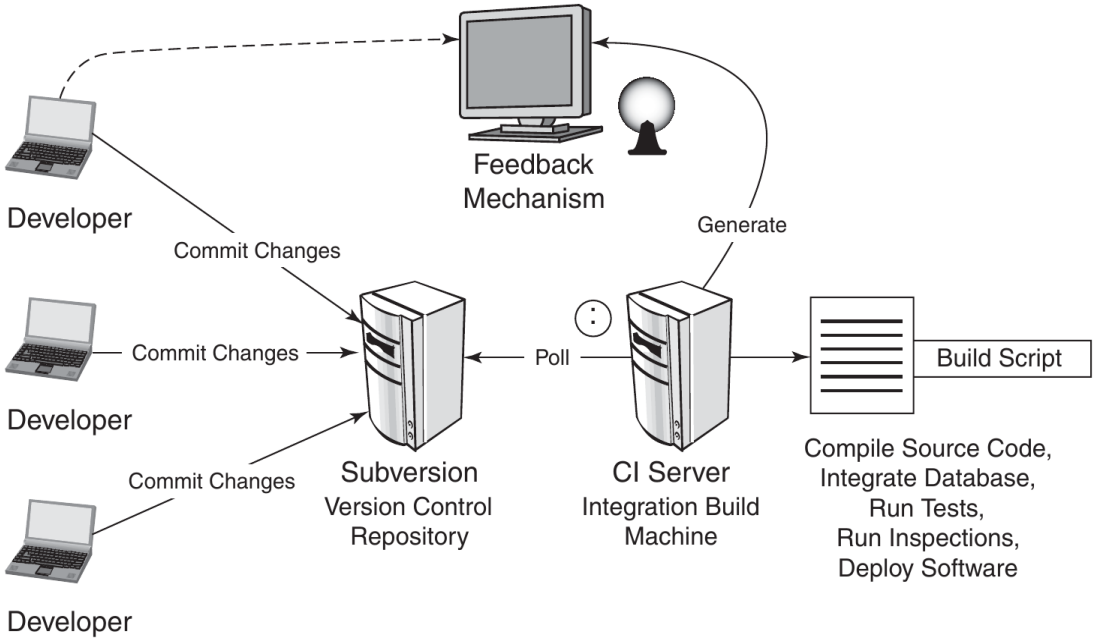
koodin kehittämistä. Tämä edellyttää kehittäjien huolehtivan, että heillä on tuorein versio koodista ennen omien muutosten lisäämistä versionhallinnan päähaaraan. Ennen lisäämistä pyritään sovelluksen toimivuutta varmistamaan suorittamalla riittävä joukko automaattitestejä. Riittävä joukko voi olla esimerkiksi kaikki muuttuneen moduulin testit.

Kehittäjien itse ajamien testien lisäksi ohjelmistokehityksessä käytetään usein erillistä jatkuvasta integroinnista huolehtivaa CI-palvelinta, jonka tehtävänä on havaita automaattisesti muutoksia versionhallinnassa olevassa haarassa, kääntää sovellus ja suorittaa automaattitestit. Tämä automaattitestijoukko saattaa olla laajempi ja suoritusajaltaan pidempi kuin mitä kehittäjä suorittaa ennen muutoksen lisäämistä päähaaraan.

Testejä suorittavien CI-palvelinten tulee vastata mahdollisimman hyvin tuotantoympäristöä. Näin voidaan osittain välttää niin sanottuja ”works on my machine” -ongelmia, eli tilanteita joissa kehittäjän oman työaseman konfiguraatiolla sovellus toimii eri tavalla kuin muissa ympäristössä. Testien suorittamisella erillisellä CI-palvelimella voidaan tehostaa kehitystyötä, koska kehittäjän ei tarvitse suorittaa kaikkien moduulien kaikkia testejä omalla työasemallaan jokaisella muutoksella. Kehittäjä voi olla esimerkiksi tehnyt muutoksen liiketoimintalogiikkaan ja sen valmistuessa suorittanut kyseiseen moduulin kuuluvat testit omalla työasemallaan. Kun muutos lisätään versionhallintaan, niin CI-palvelin suorittaa automaattisesti liiketoimintalogiikkamoduulin testien lisäksi kaikkien siitä riippuvien muiden moduulien testit. Tällöin voidaan CI-palvelimen suorituksista todeta esimerkiksi käyttöliittymämoduulin toimivan edelleen oikein muuttuneen liiketoimintalogiikkamoduulin kanssa. Kehittäjä voi testisuorituksen aikana toteuttaa jo seuraavaa tehtävää ja palata vain tarvittaessa korjaamaan aikaisempaa muutosta, mikäli CI-palvelimen testisuorituksissa havaitaan virheitä. Kaikkien testien suorittamiseen voi kulua kauan aikaa, jolloin palautteen saamista voidaan nopeuttaa suorittamalla testejä rinnakkain yhdellä tai useammalla CI-palvelimella.

Paul M. Duvallin kirjassa Continuous Integration on kuvattu CI-järjestelmän toimintaperiaate. Kuvassa 2.1 esitetään kuinka ohjelmistokehittäjät tekevät muutoksia lähdekoodiin. Kehittäjät voivat mahdollisesti toimia eri rooleissa, esimerkiksi tietokannan ylläpitäjänä, rajapintasuunnittelijana tai liiketoimintalogiikan toteuttajana. Kehittäjien versionhallintaan lisäämät muutokset havaitaan CI-palvelimella, joka tarkkailee versionhallintaa esimerkiksi muutaman minuutin välein. CI-palvelimella käynnistyy käännösprosessi, joka noutaa tuoreimman kopion versionhallinnassa olevasta haarasta. Prosessi suorittaa sille määritetyt vaiheet käännöksen suorittamiseksi, joihin useimmiten kuuluu myös automaattitestien suoritus. Suorituksen jälkeen CI-palvelin kommunikoi jollakin määritellyllä tavalla prosessin onnistumisesta tai mahdollisista virheistä. Tämä voi tapahtua esimerkiksi lähettämällä automaattises-

ti sähköpostia kehittäjille. [5, s. 4–12.]



Kuva 2.1: CI-järjestelmän rakenne [5, s. 5].

Jatkuvassa integroinnissa pyritään versionhallinnan päähaara pitämään koko ajan ehyenä. Jos kuitenkin testit CI-palvelimella epäonnistuvat, pyritään tilanne korjaamaan ennen kuin muuta työtä jatketaan. Havaitsemista helpottaa CI-palvelimen palaute suorituksista, jota voidaan visualisoida esimerkiksi liikennevalojen väreillä erilliseen näyttöruutuun, joka on jatkuvasti kehitystiimin nähtävillä. Kehittäjien tulisi olla koko ajan tietoisia versionhallinnassa olevan koodin tilasta.

Jatkuvan integroinnin toimiminen ja hyötyjen saavuttaminen edellyttää kattavaa joukkoa automaattitestejä, joiden perusteella saadaan varmuutta sovellukset toimivuudesta. Kattava testiverkosto auttaa varmistamaan, ettei uusi toteutus riko mitään aiempaa toiminnallisuutta. Automaattitestien muodostaman testiverkoston luonnissa voidaan käyttää testivetoisen kehityksen menetelmää (test-driven development, TDD), jossa uutta toiminnallisuutta testaavat testit kirjoitetaan ennen varsinaisen toiminnallisuuden toteutusta. Tällöin saadaan lisävarmuutta siitä, että testit todella testaavat halutun toiminnallisuuden. [3, s. 71.]

Työkaluja jatkuvan integroinnin toteuttamiseen on useita. Työkalujen toiminta perustuu yleensä erilaisten tehtävien suorittamiseen. Tehtävät voidaan käynnistää manuaalisesti tai asettaa käynnistymään automaattisesti esimerkiksi ajastetusti tai versionhallinnan muutoksista. Työkalut tarjoavat erilaisia tapoja seurata tehtävien suorituksia ja onnistumista. Esimerkiksi avoimen lähdekoodin työkalu Jenkins [6] tallentaa tehtävien suoritushistorian, josta voidaan seurata tehtävien suoritusajanhetkiä, kestoja sekä mitä muutoksia lähdekoodissa on ollut suoritusten välillä. Osa

jatkuvan integroinnin työkaluista soveltuu myös muuhunkin kuin lähdekoodin kääntämiseen ja testaamiseen. Työkaluilla voidaan esimerkiksi suorittaa tietokantapäivityksiä ajastetusti.

2.4 Jatkuva toimitus

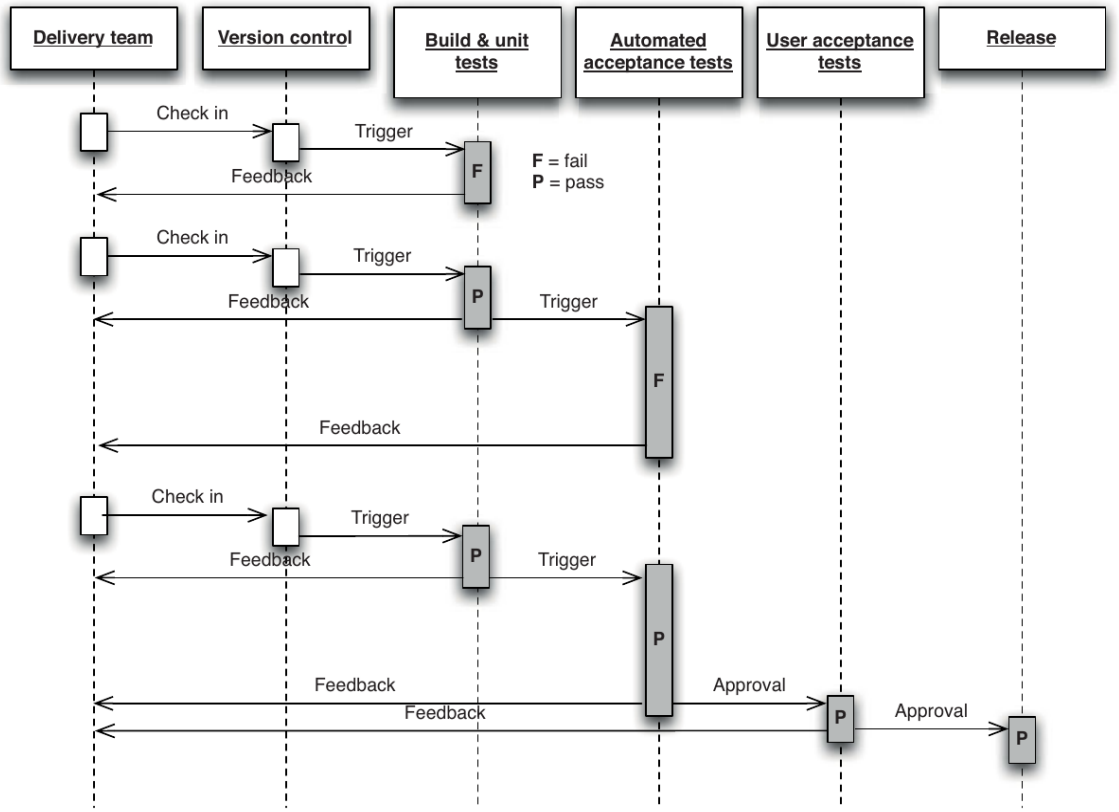
Jatkuva toimitus (continuous delivery) on malli, joka pyrkii ohjelmistojen julkaisuprosessin kehittämiseen ja automatisointiin. Malli perustuu kattavaan automaattitestaukseen, jatkuvaan integrointiin ja nopeaan toimitussykliin. Jez Humblen ja David Farleyn Continuous Delivery -kirjan mukaan hyöty sovelluksesta toteutuu asiakkaalle vasta silloin, kun sovellus on viety onnistuneesti tuotantoon. Toteutettujen ominaisuuksien ja virhekorjausten todellinen hyödyllisyys selviää vasta, kun loppukäyttäjät ovat sen todenneet. Ennen sitä kyseessä on vain olettamuksia ja valistuneita arvauksia. Tämän vuoksi on tärkeää toimittaa muutokset nopeasti ja usein. Tärkeä tekijä hyödyllisyydessä on laatu. Jatkuvan toimituksen tavoitteena on toimittaa korkealaatuinen ja hyödyllinen ohjelmisto tehokkaasti, nopeasti ja luotettavasti. [3, s. 11–12.]

Continuous Delivery -kirjan mukaan jatkuva toimitus mahdollistetaan toistettavalla, luotettavalla ja ennustettavalla julkaisuprosessilla. Tehokkaan toimitusputken myötä ehjien ja tuotantovalmiiden sovellusversioiden tunnistaminen ja muodostaminen helpottuu. Kirjan kirjoittajien kokemusten mukaan toimitussyklin pituuteen vaikuttaa suuresti ehjän tuotantovalmiin käännöksen odottamiseen kulutettu aika. Kirjoittajien mukaan hyvän käännöksen muodostaminen vaatii usein suuren määrän sähköpostin vaihtoa, tehtävienhallintajärjestelmän tikettejä ja tehottomia kommunikaatiomenetelmiä, mikä korostuu erityisesti silloin, kun projektitiimi on hajautunut. Tehokkaan toimitusputken toteutuksen myötä tämä ongelma poistuu. Kaikki toimitusputken läpäisseet käännökset ovat teoriassa tuotantovalmiita, ja ovat toimitettavissa niin sanotusti napin painalluksella. [3, s. 17–22.]

Jatkuva toimitus ei välttämättä ole kuitenkaan haluttu, toimiva tai aina edes mahdollinen malli kaikkiin käytännön tilanteisiin. Täydellisen ja tehokkaan toimitusputken luominen voi osoittautua mahdottomaksi tai hyvin hankalaksi. Toimitusputken vaiheita määriteltäessä on syytä tiedostaa, että kaikkia vaiheita ei ole välttämätöntä suorittaa automaattisesti [3, s. 110]. Jotkin vaiheet voivat koostua manuaalisesti suoritettavasta työstä. Tällaiset vaiheet saattavat kasvattaa toimitusputken kokonaiskestoa huomattavasti. Jos joitakin toimitusputken vaiheita ei voida erinäisistä syistä johtuen suorittaa riittävän usein, ei jokaisen käännöksen toimitusputkea ole järkevää viedä päätökseen. Mikäli joitakin vaiheita voidaan suorittaa esimerkiksi puolen vuoden välein, ei ole välttämättä järkevää käyttää termiä jatkuva toimitus.

Kuvassa 2.2 on esitelty Continuous Delivery -kirjassa oleva visualisointi toimitusputken kulusta sekvenssikaaviona. Jokaisesta kehitystiimin versionhallintaan li-

säämästä muutoksesta käynnistyy uusi toimitusputki. Kuvassa ylimpänä olevasta lisäyksestä käynnistynyt toimitusputki katkeaa vaiheeseen, jossa suoritetaan sovelluksen kääntäminen ja yksikkötestit. Testin epäonnistuminen katkaisee putken ja kehittäjätiimi saa siitä palautteen, jonka perusteella tiedetään muutoksen olleen epäonnistunut. Tällöin ei kyseistä käännöstä ole järkevää viedä eteenpäin toimitusputkessa. Virheiden havaitseminen ja palautteen saaminen mahdollisimman aikaisessa vaiheessa tehostavat työtä. Mitä pidemmälle toimitusputken suoritus etenee, sitä varmempia voidaan olla kyseisen käännöksen kelvollisuudesta. Kääntöpuolena sarjallistetussa suorituksessa on, että mitä pidemmällä toimitusputkessa potentiaalinen virhe on, sitä myöhemmin se havaitaan. Siten palautteen saanti toimitusputken myöhemmistä vaiheista tulee kehittäjille pidemmällä viiveellä kuin toimitusputken alkupään vaiheista. [3, s. 108–109.]



Kuva 2.2: Toimitusputken rakenne [3, s. 109].

Jatkuvan toimituksen malli ei edellytä, että kaikkia toimitusputken läpäisseitä käännöksiä vietäisiin tuotantoon asti. Toimitusputken läpäisseet käännökset ovat vain potentiaalisia tuotantoon vietäviä käännöksiä, joista käytetään nimitystä julkaisukandidaatti [3, s. 22–24]. Mallia, jossa kaikki toimitusputken läpäisseet julkaisukandidaatit viedään tuotantoon asti, kutsutaan jatkuvaksi käyttöönotoksi (continuous deployment) [3, s. 266].

Jatkuvassa käyttöönotossa toimitusputken viimeinen vaihe, eli tuotantoonvienti, tehdään automaattisesti. Jatkuvan toimituksen mallin toimivuus perustuu automaattisesti suoritettaviin vaiheisiin, mikä edellyttää jatkuvassa käyttöönotossa hyvin kattavia automaattitestejä. Jatkuvan käyttöönoton malli ei välttämättä sovellu kaikkiin tilanteisiin. Uusien versioiden toimittamiseen voi esimerkiksi sisältyä vastuu muutosten tiedottamisesta ulkoisille asiakkaille tai organisaatio voi olla sitoutunut tukemaan kaikkia toimitettuja versioita. Jatkuva käyttöönotto saattaa vaikuttaa riskialttiilta toimintamallilta. On hyvä kuitenkin huomata, että kun käyttöönottoja tehdään usein, ei yksittäisissä versioissa ole yhtä paljon muutoksia kuin julkaistaessa esimerkiksi puolen vuoden välein. [3, s. 266–267.]

2.5 Automatisointi

Konfiguraationhallinnan automatisoinnilla tarkoitetaan palvelinten konfiguraatioiden keskitettyä hallintaa, missä konfiguraatiomuutosten toteutus kaikille palvelimille tapahtuu automatisoidusti. Automatisointi edellyttää konfiguraationhallintaan liittyvien prosessien tarkkaa kuvaamista ja vakiointia.

Ohjelmistotuotannossa sovelluspalvelinten konfiguraatiot ovat perinteisesti erillään toteutettavista sovelluksista. Palvelinten konfiguraatioista saattaa vastata eri ihmiset tai eri organisaatio. Sovellukset voivat olla vahvasti sidoksissa palvelimiin ja vaatia toimiakseen muita palveluita sekä tiettyjä asetuksia. Automatisoitu konfiguraationhallinta edellyttää näiden riippuvuuksien tiedostamista ja määrittelyä.

Konfiguraationhallinnan automatisoinnilla voidaan saavuttaa merkittäviä hyötyjä toimitusprosessien nopeuttamisessa, palveluiden skaalaamisessa sekä laadunvarmistuksessa. Automatisointi ei suoraan ratkaise kuitenkaan kaikkia konfiguraationhallinnan ongelmia eikä välttämättä edes sovellu kaikkiin tilanteisiin. Varsinkin vanhojen, pitkään käytettyjen ja jo valmiiksi tarpeisiin mukautettujen prosessien automatisointi saattaa vaatia paljon työtä, aikaa ja syvällistä perehtymistä käytössä oleviin prosesseihin.

Konfiguraationhallintatyökalut automatisoivat palvelimien konfiguroinnin hallituksi ja toistettavaksi operaatioksi. Infrastruktuuriin pyritään suhtautumaan kuin ohjelmakoodiin, jota on muun muassa mahdollista testata, jakaa ja versioida. Versioitu konfiguraatiokoodi toimii myös dokumentaationa ja muutoshistoriana konfiguraatioille.

Konfiguraationhallintatyökaluja on olemassa useita, joista jotkin ovat olleet olemassa jo pitkään. Tarve konfiguraationhallinnan automatisoinnille on lisääntynyt ketterien menetelmien ja pilvipalveluiden yleistymisen myötä. Muun muassa käyttäjämäärien mukaan skaalautuvat palvelut ja ketterät toimitusprosessit luovat vaatimuksia ympäristöjen automatisoituun hallintaan ja laajennettavuuteen. Erilaiset konfiguraationhallintatyökalut helpottavat näihin tarpeisiin vastaamisessa.

Konfiguraationhallintatyökalujen käyttö verrattuna palvelinten manuaalisesti tehtävään konfigurointiin tuo paljon etuja varsinkin, jos palvelinten määrä on suuri. Suuren palvelinmäärän konfigurointi manuaalisesti palvelin kerrallaan on huomattavasti hitaampaa ja virheiden tekemisen todennäköisyys kasvaa. Virheitä ei välttämättä edes huomata heti, vaan ne saattavat paljastua vasta myöhemmin erilaisissa tilanteissa. Työkalujen avulla palvelimia voidaan hallita keskitetysti ja uusia palvelimia saadaan luotua automatisoidusti, jolloin samat asetukset tulee tehtyä kaikille palvelimille samassa järjestyksessä. Muutosten testaaminen on mahdollista automatisoida esimerkiksi luomalla uusi virtuaalipalvelin, johon suoritetaan konfiguraatiomuutokset ja automaattitesteillä tarkastetaan, että palvelin on halutussa tilassa.

Stephen Nelson-Smithin kirjassa Test-Driven Infrastucture with Chef on selvitetty hyötyjä, joita saavutetaan lähestymistavalla, jossa infrastruktuuria ylläpidetään ohjelmankoodin muodossa. Saavutettaviksi hyödyiksi on listattu toistettavuus, automatisointi, ketteryys, skaalautuvuus, varmuus ja ongelmatilanteista toipuminen. [7, s. 3–4.]

Infrastruktuurikuvausten ollessa ohjelmakoodia, saadaan ympäristöjen rakentaminen automatisoitua. Luonti on toistettavaa eli samanlaisia ympäristöjä voidaan rakentaa useita ja niitä voidaan tarvittaessa luoda uudelleen. Infrastruktuurin skaalautuvuus paranee, kun uusia sovelluspalvelimia saadaan luotua automatisoidusti. Ohjelmakoodia voidaan ylläpitää versionhallintajärjestelmässä, jolloin muutosten tekemisestä ja seurannasta tulee hallittua. Ohjelmakoodi myös dokumentoi infrastruktuurin, mikä pienentää riskiä siitä, että infrastruktuurin rakenne on vain muutamien ihmisten tiedossa. [7, s. 3–4.]

Konfiguraationhallintatyökalun käyttöönotto ei välttämättä ole järkevä ratkaisu kaikissa tilanteissa. Uusien työkalujen käyttöönotto vaatii työtä ja niiden käyttö opettelua sekä mahdollisesti uudenlaisen ajatustavan omaksumisen. Mahdollisesti palvelimia on ollut jo pitkään tuotannossa ja niihin on tehty erilaisia muutoksia, joita ei kaikkia välttämättä ole edes dokumentoitu toistettaviksi. Konfiguraatioiden luominen työkalujen ymmärtämään muotoon vaatii ohjelmointiosaamista, jota ei välttämättä ylläpitäjillä ennestään ole riittävästi.

Konfiguraationhallintatyökaluissa on eroja, joten käyttötarkoitukseen sopivan työkalun valintaan kannattaa kiinnittää huomiota. Yhtenä tärkeänä eri työkaluja erottavana ominaisuutena on konfiguraatioiden määrittämiseen käytettävä kieli. Useilla työkaluilla on oma niille kehitetty kielensä, jolloin työkalun käyttäminen vaatii kielen ja erilaisten käsitteiden opettelun. Tämä tulee huomioida myös konfiguraatioiden dokumentoinnin ja ylläpidon osalta.

Kuvassa 2.3 on Continuous Delivery -kirjassa määritelty kypsyymalli, jolla voidaan analysoida konfigurointi- ja toimitusprosessin tilaa. Kypsyysmalli on jaettu kuuteen eri osa-alueeseen: käännösten hallinta ja jatkuva integrointi; ympäristöt ja

toimitus; julkaisujen hallinta ja noudattaminen; testaus; datan hallinta sekä konfiguraation hallinta.

Practice	Build management and continuous integration	Environments and deployment	Release management and compliance	Testing	Data management	Configuration management
Level 3 - Optimizing: Focus on process improvement	Teams regularly meet to discuss integration problems and resolve them with automation, faster feedback, and better visibility.	All environments managed effectively. Provisioning fully automated. Virtualization used if applicable.	Operations and delivery teams regularly collaborate to manage risks and reduce cycle time.	Production rollbacks rare. Defects found and fixed immediately.	Release to release feedback loop of database performance and deployment process.	Regular validation that CM policy supports effective collaboration, rapid development, and auditable change management processes.
Level 2 - Quantitatively managed: Process measured and controlled	Build metrics gathered, made visible, and acted on. Builds are not left broken.	Orchestrated deployments managed. Release and rollback processes tested.	Environment and application health monitored and proactively managed. Cycle time monitored.	Quality metrics and trends tracked. Non functional requirements defined and measured.	Database upgrades and rollbacks tested with every deployment. Database performance monitored and optimized.	Developers check in to mainline at least once a day. Branching only used for releases.
Level 1 - Consistent: Automated processes applied across whole application lifecycle	Automated build and test cycle every time a change is committed. Dependencies managed. Re-use of scripts and tools.	Fully automated, self-service push-button process for deploying software. Same process to deploy to every environment.	Change management and approvals processes defined and enforced. Regulatory and compliance conditions met.	Automated unit and acceptance tests, the latter written with testers. Testing part of development process.	Database changes performed automatically as part of deployment process.	Libraries and dependencies managed. Version control usage policies determined by change management process.
Level 0 – Repeatable: Process documented and partly automated	Regular automated build and testing. Any build can be re-created from source control using automated process.	Automated deployment to some environments. Creation of new environments is cheap. All configuration externalized / versioned	Painful and infrequent, but reliable, releases. Limited traceability from requirements to release.	Automated tests written as part of story development.	Changes to databases done with automated scripts versioned with application.	Version control in use for everything required to recreate software: source code, configuration, build and deploy scripts, data migrations.
Level -1 – Regressive: processes unrepeatable, poorly controlled, and reactive	Manual processes for building software. No management of artifacts and reports.	Manual process for deploying software. Environment-specific binaries. Environments provisioned manually.	Infrequent and unreliable releases.	Manual testing after development.	Data migrations unversioned and performed manually.	Version control either not used, or check-ins happen infrequently.

Kuva 2.3: Konfigurointi- ja toimitusprosessin kypsyysmalli [3, s. 419].

Kypsyyttä arvioidaan viisitasoisella asteikolla, joista alin taso on taantuva (regressive). Tällä tasolla prosessit eivät ole toistettavia. Muun muassa käännökset, konfigurointi ja asennukset tehdään manuaalisesti. Seuraava taso on toistettava (repeatable). Prosessit ovat dokumentoituja ja työ on osittain automatisoitua. Konfiguraatiot ovat versioituja, automaattitestejä luodaan osana kehitystyötä ja tietokannan päivittäminen tehdään automatisoiduilla skripteillä. Keskimmäisellä tasolla pyritään vakauteen (consistent), missä prosessit ovat automatisoituja koko sovelluksen elinkaaren osalta. Toiseksi korkeimmalla tasolla prosessit ovat mitattavasti hallittuja (quantitatively managed). Käännöksistä kerätään metriikoita, jotka tehdään näkyviksi. Julkaisuprosessit ovat testattuja, ympäristöjä monitoroidaan ja myös eitoiminnallisia ominaisuuksia mitataan. Korkein taso on optimoiva (optimizing). Tasolla keskitytään prosessien parantamiseen ja ongelmien ratkaisemiseen muun muassa automatisoinnilla, vasteaikojen lyhentämisellä ja näkyvyyden parantamisella. [3, s. 419–421.]

2.6 Työkalut

Konfiguraationhallinnan automatisointiin on saatavilla useita työkaluja. Tunnettuja työkaluja ovat muun muassa CFEngine [8], Puppet [9], Chef [10] sekä Ansible [11]. Työkalun valinta on suureksi osaksi mielipidekysymys, johon vaikuttaa esimerkiksi tiimin mieltymykset käytettyyn kieleen, tyyliin ja kehittäjäyhteisöön. Konfiguraationhallintatyökalujen tarkoitus on pääsääntöisesti helpottaa työtä tarjoamalla apuvälineitä toistettavien ja automatisoitavissa olevien tehtävien suorittamiseen.

Osa työkaluista on ollut olemassa pidempään kuin toiset ja saattavat siten olla kypsempiä muun muassa dokumentaation osalta. Työkalujen toimintaperiaatteet vaihtelevat, ja siten jotkin työkalut saattavat sopia projektitiimin toimintamalleihin paremmin kuin toiset. Kehittäjäyhteisöjen aktiivisuus vaikuttaa muun muassa valmiiden konfiguraatiopakettien määrään ja laatuun. Joillakin työkaluilla on helppoa tehdä yksinkertaisia konfiguraatioita, mutta ne eivät välttämättä sovellu yhtä hyvin monimutkaisempien konfiguraatioiden muodostamiseen ja hallintaan.

Konfiguraationhallintatyökalun käyttöönotto valmiiseen ympäristöön on mahdollista, mutta aiheuttaa todennäköisesti ylimääräistä työtä valmiina olevien konfiguraatioiden ja itse tehtyjen asennusskriptien vuoksi. Useimmat konfiguraatiotyökalut kuitenkin mahdollistavat muun muassa komentoriviskriptien suorittamisen osana automatisoitua konfiguraationhallintaa. Tämä helpottaa siirtymää, joskin muuttamalla aikaisemmat skriptit konfiguraationhallintatyökalun käyttämään muotoon voidaan saavuttaa hyötyjä. Cory Lueninghoener ;login:-lehden artikkelissa listaa saavutettaviksi hyödyiksi kattavat raportit, testattavuuden, abstrahoinnin sekä ennustettavamman toiminnan. [12.]

Tässä kohdassa on esitelty konfiguraationhallinnan automatisointiin liittyviä erilaisia työkaluja. CFEngine, Puppet, Chef ja Ansible ovat konfiguraationhallinnan automatisointiin tarkoitettuja työkaluja. Lisäksi on käsitelty vaihtoehtoinen tapa toteuttaa automatisointia skriptien avulla. Viimeisessä alakohdassa on esitelty kehitys- ja testaustyötä auttavia tukityökaluja.

2.6.1 CFEngine

CFEngine on samannimisen yrityksen omistama konfiguraationhallintatyökalu, jonka ensimmäisen version on tehnyt CFEnginen perustaja Mark Burgess. Ensimmäinen versio julkaistiin vuonna 1993 Oslon yliopistossa. Lähtökohtana CFEnginen synnylle oli helpottaa palvelinten ylläpitoa. [13.] CFEngine on kaksoislisensoitu CFEnginen kaupallisella vapaan lähdekoodin lisenssillä (COSL) sekä GPL-lisenssin versiolla kolme [14].

CFEngine soveltuu järjestelmien luomisen lisäksi niiden ylläpitoon. CFEngineä voidaan käyttää pienissä sulautetuissa laitteissa, palvelimissa ja pilvipalveluissa useiden tuhansien laitteiden hallintaan [8]. CFEnginestä on saatavilla avoimen lähdekoodin GPL-lisensoitu Community-versio ja kaupallinen Enterprise-versio, jossa on muun muassa parempi tuki ja enemmän monitorointiominaisuuksia [15].

CFEnginessä konfigurointi tapahtuu CFEngineen kehitetyllä omalla DSL-kielellä [16]. CFEnginen toiminta perustuu asiakas–palvelin-malliin, missä konfiguraatioasetukset ovat palvelimella, josta asiakassovellukset noutavat ne suoritusta varten [17]. CFEngine on tuettuna useille eri alustoille, joihin kuuluvat muun muassa muutamat eri Linux-distribuutiot, Solaris-järjestelmät ja Windows Server -versiot [18].

2.6.2 Puppet

Puppet on Puppet Labs -yrityksen luoma konfiguraationhallintajärjestelmä, jolla voidaan kuvata IT-infrastruktuurin haluttu tila, minkä Puppet automaattisesti asetettaa käytäntöön. Puppet soveltuu useiden tuhansien fyysisten ja virtuaalisten palvelinten konfiguraationhallintaan. Ensimmäinen Puppet-versio julkaistiin vuonna 2005 GPL-lisenssillä. Puppetin versiossa 2.7 lisenssi vaihtui Apache 2.0 -lisenssiin. [9; 19.]

Puppet perustuu asiakas–palvelin-malliin, jossa konfigurointi tapahtuu Puppetin omalla DSL-kielellä. Kieli on deklaratiivinen, mikä saattaa vaatia totuttelua sovelluskehittäjiltä, jotka ovat tottuneet proseduraalisiin ohjelmointikieliin. Puppetiin on vapaasti saatavilla tuhansia valmiita moduuleita yleisimpien ylläpitotoimien automatisointiin. [9.] Puppetin versiosta 2.6 alkaen konfiguraatioita on ollut mahdollista kirjoittaa myös Ruby-ohjelmointikielellä [20]. Kielten käyttäminen sekaisin saattaa kuitenkin heikentää ylläpidettävyyttä ja vaatia kehittäjiltä sekä Rubyn että Puppetin oman DSL-kielen osaamista.

Puppet on tuettuna useille eri alustoilla, joihin kuuluu muun muassa eri Linux- ja BSD-distribuutiota, Mac OS X sekä useita Windows-versiota [21]. Puppetista on saatavilla ilmainen vapaan lähdekoodin versio sekä Enterprise-versio, joka on maksullinen yli kymmenelle solmulle [22]. Enterprise-versio tarjoaa muun muassa integraatiot VMWare-virtualisointiohjelmistoon [23] ja Amazonin EC2 -pilveen [24] sekä automatisoinnin muun muassa DNS- ja NTP-palvelimien konfiguroinnille [25].

2.6.3 Chef

Chef on avoimen lähdekoodin konfiguraationhallintatyökalu, jonka ensimmäinen julkaistu versio 0.5.1 ilmestyi vuonna 2009 [26]. Chef on lisensoitu Apache 2.0 -lisenssillä [27]. Chefistä on tarjolla ilmainen versio sekä erilaisilla lisenssimalleilla tarjottava Enterprise-versio. Enterprise-versioon kuuluu muun muassa päivystystuki, hallintakonsoli ja mahdollisuus käyttää Opscoden ylläpitämiä Chef-palvelimia. Molemmissa versioissa on tuki suurimmille pilvipalvelutarjoajille. [10.]

Chefin syntyyn on vaikuttanut kehittäjien kokemukset Puppet-projektin [9] käytöstä ja toiminnasta. Chefillä ja Puppetilla ei kuitenkaan ole yhteistä koodia. Tärkeimpinä eroina Puppetiin on Chefin sivuston mukaan se, että Chef käyttää konfigurointikielenään puhdasta Rubya eikä työkalulle kustomoitua omaa kieltä. Chef on suunniteltu integroitumaan muihin työkaluihin, eli Chef ei pyri kuvaamaan koko infrastruktuuria kanonisessa muodossa, vaan ennemmin tarjoamaan tietoa infrastruktuurista palveluna. Chefin asiakassovellus suorittaa konfiguroinnin Chefin *resepteissä* kuvatussa järjestyksessä, eli jokaisella asiakassovelluksen suorituksella resurssit konfiguroidaan aina samassa järjestyksessä. Resursseilla voi myös olla suoritettavia toi-

mintoja, kuten web-palvelimen käynnistäminen. Sama resurssi voi esiintyä useaan kertaan reseptissä, jolloin esimerkiksi attribuuttina oleva web-serverin uudelleenkäynnistäminen tulee suoritetuksi vain kertaalleen, vaikka se asetettaisiin resurssille useaan kertaan. [28.]

Chefissä reseptit ja muu konfiguraatiototeutus paketoidaan niin sanottuihin *cookbookeihin*. Cookbookeissa asioita voidaan tehdä monella eri tavalla, mikä mahdollistaa kehittäjille vapauden erilaisiin toteutustapoihin. Tämä saattaa aiheuttaa myös ongelmia, koska mahdollisesti vain tietty toteutustapa on niin sanotusti oikea, ja muiden toteutustapojen sopimattomuus voi ilmetä vasta myöhemmässä kehitysvaiheessa konfiguraation munimutkaistuessa. Vaihtelevat ja epäjohdonmukaiset toteutustavat myös heikentävät konfiguraation luettavuuttaa ja saattavat vaatia syvempää perehtymistä konfiguraation ymmärtämiseksi. Chefin toimintaa on kuvattu tarkemmin luvussa 3.

2.6.4 Ansible

Ansible on Michael DeHaanin vuonna 2012 käynnistämä projekti. Michael DeHaanin mielestä Ansiblelle oli tarvetta, koska muut konfiguraationhallintatyökalut ovat vain onnistuneet siirtämään monimutkaisuuden IT-infrastruktuurin hallinnasta muualle. Hänen mielestään sen hetkisillä konfiguraationhallintatyökaluilla aikaa kului liikaa itse konfiguraationhallintatyökalujen hallintaan. [29.]

Ansiblen konfigurointi tapahtuu YAML-merkintäkielellä kirjoitetuilla *playbookeilla*, jotka määrittelevät tarvittavat konfigurointitehtävät. Ansible ei käytä asiakas–palvelin-mallia, mikä poikkeaa monista muista konfiguraationhallintajärjestelmistä. Ansiblen toiminta perustuu pieniin ohjelmiin, joita kutsutaan moduuleiksi. Nämä moduulit ovat resurssimalleja, jotka kuvaavat halutun tilan järjestelmälle. Ansible suorittaa nämä moduulit esimerkiksi SSH-yhteyden yli ja poistaa ne suorituksen päätyttyä. Koska erillistä asiakassovellusta ei ole, konfiguroitavalta solmulta vaaditaan vain SSH-palvelin ja Python-tulkki moduulien suorittamista varten [30]. [11.]

Ansible on avoimen lähdekoodin työkalu, joka on julkaistu GPL-lisenssin versiolla kolme [31]. Ansible-konfiguraationhallintatyökalun tukipalveluiden lisäksi Ansible-yritys tarjoaa Ansible Tower -nimistä työkalua. Se on maksullinen solmujen hallintaan tarkoitettu sovellus, jossa on muun muassa graafinen käyttöliittymä, roolipohjainen pääsynhallinta sekä mahdollisuus tehtävien ajastamiseen. Ansible Towerissa on myös REST-rajapinta ja komentorivityökalu, joiden avulla sovellus voidaan integroida olemassa oleviin työkaluihin ja prosesseihin. Työkalu on ilmainen, mikäli hallittavia solmuja on korkeintaan kymmenen kappaletta. [32.]

2.6.5 Skriptit

Konfiguraationhallinnan automatisointia voidaan toteuttaa myös ilman valmiita tarkoitukseen tehtyjä konfiguraationhallintatyökaluja. Konfiguraationhallintatyökalut saattavat tehdä muutoin yksinkertaisesta konfiguroinnista hankalampaa, koska työkalut vaativat opettelua sekä usein myös erillisten konfiguraatiopalvelinten luomisen ja ylläpidon. Yksinkertaista tehtävien automatisointia voidaan tehdä esimerkiksi komentoriviskripteillä. Esimerkiksi Unix-ympäristöissä voidaan käyttää Shell-skriptejä ja GNU core utilities -pakkaukseen kuuluvia perustoimintoja toteuttavia työkaluja [33].

Konfiguraatioiden monimutkaistuessa ja erilaisiin virhetilanteisiin varautuessa, komentoriviskriptien ylläpidettävyys voi heiketä. Komentoriviskriptit saattavat syntyä tarpeesta jonkin yksinkertaisen asian automatisointiin, ja muuttua sittemmin monimutkaisemmiksi ajan kuluessa ja erilaisten vaatimusten lisääntyessä. Skriptit voidaan kirjoittaa yleisesti hyviksi todettuja ohjelmointityylejä noudattaen ja huomioiden mahdollinen tarve erilaisille ympäristöille sekä asetuksille. Tämä vaatii kuitenkin ennakointia ja sovittuja käytäntöjä, joita kaikki kehittäjät noudattavat.

Skriptien käyttö saattaa olla ennestään tuttua useille kehittäjille ja ylläpitäjille, eikä siten vaadi välttämättä erikseen opettelua, toisin kuin uusien konfiguraationhallintatyökalujen käyttöönotto. Mahdollisesti ympäristöjen ylläpito ja konfigurointivastuu on henkilöillä, jotka ymmärtävät laitteistoista ja järjestelmistä, mutta joilla ei ole ohjelmointiosaamista. Silti skriptit ja käyttöjärjestelmien perustyökalut saattavat olla ennestään tuttuja.

Skriptit ja niissä käytettävät työkalut, kuten pakkaustenhallinta, ovat yleensä alustariippuvaisia. Skripteissä tulee siis huomioida erikseen kaikki alustat, joissa skriptejä suoritetaan, sekä tarvittavien aputyökalujen saatavuus ja asennus eri alustoilla. Tällöin ei kuitenkaan ympäristöön välttämättä tarvitse asentaa erillisiä konfigurointityökaluja, joita monet konfiguraationhallintatyökalut vaativat. Sovelluspalvelimet voidaan pitää yksinkertaisempina, koska niiden pääasiallisen toiminnan kannalta epäolennaisia sovelluksia on vähemmän.

2.6.6 Tukityökalut

Konfiguroinnin automatisointi erilaisilla työkaluilla on jossakin määrin verrattavissa ohjelmistokehitykseen. Ohjelmistokehityksessä on oleellista esimerkiksi toteutuksen testaaminen, kuten on myös konfiguraationhallinnassa. Konfiguraatioiden kehittämiseen ja testaamiseen on olemassa erilaisia työkaluja, jotka tukevat ja helpottavat työtä.

Konfiguraationhallinnan kannalta kehitys- ja testaustarkoituksiin soveltuvat ympäristöt on suositeltavaa olla erillään tuotantoympäristöistä, koska ympäristöihin

tehdyt muutokset eivät ole automaattisesti peruttavissa. Konfiguraatioiden kehitykseen voidaan käyttää kehittäjän työasemalla olevia virtuaalikoneita, joiden toteutukseen soveltuu esimerkiksi useille alustoille tuettu Oraclen Virtualbox [34]. Virtualisointisovelluksen ja konfiguraationhallintatyökalun yhteistoiminnan ja työnkulun helpottamiseksi voidaan käyttää Vagrantia [35]. Vagrantin avulla voidaan automatisoida virtuaalikoneiden luonti ja luotujen koneiden provisiointi konfiguraationhallintatyökaluilla [36]. Määrittelemällä Vagrantin konfiguraatiotiedostoon käytettävän alustan, provisiointityökalun ja sen asetukset, voidaan yksittäisellä komennolla luoda ja provisioida yksi tai useampi virtuaalikoneinstanssi.

Vagrantin konfiguraatioiden luonti ja hallinta voidaan automatisoida käyttämällä Test Kitchen -työkalua. Test Kitchen tukee useita pilvipalveluita sekä virtualisointisovelluksia, kuten Amazonin EC2 -pilveä [24], Virtualboxia [34] ja Dockeria [37]. Test Kitchenin konfigurointi tehdään YAML-merkintäkielellä, jolla voidaan määrittää mitä alustoja käytetään esimerkiksi web- ja tietokantapalvelimiin. Test Kitchen luo eri kombinaatiot alustoista ja palvelinrooleista, luo tarvittavat palvelininstanssit sekä provisioi ne automaattisesti. Tämä saattaa helpottaa, jos tuettavia alustoja ja palvelinrooleja on useita. [38.]

Chefin kehitystyötä helpottamaan voidaan käyttää työkalua nimeltä Berkshelf, jolla hallitaan Chefin konfiguraatiomäärittelyjä sisältäviä cookbookeja ja niiden riippuvuuksia. Berkshelf asentaa työasemalle luotavaan varastoon konfiguraatioissa riippuvuuksina vaadittuja cookbookeja. Varastoon lisätyt cookbookit voivat olla itse toteutettuja tai valmiita toteutuksia, jotka ladataan ulkoisista lähteistä. [39.]

Konfiguraatioiden automaattitestien luomiseen ja suorittamiseen voidaan käyttää Serverspeciä. Serverspecillä luodaan RSpecillä [40] suoritettavia testitapauksia, joilla varmistetaan että konfigurointi on tehty oikein. Testeillä voidaan tarkastaa esimerkiksi, että palvelin kuuntelee tiettyä porttia tai tietty tiedosto on olemassa. Serverspecillä luotavat testit ovat syntaksiltaan alustariippumattomia ja niitä voidaan suorittaa kaikilla tuetuilla alustoilla ilman sovellusten asennusta. [41]

Useat, varsinkin Chefiin liittyvät, kehitystyötä auttavat tukityökalut on toteutettu Rubyllä. Rubyssä ulkoisten kirjastojen ja ohjelmien hallinta tapahtuu pakkauksilla, joita kutsutaan gemeiksi. Gemit ovat versioituvia ja voivat riippua toisistaan [42]. Versioristiriidat saattavat aiheuttaa ongelmia kehitystyössä gemien päivittyessä. Eri gemit saattavat riippua jonkun tietyn gemin eri versioista, eivätkä toimi muilla versioilla. Välttämättä gemistä ei ole yhteistä versiota, joka täyttäisi molempien riippuvuusehdot. Tämä voi aiheuttaa ongelmia, koska samasta gemistä ei voi olla projektissa käytössä useampia eri versioita. Joitakin riippuvuusongelmia voidaan välttää käyttämällä gemien hallintaan Bundler-työkalua, joka tekee projektille oman gemikokoelman eikä käytä gemien järjestelmälaajuisia versioita [43].

Kehitys- ja testaustyötä on mahdollista helpottaa erilaisilla tukityökaluilla. Usei-

den työkalujen käyttö voi kuitenkin vaikeuttaa varsinaisten konfiguraatioiden ymmärtämistä, koska automaattisesti toimivat työkalut piilottavat toiminnallisuutta ja luovat varsinaisen konfiguraationhallinnan kannalta epäolennaista toiminnallisuutta. Kehittäjän tulee tuntea käytettävien tukityökalujen toiminta ja merkitys, jotta kokonaisuus on ymmärrettävissä.

3. CHEF

Chef on Opscoden kehittämä avoimen lähdekoodin konfiguraationhallintatyökalu, jonka tavoitteena on muuttaa infrastruktuuri ohjelmakoodiksi. Chefin avulla infrastruktuurista saadaan versioituva, testattava ja toistettava. Chefissä konfigurointi tehdään Ruby-ohjelmointikielellä, jota on laajennettu Chefin omalla DSL-kielellä resurssien kuvauksen osalta. Pääperiaatteena Chefissä on, että käyttäjä tietää parhaiten millaisia ympäristöjen tulee olla, miten niiden tulee toimia ja kuinka niitä hallitaan. Käyttäjä ja tiimi ymmärtävät parhaiten sovelluskohteen tekniset haasteet ratkaisutapoineen sekä organisaation ihmisten väliset sovitut toimintamallit. [44.]

Tässä luvussa perustellaan miksi Chef valittiin konfiguraationhallintatyökaluksi diplomityön sovelluskohteelle, esitellään Chefin konfiguraatioinnin toimintamalli sekä eri komponenttien toiminnallisuutta.

3.1 Miksi Chef?

Tähän diplomityöhön konfiguraationhallintatyökaluksi valittiin Chef, koska se koettiin monipuoliseksi ja nykyaikaiseksi työkaluksi, johon tutustumisesta uskotaan olevan hyötyä tulevien Solitan projektien yhteydessä. Kyseistä työkalua on käytetty myös muissa Solitan projekteissa, joten tarvittaessa on mahdollista saada apua ja ohjausta muilta Solitan työntekijöiltä.

Ruby-kielellä toteutettava konfigurointi toimi yhtenä perusteena Chefin valinnalle tässä diplomityössä. Ruby on syntaksiltaan ennestään tuttu useille kehitystiimin jäsenille, mikä mahdollisesti helpottaa jatkossa konfiguraatioiden ylläpitoa. Ohjelmointiin tarkoitetun kielen käyttö myös mahdollistaa omien ominaisuuksien toteuttamisen, jos jostakin syystä Chefin ominaisuudet eivät ole riittävät tai eivät sovellu kaikkiin sovelluskohteen vaatimiin tilanteisiin.

Chefissä käytettävä asiakas–palvelin-malli koettiin myös olevan positiivinen ominaisuus Chefissä verrattuna suoraan työasemalta tapahtuvaan provisiointiin. Tällöin palvelinten provisiointi ei riipu suoraan kehittäjien työasemista ja niissä käytettävistä työkalujen versioista. Tämä mahdollistaa tarvittaessa myös palvelinten provisioinnin ajastetusti työajan ulkopuolella. Ilman asiakas–palvelin-mallia tämän voisi toteuttaa esimerkiksi käyttäen jatkuvaan integrointiin tarkoitettuja palvelimia ja niihin luotavia ajastettuja tehtäviä.

Chefin käyttämä pull-periaate voidaan myös nähdä eduksi. Pull-periaate tarkoit-

taa tässä yhteydessä mallia, jossa asiakassolmut vastaavat omasta provisioinnistaan itse noutamalla konfigurointiin tarvitsemansa resurssit palvelimelta. Uusi palvelin on mahdollista rekisteröidä konfiguraationhallinnan piiriin ilman erillistä konfiguraatiota konfiguraatiopalvelimelle. Vastakohtana pull-periaatteelle on muun muassa Ansiblessa käytettävä push-periaate, jossa eksplisiittisesti käynnistettävän provisioinnin yhteydessä asiakassolmuille välitetään kaikki provisiointiin tarvittavat resurssit. Tällöin provisioinnin yhteydessä tulee olla erikseen määriteltynä mitkä asiakassolmut provisioidaan. Tämän diplomityön ja sovelluskohteen osalta tosin ero push- ja pull-periaatteiden välillä on lähinnä teoreettinen, koska sekä palvelin että asiakassolmut ovat saman kehitystiimin hallinnassa.

Chef-yhteisö on aktiivinen ja tarjolla on paljon valmiita konfiguraatiototeutuksia, joita on mahdollista hyödyntää lisensoinnin sallimissa rajoissa. Muun muassa Chef Supermarketissa on tarjolla useita monilla eri alustoilla toimivia cookbookeja, joista suuri osa on lisensoitu Apache 2.0 -lisenssillä [45]. Chefille on saatavilla myös kattava dokumentaatio sekä useita esimerkkitoteutuksia erilaisista konfiguraatioista.

Cookbookit ovat versioituvia, mikä mahdollistaa riippuvuuksien kohdistumisen tiettyihin ja toimiviksi testattuihin cookbookeihin. Tämän avulla voidaan pienentää riskiä mahdollisten taaksepäin yhteensopimattomien muutosten tulemiseen osaksi toteutusta. Versioituvuus on hyödyllinen ominaisuus käytettäväksi myös itsetehdyissä cookbookeissa, koska siten voidaan varmistaa konfiguraatioiden olevan asennettavien sovellusversioiden mukaisia.

3.2 Toiminta

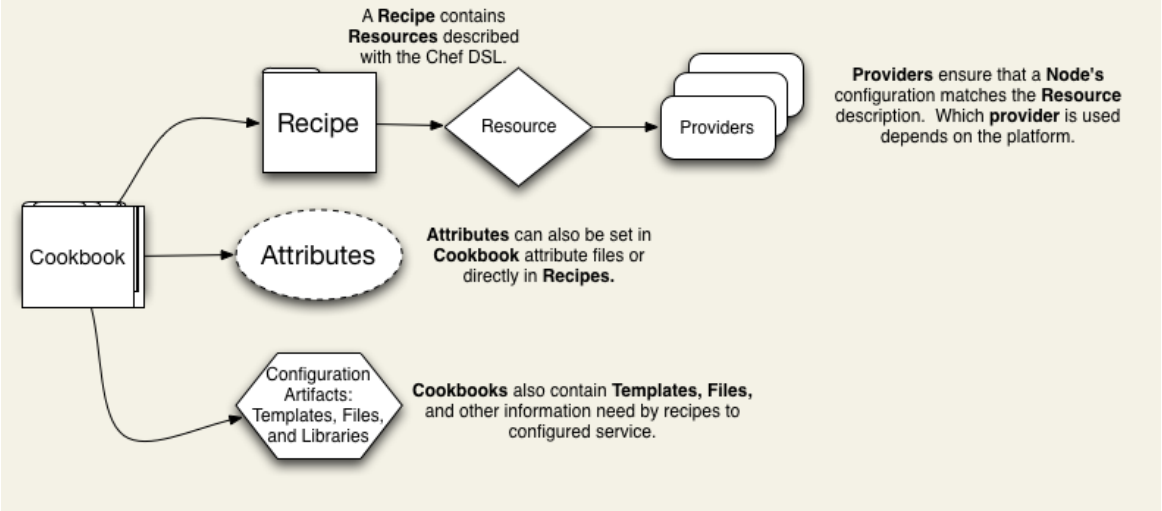
Chefin toiminta perustuu niin sanottuihin resepteihin, jotka ovat uudelleenkäytettäviä määrittelyjä infrastruktuurin luomiseen tarvittaville tehtäville. Resepteillä voidaan kuvata kuinka luodaan esimerkiksi web-palvelin, tietokanta ja kuormantasaus. Yhdessä eri reseptit kuvaavat kuinka infrastruktuuri luodaan, konfiguroidaan ja hallitaan. Reseptit koostuvat rakennuspalasista, joita kutsutaan resursseiksi. Resurssit kuvaavat tilan jollekin infrastruktuurin osalle. [10.]

Chefin resursseissa määritettyjen tehtävien suorittamista tietylle asiakassolmulle kutsutaan provisioinniksi. Chefin suorittama provisiointi on idempotenttia, eli reseptin suorittaminen useaan kertaan johtaa aina samaan tulokseen. Chef Client -sovellus huolehtii siitä, että mitään toimenpiteitä ei tehdä, jos resurssit eivät ole muuttuneet edellisen provisioinnin jälkeen. [28.]

Chefillä on mahdollista luoda alustariippumattomia konfiguraatioita. Toiminta perustuu asiakassolmuilla olevaan Ohai-työkaluun. Ohai kerää tietoja alustasta Chef Client -sovelluksen käyttöön. Näitä tietoja ovat alustan tyypin lisäksi muun muassa tiedot suorittimesta, muistista ja verkon käytöstä. Alustariippumattomien konfiguraatioiden luonti vaatii alustaerojen huomioimista resepteissä, joskin monet peruse-

roavaisuudet on abstrahoitu Chefin DSL-kielen puolesta. [46.]

Kuvassa 3.1 on esitetty Chefin cookbookin rakenne. Cookbookit ovat kokoelmia resepteistä (recipe), konfiguraatiota ohjaavista attribuuteista (attribute) sekä artefakteista (configuration artifact). Reseptit määrittävät resursseja (resource), jotka suoritetaan alustakohtaisilla toteuttajilla (provider). Attribuutteja voivat olla esimerkiksi porttinumero, jota web-palvelin kuuntelee, tai URL, josta asennettava sovelluspaketti noudetaan. Artefakteja ova esimerkiksi palomuurin asetustiedosto, joka asetetaan sovelluspalvelimelle tai tietokanta-ajurien asennuspaketti.



Kuva 3.1: Cookbookin rakenne [47].

Chefin reseptit kuvaavat toimivia konfiguraatiota koostamalla joukon resursseja yhteen. Reseptejä voidaan lisätä suorituslistaan, jolloin kohdistamalla suorituslistan tietylle solmulle, saadaan määriteltyä tila, johon kyseinen solmu konfiguroidaan. [48.]

Reseptit ovat Ruby-ohjelmointikielisiä tiedostoja, jotka koostuvat pääasiassa Chefin resurssien kuvauksista. Resursseilla on tyyppi, nimi, toiminto sekä tyypin mukaan vaihteleva lista mahdollisia attribuutti–arvo-pareja. Resepteihin voidaan sisällyttää (include) muita resepti-tiedostoja samaan tapaan kuin perinteisiin Ruby-kielisiin tiedostoihin voidaan sisällyttää muita Ruby-tiedostoja. Resepteissä voidaan tehdä myös Chef-palvelimen avulla suoritettavia hakuja, joiden tulokset ovat käytettävissä resepteissä. Koska reseptit ovat Ruby-kielisiä tiedostoja, ne voivat sisältää Ruby-koodia, kuten funktioita, muuttujia, ehto-lauseita ja silmukoita. [49; 50.]

Chefissä resurssilla tarkoitetaan yksittäistä lausetta konfiguraatiossa. Se määrittelee halutun tilan jollekin infrastruktuurin elementille, sekä vaiheet, joilla kohde saadaan haluttuun tilaan. Resurssin kohde voi olla esimerkiksi tiedosto, malli (template), hakemisto, pakkaus tai suoritettava komento. Jokaisella resurssilla on tyyppi, nimi, erilaisia resurssia kuvaavia attribuutteja sekä toiminto, joka määrittelee kuinka Chef Client -sovellus toteuttaa resurssin osaksi konfiguraatiota. [48.]

Asiakassolmun tilan saattamiseen senhetkisestä tilasta resurssien kuvaamaan tilaan vastaavat erilaiset toteuttajat (provider). Toteuttajat ovat riippuvaisia sovelluspalvelimen alustasta sekä alustan versiosta. Tiedot asiakassolmun alustasta kerätään Chef Clientille Ohai-työkalulla. Näistä tiedoista Chef Client automaattisesti päättelee, mitä toteuttajaa provisiointiin käytetään. Toteuttajasta luodaan instanssi, tutkitaan resurssin senhetkinen tila asiakassolmulla, tehdään tarvittavat muutostoimet ja merkitään resurssi päivitetyksi, mikäli muutoksia tehtiin. [48.]

Ruby-kielistä logiikkaa resepteihin toteutettaessa on huomioitava, että Chefin reseptien suorituksessa on kaksi vaihetta: kääntäminen ja suorittaminen. Kääntövaiheessa reseptit suoritetaan Ruby-koodina, jolloin resurssit lisätään toteuttajalla (provider) suoritettavaan listaan. Suoritusvaiheessa Chef suorittaa sisäisesti valitsemallaan toteuttajalla resursseissa kuvatut toiminnot. Tämän kaksivaiheisuuden vuoksi resepteihin ei voi sisällyttää resurssien suorituksesta riippuvaa Ruby-kielistä ehdollisuuslogiikkaa, kuten: ”Jos resurssi A suoritettiin, resurssin B määrittely on X, muutoin Y.” Tämänkaltainen ehdollisuus on kuitenkin mahdollista toteuttaa resurssien yleisillä suojain-attribuuteilla (guard) *not_if* ja *only_if*, joiden evaluointi tehdään vasta suoritusvaiheessa. [51.]

Kuvassa 3.2 on esimerkkinä *cookbook_file*-tyyppisen resurssin kuvaus. Chef Client valitsee toteuttajan tyypin määrittelemälle resurssille sovelluspalvelimen mukaisella alustatyypillä ja -versiolla. Lähdetiedosto *oracle.conf* on cookbookissa oleva tiedostoartefakti, josta on mahdollista olla myös eri versioita eri alustoille. Toteuttaja tarkastaa asiakassolmulta onko polun */etc/sysctl.d/60-oracle.conf* kuvaama tiedosto olemassa, tai onko sillä eri tarkistussumma kuin cookbookissa olevalla tiedostoartefaktilla. Jos tarkistussumma eroaa tai tiedosto puuttuu, toteuttaja kopioi tiedostoartefaktin asiakassolmulle resurssissa määritettyyn polkuun. Tiedostolle asetetaan attribuutilla *mode* kuvatut oikeudet (omistajalle luku- sekä kirjoitusoikeus ja muille vain lukuoikeus). Mikäli toteuttaja on kopioinut tiedoston tai muuttanut sen oikeuksia, merkitään resurssi päivitetyksi. [52.]

```
cookbook_file '/etc/sysctl.d/60-oracle.conf' do
  action :create
  source 'oracle.conf'
  mode 644
end
```

Kuva 3.2: Esimerkki resurssin kuvauksesta Chefin reseptissä.

Chefissä roolit (role) ovat tapa kuvata konfigurointivaiheet, jotka tulee olla tehtynä, jotta asiakassolmu voi toimia roolin mukaisessa tehtävässä. Rooli kuvataan JSON- tai Ruby-tiedostolla. Esimerkiksi roolina voi olla web-palvelin, jolloin roolissa määritellään, että solmulle on suoritettava web-palvelin-resepti, ja että web-

palvelimen tulee kuunnella roolissa attribuuttina asetettua porttia. Roolit voivat sisältää attribuutteja sekä suorituslistan (runlist), jossa on listattuna suoritettavat reseptit ja tarvittavat muut roolit. Asiakassolmun konfiguraation ei tarvitse koostua roolista, mutta voi tarvittaessa koostua useistakin eri rooleista. [53.]

Chefin ympäristöt (environment) ovat tapa kuvata eri tarkoituksiin olevia ympäristöjä, kuten kehitys-, testaus-, koekäyttö- ja tuotantoympäristö. Ympäristöt kuvataan JSON- tai Ruby-tiedostolla. Jokaisella asikassolmulla on oltava ympäristö. Jos ympäristöä ei erikseen ole asetettu, on ympäristönä *_default*, jota ei voi muokata. Ympäristömäärittelyillä voidaan asettaa attribuutteja ja ympäristössä käytettävien cookbookien versiot. [54.]

Chefin attribuutit (attribute) sisältävät arvoja, joita käytetään konfiguroinnissa. Attribuutteja voidaan asettaa resepteissä, cookbookien attribuutti-tiedostoissa, rooleissa ja ympäristöissä. Lisäksi attribuuttien arvoja asetetaan automaattisesti kuvaamaan tietoja provisioitavasta solmusta Ohain ja Chef Serverin avulla. Attribuuteille on määritelty tyypit sekä presedenssijärjestys. Näiden avulla samalle attribuutille voidaan asettaa arvo useissa paikoissa, jolloin tyypin ja presedenssijärjestyksen perusteella määräytyy miksi attribuutin arvo evaluoituu. Attribuutille voidaan esimerkiksi cookbookin attribuutti-tiedostossa määrittää *default*-tyyppinen oletusarvo, joka ylikirjoitetaan ympäristössä *override*-tyyppisellä arvolla. Tällöin reseptiä suoritettaessa attribuutin arvoksi evaluoituu ympäristössä ylikirjoitettu arvo. [55.]

Chefin data bagit ovat globaaleja muuttujia, jotka on tallennettu JSON-muodossa Chef Serverille. Muuttujat ovat saatavissa kaikissa ympäristöissä Chef Serverille tehdyillä hauilla. Koska muuttujat eivät ole ympäristökohtaisia, pitää ympäristökohtainen tieto tallentaa erilailla nimettyihin muuttujiin tai käyttää vaihtoehtoisesti ympäristöissä asetettuja attribuutteja arvojen hallintaan. Data bagien sisältö on mahdollista kryptata salausavaimella. Salaukseen käytetään Rubyn vakiokirjastojen OpenSSL-pakkauksen AES–256–CBC-salausta. Kryptattujen data bagien avulla voidaan hallita suojatusti esimerkiksi tietokantojen salasanoja. [56.]

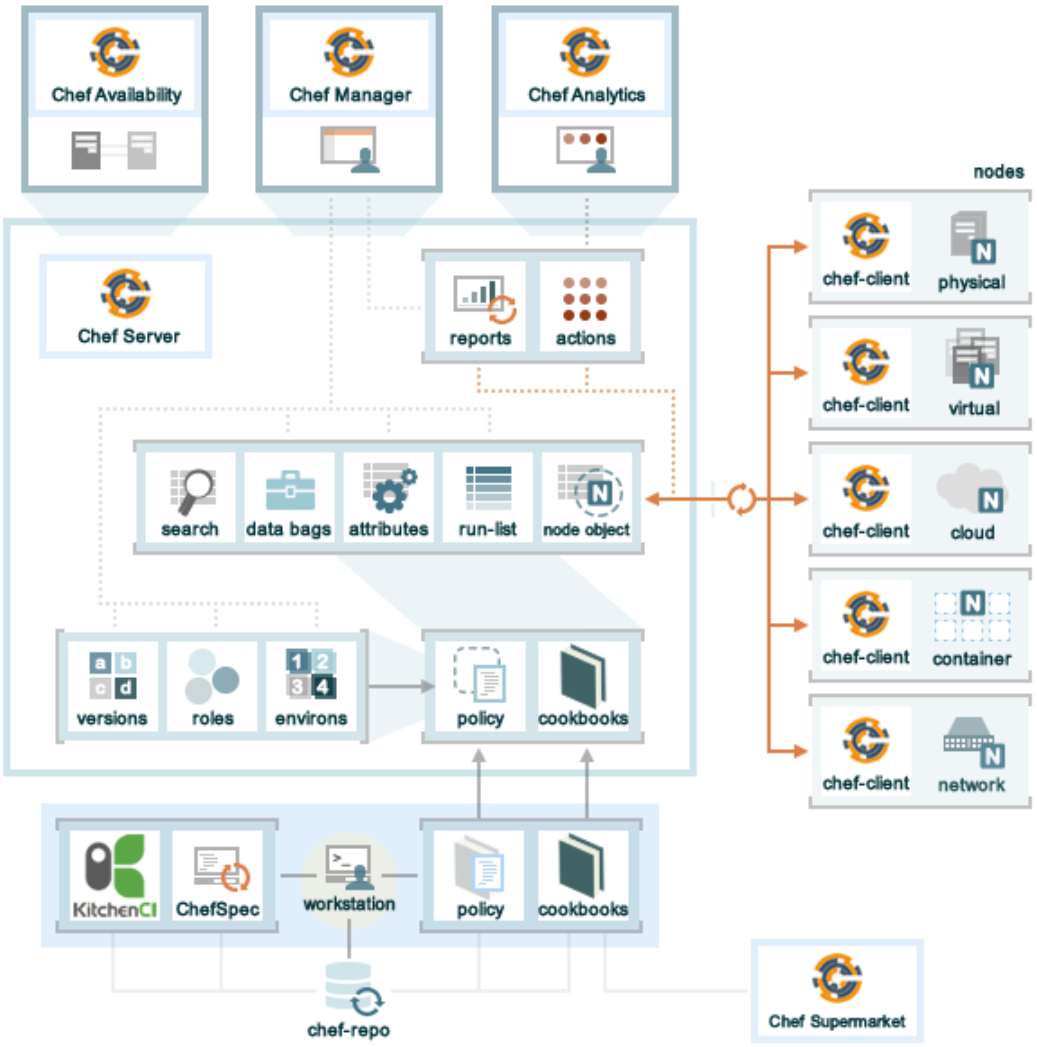
3.3 Palvelin, asiakas ja työasema

Chefin toiminta perustuu Chef-palvelimeen ja -asiakkaisiin. Chef-palvelimen toiminnasta vastaa Chef Server -sovellus ja asiakkaiden toiminnasta Chef Client -sovellus. Reseptit ja konfiguraatiotieto tallennetaan palvelimelle, josta asiakassolmuille (client node) asennetut asiakassovellukset noutavat suorituksen yhteydessä uusimmat reseptit ja tarkistavat onko sovelluksen konfiguraatio määrittelyn mukainen. Asiakassovellusten suoritus, eli provisiointi, voidaan suorittaa manuaalisesti tai ajastetusti käyttämällä esimerkiksi Unix-pohjaisissa käyttöjärjestelmissä olevaa Cron-työkalua. [10.]

Asiakassolmu voi olla fyysinen palvelin, virtuaalikone tai niin sanottu container.

Containerit ovat käyttöjärjestelmätasoinen virtualisointimenetelmä Linuxille. Containerit näyttäytyvät virtuaalikoneina tarjoten muun muassa oman eristetyn tiedostojärjestelmän, prosessiavaruuden ja verkkorajapinnan, mutta jakavat kuitenkin varsinaisen käyttöjärjestelmän ytimen [57]. Containerit ovat virtuaalikoneita kevyempi versio käyttöjärjestelmän abstrahointiin. Yksi suosittu toteutus on Docker [37], joka mahdollistaa Chefillä hallittavien Linux-containerien luomisen.

Kuvassa 3.3 on esitelty Chefin eri komponentit ja niiden väliset suhteet yleisellä tasolla. Kuvassa oikeassa reunassa on eri tyyppisiä Chefin asiakassolmuja, joita kaikkia yhdistää niissä oleva Chef Client -sovellus. Kuvan keskellä olevassa laatikossa on informaatiokeskuksena toimivan Chef-palvelimen rakenne. Palvelimella on Chef Server -sovellus, johon asiakassolmut ovat yhteydessä. Palvelimella säilötään muun muassa cookbookit, ympäristöt, roolit, data bagit ja attribuutit, jotka asiakassolmut noutavat tarvittaessa provisiointien yhteydessä. [44.]



Kuva 3.3: Chefin komponentit [44].

Chef-palvelimen hallinta tehdään kuvan 3.3 alareunassa esitellyiltä työasemilta (workstation). Työasemalla kehittäjät voivat ylläpitää ja testata cookbookeja käyttäen apunaan esimerkiksi aiemmin esiteltyjä Vagrantia ja Test Kitcheniä sekä Chefin yksikkötestaukseen tarkoitettua sovelluskehystä ChefSpeciä [35; 38; 58]. Työasemalle voidaan ladata valmiita Chef-yhteisön ylläpitämiä cookbookeja Chef Supermarketista [45] sekä itse tehtyjä cookbookeja esimerkiksi versionhallintajärjestelmästä. Cookbookit ladataan työasemalta Chef-palvelimelle, josta ne ovat käytettävissä asiakassolmuille. [44.]

Kuvan 3.3 yläreunassa olevat Chef Availability-, Chef Manager- ja Chef Analytics -laatikot ovat Chefin Enterprise-version ominaisuuksia. Ne mahdollistavat Chef-palvelinten klusteroinnin, hallinnan erillisen hallintakäyttöliittymän avulla sekä tarjoavat reaaliaikaista tietoa Chef-palvelimen tapahtumista. [59; 60; 61.]

Chef-palvelimen hallinta tapahtuu työasemalta siihen tarkoitetulla Knife-työkalulla. Knife on komentorivityökalu, jolla voidaan muun muassa alustaa uusia asiakassolmuja, hallita palvelimella olevia cookbookeja sekä suorittaa palvelimella erilaisia hakutoimintoja, joilla saadaan tietoja asiakassolmuista ja niiden attribuuteista. Knifen avulla voidaan myös suorittaa SSH-yhteyden yli komentoja kerralla joukolle asiakassolmuja hyödyntäen Chef-palvelimelle tehtyjä hakutoimintoja. [62.]

Chefiä on mahdollista käyttää myös ilman erillistä Chef-palvelinta. Tähän tarkoitukseen on olemassa Chef Solo, joka suoritetaan pelkästään asiakassolmulla [63]. Tällöin asiakassolmulle kopioidaan etukäteen reseptit ja muut tarvittavat resurssit suoritusta varten. Chef Solo asennetaan asiakassolmulle ja suoritetaan määrittämällä parametreina Chefin resurssien sijainti järjestelmässä tai internetissä. Chef Solo ei sovellu hyvin tilanteisiin, joissa ylläpidettäviä solmuja on useita, koska konfigurointiprosessi sisältää paljon manuaalista työtä. Chef Solo on myös toiminnaltaan rajoittuneempi kuin perinteinen asiakas–palvelin-toteutus. Muun muassa Chef-palvelinta käyttävät hakutoiminnallisuudet eivät ole käytettävissä Chef Solossa. Chef Solo soveltuu kuitenkin käytettäväksi yksittäisten sovelluspalvelinten konfigurointiin sekä testaus- ja kehitystarkoituksiin.

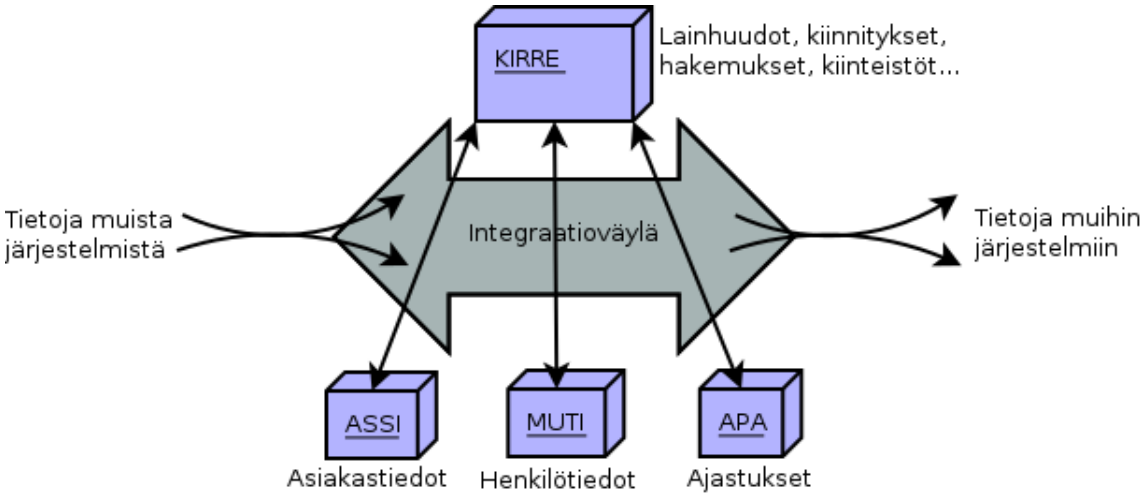
Chefin version 11.8.0 myötä Chef Solo on mahdollista korvata suorittamalla Chef Client -sovellusta paikallisella muodolla (local mode). Tällöin Chefin toiminta on lähempänä perinteistä asiakas–palvelin-mallia, eikä reseptien kirjoituksessa tarvitse erikseen huomioida suoritustapaa, toisin kuin Chef Solossa. Paikallisen muodon toiminta perustuu Chef Zero -sovellukseen [64], jota suoritetaan provisioitavalla palvelimella. Chef Zero käynnistyy prosessiksi, joka simuloi Chef-palvelinta. Prosessille ladataan tarvittavat resurssit, jotka Chef Client -sovellus noutaa prosessilta samalla tavalla kuin perinteiseltä Chef-palvelimelta. [65.]

4. SOVELLUSKOHDE

Sovelluskohteeksi tähän diplomityöhön on valittu Solitan KIOS-projektissa Maanmittauslaitokselle toteutetut sovellukset. KIOS-projektissa on toteutettu osa Maanmittauslaitoksen kiinteistöjen kirjaamisosan uudistamishankkeeseen (UKIR) kuuluvia sovelluksia.

UKIR-hanke on käynnistetty vuonna 2008 Maanmittauslaitoksen toimesta. Hanke juontaa juurensa Matti Vanhasen II hallituksen hallitusohjelmasta [66]. Solita valittiin kesällä vuonna 2010 KIOS-järjestelmän toteuttajaksi ja työ alkoi saman vuoden syksynä [67]. Myöhemmin vuonna 2011 nimi KIOS vaihtui KIRREksi [68].

Solitan toteuttamia sovelluksia ovat kiinteistötietojärjestelmän kirjaamisosa KIRRE, asiakashallintajärjestelmä ASSI, muutostietopalvelu MUTI, ajastuspalvelu APA sekä sovellusten välisen kommunikoinnin toteuttavalle ESB-integraatioväylälle luodut järjestelmäintegraatiot. Tässä diplomityössä sovelluskohteista APA rajataan työn ulkopuolelle. Solitan toteuttamista sovelluksista monimutkaisin ja työmäärältään suurin on KIRRE ja sen integraatiot muihin järjestelmiin. Kuvassa 4.1 on esitelty Solitan toteuttamien sovellusten väliset integraatiot.



Kuva 4.1: Solitan toteuttamat UKIR-hankkeen sovellukset.

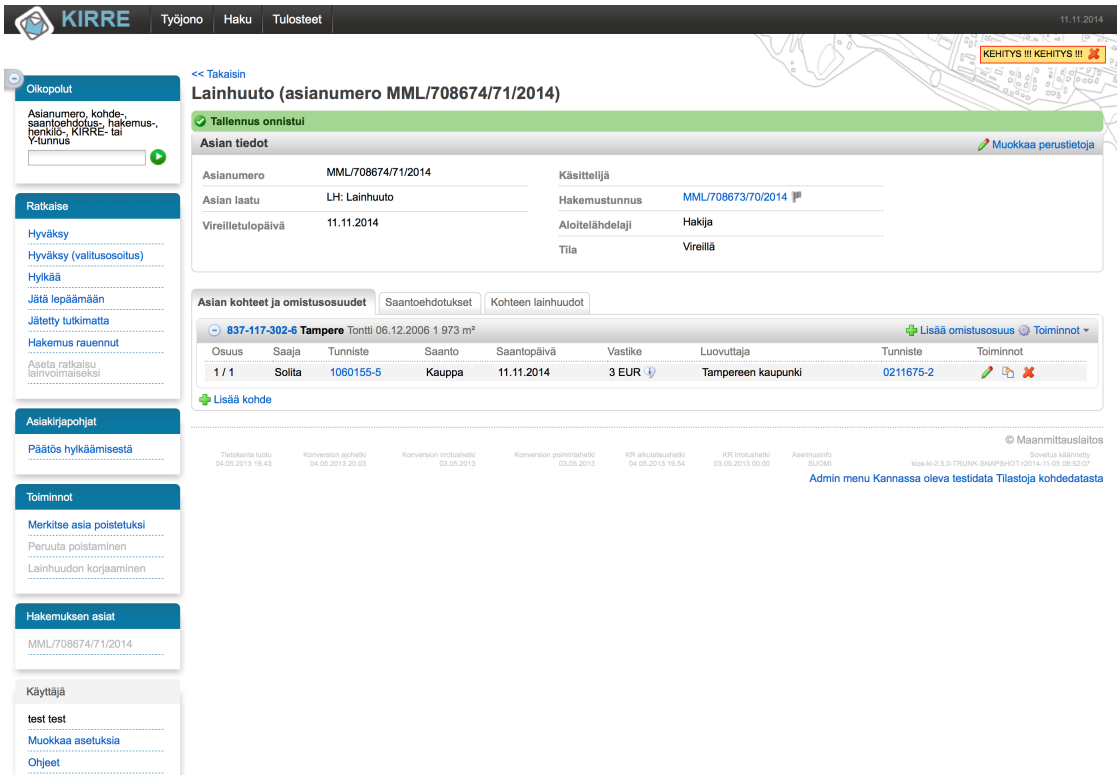
ASSI, MUTI ja APA eivät kommunikoi keskenään, mutta kommunikoivat KIRREn ja muiden järjestelmien kanssa. Kaikki sovellusten välinen kommunikaatio tapahtuu integraatioväylän kautta. Solitan toteuttamat sovellukset kommunikoivat

keskenään pääsääntöisesti XML-muotoisilla viesteillä käyttäen sovellusten REST-rajapintoja. Integraatioväylän kautta sovelluksiin liittyy myös useita muita järjestelmiä, joista vastaanotetaan tietoja muun muassa KIRREen. Väylän avulla myös välitetään tietoja muille järjestelmille.

Tässä luvussa on esitelty sovelluskohteen eri sovellukset ja niiden käyttötarkoitukset.

4.1 KIRRE

Kiinteistötietojärjestelmän kirjaamisosa KIRRE koostuu käyttöliittymästä, tietopalveluista, eräajoista sekä palvelurajapinnasta. KIRRE sisältää tiedot Suomen kiinteistöjen ja määräalojen omistusoikeuksista sekä omistuksiin tehdyistä kiinnityksistä. Kiinnitys tarkoittaa ”kiinteään omaisuuteen kohdistuvaa, lainhuuto- ja kiinnitysrekisteriin tehtävää merkintää kiinteän omaisuuden käyttämiseksi velan vakuutena” [69]. Näiden lisäksi KIRREssä on tiedot maa-alueisiin liittyvistä erityisistä oikeuksista, joiksi luokitellaan muun muassa vuokra- ja metsänkäyttöoikeudet. Kuvassa 4.2 on esimerkki siitä miltä lainhuudon kirjaaminen näyttää KIRREn käyttöliittymässä.



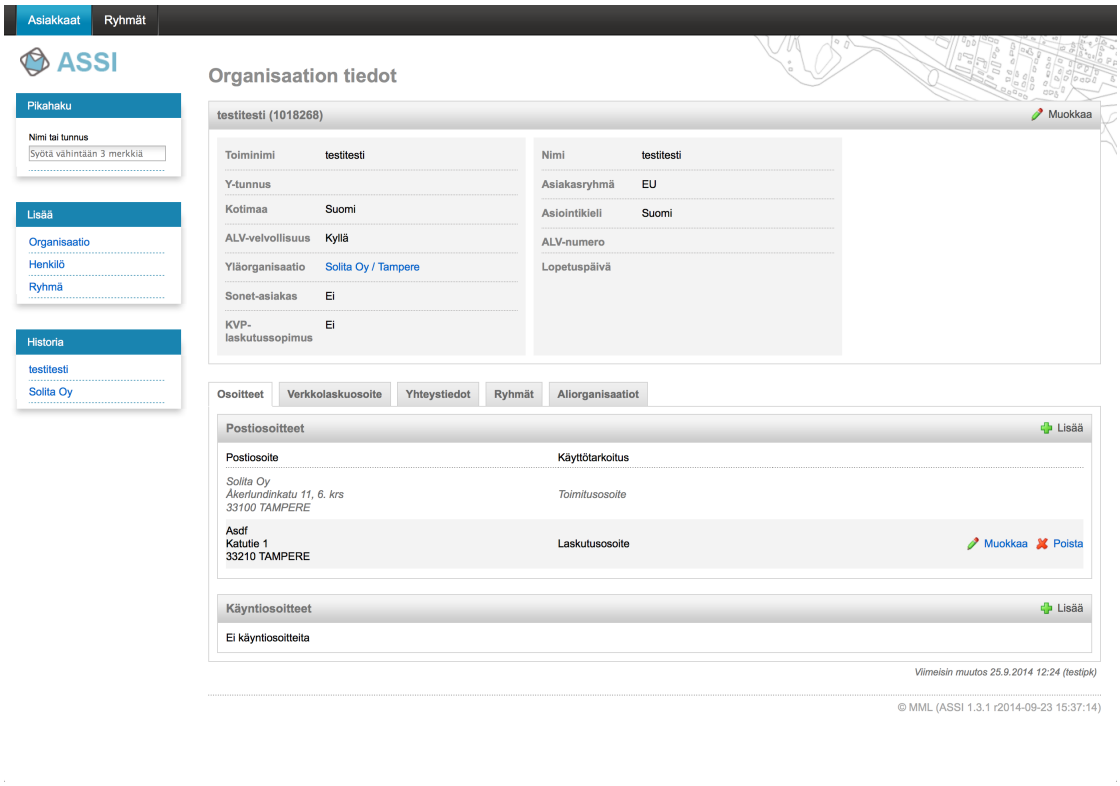
Kuva 4.2: Lainhuudon kirjaaminen KIRREn käyttöliittymässä.

KIRREn käyttäjiä ovat Maanmittauslaitoksen henkilökuntaan kuuluvat kirjaamissihteerit ja -lakimiehet. Näiden lisäksi muut järjestelmät käyttävät KIRREä ul-

koisten rajapintojen kautta. Näiden rajapintojen kautta KIRRE muun muassa tarjoaa tietoa pankeille ja Verohallinnolle, kirjaa Kiinteistökaupan verkkopalvelun kautta tulleita hakemuksia ja päivittää kiinteistöjen tietoja Kiinteistörekisterin avulla. Kiinteistökauppoja tehdessä halutaan usein varmistua kiinteistön oikeasta omistajasta, ja ettei kiinteistöä ole kiinnitetty jonkin velan vakuudeksi. Tällöin KIRREllä voidaan luoda lainhuuto- ja rasitustodistukset, joista edellämainitut tiedot selviävät.

4.2 ASSI

ASSI on kiinteistötietojärjestelmän asiakkuudenhallintasovellus. Se tarjoaa muun muassa toiminnallisuuden asiakkaiden yhteyshenkilöiden sekä osoitteiden hallintaan. ASSIssa on käyttöliittymä, jolla organisaatio- ja henkilöasiakkaita voidaan lisätä, muokata ja poistaa. Samat toiminnot on mahdollista tehdä myös REST-rajapinnan kautta. Asiakkaille voidaan tallentaa erilaisia tietoja, kuten toimitus-, laskutus-, sähköposti- ja verkkolaskuosoitteita. Organisaatioasiakkaille on mahdollista luoda hierarkioita, jolloin esimerkiksi aliorganisaatiolla on oma toimitusosoite, mutta laskutusosoite periytyy sen yläorganisaatiolta. Kuvassa 4.3 on esimerkki siitä miltä organisaation tiedot näyttävät ASSIn käyttöliittymässä.



Kuva 4.3: Organisaation tiedot ASSIn käyttöliittymässä.

ASSIn käyttöliittymällä on joitakin kymmeniä asiakastietoja hallinnoivia käyttäjiä. ASSIn REST-rajapintaa käyttää muun muassa KIRRE integraatioväylän kautta.

Integraatioväylällä välitetään asiakastietoja ASSIn REST-rajapinnasta myös muihin kuin Solitan toteuttamiin järjestelmiin, kuten Kiinteistökaupan verkkopalveluun. Lisäksi ASSIsta välitetään asiakkaiden muuttuneita osoitetietoja Sonet-järjestelmään määrämittaisina tiedostoina.

4.3 MUTI

Muutostietopalvelu MUTI toimii välivarastona Väestörekisterikeskuksen tarjoamille henkilöiden yhteystiedoille. MUTIssa on tallennettuna henkilöiden nimi- ja osoitetietoja. MUTIa käytetään KIRREstä integraatioväylän kautta, kun esimerkiksi lainhuutoa kirjatessa haetaan henkilötunnusta vastaavan henkilön nimitiedot. Mikäli kyseisen henkilön tietoja ei ole sillä hetkellä MUTIssa, haetaan tiedot Väestörekisterikeskuksesta ja tallennetaan MUTIin ennaltamääritellyksi ajaksi. MUTIssa olevat henkilötiedot päivittyvät Väestörekisterikeskukselta saatujen muutostietojen avulla.

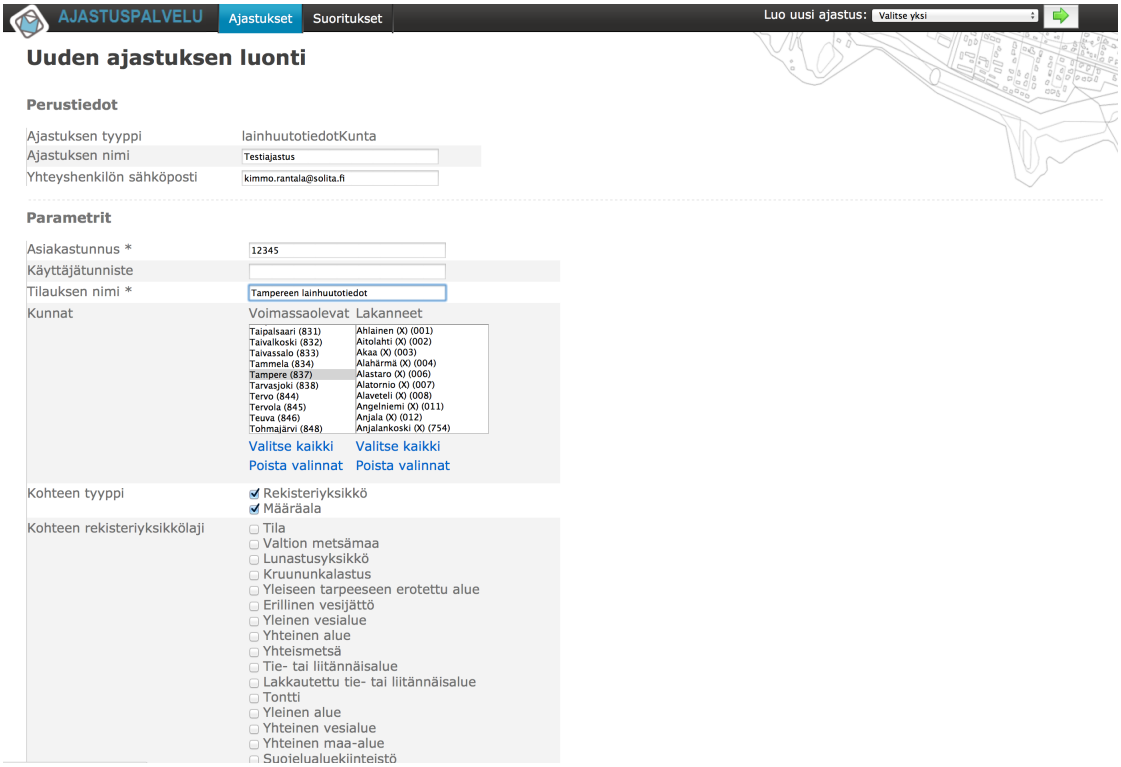
MUTIssa on REST-rajapinta, jonka avulla tietoja voidaan hakea muihin järjestelmiin. MUTIssa ei ole erillistä käyttöliittymää, vaan käyttö ja tietojen ylläpito tapahtuu integraatioväylän avulla.

4.4 APA

Ajastuspalvelu APA on järjestelmä, jota käytetään Maanmittauslaitoksella erilaisten tietyin väliajoin suoritettavien tehtävien (eräajojen) ajastukseen. Tyyppillisesti eräajolla noudetaan tietoja muista tietojärjestelmistä ja tuotetaan niistä asiakkaille välitettäviä aineistoja. APAlla ajastetaan muun muassa KIRREn ja Kiinteistötietojärjestelmän aineistopalvelun eräajoja. Eräajojen tuloksena syntyvä materiaali välitetään asiakkaille aineistovälityspalvelun avulla. Aineistovälityspalvelu on APAsta erillinen järjestelmä.

Ajastuspalvelua käytetään Maanmittauslaitoksella ulospäin asiakkaille tarjottavien palveluiden tarjoamiseen sekä Maanmittauslaitoksen sisäisten tehtävien suorittamiseen. Ajastuspalvelun käyttäjät ovat Maanmittauslaitoksella työskenteleviä henkilöitä. Kuvassa 4.4 on esimerkki ajastuksen luonnista APAssa.

Ajastuspalvelua ei oteta osaksi tätä diplomityötä. Sen nykyinen konfiguraationhallinta- ja toimitusprosessi eroaa jonkin verran muista sovelluksista. Tulevaisuudessa prosessia tullaan mahdollisesti yhdenmukaistamaan muiden sovellusten mukaiseksi. Muista sovelluskohteen sovelluksista ei ole riippuvuuksia ajastuspalveluun, mikä mahdollistaa sen rajaamisen diplomityön ulkopuolelle.



Kuva 4.4: Esimerkki APAlla tehtävästä ajastuksen luonnista.

4.5 Integraatioväylä

Integraatioväylänä käytetään Mulesoftin avoimen lähdekoodin Mule ESB:ia [70]. Käytössä on Community edition -versio. Väylän päälle on toteutettu useita eri integraatioita moniin järjestelmiin.

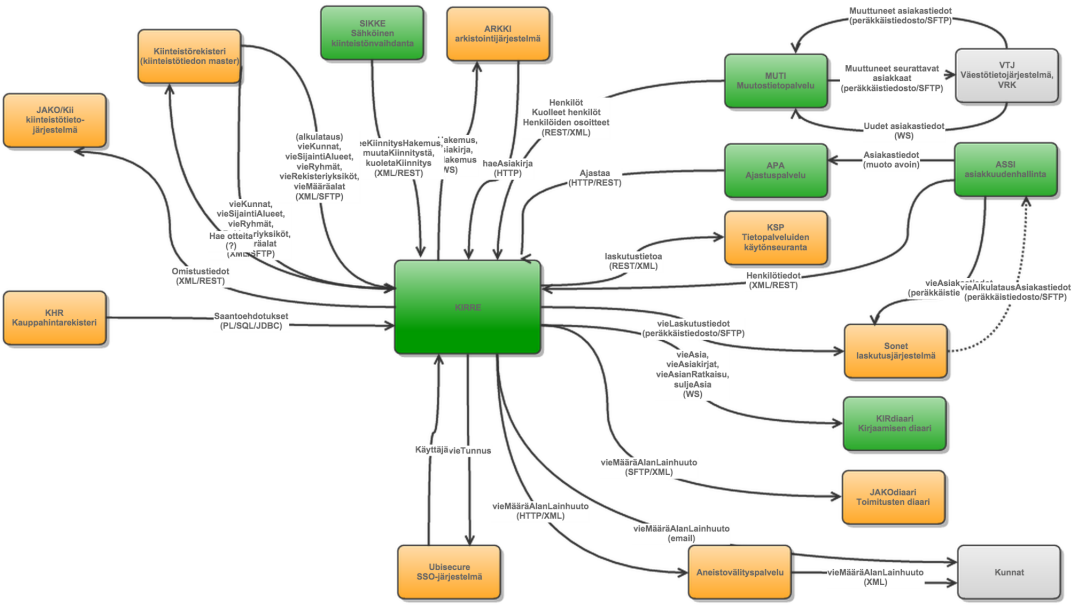
Ylläpidon ja integraatioiden seuraamisen helpottamiseksi Solita on toteuttanut Muleen Solita Pulse -integraatiomonitorin. Solita Pulse on sovellus, joka säilyttää tietoja integraatioiden suorituksista, mahdollisista epäonnistumisista ja virheviesteistä. Sovelluksessa on käyttöliittymä, jonka avulla tietoja voidaan selata ja suodattaa. Kuvassa 4.5 on osa erään prosessin transaktiolokista kehitysympäristön Solita Pulse -integraatiomonitorin käyttöliittymässä. Solita Pulse on tuote, jota käytetään myös muissa Solitan projekteissa.

Kuvassa 4.6 on esitelty osa järjestelmän integraatioista. Kuva on yksinkertaistettu eikä sisällä kaikkia nykyisen toteutuksen integraatioita. Järjestelmässä keskeisessä osassa on KIRRE, jonka kanssa useat muut sovellukset kommunikoivat integraatioväylän välityksellä. Järjestelmien väliseen kommunikointiin käytetään eri protokollia ja rajapintoja, jotka voivat yksittäisessä integraatiossakin vaihdella. Esimerkiksi Kauppahintarekisteristä tiedot noudetaan integraatioväylälle JDBC-tietokantarajapinnan kautta ja syötetään KIRREen REST-rajapintaa käyttäen. Tällöin KIRREssä ei tarvitse erikseen huomioida tietojen noutotapaa.

3,035 items found, displaying 1 to 100. [First/Prev] 1, 2, 3, 4, 5, 6, 7, 8 [Next/Last]

	Tapahtuma	Aikaleima	Transaktio id	Prosessinimi	Laukaisija	Viesti
	Integration success	2014-11-11 22:45:58 +0.0 s	12683256	services	0.0.0.0.8080.esb.services/email.1.0	Integration finished successfully.
	Info	2014-11-11 22:45:58	12683256	services	to.mmlsmtp.send	Interface finished successfully. Sähköpostin lähetys ([redacted]@maanmittauslaitos.fi) onnistui.
	Integration success	2014-11-11 22:45:58 +0.0 s	12683257	services	0.0.0.0.8080.esb.services/email.1.0	Integration finished successfully.
	Info	2014-11-11 22:45:58	12683257	services	to.mmlsmtp.send	Interface finished successfully. Sähköpostin lähetys ([redacted]@maanmittauslaitos.fi) onnistui.
	Integration success	2014-11-11 22:43:50 +0.0 s	12683255	services	0.0.0.0.8080.esb.services/email.1.0	Integration finished successfully.
	Info	2014-11-11 22:43:50	12683255	services	to.mmlsmtp.send	Interface finished successfully. Sähköpostin lähetys ([redacted]@maanmittauslaitos.fi) onnistui.
	Integration success	2014-11-11 22:42:46 +0.0 s	12683253	services	0.0.0.0.8080.esb.services/email.1.0	Integration finished successfully.
	Integration success	2014-11-11 22:42:46 +0.0 s	12683239	services	0.0.0.0.8080.esb.services/email.1.0	Integration finished successfully.
	Info	2014-11-11 22:42:46	12683239	services	to.mmlsmtp.send	Interface finished successfully. Sähköpostin lähetys ([redacted]@maanmittauslaitos.fi) onnistui.
	Info	2014-11-11 22:42:46	12683253	services	to.mmlsmtp.send	Interface finished successfully. Sähköpostin lähetys ([redacted]@maanmittauslaitos.fi) onnistui.
	Integration success	2014-11-11 22:42:46 +0.0 s	12683251	services	0.0.0.0.8080.esb.services/email.1.0	Integration finished successfully.
	Integration success	2014-11-11 22:42:46 +0.0 s	12683252	services	0.0.0.0.8080.esb.services/email.1.0	Integration finished successfully.
	Integration success	2014-11-11 22:42:46 +0.0 s	12683242	services	0.0.0.0.8080.esb.services/email.1.0	Integration finished successfully.
	Integration success	2014-11-11 22:42:46 +0.0 s	12683245	services	0.0.0.0.8080.esb.services/email.1.0	Integration finished successfully.
	Integration success	2014-11-11 22:42:46 +0.0 s	12683254	services	0.0.0.0.8080.esb.services/email.1.0	Integration finished successfully.
	Info	2014-11-11 22:42:46	12683251	services	to.mmlsmtp.send	Interface finished successfully. Sähköpostin lähetys ([redacted]@maanmittauslaitos.fi) onnistui.
	Integration success	2014-11-11 22:42:46 +0.0 s	12683247	services	0.0.0.0.8080.esb.services/email.1.0	Integration finished successfully.
	Info	2014-11-11 22:42:46	12683253	services	to.mmlsmtp.send	Interface finished successfully. Sähköpostin lähetys

Kuva 4.5: Prosessin transaktioloki integraatiomonitorin käyttöliittymässä.



Kuva 4.6: Integraatiokartta.

Solitan toteuttamien sovellusten välisten integraatioiden lisäksi väylälle on toteutettu muun muassa ASSIsta ja KIRREstä Sonet-järjestelmään siirtyvät asiakas- ja laskutustiedot. Kiinteistökaupan verkkopalvelulle välitetään tietoja muun muassa kiinteistöjen omistuksista ja niihin haetuista kiinnityksistä, sekä välitetään KIRREen asiakirjoja ja sähköisesti luotuja hakemuksia. Integraatioväylän avulla voidaan seurata myös rajapintojen käyttömääriä, koska kaikki ulkoisista rajapinnoista sovel-

luksille tehdyt pyynnöt kulkevat integraatioväylän kautta. Osa integraatioista toimii reaaliajassa käyttäen sovellusten rajapintoja tai suoria kantayhteyksiä sovellusten tietokantoihin. Jotkin integraatiot suoritetaan ajastetusti, jolloin tietoja saatetaan välittää tiedostoina esimerkiksi SFTP-yhteyksillä.

5. SUUNNITTELU JA TOTEUTUS

Sovelluspalvelinten konfiguraationhallinnan automatisointi toteutetaan luvussa 4 esitellyille sovelluksille. Sovelluskohteen sovelluksista ajastusten hallintaan käytettävä APA jätetään toteutuksen ulkopuolelle. APAn sovelluspalvelin muistuttaa rakenteeltaan muita sovelluspalvelimia, mutta APAn konfiguraationhallinta- ja toimitusprosessi eroaa kuitenkin jonkin verran muista. Muista sovelluksista ei ole riippuvuuksia APAan, minkä vuoksi APA on mahdollista rajata pois. Samoja toimintamalleja voidaan tarvittaessa myöhemmin soveltaa APAn konfiguraationhallinnan automatisointiin.

Automatisointi tapahtuu kartoittamalla tarpeet konfiguraationhallinnan automatisoinnille sekä konfiguraationhallinnan ja olemassa olevien ympäristöjen alkutila. Tämän perusteella määritellään tavoitteet automatisoinnille ja sille, millaisiin tarkoituksiin erilaisia ympäristöjä tarvitaan. Tavoitteiden pohjalta luodaan suunnitelma, jossa käsitellään millaisia ongelmia sovelluskohteen automatisointiin liittyy ja miten sovelluskohteen automatisointi korkealla tasolla tehdään. Suunnitelman perusteella toteutetaan varsinainen automatisointi käyttämällä Chefiä ja mahdollisesti muita työkaluja. Chefin valinta työssä käytettäväksi konfiguraationhallintatyökaluksi on perusteltu edellisessä luvussa.

Tässä luvussa on kuvattu sovelluskohteen tilanne ennen automatisoidun konfiguraationhallinnan käyttöä sekä mitä ongelmia ja tarpeita liittyy sovelluskohteen konfiguraationhallintaan. Automatisoidulle konfiguraationhallinnalle määritellään tavoitteet, kuvataan suunnitteluratkaisuja tavoitteisiin pääsemiseen sekä esitellään suunnitelman perusteella tehtyjä toteutusratkaisuja.

5.1 Alkutila

Alkutilassa ympäristöjen ylläpito on tehty manuaalisesti. Muutokset ympäristökonfiguraatioihin on tehty käytössä oleville sovelluspalvelimille tarpeiden tullessa ajankohtaisiksi. Ympäristöjen luominen ja muutosten tekeminen on ollut projektitiimin vastuulla ja toteuttajia on ollut useita. Muutoksia ja ympäristökonfiguraatioita on dokumentoitu projektin wikisivustolle ja joitakin oleellisimpia konfiguraatiotiedostoja on ylläpidetty versionhallinnassa.

Käytännössä muutosten hallinta ja dokumentointi ei kuitenkaan ole toiminut aukottomasti. Kaikkia muutoksia ei ole dokumentoitu tai dokumentaatio on hajau-

tunut eri paikkoihin, joista osa on jäänyt päivittämättä ja vanhentunut. Konfiguraatiomuutoksia on tehty suoraan palvelimille eikä muutoksia ole välttämättä tehty versionhallinnassa oleviin tiedostoihin. Välttämättä muutoksen tekijä ei ole tiennyt mihin eri ympäristöihin muutos pitäisi kohdistaa tai jostakin muusta syystä muutosta ei ole tehty kaikkiin tarvittaviin ympäristöihin.

Tarve konfiguraatioiden hallittavuudelle ja dokumentoinnille on ollut projektin alusta asti. Ylläpidettäviä ympäristöjä on useita, joista osa on Solitan palvelimilla ja osa asiakkaalla. Sovellusten edellyttämät konfiguraatioasetukset tulee toteuttaa kaikkiin ympäristöihin, jotta sovellusten toimivuus oikealla tavalla voidaan varmistaa. Yhtenä tärkeänä tarpeena dokumentoidulle konfiguraatiolle on, että Solitan projektitiimillä ei ole pääsyä kaikkiin asiakkaan ympäristöihin, joissa sovellukset ovat asennettuina. Konfigurointivastuu näiden ympäristöjen osalta on asiakkaalla.

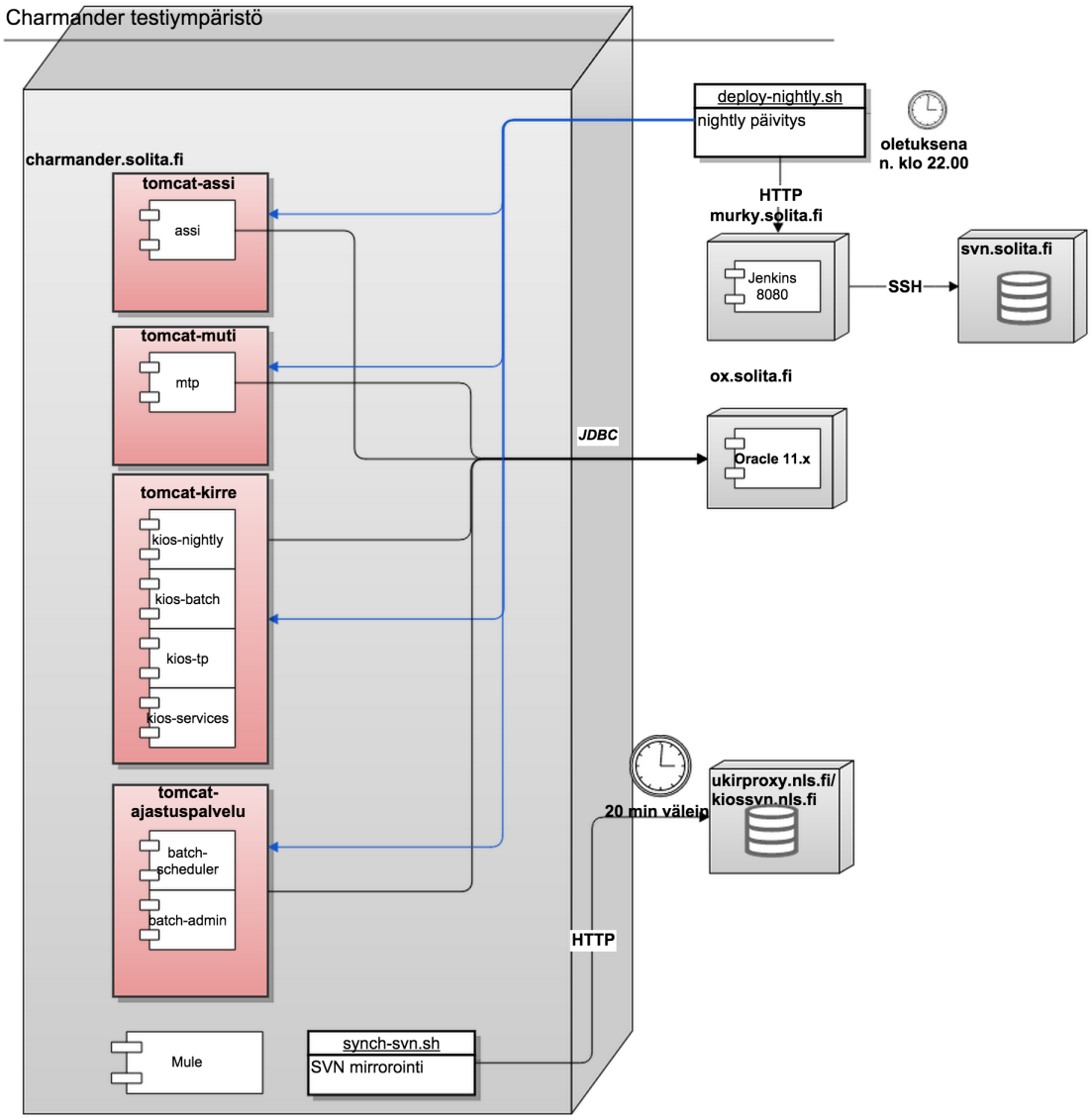
Kehitystiimillä on käytössä Solitan ympäristössä virtuaalipalvelin (Charmander), johon on asennettu KIRRE, MUTI, ASSI, APA ja integraatioväylä. Palvelimen rakennetta on havainnollistettu kuvassa 5.1. Sovelluksista käännetään ja asennetaan uudet versiot palvelimelle joka yö, mikäli automaattitestit CI-ympäristössä ovat suoriutuneet onnistuneesti. Tarvittaessa sovellusten kääntö ja asennus voidaan käynnistää myös päivällä, mikäli yölliset versiot koetaan liian vanhoiksi. Tämän palvelimen on tarkoitus tarjota mahdollisuus koekäyttää sovellusten tuoreita kehitysversioita.

Kehitystiimillä on lisäksi käytössä Solitan ympäristössä suorituskykytestaukselle omistettu palvelin, jolla suoritetaan joka yö suorituskykytestaus sen hetkiselle KIRREn kehitysversiolle. Tietokantana käytetään samaa kantainstanssia kuin koekäyttöpalvelimen KIRRE käyttää. Suorituskykytesteissä integraatioväylänä toimii Charmander-palvelimella oleva väylä, joka on yhteydessä samalla palvelimella oleviin MUTIin ja ASSIin. Suorituskykytestauksen ei ole koettu tarpeelliseksi käyttää puhtaasti sille omistettua integraatioväylää, MUTIa ja ASSIa. Myös näiden tulisi olla erillään muusta käytöstä, jotta suorituskykymittaukset olisivat realistisemmat. Tarvetta kuitenkin pienentää se, että suorituskykytestit ajetaan öisin sellaiseen aikaan, jolloin sovelluksilla ei ole muita käyttäjiä.

Alkutilassa Solitan ympäristöjen konfiguraatio eroaa monilta osin asiakkaan ympäristöistä. Kaikki koekäyttöön tarkoitetut sovellukset on asennettuina samalle palvelimelle. Samalla palvelimella on muitakin tehtäviä kuin sovellusten suorittaminen, kuten Solitan versionhallinnan synkronointi asiakkaan versionhallintaan.

Koekäyttöön tarkoitetun palvelimen konfiguraatio ei ole kokonaisuudessaan dokumentoituna. Vaikka joitakin oleellisimpia kohtia on kirjoitettu projektin wikisivustolle, monet yksityiskohdat ovat vain niiden alkuperäisten toteuttajien tiedossa. Dokumentoidun konfiguraation puuttuminen vaikeuttaa uusien ympäristöjen luomista, joita tarvitaan muun muassa eri sovellusversioiden yhteensopivuuden testaamiseen.

Alkutilan konfiguraationhallintaprosessi on ongelmallinen, vaikka erilaisten tilan-



Kuva 5.1: Kehitysversioiden koekäyttöön tarkoitettu Charmander-palvelin.

teiden mukaan tehtyjen korjausten myötä palvelimien toiminnasta on saatu melko vakaata. Palvelimilla on silti jäämiä erilaisista aikaisemmin käytetyistä konfiguraatioista, kuten skriptejä, joilla ennen KIRREn käyttöönottoa saatiin tietokantaskeema asetettua oikeaksi sen perusteella mihin skeemaan on edellisen kerran saatu onnistuneesti tuotua konvertoitu data-aineisto aikaisemmasta kirjaamisjärjestelmästä. Tällaisten siivoaminen pois tulee tehdä manuaalisesti, mitä vaikeuttaa dokumentaation puuttumisen lisäksi se, että muutokset voidaan tehdä vain suoraan kyseiselle palvelimelle eikä niitä voida helposti testata erillisellä palvelimella. Virheellinen korjaus voi tehdä ainoan koekäyttöpalvelimen toimimattomaksi, joten korjaukset ja muutokset tulee tehdä siten, että tilanteen ehtii korjaamaan ennen kuin ympäristöä tarvitaan.

Dokumentaation puuttuminen ja asennusskriptit, jotka toimivat vain asiakkaan ympäristöissä johtuen muun muassa vakioiduista palvelinten nimistä ja oletuksista palvelimien tilasta, vaikeuttavat uusien sovelluspalvelimien ja ympäristöjen luomista. Integraatioiden asennusskriptit tukevat konfiguraatioiden migratoimista tiettyjen Mule ESB:n versiopäivitysten välillä, mutta nämä skriptit toimivat vain Maanmittauslaitoksen ympäristöissä ja tekevät oletuksia sovelluspalvelinten konfiguraatioiden tilasta. Skriptit on luotu erityisesti asiakkaan ympäristöjen päivityksiä varten tiettyihin tilanteisiin, joissa ympäristöt ovat olleet jossakin tietyssä tilassa. Tämän jälkeen muut asennusskriptit ovat päivittyneet siten, ettei ympäristöä välttämättä voi enää saada niiden avulla migratointia edeltävään tilaan. Osa skripteistä on siis luotu kertaluontoisia tilanteita varten eivätkä ne ole enää uudelleenkäytettävissä. Solitan ympäristöissä versiopäivityksiin liittyneet konfiguroinnit on ainakin osittain tehty manuaalisesti.

5.2 Tavoitteet

Tavoitteena on saada Solitan toteuttamien sovellusten sovelluspalvelimien konfigurointi automatisoitua sellaiseen muotoon, jolla ympäristöjä voidaan luoda ja päivittää automaattisesti. Automatisointi toteutetaan KIRRElle, ASSIlle, MUTIlle ja integraatioväylälle. Kaikkia integraatioita ei tulla toteuttamaan tämän työn puitteissa. Tärkeintä on saada Solitan toteuttamien sovellusten väliset integraatiot toimimaan automatisoidusti konfiguroiduissa ympäristöissä. Myöhemmin toteutusta on mahdollista laajentaa kattamaan integraatioita muihin järjestelmiin toteuttamalla esimerkiksi yksinkertaistettuja tynkätoteutuksia mallintamaan ulkoisia järjestelmiä. Tavoitteena ei ole luoda valmiita testiympäristöjä, vaan ennemmin mahdollistaa uusien ympäristöjen luominen pienellä työmäärällä, siten että ympäristöjen konfiguraatiot pysyvät jatkuvasti ajan tasalla. Tällöin ympäristöjen ylläpito helpottuu, koska ajan tasalla pitäminen ei vaadi manuaalista työtä, joka voisi jäädä tekemättä tai erota eri ympäristöissä.

Tavoitteena työlle on, että uusien sovellusversioiden asentaminen saadaan tarvittaessa tehtyä automatisoidusti, eli niin sanotulla jatkuvan toimituksen (continuous delivery) mallilla. Tämä helpottaa kehittäjien ja testaajien työtä, koska tehdyt muutokset ovat koekäytettävissä nopeasti ja ympäristöt pysyvät ajan tasalla.

Solitan palvelimilla olevat kehitys- ja ylläpitotyöhön käytettävät ympäristöt eivät ole identtisiä asiakkaan ympäristöjen kanssa. Tuotantoympäristöä ei ole mahdollista toteuttaa tarkalleen samanlaiseksi Solitan ympäristöön, koska kaikkia tuotantoympäristön järjestelmiä ei ole mahdollista saada asennettaviksi Solitalle. Tuotantoympäristössä on useita eri järjestelmiä, jotka eivät ole Solitan toteuttamia. Asiakkaan hallinnassa olevassa kehitysympäristössä on joistakin palveluista käytettävissä testiversiot, joten joiltakin osin järjestelmätestaaminen voidaan toteuttaa siellä. Tämän

vuoksi realistisempana tavoitteena ympäristöjen luonnille on rajata toteutettavat sovelluspalvelinkonfiguraatiot Solitan toteuttamille sovelluksille ja tarpeen mukaan toteuttaa riittävällä tasolla toimivia tynkäpalveluita korvaamaan puuttuvat palvelut.

Kehitysvaiheen aikana tarve eri ympäristöille oli vähäisempi kuin tuotantoaikana. Sovellusten käyttöönotto toi uusia tarpeita muun muassa tuotannossa havaittujen ongelmien selvittämiselle ympäristössä, johon on asennettu sovellusten tuotannossa olevat versiot. Tämän tuotannon tuki -ympäristön tulee vastata mahdollisimman hyvin todellista tuotantoa, jotta ongelmien juurisyy saadaan selvitettyä. Tämän lisäksi syntyi tarve ympäristölle, jossa voidaan todeta tuotantoversioon tehtävien kriittisten korjausten toimivuus. Korjauksia ei välttämättä tehdä pelkästään seuraavaan kehitysversioon, vaan tuotannon korjaamiseksi saatetaan tehdä erillinen korjausversio, joka voidaan toimittaa normaalia kehityssykliä nopeammin. Sovelluksista on olemassa samaan aikaan useita versioita, joihin tulee mahdollisesti kohdistaa muutoksia. Tuotannossa on asennettuna jokin versio, johon pitää olla mahdollista tehdä korjauksia kriittisiin ongelmiin. Asiakkaan testausympäristöön on mahdollisesti toteutettu tietyt versiot sovelluksista hyväksyntätestausta varten. Näihin versioihin on mahdollisesti tehtävä vielä korjauksia, ennen kuin versiot ovat valmiita vietäviksi tuotantoympäristöön. Samaan aikaan Solitalla toteutetaan jo seuraavaa kehitysversiota, jonka tulisi olla koekäytettävissä Solitalla. Kun mikä tahansa versio on tarvittaessa helposti asennettavissa, vähenee tarve niin sanotuille hotfix-korjauksille, jotka tehtäisiin suoraan tuotantoon testaamatta.

Tietyssä versiossa havaitut ongelmat eivät välttämättä koske pelkästään kyseistä sovellusversiota. Ongelman aiheuttanut virhe on saattanut olla olemassa jo pitkään useissa aikaisemmissakin versioissa, jolloin saattaa olla tarpeen selvittää missä versiossa virhe on syntynyt. Eri sovellusversioista koostuvia kombinaatioita muodostuu suuri määrä. Ympäristöjä kaikille sovellusversioiden kombinaatioille ei ole tarpeen olla toiminnassa koko aikaa eikä kaikkia näistä välttämättä tarvita milloinkaan. Tällaisten tietyistä sovellusversioista koostuvien kombinaatioden muodostamiselle yhtenä ratkaisuna on mahdollistaa ympäristön luonti siten, että asennettavat sovellusversiot voidaan parametrisoida ympäristön luontivaiheessa. Tällöin voidaan varmistaa esimerkiksi tietyn KIRRE-version yhteensopivuus eri integraatio- ja ASSI-versioiden kanssa. Kun asiakkaan ympäristöä vastaavan ympäristön luominen on mahdollista automatisoidusti, voi yksittäinen kehittäjä luoda esimerkiksi omalla työasemallaan oleville virtuaalikoneille oikeista sovellusversioista koostuvan ympäristön.

Suorituskykytestausta voitaisiin parantaa Solitan ympäristössä eristämällä sille tarkoitettu ympäristö kokonaisuudessaan muusta käytöstä, jolloin mittaustulokset olisivat luotettavampia. Tämä edellyttäisi mahdollisimman lähelle tuotantoympäristön kaltaista ympäristöä, jossa olisi oma tietokantapalvelin, sovelluspalvelimet ja

integraatioväylä. Eroavaisuuksia kuitenkin muodostuisi mahdollisista verkkoviiveistä ja muista monitoimittajaympäristön järjestelmistä, joita Solitan ympäristössä ei ole. Suorituskykytestauksessa joudutaan siis joka tapauksessa tekemään jonkinlaisia kompromissiratkaisuja, koska täysin vastaavaa ympäristöä ei ole mahdollista luoda Solitalle. Tavoitteena suorituskykytestauksen parantamiselle on luoda ympäristö, jossa on mitattavien osakokonaisuuksien kannalta oleellisimmat osat mahdollisimman hyvin eristettyinä muusta käytöstä.

Asiakkaan ympäristöjä vastaavien uusien ympäristöjen luonti mahdollistaa myös erilaisten konfiguraatioiden testaamisen ja koekäytön. On esimerkiksi mahdollista luoda ympäristö, jossa palvelimilla käytetään uudempaa versioita Mulesta. Näin voidaan etukäteen selvittää sovellusten toimivuus erilaisilla konfiguraatioilla. Tämä mahdollistaa myös esimerkiksi palomuurisääntöjen muutosten testaamisen. Alkutilassa kuvatulla Charmander-koekäyttöpalvelimella tämä ei ole mahdollista, koska kaikki sovellukset ovat samalla palvelimella.

Välillä tulee tarpeita luoda erilaisia skriptejä asiakkaan ympäristöissä suoritettaviksi. Tällaiset skriptit voivat olla tarpeellisia esimerkiksi versiomigraatioiden yhteydessä. Toimivuuden varmistamiseksi näiden kehittäminen ja testaaminen olisi hyvä tehdä kehittäjälle eristetyssä omassa ympäristössä, jossa virheelliset suoritukset eivät vaikuttaisi muuhun kehitystiimin ja ympäristöjen toimintaan. Alkutilan mallissa tämä ei ole käytännössä mahdollista, paitsi luomalla vastaava ympäristö esimerkiksi virtuaalikoneeseen manuaalisesti.

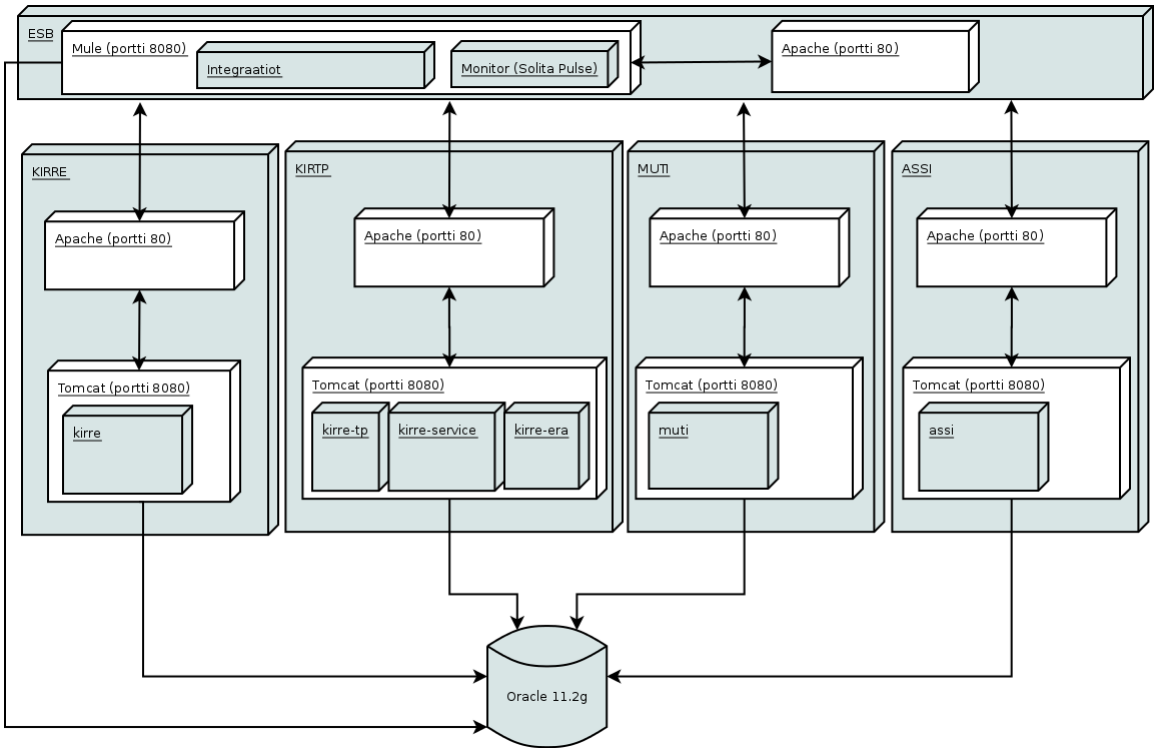
5.3 Suunnittelu

Sovelluspalvelimien konfiguraatiot luodaan vastaamaan nykyisiä tuotantoympäristön sovelluspalvelimia. Näin saadaan vältettyä yllättäviä toiminnallisia eroavaisuuksia, jotka johtuvat palvelimien erilaisista konfiguraatioista.

Solitan ympäristöön ei ole mahdollisuutta asentaa kaikkia muiden toimittajien järjestelmiä. Näiden integraatioiden testaamista varten luodaan erillinen apupalvelin, johon voidaan toteuttaa testaamisen kannalta riittävällä tasolla toimivia tynkäpalveluita. Apupalvelimelle voidaan asentaa myös palvelimien lokit keräävä palvelu, joka asiakkaan ympäristöissä ei ole Solitan hallittavissa oleva palvelu.

Kuvassa 5.2 on havainnollistettu sovelluskohteen ympäristön rakenne. Kuvassa suuret laatikot kuvaavat palvelinkoneita. KIRREn käyttöliittymäsovellus on palvelimella *KIRRE*. KIRREn tietopalvelut, eräajot ja palvelurajapinta ovat palvelimella *KIRTP*. MUTI koostuu vain yhdestä sovelluksessa, joka on *MUTI*-palvelimella. ASSIn käyttöliittymä ja rajapinta ovat samassa sovelluksessa, joka on *ASSI*-palvelimella. Integraatiot ja monitorointisovellus ovat *ESB*-palvelimella. Kaikissa palvelimissa on Apache web-palvelin, joka toimii edustapalvelimena (reverse proxy). HTTP-pyynnöt kulkevat sovelluksille aina edustapalvelimien kautta. Suorat yhteydet sovelluksiin

edustapalvelimien ohi on estetty palomuurilla. Integraatiosovellusta sekä -monitoria suoritetaan Mule ESB:lla. MUTIa, ASSIa ja eri KIRRE-sovelluksia suoritetaan Apache Tomcat -web-palvelimilla [71]. Kaikki sovellukset käyttävät samaa tietokantapalvelinta, jossa on Oracle Database -tietokannanhallintajärjestelmä [72]. Tavoitteena on mahdollistaa samankaltaisen konfiguraatiorakenteen luominen Chefillä.



Kuva 5.2: Sovelluskohteen ympäristön rakenne.

Yhtenä ongelmana on tietokantojen käyttö ja päivittäminen sovellusversioiden mukaiseen tilaan. KIRRE, ASSI ja MUTI käyttävät tietokantapäivitysten hallintaan Dbmaintain-työkalua [73]. Dbmaintain huolehtii, että päivitysskriptit tulevat suoritetuksi oikeassa järjestyksessä ja vain kertaalleen kuhunkin tietokantaskeemaan. Integraatiosovelluksella ei varsinaisesti ole käytössä tietokannanhallintajärjestelmää. Mule ESB:ssä toimiva Solita Pulse -integraatiomonitori kuitenkin käyttää samaa Oraclen tietokannanhallintajärjestelmää kuin muutkin sovellukset. Monitorointikannan päivittäminen tapahtuu toistaiseksi komentoriviskripteillä ja Oraclen Sqlplus-työkalulla tai vaihtoehtoisesti manuaalisena SQL-tiedostojen suorittamisella päivitettävään tietokantaan. Mikään sovelluskohteen sovelluksista ei tue kantapäivitysten perumista eli niin sanottua rollbackia. Mikäli tällaiselle on tarve, on vaihtoehtona korjata kanta manuaalisesti, palauttaa kanta aikaisempaan tilaan varmuuskopiosta tai luoda kanta alusta asti uudestaan. Viimeisellä tavalla korjattaessa menetetään kannassa oleva data. Kehitys- ja koekäyttötilanteissa tämä voi olla mahdollinen vaihtoehto, mutta tuotantoympäristössä se ei ole. Mikäli tuotantoympäristöön kohdis-

tuisi tarve palauttaa kanta päivityksiä edeltävään tilaan, niin tilanne korjattaisiin palauttamalla tietokanta varmuuskopiosta.

Oracle Database -tietokannanhallintajärjestelmänä on automatisoidun konfiguraationhallinnan kannalta vaativa. Tähän vaikuttaa sen asennuksen monimutkaisuus sekä Oraclen lisensointimalli. Oracle Databasesta on tarjolla Express edition -versio, joka soveltuu kehityskäyttöön sekä pienimuotoiseen testikäyttöön. Express edition -version lisenssi sallii yksittäisen tietokantainstanssin ajon palvelimella, siten että tietokannanhallintajärjestelmää suoritetaan vain yhdellä prosessorilla. Käytössä olevaa keskusmuistia saa olla korkeintaan yhden gigatavun verran eikä tietokannan koko saa ylittää 11 gigatavua. Express edition -versiosta myös puuttuu sovellusten toiminnan kannalta oleellisia ominaisuuksia, kuten tuki teksti-indekseille ja materialisoiduille näkymille. Muut lisenssivaihtoehdot ovat maksullisia ja vaativat erillisen tilauksen. [74.]

Kehitystiimillä on käytössä yksi valmiiksi asennettu Standard edition -versio Oracle Database -tietokannanhallintajärjestelmästä. Standard edition ei rajoita tietokantaskeemojen kokoa eikä määrää. Standard edition -versio sisältää myös kaikki ominaisuudet, joita sovelluskohteen sovellukset tarvitsevat toimiakseen.

Tuotantoympäristössä kaikilla palvelimilla on omat DNS-nimet, joita käytetään integraatioissa osoittamaan palvelun sijainti. DNS-nimien käyttö on mahdollista myös Solitan ympäristössä, mutta niiden hallinta voi olla haastavaa, mikäli ympäristöjä on useita. DNS-nimien käyttö vaatisi käytännössä projektille oman DNS-palvelimen, jota hallittaisiin osana muuta konfiguraationhallintaa. Tämä edellyttäisi lisäksi, että ympäristöjä käyttäville työasemille pitäisi asettaa kyseinen DNS-palvelin käyttöön. Käytön helpottamiseksi on järkevämpää mahdollistaa ympäristöjen konfigurointi suoraan IP-osoitteilla ja tarvittaessa lisätä niille DNS-nimet Solitan yhteiselle DNS-palvelimelle. Tällöin palvelut saadaan käytettäviksi DNS-nimillä Solitan verkossa. Pyritään luomaan toteutus niin, että palvelimien DNS-nimet tai IP-osoitteet voidaan asettaa konfiguraatioihin eksplisiittisesti tai vaihtoehtoisesti resepteissä haetaan provisioinnin yhteydessä dynaamisesti kyseisen ympäristön palvelimien IP-osoitteet Chef Server -palvelimen kautta.

Toteutettavaa Chef-konfiguraation kehitystä varten tarvitaan versionhallintajärjestelmä. Versionhallintajärjestelmänä käytetään Gitiä [75]. Tätä varten luodaan Solitan omalle GitLab-palvelimelle [76] repositorio. Eri ympäristöihin liittyvä konfiguraatio pidetään erillään sovellusten toteutuskoodista. Mikäli toteutuksessa koetaan tarve luoda esimerkiksi skriptejä sovelluspakettien tuottamiseen tai tietokantojen päivittämiseen, nämä skriptit tallennetaan sovelluskohteen sovellusten yhteyteen niissä käytettävään Subversion-versionhallintaan. Tämä on looginen paikka läheisesti sovelluksiin liittyvälle toteutukselle, ja kyseinen versionhallintajärjestelmä on myös käytettävissä projektin jatkuvan integroinnin Jenkins-palvelimelta.

Chef-konfigurointia varten luodaan Solitan ympäristöön Chef Server -palvelin, jota uusien ympäristöjen asiakassolmut käyttävät. Jokaista ympäristöä varten tarvitaan omat palvelimet ASSIlle, MUTIlle, integraatioväylälle sekä KIRREn käyttöliittymälle ja KIRREn rajapintapalveluille, kuten on esitetty kuvassa 5.2. Konfiguraatiototeutuksen kehitystyössä ei tarvita erillistä Chef Server -palvelinta eikä sovelluspalvelimia, koska voidaan käyttää työasemalla suoritettavaa Chef Zero -sovellusta sekä Vagrantilla luotavia VirtualBox-virtuaalikoneita [34; 35; 64]. Kehitystyössä tämä on järkevää, koska konfigurointia voidaan testata helposti täysin puhtaaseen palvelimeen, jossa ei ole aikaisemmissa provisioinneissa tehtyjä muutoksia.

Sovelluspalvelinten konfiguraatiot pyritään luomaan sellaisiksi, että ne on vähäisellä työmäärällä sovellettavissa uuteen ympäristöön. Tämä edellyttää, että sovelluspalvelinten konfiguroinnin toteuttava toiminnallisuus on erillään ympäristökohtaisista arvoista. Cookbookeissa olevissa resepteissä ei siis tule olla tiettyyn ympäristöön perustuvia arvoja määriteltyinä kiinteästi. Tällaisiksi luokitellaan esimerkiksi käytettävän tietokantapalvelimen ja -skeeman nimi. Ympäristökohtaiset muuttujat tulee pitää erillään esimerkiksi cookbookeissa määriteltävinä attribuutteina, joille voidaan asettaa oletusarvot, ja tarvittaessa ylikirjoittaa nämä ympäristökohtaisesti.

Alkutilassa integraatioiden asennuspaketti eroaa muista sovelluksista siten, että asennuspaketti luodaan erikseen jokaiselle ympäristölle. Asennuspaketissa on varsinaisesta sovelluspaketista erillisissä properties-tiedostoissa määritelty muun muassa ympäristökohtaiset palvelinten nimet, käytettävät tietokantaskeemat ja mitkä integraatiot ovat käytössä. Asennuspaketissa on lisäksi useita asennusta tukevia shell-skriptejä, jotka toimivat pääasiallisena dokumentaationa asennuksen vaiheille. Muidenkin sovellusten asennuspaketit sisältävät asennusta tukevia skriptejä, mutta johtuen sovellusten yksinkertaisemmasta asennuksesta, on toteutus näiden osalta suoraviivaisempaa toteuttaa kuin integraatioille. Integraatioiden osalta tavoitteena on, että ympäristökohtaiset asetukset toteutetaan Chefin konfiguraatiotiedostoilla, eikä asennuspaketissa olevia properties-tiedostoja käytetä. Tällöin samaa asennuspakettia voidaan myös integraatioille käyttää kaikissa eri ympäristöissä.

Alkutilassa integraatioiden tietokannan luonti- ja päivitysskriptit on hajautettu osittain Solita Pulse -sovelluksen pakettiin ja osittain Maanmittauslaitokselle toteutettavien integraatioiden yhteyteen. Konfiguraationhallinnan automatisointi ja sovellusversioiden päivittäminen jatkuvan toimituksen mallilla edellyttää, että tietokantapäivitykset voidaan suorittaa automatisoidusti. Nykyiset kannanluontiskriptit tekevät joitakin oletuksia päivitettävästä kannasta, minkä vuoksi skriptit eivät ole suoritettavissa automatisoidusti. Tavoitteena on osana diplomityötä muuttaa nämä skriptit käyttämään muiden sovellusten tapaan Dbmaintain-työkalua, mikä edellyttää jonkin verran muutoksia skripteihin ja niiden rakenteeseen. Tavoitteena on tämän diplomityön puitteissa koostaa kaikki kannan luontiin ja päivittämiseen

tarvittavat skriptit osaksi Maanmittauslaitoksen integraatiototeutusta. Myöhemmin Dbmaintain-toteutus näille on mahdollisesti laajennettavissa osaksi Solita Pulse -sovelluksen toteutusta. Tässä työssä tätä ei kuitenkaan tehdä, koska muutoksilla voi olla epätoivottavia vaikutuksia muihin Solita Pulsea käyttäviin projekteihin.

5.4 Toteutus

Chefillä toteutettavat ympäristökonfiguraatiot luodaan erilleen varsinaisesta sovellusten toteutuskoodista. Tämä toteutuu siten, että konfiguraatioille käytetään eri versionhallintaa kuin sovelluskoodille. Sovelluspakettien luominen ja tietokantojen päivittäminen toteutetaan erillään ympäristökonfiguraatioista. Toteutuksessa huolehditaan, että valmiit ympäristöriippumattomat sovelluspaketit ovat noudettavissa provisioinnin yhteydessä osaksi konfiguraatiota. Luotavien ympäristöjen sovellusten käyttämät tietokantaskeemat tulee myös olla valmiiksi päivitettyinä vastamaan käytettäviä sovellusversioita.

Ajan tasalla olevien tietokantaskeemojen ja uusimpaan sovelluspakettiin osoittavan URL:n avulla saadaan uudet sovellusversiot asennettua automaattisesti kaikille sovelluspalvelimille ilman manuaalisia toimenpiteitä. Ympäristöjen luominen ja hallinta perustuu Chefille toteutettuihin cookbookeihin, jotka kuvaavat eri palvelimien ja ympäristöjen konfiguraatiot.

Ensimmäisessä alakohdassa on kuvattu toimenpiteet sovellusten käyttämien tietokantaskeemojen luomiseen ja päivitykseen, sekä toimenpiteet valmiiden sovelluspakettien muodostamiseen. Jälkimmäinen alakohta esittelee sovelluspalvelimien konfigurointitoteutuksen sekä tarvittavat vaiheet ympäristöjen luontiin ja hallintaan liittyen.

5.4.1 Tietokantojen ja sovelluspakettien luominen

Sovellusten lähdekoodit ovat Solitan hallinnoimassa Subversion-versionhallinnassa. Sovelluspakettien luominen tehdään jatkuvaan integrointiin tarkoitetulla Jenkins-työkalulla [6]. Tällä tavoin voidaan suorittaa automaattitestit sovelluksille käännön yhteydessä ja tallentaa tuotetut sovelluspaketit artefakteiksi, jotka ovat noudettavissa HTTP-protokollaa käyttäen Jenkins-palvelimelta suoraan provisioitaville sovelluspalvelimille.

Jenkins on käytössä jo ennestään projektitiimillä jatkuvan integraation toteutuksessa. Jenkins mahdollistaa muun muassa ajastettujen tehtävien luonnin ja sen avulla saadaan heti tieto kehittäjille epäonnistuneista käännöksistä ja testisuorituksista. Projektitiimillä on työpaikan avotilassa televisioruutu, jossa on avoinna Jenkinsin-tehtävien tilaa näyttävä projektitiimin toteuttama Radiator-sovellus. Radiator näyttää reaaliajassa tärkeimpien Jenkins-tehtävien tilan, eli onko edellinen

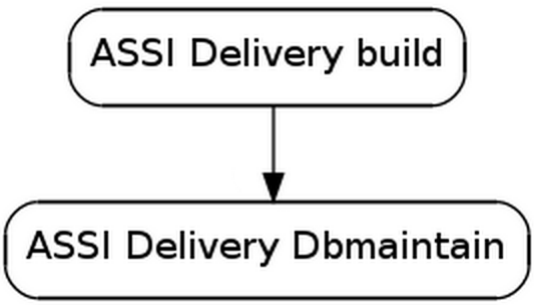
suoritus onnistunut, ja mitkä tehtävät ovat käynnissä.

Sovellusten käyttämien tietokantojen päivittäminen toteutetaan myös Jenkinsillä. Tällöin mahdolliset virheet päivityksissä ovat selkeästi näkyvissä Jenkinsin tuottamassa lokissa. Mikäli kantapäivitykset suoritettaisiin osana sovelluspalvelinten provisiointia, olisi epäselvää miltä sovelluspalvelimelta päivitykset ajettaisiin, jos provisioitavia sovelluspalvelimia olisi useita. Päivitysten ajaminen samaan tietokantaan samanaikaisesti useammalta palvelimelta johtaisi todennäköisesti virheeseen, jolloin tietokannan korjaaminen pitäisi tehdä manuaalisesti tai mahdollisesti luoda koko kanta uudestaan.

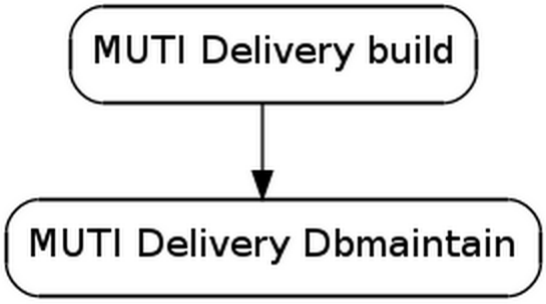
Työvaiheet toteutetaan eri Jenkins-tehtäviksi, jotka ketjutetaan yhdeksi kokonaisuudeksi, jota kutsutaan tehtäväputkeksi (pipeline). Tehtävien eriyttämisellä saadaan mahdolliset virhetilanteet helpommin paikannettua. Tämä mahdollistaa myös tehtäväputken jatkamisen virheiden korjaamisen jälkeen keskeytyneestä tehtävästä eikä koko toimitusputkea välttämättä tarvitse käynnistää alusta alkaen. Ketjutus tehdään asettamalla seuraava tehtävä edellisen tehtävän alavirtaan, jolloin edellisen tehtävän onnistuessa, Jenkins automaattisesti käynnistää seuraavan tehtävän alavirrassa.

Kuvissa 5.3 ja 5.4 on esitelty ketjutetuista Jenkinsin tehtävistä muodostuvat tehtäväputket ASSIn ja MUTIn sovelluspakettien tuottamiseen. Kuvissa olevat tehtäväputket tuottavat asennuspaketin tietylle versionhallinnan tagille. Tageja käytetään projektissa asiakkaalle toimitettavien versioiden lähdekoodien tallentamiseen. Tehtäväputket käynnistetään parametrisoituina Jenkins-tehtävinä, joille annetaan halutun tagin nimi versionhallinnassa. Tehtäväputken ensimmäinen tehtävä noutaa versionhallinnasta tagin Jenkins-palvelimelle ja tuottaa käännöksen tuloksena samanlaisen asennuspaketin, joka asiakkaallekin tarjotaan. Onnistuneen suorituksen lopuksi tehtävä tallentaa artefaktina asennuspaketin, joka välitetään alavirrassa olevalle tietokannan päivittävälle tehtävälle (*ASSI/MUTI Delivery Dbmaintain*). Tietokannan päivittävä tehtävä purkaa asennuspaketin ja suorittaa Dbmaintain-työkalulla tietokannan päivittämisen tehtävälle konfiguroituun tietokantaskeemaan. Tietokannan päivittävä tehtävä tallentaa artefakteina valmiit sovelluspaketit, jotka voidaan Jenkinsistä ladata suoritettaviksi sovelluspalvelimilla.

Käytettävät tietokantaskeemat on luotu tietokannanhallintajärjestelmään etukäteen manuaalisesti järjestelmäkäyttäjällä. Edellytyksenä tehtäväputkien suoritukselle on, että tietokantaskeemat ovat tehtäväputkelle asetettua tagiversiota aiemmassa tilassa. Kantapäivitykset eivät ole tuettuina taaksepäin, mutta aiemmassa tilassa oleva skeema voidaan päivittää uudempaan versioon. Kuvatussa tehtäväputkessa oleviin ASSIn ja MUTIn skeemoihin on lisätty päivityksen jälkeen synteettisesti generoitua testidataa, jotta sovellusten koekäyttö on mahdollista. ASSIn ja MUTIn yhteydessä on todettu synteettisen testidatan vastaavan tarpeisiin riittävän hyvin.



Kuva 5.3: ASSIn sovelluspaketin tuottava tehtäväputki.



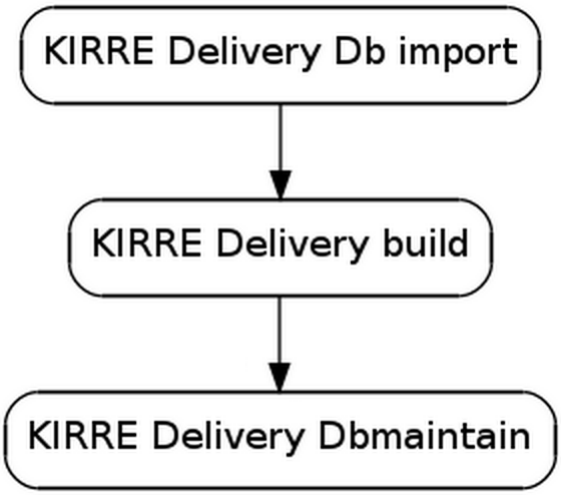
Kuva 5.4: MUTIn sovelluspaketin tuottava tehtäväputki.

Mahdollisesti testidata tullaan korvaamaan asiakkaalta saaduilla tuotantoympäristöstä otetuilla datakopioilla, joista muun muassa henkilöiden henkilötunnukset obfuskoidaan kuvitteellisiksi.

KIRREn sovelluspaketin tuottamiseen tarkoitettu tehtäväputki on rakenteeltaan lähes samanlainen kuin ASSIlla ja MUTIlla. Tehtäväputken malli on kuvassa 5.5. KIRREn tietokannan luominen ja päivittäminen eroaa jonkin verran muista sovelluksista, koska tietomalli ja kantarakenne ovat sovelluskohteen sovelluksista monimutkaisimmat. Todellisuutta vastaavan synteettisen testidatan luominen on tämän vuoksi KIRRElle hankalaa ja työlästä. Tämän vuoksi KIRREn koekäyttöä varten käytetään tuotantoympäristöstä irroitettua datakopiota, josta henkilötietojen osalta data on obfuskoitu.

Tietokantaskeeman alustaminen ja datakopion lataaminen suoritetaan *KIRRE Delivery Db import* -tehtävässä. Datakopio on otettu tuotannossa olevasta tietystä kantaversiosta, joten sen lataamiseksi alustettava tietokantaskeema on ensin saatettava saman version mukaiseen tilaan. Tehtävän alussa tietokantaskeema tyhjennettään ja päivitetään Dbmaintain-työkalulla datakopion mukaiseen versioon. Datakopio ladataan skeemaan käyttäen Oraclen Data pump -työkalua [77]. Lataamisen jälkeen tarvitsee vielä tehdä jälkitoimenpiteitä, joihin kuuluu muun muassa indeksien uudelleenluonti. Tehtäväputkelle asetetusta sovellusversiosta riippuen, pitää tietokantaskeemaa päivittää asteittain myöhempiin versioihin ja suorittaa oikeissa vaiheissa SQL-skripteillä toteutettuja konversioajoja. Konversioajot on pakko suorittaa tietyille versioille, jotta päivittäminen uudempiin versioihin onnistuu. Tämän vuoksi *KIRRE Delivery Db import* -tehtävälle tulee konfiguroida ennen suoritusta manuaalisesti mitä konversioajoja tulee suorittaa.

KIRRE Delivery Db import -tehtävän jälkeen tietokantaskeemaan on ladattu datakopio ja suoritettu tarvittavat konversioajot. Seuraavana tehtäväputkessa on asennuspaketin luonti parametrina asetetusta versionhallinnan tagista *KIRRE Delivery build* -tehtävällä. Luotu asennuspaketti välitetään *KIRRE Delivery Dbmaintain* -tehtävälle, joka päivittää tietokannan ja tallentaa artefakteina asennuspaketista pu-



Kuva 5.5: Tehtäväputki KIRREn tuotantodatakopion lisäämiseen ja sovelluspaketin tuottamiseen.

retut valmiit sovelluspaketit noudettaviksi sovelluspalvelimille.

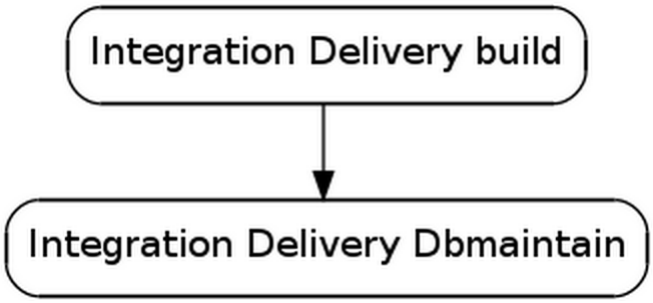
Integraatioiden aikaisemmin ympäristökohtaisen asennuspaketin sijasta luodaan yksi, kaikkiin eri ympäristöihin soveltuva, paketti. Ympäristökohtaisten asetusten asettaminen tehdään Chefin konfiguraatioissa, jolloin varsinaisen asennuspaketin ei tarvitse sisältää kuin integraatiosovellus sekä Solita Pulse -monitorointisovellus. Näin ollen eri ympäristöjen olemassaolo ja uusien ympäristöjen luonti eivät aiheuta muutoksia sovelluskohteen toteutukseen eivätkä asennuspakettien luomiseen.

Integraatioiden ja monitorointisovelluksen käyttämän tietokannan luonti- ja päivitysskripti on hajautettu osittain monitorointisovelluksen pakettiin ja osittain projektin integraatiototeutuksen rinnalle. Kaikki tietokantaskriptit koostetaan yhteen paikkaan projektin integraatiototeutuksen yhteyteen, ja skriptit muokataan muotoon, jossa ne ovat suoritettavissa automaattisesti tyhjään kantaskeemaan.

Aikaisemmissa skripteissä on taulujen luonnin yhteydessä asetettu muun muassa rajoitteet (constraint) taulujen sarakkeille. Kun rajoitteita on jossakin versiossa muutettu, on muutokset tehty taulujen luonnin yhteyteen ja lisäksi tehty erillinen päivitysskripti, joka korjaa aikaisemmat rajoitteet muutosten mukaisiksi. Mikäli näillä skripteillä tietokanta luotaisiin alusta asti, jälkimmäisen korjausskriptin suoritus epäonnistuisi, koska rajoitteet ovat jo uuden toteutuksen mukaisia taulujen luonnin jälkeen. Uudessa Dbmaintain-työkalun mukaisessa tietokantaskriptitoteutuksessa tällaiset olettamukset on korjattu, ja skriptit on nimetty sekä tallennettu Dbmaintain-työkalun mukaiseen hakemistorakenteeseen.

Näiden muutosten ansiosta integraatioiden sovelluspaketin luonti ja tietokannan päivitys voidaan toteuttaa Jenkins-tehtävinä samalla tavalla kuin muillekin sovelluskohteen sovelluksille. Tätä varten Jenkinsiin luodaan kuvan 5.6 mukainen teh-

täväputki, joka koostuu asennuspaketin luonnista (*Integration Delivery build*) ja tietokannan päivityksestä (*Integration Delivery Dbmaintain*). Luotu asennuspaketti kopioidaan ensimmäiseltä tehtävältä tietokannan päivittävän tehtävän artefaktiksi, josta se on noudettavissa Chef-suorituksessa asennettavaksi sovelluspalvelimille.



Kuva 5.6: Tehtäväputki integraatioiden sovelluspaketin tuottamiseen.

Vastaavat tehtäväputket luodaan jokaiselle luotavalle ympäristölle. Tämä voidaan tehdä luomalla Jenkinsillä olemassa olevista tehtävistä kopiot ja konfiguroimalla näihin oikeat tietokantaskeemat sekä muut tarvittavat tiedot. Mikäli hallittavia ympäristöjä on useita, voidaan Jenkins-tehtävien konfiguraatiot irroittaa erillisiksi parametroitaviksi skripteiksi, jotka lisätään versionhallintaan. Tällöin Jenkins-tehtävien konfiguraatiot yksinkertaistuvat versionhallinnasta noudettavien skriptien suorittamiseen ja mahdolliset muutokset tehtävissä saadaan toteutettua kerralla kaikkiin samanlaisiin Jenkins-tehtäviin.

5.4.2 Sovelluspalvelimien konfigurointi

Jenkinsin rajapinta tarjoaa valmiiksi kiinteän URL-osoitteen, joka osoittaa tietyn tehtävän viimeisimpään onnistuneeseen suoritukseen. Tämä URL asetetaan Chef-konfiguraatioihin sovelluspaketin noutamista varten, jolloin sovelluspalvelimen provisioinnissa haetaan automaattisesti sovelluspaketti uusimmasta onnistuneesta käännöksestä. Chef Client -sovelluksen suorittaminen asetetaan suoritettavaksi määritellyin aikavälein asiakassolmuilla osana konfigurointia. Ajastukseen käytetään Unix-pohjaisissa käyttöjärjestelmissä olevaa Cron-työkalua.

Ajastetun provisioinnin ja uusimpaan sovelluspakettiin osoittavan URL:n avulla saadaan uudet sovellusversiot asennettua automaattisesti kaikille sovelluspalvelimille ilman manuaalisia toimenpiteitä. Tehtäväputken ensimmäinen tehtävä voidaan asettaa seuraamaan esimerkiksi versionhallinnan kehityshaaran muutoksia, jolloin kehityshaaraan kohdistuneet muutokset käynnistävät tehtäväputken automaattisesti. Kun tehtäväputki on päättynyt, ajastettu Chef Client -sovelluksen suoritus noutaa uuden sovelluspaketin ja suorittaa asennuksen. Jos sovelluspaketti ei ole muuttunut edellisen provisioinnin jälkeen, ja konfiguraatio on muiltakin osilta samassa

tilassa kuin edellisen provisioinnin jälkeen, Chef Client -sovelluksen suoritus ei tee mitään muutoksia. Toimitusversiota vastaavaan ympäristöön sovellusversioiden päivittäminen tehdään käynnistämällä Jenkinsin tehtäväputki manuaalisesti antamalla parametrina asiakkaalle toimitetun version tagin nimi.

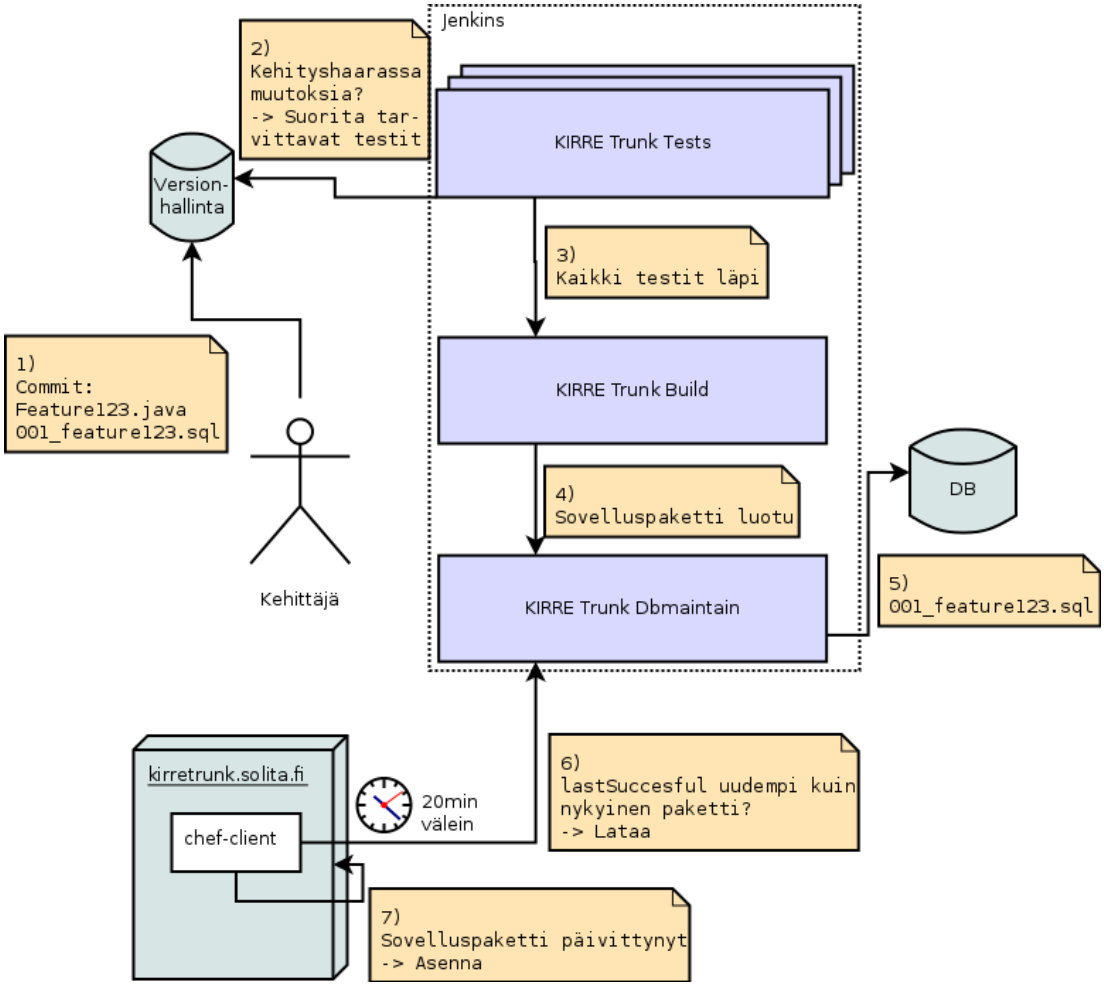
Kuvassa 5.7 on havainnollistettu yksinkertaistettuna kuinka kehittäjän tekemästä muutoksesta saataisiin käynnistettyä automaattisesti tehtäväputki, jonka lopputuloksena uusi, muutoksen sisältävä, sovellusversio on valmis koekäytettäväksi siihen tarkoitetulla sovelluspalvelimella. Ensin kehittäjä työntää muutoksen versionhallinnan kehityshaaraan, jonka muutoksia Jenkinsin testejä suorittavat tehtävät aktiivisesti seuraavat. Kun kaikki muutosta testaavat tehtävät on suoritettu onnistuneesti, käynnistyy tehtäville alavirtaan asetettu käännöstehtävä. Käännöstehtävä tuottaa valmiin sovelluspaketin ja käynnistää tietokannan päivittävän tehtävän.

Kehitysympäristön koekäyttöpalvelimella suoritetaan tietyin aikavälein automaattisesti provisiointi. Provisioinnin osana Chef Client -sovellus lähettää HTTP-pyynnön Jenkinsille. Pyynnöllä kysytään tehtäväputken viimeiseltä tehtävältä onko viimeisimmän onnistuneen suorituksen tallentaman sovelluspaketin muutosaika uudempi kuin sovelluspalvelimella jo olevalla edellisellä asennuspaketilla. Tässä hyödynnetään HTTP:n *If–modified–since*-parametria. Mikäli haettava sovelluspaketti on uudempi, niin provisioinnissa suoritetaan tarvittavat toimenpiteet uuden sovellusversion asentamiseksi. Mahdolliset muutokset konfiguraatioihin toteutetaan sovelluspaketin päivittymisestä riippumatta. Asennuksen jälkeen kehittäjän tekemän muutoksen sisältävä sovellusversio on koekäytettävissä palvelimella.

Mikäli Croniin asetettu kahdenkymmenen minuutin väli provisiointisuorituksille on liian pitkä, voidaan provisiointi käynnistää myös eksplisiittisesti suorittamalla asiakassolmuilla Chef Client -sovellus. Tämä voidaan tehdä myös käyttämällä Chef Serverin hallinnointiin tarkoitettua Knife-työkalua. Knifen avulla voidaan suorittaa komentoja kerralla useilla palvelimilla.

Kuvassa 5.8 on esimerkkikomento, jolla suoritetaan SSH-yhteyttä käyttäen provisiointi Chef Client -sovelluksella kaikilla kehitysympäristön integraatiopalvelimilla. Tällainen toiminto voitaisiin lisätä Jenkins-tehtäväksi käynnistymään automaattisesti tietokannan päivityksen jälkeen, mikäli Jenkinsin suorituspalvelimille on asennettu Knife-työkalu ja sille asetettu mitä Chef Server -palvelinta käytetään.

Cookbookien ja asiakassolmujen hallinnointiin käytetään Solitan ympäristöön luotua Chef-palvelinta eli palvelinta, johon on asennettu Chef Server -sovellus. Kehittäjä lataa tälle palvelimelle työasemaltaan sovelluspalvelinten luontiin tarvittavat Chefin cookbookit. Kaikille sovelluskohteen ympäristöille voidaan käyttää samaa Chef-palvelinta. Jokaiselle ympäristölle luodaan virtuaalikoneet, joita käytetään sovelluspalvelimina. Virtuaalikoneisiin asennetaan CentOS Linux [78], joka asiakkaan ympäristöissä käytettävään Red Hat Enterprise Linuxiin [79] pohjautuva maksuton



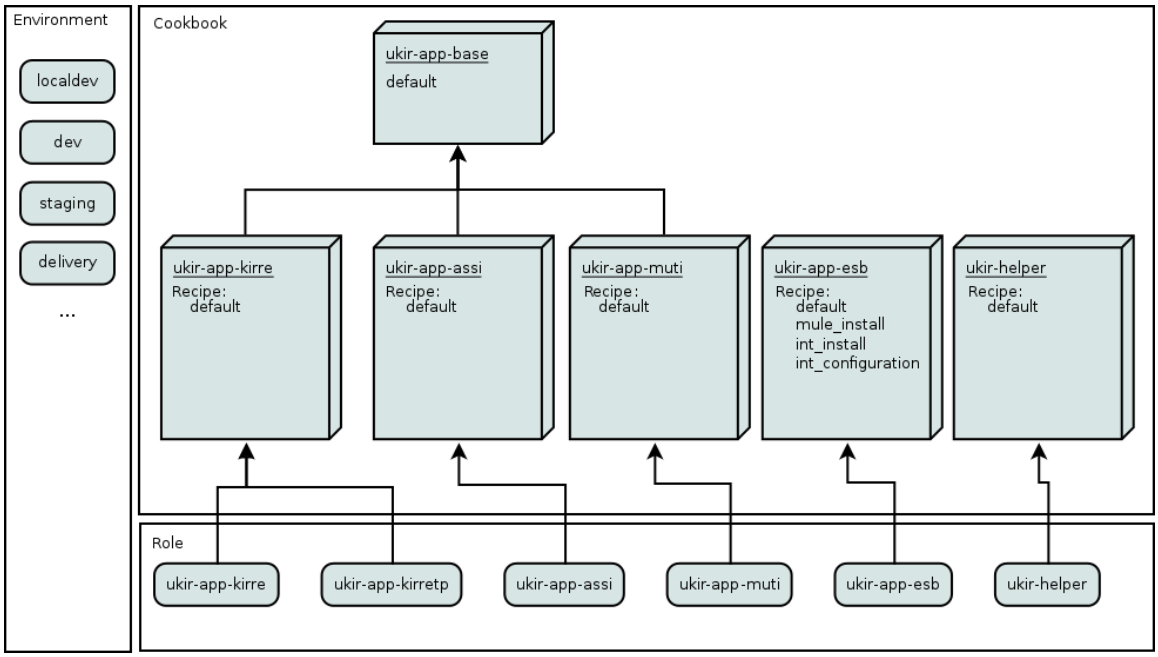
Kuva 5.7: Esimerkki kehityshaaraan tehdyn muutoksen viennistä koekäytettäväksi sovelluspalvelimelle.

Linux-distribuutio.

Kuvassa 5.9 on esitetty roolien, ympäristöjen ja cookbookien käyttö sovelluskohteessa. Roolien perusteella määräytyy, mikä sovelluspalvelimen tehtävä on, eli mikä ympäristön palvelin on kyseessä. Rooleille on määritelty suorituslista (runlist), joka määrittelee mitä reseptejä tulee suorittaa mistäkin cookbookista. Tässä tapauksessa roolien suorituslistassa on vain yksi resepti, joka on roolia vastaavan cookbookin oletusresepti (*default*). Roolia vastaavat cookbookit on esitetty kuvassa rooleista lähtevillä nuolilla. Oletusresepteissä on määritelty mitä muita reseptejä suoritukseen kuuluu. Esimerkiksi *Ukir-app-kirre*-cookbookin oletusreseptissä suoritetaan alussa *Ukir-app-base*-cookbookin oletusresepti. Tässä reseptissä suoritetaan muun muassa Java OpenJDK:n ja Tomcat-web-palvelimen asennuksesta vastaavat reseptit. Näistä OpenJDK:n asennukseen käytetään Chef Supermarketissa [45] olevaa Apache 2.0 -lisensoitua cookbookia [80]. Tomcatin asennus tehdään Solitan toteuttamalla cookbookilla, jota käytetään myös muissa projekteissa.

```
knife ssh 'chef_environment:dev AND role:ukir-app-esb' chef-client
```

Kuva 5.8: Esimerkki provisioinnin käynnistämisestä asiakassolmuille Knife-työkalulla.



Kuva 5.9: Chefin roolien, ympäristöjen ja cookbookien käyttö sovelluskohteessa.

Integraatioiden konfigurointi on kompleksisempi kuin muiden sovelluskohteen sovellusten, minkä vuoksi toteutus on *Ukir-app-esb*-cookbookissa jaettu useampaan reseptiin. Oletusreseptissä suoritetaan Mulen asennus *mule_ install*-reseptillä, joka noutaa Chefin ympäristöissä määritellyn Mule asennuspaketin ja suorittaa tarvittavat asennustoimenpiteet. Tämän jälkeen suoritetaan *int_ install*-resepti, joka noutaa ja purkaa Jenkinsin tallentaman integraatioiden asennuspaketin Mulen sovellushakemistoon. Integraatioiden konfigurointi suoritetaan *int_ configuration*-reseptillä, joka muun muassa hakee Chef Serveriltä samassa Chef-ympäristössä olevien oikeilla rooleille alustettujen sovelluspalvelinten IP-osoitteet konfiguraatioihin. Hakeminen voidaan ohittaa asettamalla eksplisiittisesti sovelluspalvelinten IP-osoitteet tai DNS-nimet Chef-ympäristön asetuksiin attribuutteina.

Rooleille on toteutuksessa määritelty pääasiassa vain reseptien suorituslista. Konfigurointiin käytettävät ympäristö- ja sovelluskohdekohtaiset attribuutit on määritelty Chef-ympäristöissä. Näillä attribuuteilla on määritelty muun muassa eri sovellusten käyttämät tietokantaskeemat ja URL-osoitteet valmiiden sovelluspakettien noutamiseen. Yleiskäyttöisempiä, useissa ympäristöissä samoina pysyviä, attribuutteja on asetettu cookbookien omissa attribuuttitiedostoissa. Chefin attribuuttien asettamisessa käytetyn presedenssijärjestyksen vuoksi cookbookin attribuuttitiedos-

tossa määritellyt oletusarvot voidaan ylikirjoittaa tarvittaessa Chef-ympäristöjen määrittelyissä. Esimerkki tällaisesta attribuutista on Mulen distribuutiopakettien URL-osoite, johon pääsääntöisesti voidaan käyttää samaa arvoa ympäristöstä riippumatta. Tarvittaessa tämä voitaisiin kuitenkin asettaa erikseen yksittäiselle ympäristölle, jos haluttaisiin käyttää esimerkiksi tuoreempia snapshot-versioita vakaiden versioiden sijasta.

Konfiguraatiomuutokset voidaan viedä olemassaoleville sovelluspalvelimille lataamalla muuttuneet cookbookit, environmentit ja roolit Chef-palvelimelle Knife-työkalulla. Ennen tätä muutosten toimivuus on suositeltavaa varmistaa omalla työasemalla Chef Zeron, Vagrantin ja VirtualBoxin avulla [34; 35; 64].

Uuden ympäristön lisääminen konfiguraationhallintaan tehdään muodostamalla sovelluksille tarvittavat asennuspaketit ja tietokannan päivityksen tekevät tehtävät Jenkinsiin. Tämä voidaan tehdä kopioimalla olemassa olevat tehtävät pohjaksi Jenkinsillä ja muuttamalla näistä tarvittavat kohdat, kuten versionhallintahaara ja tietokantaskeema. Lisäksi Chef-konfiguraatiohin tulee luoda uusi Chef-ympäristön määrittävä tiedosto *environments*-hakemistoon. Mikäli jokin attribuutin arvo on asettamatta, niin provisioinnista seuraa virheilmoitus. Uusi Chef-ympäristö lisätään versionhallintaan ja ladataan Chef-palvelimelle. Tämän jälkeen uudet sovelluspalvelimet voidaan liittää Chefin piiriin Knife-työkalun bootstrap-toiminnolla antamalla parametreina palvelimen rooli ja Chef-ympäristön nimi, kuten kuvassa 5.10

```
knife bootstrap kirretp.solita.fi \
  --run-list "role[ukir-app-kirretp]" \
  --environment staging \
  --ssh-user=root
```

Kuva 5.10: Esimerkkikomento uuden sovelluspalvelimen alustamisesta.

Bootstrap-toiminto asentaa asiakassolmulle Chef Client -sovelluksen, jolla suoritetaan asiakassolmun provisiointi. Tämän seurauksena asiakassolmu on valmis toimimaan sovelluspalvelimena. Bootstrap-toiminto tehdään palvelin kerrallaan kaikille ympäristön uusille palvelimille, minkä jälkeen uusi ympäristö on toiminnassa.

6. ARVIOINTI

Diplomityössä toteutettiin konfiguraationhallinnan automatisointi luvussa 4 esitellyille UKIR-hankkeen sovelluksille. Työssä toteutettu automatisointi keskittyi Solitan ympäristöissä olevien koekäyttöön ja testaukseen tarkoitettujen sovelluspalvelinympäristöjen osuuteen. Luvussa 5 esiteltyjä ongelmia pyrittiin ratkaisemaan Chefin ja muiden työkalujen avulla toteutetulla konfiguraationhallinnan automatisoinnilla.

Tässä luvussa arvioidaan asetettujen tavoitteiden vastaavuutta ongelmien ratkaisemiseen, sekä suunnittelun ja toteutuksen onnistumista tavoitteiden täyttämisessä.

6.1 Valmis toteutus

Työssä määriteltiin sovelluskohteen eri sovelluspalvelintyyppien konfiguraatiorakenne Chef-konfiguraationhallintatyökalun käyttämässä muodossa. Määritellyn rakenteen implementoiminen voidaan suorittaa ainakin CentOS- ja Red Hat Enterprise Linux -käyttöjärjestelmiä käyttäviin palvelimiin. Puhtaan palvelimen alustaminen joksikin sovelluskohteen sovelluspalvelimista voidaan tehdä automatisoidusti, jolloin palvelimien luontiin tarvittavan manuaalisen työn määrä on huomattavasti vähäisempi kuin täysin manuaalisesti tehdyllä konfiguroinnilla. Palvelimia voidaan tarvittaessa poistaa ja luoda vapaammin. Muutokset konfiguraatioihin voidaan tehdä keskitetysti, jolloin jokaista palvelinta ei tarvitse konfiguroida erikseen.

Työssä toteutettujen Chefin konfiguraatioden ja Jenkinsin tehtävien avulla voidaan toteuttaa jatkuvan toimituksen malli useisiin eri ympäristöihin, mikä vähentää kehitykseen ja testaukseen liittyvää manuaalisen työn määrää. Automaattisesti tehtävät toiminnot vähentävät toistuviin tehtäviin liittyvää virheiden todennäköisyyttä, koska ihmisten suorittamiin tehtäviin liittyy mahdollisuus inhimillisiin virheisiin ja vaiheiden unohtamiseen.

Projektitiimillä on yhä käytössä aikaisempi manuaaliseen konfigurointiin perustuva ympäristötoteutus asennettuna koekäyttöpalvelimelle. Rinnalla on kuitenkin myös diplomityössä toteutetulla konfiguraationhallintamekanismilla hallittu ympäristö, jonka tarkoituksena on vastata sovellusversioiltaan tuotantoympäristöä. Konfiguraationhallinnan toteutuksen ja toiminnallisuuden rakenne ei ole vielä koko projektitiimin tiedossa, mutta tavoitteena on saada koko kehittäjätiimi ymmärtämään toteutus ja mahdollistaa työn jatkokehittäminen tiimin voimin.

Konfiguraationhallinnasta ei ole vastannut koko projektitiimi, vaan vain osa tii-

mistä. Toteutetulla automatisoidulla mallillakaan tilanne ei tästä muutu itsestään, vaan vaatii aktiivista tekemistä. Projektin kannalta olisi edullista, että sovelluspalvelimien rakenne ja konfiguraatioiden hallinta olisi useampien henkilöiden tiedossa, jolloin tiimin kokoonpanon muuttuminen ja tehtävien vaihtuminen olisi helpommin hallittavissa. Se, onko automatisoitu konfiguraationhallinta toimiva ja haluttu malli projektissa, selviää todennäköisesti myöhemmin, kun malli on ollut käytössä pidempään ja tarve uusille palvelimille sekä konfiguraatiomuutoksille on paremmin konkretisoitunut. Huomattavaa on, että diplomityössä toteutettu automatisoidun konfiguraationhallinnan malli ei ole, ainakaan toistaiseksi, käytössä asiakkaan ympäristöissä.

Konfiguraationhallintatyökaluksi valittu Chef osoittautui toimivaksi ratkaisuksi, jolla toteutus saatiin tehtyä onnistuneesti. Chefillä toteutettaessa vaaditaan kuitenkin huolellista perehtymistä Chefin toimintaan ja dokumentaatioon. Chefissä eri asioita voi ratkaista useilla tavoilla, joista kaikki eivät välttämättä ole yhtä hyviä ja toimivia pidemmällä aikavälillä tarpeiden muuttuessa. Konfiguraationhallinnan olisi hyödyllistä olla koko projektitiimin hallittavissa. Tämän vuoksi olisi edullista, että konfiguraationhallintatyökalujen kompleksisuus ei aiheuttaisi ylimääräisiä haasteita konfigurointiin.

6.2 Vastaavuus vaatimuksiin

Työlle määritellyt tavoitteet on asetettu alkutilan ongelmien perusteella. Tavoitteilla pyritään ratkaisemaan konfigurointiin sekä ympäristöjen eroavaisuuksiin ja puuttumiseen liittyviä ongelmia. Monet tiedon jakamiseen liittyvistä ongelmista ovat kuitenkin sellaisia, joita ei pelkästään teknisellä ratkaisulla voida selvittää.

Toteutettaville sovelluksille tarvitaan yleensä ohjelmistohankkeissa erillinen dokumentaatio, jossa määritellään miten asioiden tulee toimia. Sama pätee myös ympäristöihin ja niiden konfiguraatioihin. Pelkkä toteutus ei ole riittävä ainoaksi määrittelyksi. Toteutus ei välttämättä vastaa asioiden tahtotilaa, koska toteutuksessa voi olla virheitä ja puutteita. Konfiguraatioiden ja ympäristöjen dokumentointi voidaan kuitenkin toteuttaa korkemmalla tasolla kuin yksittäisten konfiguraatiotiedostojen kuvaaminen. Sovelluspalvelinten väliset liittymät ja konfiguraatiorakenne voidaan visualisoida kaavioina ja kuvata tekstimuodossa ottamatta kantaa toteutuksellisiin yksityiskohtiin. Tämän tasoinen dokumentaatio jättää vapauksia toteutukselle ja määrittelee silti minkälaisia ympäristöjen tulee olla.

Chefin cookbookeilla ja reseptien toteutuksella kuvattu dokumentaatio saattaa olla helpompilukuista ja dokumentoida korkeammalla tasolla rakenteen kuin yksittäiset sovellusten konfiguraatiotiedostot ja komentoriviskriptit. Chefin konfiguraatioiden lukeminen vaatii kuitenkin ymmärryksen Chefin toiminnasta ja vaatii rinnalleen erillisen määrittelyn, joka dokumentoi tahtotilan ja korkeamman tason määrittelyn

ympäristöille.

Diplomityön toteutukselle asetetut tavoitteet ja niiden pohjalta tehty toteutus automatisoidulle konfiguraationhallinnalle toteuttavat eri ympäristöjen konfiguroinnin keskitetysti. Toteutus on versioitu, mikä mahdollistaa tarvittaessa erilaisen konfiguraation eri sovellusversioita käyttäville ympäristöille. Tämä on oleellista, koska konfiguraatiot voivat muuttua eri sovellusversioiden välillä.

Diplomityössä tehty toteutus on toistaiseksi käytössä vain Solitan ympäristöille, mikä monimutkaistaa ylläpidettävää konfigurointia. Muun muassa integraatioväylän konfiguraatiototeutus tulee olla edelleen toteutettuna Chef-konfiguraatioiden lisäksi erillisillä properties-tiedostoilla ja komentoriviskripteillä, koska kyseinen prosessi on yhä käytössä asiakkaan ympäristöille. Diplomityössä tehty automatisoidun konfiguraationhallinnan malli on kuitenkin mahdollista ottaa käyttöön myös asiakkaan ympäristöjen hallinnassa. Tätä edesauttaa se, jos malli on testattu ja koettu toimivaksi sekä on aktiivisesti käytössä Solitan ympäristöissä.

Toteutuksella saavutettuja hyötyjä voidaan arvioida soveltamalla aiemmin esiteltyä Continuous Delivery -kirjan konfigurointi- ja toimitusprosessin kypsyysmallia. Kypsyysmallia sovellettaessa on määriteltävä minkä kaiken katsotaan kuuluvaan arvioitavaan prosessiin. Tämän työn sovelluskohdetta arvioitaessa on muun muassa määriteltävä käsittääkö prosessi kaikki sovelluskohteen sovellukset sekä kohdistuuko prosessi koko toimituksen elinkaareen tuotantoympäristöön asti vai vain Solitan ympäristöjen osuuteen. Tässä tapauksessa arvioinnin kannalta on järkevää määritellä prosessin käsittävän kaikki diplomityöhön kuuluvat sovelluskohteen sovellukset Solitan ympäristöjen osalta. Rajoittaminen koskemaan vain Solitan ympäristöjä perustuu siihen, että Solitan ja Maanmittauslaitoksen ympäristöjen ylläpito ja hallinta on eri osapuolten vastuulla eikä kummallakaan osapuolella ole täyttä näkyvyyttä kaikkiin ympäristöihin.

Diplomityössä tehdyllä toteutuksella ei pyritä parantamaan kaikkia kypsyysmallin osa-alueita. Monet osa-alueet vaativat teknisen toteutuksen lisäksi koko tiimiä sekä asiakasta koskevia muutoksia prosesseihin. Tämän diplomityön osalta suurimmat muutokset kypsyysmalliin toteutuivat ympäristöt ja toimitus- (environments and deployment) sekä konfiguraationhallinta-osa-alueilla (configuration management). Ympäristöt ja toimitus -osa-alue oli aikaisemmin lähimpänä taantuvaa tasoa (taso –1). Ympäristöjen hallinta tehtiin manuaalisena työnä, joskin sovelluspäivitykset joihinkin ympäristöihin tapahtui automatisoidusti, mikä on ominaista toistettavalle tasolle (taso 0). Diplomityön toteutuksen myötä konfiguraatiot ovat eriytetty ja versioitu, sekä uusien ympäristöjen luonti on huomattavasti yksinkertaisempaa. Näiden ominaisuuksien myötä osa-alueen taso on noussut selkeästi toistettavalle tasolle (taso 0). Työssä tehdyn toteutuksen myötä samaa toimitusprosessia voidaan käyttää kaikkiin ympäristöihin, ja sovellusten asennus tapahtuu täysin automatisoidusti.

Tämän myötä voidaan prosessin täyttävän myös osa-alueen tason vakaus (taso 1).

Prosessin konfiguraationhallinta-osa-alueen taso oli aikaisemmin ainakin osittain taantuvalla tasolla (taso –1). Ainoastaan integraatioiden konfiguraatiot sekä konfiguraation toteutusskriptit olivat versionhallinnassa. Muiden sovellusten osalta konfigurointi tehtiin täysin manuaalisena työnä. Diplomityön toteutuksen myötä kaikki konfiguraatiot ja niiden toteuttamiseen vaaditut toiminnot ovat versionhallinnassa, mikä täyttää osa-alueen toistettavan tason (taso 0) vaatimukset.

Yksittäisessä ympäristössä käytettyjen sovellusten sovellusversiot on määritelty Chefin ympäristötiedostoissa, minkä myötä voidaan hallita ympäristöissä käytettyjen sovellusversioiden välisiä riippuvuussuhteita. Kirjastoriippuvuudet on jo ennestään hallittu versionhallinnassa olevien Apache Maven -työkalun [81] konfiguraatiotiedostojen avulla. Myös versionhallinnan käyttöön liittyvät periaatteet on määritelty tiimin kesken liittyen muun muassa muutosten kommentteihin, haarojen luontiin ja muutosten yhdistämiseen (merge) eri haaroista. Konfiguraationhallinta-osa-alueen tason voidaan katsoa prosessin osalta täyttävän vähintään vakaan tason (taso 1) vaatimukset. Koko projektitiimi kehittää pääsääntöisesti aina samaa päähaaraa, josta kehittäjät noutavat päivitykset useita kertoja päivässä. Uusien haarojen luonti tapahtuu ainoastaan julkaisujen yhteydessä. Tämän vuoksi konfiguraationhallinnan osa-alue on nyt tasolla mitattavasti hallittu (taso 2).

7. YHTEENVETO

Konfiguraationhallinta on tärkeä osa ohjelmistohankkeita. Sovelluksilla voidaan ratkaista monia ongelmia ja helpottaa erilaisia tehtäviä. Sovellusten tulee olla toimivia ja hyödyllisiä käyttäjilleen. Sovellukset vaativat toimiakseen tietynlaisen ympäristön, jonka toimivuus on yhtä merkittävässä asemassa kuin sovelluksien ominaisuudet. Sovellusten ympäristöille asetetut vaatimukset vaihtelevat tilannekohtaisesti, mutta niitä yhdistävänä tekijänä on kuitenkin, että konfiguroinnin tulee olla jollakin tavalla hallittua. Joissakin tapauksissa manuaalisesti tehtävä tilannekohtainen konfigurointi voi olla riittävä toimintamalli konfiguraationhallintaan. Monimutkaiset ympäristökonfiguraatiot ja toimivuudeltaan kriittiset sovellukset vaativat usein tarkemmin määritellyn ja vakioidun konfiguraationhallintamallin. Automatisoidun konfiguraationhallinnan avulla voidaan vähentää konfigurointiin liittyviä riskitekijöitä ja keskittyä toistettavien tehtävien manuaalisen suorittamisen sijasta laadukkaiden konfiguraatioiden muodostamiseen.

Automatisoidun konfiguraationhallinnan käyttöönottaminen vaatii konfigurointi- ja toimitusprosessien tarkkaa määrittelyä ja vakioitua hallintaa. Automatisoinnilla voidaan kuitenkin saavuttaa merkittäviä etuja liittyen muun muassa prosessien nopeuttamiseen, toistettavuuteen sekä laadunvarmistukseen. Konfiguraationhallinnan automatisointi voi olla haastavaa ja se vaatii usein alussa enemmän työtä kuin manuaalisesti tehtävä hallinta. Pidemmällä aikavälillä automatisointi usein vähentää konfigurointiin tarvittavaa työmäärää muiden saavutettujen etujen lisäksi.

Tässä diplomityössä UKIR-hankkeen sovelluksille toteutetun konfiguraationhallinnan automatisoinnin todettiin olevan toimiva ratkaisu moniin konfigurointiin liittyviin haasteisiin. Suurimmat haasteet toteutuksessa kohdistuivat aikaisemmin dokumentoimattomiin osiin ja konfiguraatioiden muutoksiin sekä siihen, että konfiguraatioita on pitkään hallittu manuaalisena työnä ja erilaisilla skriptitoteutuksilla. Eri ympäristöt eivät myöskään ole olleet yhtenäisiä. Työssä tehdyn toteutuksen myötä konfiguraatiot ovat versioituja ja muutokset hallittavissa versionhallinnan avulla. Toteutetut Chefin konfiguraatiotiedostot dokumentoivat toteutuksen yksityiskohtaisesti, vaikka näiden lisäksi onkin tarpeellista olla erillinen korkeammalla tasolla kuvattu dokumentaatio ympäristöistä.

Yhteenvetona voidaan todeta automatisoidun konfiguraationhallinnan olevan toimiva ratkaisu moniin haasteisiin, jotka saattavat manuaalisella hallinnalla muodos-

tua hankaliksi ongelmiksi eri tilanteissa. Konfiguraationhallintatyökalujen avulla automatisointi helpottuu, mutta työkalut yksinään eivät ole riittävä ratkaisu haasteisiin. Automatisoitu konfiguraationhallinta vaatii eri osapuolilta uudenlaisen suhtautumistavan konfiguraatioihin ja toimitusprosessiin verrattuna manuaaliseen hallintamalliin. Totutuista manuaalisesti suoritettavista toimitusprosessin vaiheista siirtyminen automatisoituihin vaatii työtä, mutta vastineena voidaan saavuttaa merkittäviä etuja muun muassa toimintavarmuudessa sekä prosessien nopeudessa.

LÄHTEET

[1] ITIL V3 Glossary of Terms and Definitions [sanasto]. 2007, AXELOS Limited. [viitattu 14.09.2014] 55 s. Saatavissa: http://www.itil-officialsite.com/nmsruntime/saveasdialog.aspx?lID=910&sID=242.

[2] Aiello, Bob & Sachs, Leslie. Configuration Management Best Practices : Practical Methods that Work in the Real World. Upper Saddle River, NJ, USA 2010, Addison-Wesley. 272 p.

[3] Humble, Jez & Farley, David. Continuous Delivery : Reliable Software Releases Through Build, Test, and Deployment Automation. Upper Saddle River, NJ, USA 2010, Addison-Wesley. 512 p.

[4] Fowler, Martin. Continuous Integration. 2006. [WWW]. [viitattu 15.09.2014]. Saatavissa: http://www.martinfowler.com/articles/continuousIntegration.html.

[5] Duvall, Paul M. Continuous Integration : Improving Software Quality and Reducing Risk. Upper Saddle River, NJ, USA 2007, Addison-Wesley. 336 p.

[6] Jenkins. 2014. [WWW]. [viitattu 23.10.2014]. Saatavissa: http://jenkins-ci.org/.

[7] Nelson-Smith, Stephen. Test-Driven Infrastructure with Chef. Sebastopol, CA, USA 2011, O'Reilly Media, Inc. 90 p.

[8] What is CFEngine? 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://auth.cfengine.com/what-is-cfengine/.

[9] What Is Puppet? 2014. [WWW]. [viitattu 04.09.2014]. Saatavissa: http://puppetlabs.com/puppet/what-is-puppet/.

[10] What Is Chef? 2014. [WWW]. [viitattu 04.09.2014]. Saatavissa: http://www.getchef.com/chef/.

[11] How Ansible Works? 2014. [WWW]. [viitattu 11.09.2014]. Saatavissa: http://www.ansible.com/how-ansible-works/.

[12] Lueninghoener, Cory. Getting Started with Configuration Management. ;login: [verkkolehti]. 36(2011)2, pp. 12–17 [viitattu 04.09.2014]. Saatavissa: https://www.usenix.org/system/files/login/articles/105457-Lueninghoener.pdf.

[13] CFEngine : Customer Success Story. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://auth.cfengine.com/use-cases-university-of-oslo/.

[14] CFEngine : License. 2013. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://github.com/cfengine/core/blob/master/LICENSE.

[15] Which CFEngine Edition is Right for Me? 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://auth.cfengine.com/cfengine-comparison/.

[16] CFEngine 3.6 : Language Concepts. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://docs.cfengine.com/docs/3.6/guide-language-concepts.html.

[17] CFEngine 3.5 : Networking. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://docs.cfengine.com/docs/3.5/manuals-architecture-networking.html.

[18] CFEngine 3.6 : Supported Platforms and Versions. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://docs.cfengine.com/docs/3.6/guide-latest-release-supported-platforms.html.

[19] Relicensing Puppet to Apache 2.0. 2011. [WWW]. [viitattu 16.10.2014]. Saatavissa: http://puppetlabs.com/blog/relicensing-puppet-to-apache-2-0/.

[20] Puppet : Ruby DSL. 2010. [WWW]. [viitattu 16.10.2014]. Saatavissa: http://puppetlabs.com/blog/ruby-dsl.

[21] Puppet : Supported Platforms and System Requirements. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://docs.puppetlabs.com/guides/platforms.html.

[22] Try Puppet Enterprise. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: http://puppetlabs.com/download-puppet-enterprise/.

[23] VMWare. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: http://www.vmware.com/.

[24] Amazon EC2. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: http://aws.amazon.com/ec2/.

[25] What Can You Do with Puppet Enterprise. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: http://puppetlabs.com/puppet/puppet-enterprise-detail/.

[26] Chef 0.5.1 Summary. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://tickets.opscode.com/browse/CHEF/fixforversion/10000.

[27] Chef : License. 2008. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://github.com/opscode/chef/blob/master/LICENSE/.

[28] Chef Documents : Why Chef? 2014. [WWW]. [viitattu 11.09.2014]. Saatavissa: https://docs.getchef.com/chef_why.html.

[29] The Origins of Ansible. 2013. [WWW]. [viitattu 16.10.2014]. Saatavissa: http://www.ansible.com/blog/2013/12/08/the-origins-of-ansible/.

[30] Ansible : Installation. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: http://docs.ansible.com/intro_installation.html.

[31] Ansible : License File. 2012. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://github.com/ansible/ansible/blob/devel/COPYING/.

[32] Ansible : License Information. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: http://www.ansible.com/license/.

[33] Coreutils - GNU Core Utilities. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://www.gnu.org/software/coreutils/.

[34] VirtualBox. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://www.virtualbox.org/.

[35] Vagrant. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://www.vagrantup.com/.

[36] Why Vagrant? 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: https://docs.vagrantup.com/v2/why-vagrant/index.html.

[37] What is Docker? 2014. [WWW]. [viitattu 11.09.2014]. Saatavissa: https://www.docker.com/whatisdocker/.

[38] Test Kitchen. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: http://kitchen.ci/.

[39] Riot Games. Berkshelf. 2014. [WWW]. [viitattu 17.10.2014]. Saatavissa: http://berkshelf.com/.

[40] RSpec. 2014. [WWW]. [viitattu 17.10.2014]. Saatavissa: http://rspec.info/.

[41] Serverspec. 2014. [WWW]. [viitattu 17.10.2014]. Saatavissa: http://serverspec.org/.

[42] What is a Gem? 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: http://guides.rubygems.org/what-is-a-gem/.

[43] Bundler. 2014. [WWW]. [viitattu 16.10.2014]. Saatavissa: http://bundler.io/.

[44] Chef Documents : An Overview of Chef. 2014. [WWW]. [viitattu 9.10.2014]. Saatavissa: https://docs.getchef.com/chef_overview.html.

[45] Chef Supermarket. 2014. [WWW]. [viitattu 15.11.2014]. Saatavissa: https://supermarket.getchef.com/.

[46] Chef Documents : About Ohai. 2014. [WWW]. [viitattu 17.1.2014]. Saatavissa: http://docs.opscode.com/ohai.html.

[47] Alex Dergachev. Vagrant Talk (Drupalcamp NJ & Drupalcamp Ottawa). 2013. [WWW]. [viitattu 9.10.2014]. Saatavissa: https://gist.github.com/dergachev/4698240.

[48] Chef Documents : Resources and Providers Reference. 2014. [WWW]. [viitattu 9.10.2014]. Saatavissa: http://docs.getchef.com/chef/resources.html.

[49] Chef Documents : About Recipes DSL. 2014. [WWW]. [viitattu 20.11.2014]. Saatavissa: https://docs.getchef.com/dsl_recipe.html.

[50] Chef Documents : About the Recipe DSL. 2014. [WWW]. [viitattu 20.11.2014]. Saatavissa: https://docs.getchef.com/dsl_recipe.html.

[51] Julian Dunn. Demystifying Common Idioms in Chef Recipes. 2013. [WWW]. [viitattu 20.11.2014]. Saatavissa: https://www.getchef.com/blog/2013/09/04/demystifying-common-idioms-in-chef-recipes/.

[52] Chef Documents : cookbook_file. 2014. [WWW]. [viitattu 9.10.2014]. Saatavissa: https://docs.getchef.com/resource_cookbook_file.html.

[53] Chef Documents : About Roles. 2014. [WWW]. [viitattu 20.11.2014]. Saatavissa: https://docs.getchef.com/essentials_roles.html.

[54] Chef Documents : About Environments. 2014. [WWW]. [viitattu 20.11.2014]. Saatavissa: https://docs.getchef.com/essentials_environments.html.

[55] Chef Documents : About Attributes. 2014. [WWW]. [viitattu 20.11.2014]. Saatavissa: http://docs.getchef.com/chef_overview_attributes.html.

[56] Chef Documents : About Data Bags. 2014. [WWW]. [viitattu 20.11.2014]. Saatavissa: https://docs.getchef.com/essentials_data_bags.html.

[57] LXC - Linux Containers. 2014. [WWW]. [viitattu 11.09.2014]. Saatavissa: https://linuxcontainers.org/.

[58] Chef Documents : ChefSpec. 2014. [WWW]. [viitattu 21.11.2014]. Saatavissa: https://docs.getchef.com/chefspec.html.

[59] Chef Documents : High Availability. 2014. [WWW]. [viitattu 21.11.2014]. Saatavissa: https://www.getchef.com/chef/high-availability/.

[60] Chef Documents : Chef Manage. 2014. [WWW]. [viitattu 21.11.2014]. Saatavissa: http://docs.getchef.com/manager.html.

[61] Chef Documents : Chef Analytics. 2014. [WWW]. [viitattu 21.11.2014]. Saatavissa: https://docs.getchef.com/analytics.html.

[62] Chef Documents : About Knife. 2014. [WWW]. [viitattu 21.11.2014]. Saatavissa: https://docs.getchef.com/knife.html.

[63] Chef Documents : Chef Solo. 2014. [WWW]. [viitattu 9.10.2014]. Saatavissa: https://docs.getchef.com/chef_solo.html.

[64] Chef Zero. 2014. [WWW]. [viitattu 13.11.2014]. Saatavissa: https://github.com/opscode/chef-zero/.

[65] Julian Dunn. From Solo to Zero: Migrating to Chef Client Local Mode. 2014. [WWW]. [viitattu 9.10.2014]. Saatavissa: https://www.getchef.com/blog/2014/06/24/from-solo-to-zero-migrating-to-chef-client-local-mode/.

[66] Kokkonen, Arvo. Quo Vadis – UKIR tulee. Tietoa maasta [verkkolehti]. (2010)2, s. 1 [viitattu 16.01.2014]. Saatavissa: http://www.maanmittauslaitos.fi/sites/default/files/Tietoa_maasta_22010.pdf.

[67] Tennosmaa, Tiina. UKIR-hanke työllistää KEKEn henkilöstöä vielä useita vuosia. Viisari [verkkolehti]. (2011)1, s. 20–21 [viitattu 30.09.2014]. Saatavissa: http://www.maanmittauslaitos.fi/sites/default/files/viisari_111_9.2.11.pdf.

[68] KIOS-järjestelmän uusi nimi on KIRRE. 2011. [WWW]. [viitattu 30.09.2014]. Saatavissa: http://www.maanmittauslaitos.fi/node/7439.

[69] Kiinnitys. 2014. [WWW]. [viitattu 22.01.2014]. Saatavissa: http://www.maanmittauslaitos.fi/kiinteistot/kiinteistokauppa-kirjaamisasiat/kiinnitys/.

[70] Mule ESB. 2014. [WWW]. [viitattu 30.09.2014]. Saatavissa: http://www.mulesoft.com/platform/soa/mule-esb-open-source-esb.

[71] Apache Tomcat. 2014. [WWW]. [viitattu 21.10.2014]. Saatavissa: http://tomcat.apache.org/.

[72] Oracle Database. 2014. [WWW]. [viitattu 2.10.2014]. Saatavissa: https://www.oracle.com/database/index.html.

[73] Dbmaintain. 2014. [WWW]. [viitattu 2.10.2014]. Saatavissa: http://www.dbmaintain.org/overview.html.

[74] Oracle Database Special-Use Licensing. 2014. [WWW]. [viitattu 2.10.2014]. Saatavissa: http://docs.oracle.com/cd/E11882_01/license.112/e47877/editions.htm#DBLIC119.

[75] Git. 2014. [WWW]. [viitattu 13.11.2014]. Saatavissa: http://git-scm.com/.

[76] GitLab. 2014. [WWW]. [viitattu 13.11.2014]. Saatavissa: https://about.gitlab.com/.

[77] Oracle Data Pump Import 11g. 2014. [WWW]. [viitattu 30.10.2014]. Saatavissa: http://docs.oracle.com/cd/B28359_01/server.111/b28319/dp_import.htm#g1025464.

[78] CentOS Linux. 2014. [WWW]. [viitattu 30.10.2014]. Saatavissa: http://www.centos.org/.

[79] Red Hat Enterprise Linux. 2014. [WWW]. [viitattu 30.10.2014]. Saatavissa: http://www.redhat.com/en/technologies/linux-platforms/enterprise-linux.

[80] Java Cookbook (1.27.0). 2014. [WWW]. [viitattu 15.11.2014]. Saatavissa: https://supermarket.getchef.com/cookbooks/java/versions/1.27.0.

[81] Apache Maven Project. 2014. [WWW]. [viitattu 22.11.2014]. Saatavissa: http://maven.apache.org/.