PALVELINYMPÄRISTÖJEN KESKITETTY KOKOONPANON HALLINTA AVOIMEN LÄHDEKOODIN OHJELMISTOILLA

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

Hämeenlinna, Visamäki, 11.06.2010

Asko Oukka

HAMK
HÄMEEN AMMATTIKORKEAKOULU

OPINNÄYTETYÖ

Tietojenkäsittelyn koulutusohjelma
Hämeenlinna

Työn nimi	Palvelinympäristöjen keskitetty kokoonpanon hallinta avoimen lähdekoodin ohjelmistoilla
Tekijä	Asko Oukka
Ohjaava opettaja	Paula Joukainen
Hyväksytty	_____._____.20_____
Hyväksyjä	

HAMK
HÄMEEN AMMATTIKORKEAKOULU

TIIVISTELMÄ

HÄMEENLINNA
Tietojenkäsittelyn koulutusohjelma
Systeemityö

Tekijä Asko Oukka **Vuosi** 2010

Työn nimi Palvelinympäristöjen keskitetty kokoonpanon hallinta avoimen lähdekoodin ohjelmistoilla

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli selvittää Ixonos Oy:n asettamat vaatimukset keskitetylle kokoonpanon hallinnalle. Lisäksi opinnäytetyössä selvitettiin, mikä avoimen lähdekoodin keskitetty kokoonpanon hallintaohjelmisto soveltuu parhaiten toimeksiantajan, Ixonos Oy:n käyttöön.

Tutkimusmenetelmänä oli vertaileva tutkimus. Vertailu toteutettiin suorittamalla valituilla ohjelmistoilla sovitut keskitetyn kokoonpanon hallinnan eri osa-alueiden testit, joiden avulla voitiin selvittää, kuinka ohjelmistojen avulla kokoonpanon hallinnan eri osa-alueet olivat toteutettavissa.

Työn teoriaosuudessa käsitellään konfiguroinnin määritelmää sekä manuaalisen konfiguroinnin automatisointia. Lisäksi käydään läpi tunnettuja paradigmoja, joiden avulla konfigurointia voidaan automatisoida.

Opinnäytetyön pohjalta tehdyn vertailun perusteella ehdotetaan kahdesta vertaillusta keskitetyn kokoonpanon hallinnan ohjelmistosta Ixonos Oy:n käyttöön Cfengine 3:a. Cfengine 3 menestyi yleisesti ottaen paremmin myös tärkeiksi katsotuilla kokoonpanon hallinnan osa-alueilla. Vuonna 1993 julkaistu Cfengine on avoimeen lähdekoodiin perustuvista palvelinten automatisoidun ylläpidon ohjelmistoista pitkäikäisin ohjelmisto, jota edelleen määrätietoisesti kehitetään.

Avainsanat Palvelinympäristöt, ylläpito, kokoonpanon hallinta, Cfengine, Puppet

Sivut 46 s. + liitteet 8 s.

HÄMEENLINNA
Business Information Technology
Systems Work

Author Asko Oukka **Year** 2010

Subject of Bachelor's thesis Centralized Configuration Management of Server Environments with Open Source Software

ABSTRACT

The goal of this thesis was to gather the requirements set by Ixonos Oy for centralized configuration management. This thesis also presents the proposed centralized configuration management software that is best suited for Ixonos Oy's usage.

A comparative research method was used in this thesis. The comparison was made by executing a set of agreed tests against the chosen sets of software. These tests enabled to measure how each sub-area of configuration management could be implemented.

The theoretical part of the thesis discusses the definition of configuration as well as automation of manual configuration. In addition, it describes the well known paradigms that can be used to automate configuration tasks.

Based on the comparison made in this thesis, out of the two compared sets of software for centralized configuration management, the Cfengine 3 was proposed for Ixonos Oy. On the whole, Cfengine 3 outperformed also in the configuration management sub-areas that were considered more important. Cfengine was initially published in 1993 and it is open source software in the area of centralized configuration management with the longest track and Cfengine is still actively being developed.

Keywords Server environments, system administration, configuration management, Cfengine, Puppet

Pages 46 p. + appendices 8 p.

SANASTO

Avoin lähdekoodi Avoimen lähdekoodin ohjelmiston lähdekoodi on julkisesti saatavilla. Avoimen lähdekoodin ohjelmistoa voi koskettaa lisenssien mukaiset ehdot, jotka rajaavat lähdekoodin kopiointioikeutta, muokkausta sekä jakelua.

ITIL Information Technology Inftrastructure Library on joukko hyväksi havaittuja IT-alalla käytössä olevia menetelmiä ja käytäntöjä.

ATK Automaattinen Tietojen Käsittely tarkoittaa tietokoneiden ja digitaalisen tietoliikenteen avulla tehtävää tietojen muokkaamista, siirtoa, tallennusta ja hakua.

Algoritmi Algoritmi on rajallinen määrä suoritettavia ohjeita tai työvaiheita annetun tehtävänannon ratkaisemiseksi.

Objekti Objekti on luokan ilmentymä (eng. object instance) eli todellinen asia tai toiminnallisuus.

Luokka Luokan avulla voidaan esittää yleisellä tasolla todellisuudessa olevia asioita tai toiminnallisuuksia.

UML Unified Modeling Language on standardoitu yleiskäyttöinen mallintamiskieli. Standardia kehitetään edelleen. UML:n on kehittänyt Object Management Group (OMG).

Paradigma Nykysuomen sanakirjan mukaan: Tieteessä jotain tutkimuksen aluetta hallitseva perusnäkemys, malli, esikuva tai viitekehys, jota sen puitteisiin sopeutuva tutkimus ei aseta kyseenalaiseksi.

Client-server malli Client-server mallissa jaetaan sovelluksen toiminnallisuus palvelun tarjoajalle eli serverille, sekä palvelun käyttäjille eli clienteille.

GPL General Public License on GNU-projektin julkaisema vapaa ohjelmistolisenssi, jonka tarkoituksena on mahdollistaa tietokoneohjelmien käyttö, muokkaus sekä jakelu. Uusin versio kirjoitushetkellä on GPLv3.

Virtuaalipalvelin Virtuaalipalvelin tarjoaa ohjelmistolle fyysistä palvelinta vastaavan käyttöalustan virtuaalipalvelimien kesken jaetussa palvelinympäristössä.

ERB ERB (eng. Embedded Ruby) on tiedostojen sisältöä määrittelevä Ruby-pohjainen ohjelmointikieli.

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aiheena on palvelinympäristöjen keskitetty kokoonpanon hallinta avoimen lähdekoodin ohjelmistoja käyttämällä.

Palvelimen esimerkiksi web-, tietokanta-, tiedosto tai sähköpostipalvelimen käyttöönoton jälkeen palvelin palvelee tuottamalla sille määriteltyjä palveluita. Palvelimen pitäminen toimintakykyisenä vaatii palvelimen ylläpitohenkilöltä toimenpiteitä. Toimenpiteet jakautuvat ennakoivaan, sopeuttavaan, täydentävään ja korjaavaan ylläpitoon.

Palvelimien määrän noustessa kasvaa myös ylläpidon määrä. Mikäli työajasta kuluu päivässä suuri osa ylläpitoon, tulisi pohtia, kuinka toimenpiteitä voisi tehostaa. Palvelimien ylläpitoa voidaan tehostaa käyttämällä ohjelmistoja, joiden avulla ylläpitotoimenpiteitä voidaan tehdä keskitetysti. Keskittämisen avulla voidaan ylläpitotoimenpiteet toteuttaa yhdelle tai useammalle palvelimelle tehokkaimmillaan lähes samalla työmäärällä. Esimerkkinä työmäärä, joka vaaditaan käyttäjien luomisesta yhdelle palvelimelle, on lähes sama riippumatta palvelimien määrästä.

Keskitetyn ylläpidon ohjelmistoja kutsutaan keskitetyn kokoonpanon hallinnan ohjelmistoiksi (eng. configuration management). Keskitetyn kokoonpanon hallinnan ohjelmistoja on saatavilla sekä kaupallisina tuotteina että maksuttomina avoimen lähdekoodin projekteilta (esimerkiksi cfengine, puppet, bcfg2).

Ixonos Oy on kahdeksassa maassa toimiva suomalainen IT-alan yritys, jolla on noin tuhat työntekijää. Ixonos toimii tieto- ja viestintäteknologia-alan palvelumarkkinoilla sekä tarjoaa hosting, järjestelmäylläpidon ja –tuen palveluita.

Tässä opinnäytetyössä on tarkoitus verrata valittuja avoimeen lähdekoodiin perustuvia keskitetyn kokoonpanon hallinnan tuotteita Ixonoksen vaatimusten mukaisesti. Tutkimusongelmana onkin selvittää Ixonoksen asettamat vaatimukset keskitetylle kokoonpanon hallinnalle. Toisena tutkimusongelmana on määritellä Ixonoksen käyttöön soveltuva kokoonpanon hallintajärjestelmä.

Valtaosa Ixonoksen hosting-palveluiden ylläpidon käyttämistä ohjelmistoista on avoimen lähdekoodin ohjelmistoja, joten avoimen lähdekoodin sovelluksen valitseminen kaupallisen sijaan oli luontevaa.

Kokoonpanon hallinta tarkoittaa tässä yhteydessä palvelimen käyttöjärjestelmän alaisuudessa suoritettavia konfigurointi-, ohjelmistopaketin asennus-, komentojen tai komentojonojen suoritus- ym. toimenpiteitä. Käyttäjien työasemien ylläpito ja hallinta on rajattu työn ulkopuolelle. Lisäksi opinnäytetyössä ei tulla keskittymään Ixonoksen organisaatiossa käytettäviin toimintatapoihin, prosesseihin tai työkäytänteisiin liittyen kokoonpanon- ja muutoksen hallintaan, sillä nämä on määritelty erillisissä Ixonoksella käytettävien ITIL-viitekehyksen mukaisissa työkäytänteissä.

Aiheen valintaan vaikuttivat aiheen kiinnostavuus ja haastavuus. Opinnäytetyön kirjoittaja on toiminut usean vuoden ajan erilaisten palvelinympäristöjen käyttöönotto- sekä ylläpitotehtävissä, joihin kuuluu muun muassa palvelimien ja verkkolaitteiden keskitetty valvonta sekä diagnosointi. Palvelinympäristöjen keskitetty ylläpito kokoonpanon hallinnan keinoin onkin ollut varsin luonteva valinta opinnäytetyön aiheeksi.

Työn keskeiset käsitteet ovat palvelinympäristöt, ylläpito sekä kokoonpanon hallinta.

2 PALVELINYMPÄRISTÖN YLLÄPITO

Palvelimen pitäminen toimintakykyisenä vaatii palvelimen ylläpitohenkilöltä toimenpiteitä. Toimenpiteet voidaan jakaa Swansonin (1976, 492-497) ohjelmistojen ylläpidon luokittelun mukaisesti mukauttavaan-, täydentävään-, korjaavaan- sekä Pressmanin (2001, 23) mukaan myös ennakoivaan ylläpitoon.

Ennakoivat ylläpitotoimenpiteet ovat ylläpitotoimenpiteistä vähiten akuutteja ja niiden tarkoituksena onkin muokata tietojärjestelmää vastaamaan tulevia vaatimuksia.

Mukauttavien toimenpiteiden avulla voidaan tietojärjestelmä muuntaa uuteen käyttötarkoitukseen tai vastaamaan muuttuneita vaatimuksia. Täydentävien toimenpiteiden avulla voidaan tietojärjestelmiin tehtävillä muutoksilla tehostaa tietojärjestelmien toimintaa. Lopuksi puolestaan korjaava ylläpitoa käytetään korjaamaan tietojärjestelmissä havaittuja ongelmia. Swanson (1976, 492-497).

Campi ja Bauer (2009, 10 – 11.) luettelevat ylläpitoon kuuluvan myös seuraavia aikaavieviä toimenpiteitä:

- Tietojärjestelmien suunnittelu ja toteutus annettujen vaatimusten perusteella.
- Tietojärjestelmän käyttäjien hallinta.
- Tietojärjestelmien tietoturva-asioista huolehtiminen.
- Tietojärjestelmän käyttöjärjestelmän asennus, muutokset sekä päivitys.

Käytännössä palvelinympäristön ylläpitoon voi kuulua tehtäväkentästä ja toimenkuvasta riippuen tietojärjestelmien yleistä ylläpitoa. Nämä tehtävät voivat vaihdella tietojärjestelmän varusohjelmistojen ylläpidosta, verkkoyhteyksien- sekä tietojärjestelmien fyysisten laitteiden ylläpitoon. Lisäksi tietojärjestelmien laitteiden-, ohjelmistojen-, lisenssien-, käyttäjätunnusten-, inventaariotietojen- sekä kokoonpanon keskitetty hallinta on osa-alue, joka on usein ylläpitohenkilökunnan vastuulla.

2.1 Ylläpidolle asetetut tavoitteet

Tietojärjestelmien ylläpidon yleinen tavoite on varmistaa tietojärjestelmien toiminnallisuus. Tietojärjestelmän tarjoamalle palvelulle voidaan asettaa hyvinkin korkeita palveluvaatimuksia, jotka voivat koskettaa palvelun vasteaikoja, palveltavien yhteyksien yhtaikaista määrää tai esimerkiksi järjestelmän palvelukatkojen pituuksia.

Mikäli tietojärjestelmät luokitellaan tärkeiksi tai kriittisiksi, asetetaan tietojärjestelmien ylläpidolle lisää vaatimuksia. Campi ja Bauer (2009, 10 – 11.) ovat listanneet ylläpidon toimenpiteiksi myös tietojärjestelmien palveluiden valvonnan sekä tietojärjestelmien varmistukset ja palautukset.

Kriittisten järjestelmien tulee olla myös vikasietoisia. Vikasietoinen tietojärjestelmä vaatii kriittisten osakokonaisuuksien kahdentamisen varajärjestelmän avulla. Kahdennuksen ansiosta tietojärjestelmä voi pysyä toimintakykyisenä vaikka järjestelmän yksi kokonaisuus lakkaisi toimimasta. Tietojärjestelmissä voidaan toteuttaa kahdennusta tietojärjestelmän pienistä osakokonaisuuksista aina koko järjestelmän kahdentamiseen asti. Tyypillisesti kahdennusta käytetään tiedon tallennuksessa, tietoliikenneyhteyksissä sekä tietojärjestelmien palveluiden osakokonaisuuksissa.

Osittain samat vaatimukset voivat päteä myös vähemmän kriittisille järjestelmille. Voidaankin sanoa, että järjestelmän luonteesta riippumatta ylläpidon tärkein tavoite on pitää tietojärjestelmät toimintakykyisinä.

2.2 Ylläpidon työn tehostaminen

"Ihmiset eivät kykene toteuttamaan järjestelmän muutoksia yhtä johdonmukaisesti kuin tietokoneet. Mikäli ylläpidät useita samanlaisia palvelimia manuaalisesti, palvelimiesi kokoonpanot tulevat ajan myötä eroamaan toisistaan. Riippumatta manuaalisen ylläpidon luontaisista ongelmista, automatisoidussa ylläpidossa on useita etuja, kuten työvaiheiden toistettavuus. Työvaiheiden toistettavuus auttaa sinua ymmärtämään sekä dokumentoimaan ylläpitämiesi palvelimien kokoonpanot. Mikäli automatisoit järjestelmämuutokset, voi kokoonpanoprosessista tulla toistettava. Voit rakentaa vikatilanteesta toipuvia järjestelmiä, jotka luottavat käyttöjärjestelmien uudelleen asennukseen sekä muutosten käyttöönottoon käyttöjärjestelmän varmuuskopioista palautuksen sijaan. Järjestelmämuutosten automatisointi tarkoittaa myös vähemmän työtä pidemmällä aikavälillä." (Borwick, J. 2006, 39.)

Kuten edellä Borwick viittaa, manuaalisen ylläpidon keinot voivat riittää tiettyyn pisteeseen asti. Ylläpidettävien palvelimien määrän sekä ylläpitotoimenpiteiden määrän kasvaessa tullaan ennen pitkää ajautumaan ongelmiin. Tässä vaiheessa havaitaan, että olemassa olevilla henkilöstöresursseilla, ylläpitotoimenpiteet ovat muuttuneet ongelmien perässä juoksemiseksi kehittävän ylläpidon sijaan. Tällöin voidaan sanoa, että kriittinen piste on saavutettu. Ennen tähän tilanteeseen joutumista olisi pyrittävä löytämään keinoja ylläpitotoimenpiteiden tehostamiseksi. Tässä opinnäytetyössä vertaillaan kahta ylläpitotoimenpiteiden automatisoimiseksi soveltuvaa keskitetyn kokoonpanon hallinnan ohjelmistoa, joiden avulla ylläpitotoimenpiteitä voidaan tehostaa.

3 KONFIGUROINTI – KOKOONPANON HALLINTAA

Sana konfigurointi (eng. configuration) tarkoittaa toimenpidettä, jonka avulla pyritään saavuttamaan järjestelmässä haluttu tavoite tai tila eli konfiguraatio (eng. configuration). Lause ”yritin konfiguroida sitä” ilmaisee tekemistä (eng. configure), jossa tekijä teki konfiguraatiotoimenpiteitä. Englannin kielen sana ”configuration” tarkoittaa ensisijaisesti muotoa, hahmoa tai ääriviivaa. Esimerkkinä rosoisesta pinnan muodosta voidaan sanoa englanniksi *”the configuration of the ground was rugged”*. Tuotantoteknisesti sekä ATK:ssa, konfiguraatiolla tarkoitetaan yleensä kokoonpanoa.

Sabin ja Weigel (1998, 42-43) määrittelevät konfiguroinnin suunnittelun erikoistapaukseksi, jossa konfiguroitavaa kokonaisuutta rakennetaan rajatusta määrästä määriteltyjä komponenttityyppejä, jotka voivat olla liitoksissa toisiinsa ennalta määrätyillä tavoilla. Sabinin sekä Weigelin mukaan, konfiguroinnin ydin piilee soveltuvien komponenttien valinnassa sekä annettujen tavoitteiden täyttävän komponenttien kombinaatioiden löytämisessä.

Kokoonpanon hallinnalla tässä opinnäytetyössä tarkoitetaan pääsääntöisesti palvelimen käyttöjärjestelmän, varusohjelmien sekä palvelimen eri resurssien konfigurointia. Englanninkielisissä artikkeleissa tästä käytetään termiä ”configuration management”. Tässä dokumentissa ei käytetä termiä konfiguraatiohallinta, koska tämä termi yleensä liitetään ohjelmistotuotannon automatisointiin (ml. versionhallinta) – eng. ”software configuration management”.

Alla on esitettynä kaksi käytännön esimerkkiä konfiguroinnista.

Esimerkki 1 : Asiakas tilaa ravintolassa haluamansa pizzan

Tilatessasi pizzeriassa listalta haluamasi pizzan, rakentaa leipuri haluamasi reseptin mukaisen pizzan. Voimme myös sanoa, että pizzan kokoonpano perustuu reseptin mukaiseen konfiguraatioon.

Esimerkki 2 : Asiakas tilaa autokaupassa uuden auton

Jos tilaisit automyyjältä uuden auton, tiedustelisi hän sinulta haluamasi auton kokoonpanoa eli auton merkin, mallin, värin, moottorityypin, moottorin koon, istuinten verhoilun, jne. Asiakkaalle myöhemmin toimitettava auto on valmistettu ja konfiguroitu asiakkaan toiveiden mukaisesti.

Tilauksen tekijän kannalta esimerkki 1 ja esimerkki 2 eivät juuri eroa toisistaan (lukuun ottamatta tuotteiden kovin erilaista hintaa ja toimitusaikoja). Molemmissa tapauksissa tuote konfiguroidaan asiakkaan toiveiden mukaisesti.

Tuotteen konfiguroinnin sekä valmistuksen kannalta on esimerkeissä merkittävä ero. Esimerkissä 1 tehdään konfigurointi ja valmistus alusta loppuun manuaalisesti. Esimerkissä 1 tämä on tarkoituksenmukaista,

koska tämä pitää valmistuskustannukset alhaisina. Esimerkissä 2, valmistuksen ja kokoonpanon monivaiheisuudesta, komponenttien runsaasta määrästä ja pitkästä valmistusajasta (useita päiviä) johtuen, manuaalisten työvaiheiden sijaan käytetään valmistuksessa runsaasti automaatiota. Automatisoinnin ansiosta saavutetaan valmistuksessa ja kokoonpanossa merkittäviä etuja manuaaliseen työhön verrattuna; nopeus, tasalaatuisuus sekä myös alhaisemmat valmistuskustannukset. Tuotteen konfigurointi tapahtuu esimerkissä 2 pääsääntöisesti automaattisesti; asiakkaan tilaaman tuotteen konfiguraatiotiedot kulkevat kokoonpanoprosessissa mukana sähköisesti. Tämän ansiosta, auton kokoonpanossa toteutetaan vain ne työvaiheet, jotka aikaansaavat asiakkaan toivoman kokoonpanon.

3.1 Kokoonpanon hallinnan tavoitteet

Oli kyseessä ruoka-annoksen valmistaminen, palvelinympäristön rakentaminen tai auton valmistus, on tekijän varmistettava, että lopputuote vastaa toivottua kokoonpanoa. Mikäli kokoonpanon rakentaminen koostuu manuaalisista toimenpiteistä, on inhimillisen erehdyksen vaara aina olemassa. Moni meistä on varmasti ollut tilanteessa, jossa ravintolassa pöytään tuotu annos ei olekaan se mitä oli tilattu. Kun ostaja tilaa automyyjältä uuden auton, hyväksyy tilaaja allekirjoituksellaan tilauksessa mainitun kokoonpanon. Tämä kokoonpano välitetään sähköisesti autotehtaan järjestelmään, jonka avulla tilaajan toiveet löytävät tiensä autonvalmistuksen eri vaiheisiin. On siis käytännössä mahdotonta, että asiakkaalle toimitettava auto ei vastaisi niitä alkutietoja, jotka hän on automyyjälle tilausta tehdessä toimittanut. Selvää on, että kokoonpanon hallinnan yksi tärkeimmistä tavoitteista on varmistaa, että lopputulos vastaa aiemmin ilmoitettua kokoonpanoa.

Kuten aiemmin on todettu, palvelimien määrän noustessa kasvaa myös vaadittavan ylläpidon määrä. Palvelimien määrän kasvaessa on myös todennäköistä, että palvelimien kokoonpanot eivät ole identtisiä, vaan eroavat toisistaan (eri käyttöjärjestelmä, eri varusohjelmistot, eri käyttäjätilit, eri ryhmät, eri tavalla jaetut oikeudet jne.). Palvelimien runsas määrä ja erilaiset kokoonpanot aiheuttavat haasteita ylläpidolle. Jotta vastaavien ympäristöjen kokoonpanoja voitaisiin hallita, on tavoitteiksi asetettava kokoonpanon hallinnan automatisointi sekä mahdollisuus hallita erilaisia kokoonpanoja.

Edelliset esimerkit koskivat tapauksia, joissa rakennetaan tai valmistetaan uusi tuote tai kokoonpano. Alla on esitettynä kaksi esimerkkiä olemassa olevien tuotteiden tai järjestelmien kokoonpanon hallinnasta. Esimerkkien 1 ja 2 toimenpiteistä osa vaatii ajoittaisia toimenpiteitä (alleviivatut), ja osa toimenpiteistä tulee suorittaa vain, jos niille on erityinen tarve.

Esimerkki 1 : Olemassa olevan palvelinympäristön pitäminen toimintakuntoisena vaatii muun muassa seuraavia toimenpiteitä:

- Palomuurisääntöjen lisääminen
- Saatavilla olevien tietoturvapäivitysten listaaminen
- Tarkistus, onko palvelimelle listattu asennettavia komponentteja
- Asennettavaksi listattujen komponenttien asentaminen
- Käyttöjärjestelmän lisäkonfigurointi
- Varusohjelmien konfigurointi
- Käyttäjätilien lisääminen tai poistaminen
- Oikeuksien lisääminen tai poistaminen käyttäjiltä

Esimerkki 2 : Matkapuhelinverkon pitäminen toimintakuntoisena vaatii esimerkiksi seuraavia toimenpiteitä:

- Tukiasemille suoritettavien ohjelmistopäivitysten suorittaminen
- Tukiasemien asetusten (esimerkiksi taajuudet) määrittäminen
- Tukiasemalta kerättävien suorituskykyarvojen määrittäminen
- Kerättyjen suorituskykyarvojen noutaminen

Edellisten perusteella kokoonpanon hallinnalle voidaan myös asettaa tavoitteiksi kokoonpanon täsmällinen (ajastettu) hallinta, kokoonpanon hallinta tiettyjen ehtojen täyttyessä sekä kokoonpanon hallinta vain tarvittaessa. Yhteisiksi kokoonpanon hallinnan tavoitteiksi voidaan luetella seuraavat asiat:

- Kyettävä määrittelemään yksiselitteisesti toteutettava kokoonpano.
- Kokoonpanon hallinnan automatisointi eli konfigurointitoimenpiteiden suorittaminen yhdelle tai useammalle järjestelmälle ilman tarvetta vastaaviin manuaalisiin toimenpiteisiin.
- Kokoonpanojen hallinnassa tapahtuvien konfigurointivirheiden minimoiminen.
- Kokoonpanojen hallinta riippumatta, ovatko hallittavat kokoonpanot samanlaisia vai erilaisia.
- Toistuvien konfigurointitoimenpiteiden täsmällinen soveltaminen eli soveltaminen haluttuna ajankohtana.
- Konfigurointitoimenpiteiden soveltaminen vain tarvittaessa tai tiettyjen ehtojen toteutuessa.

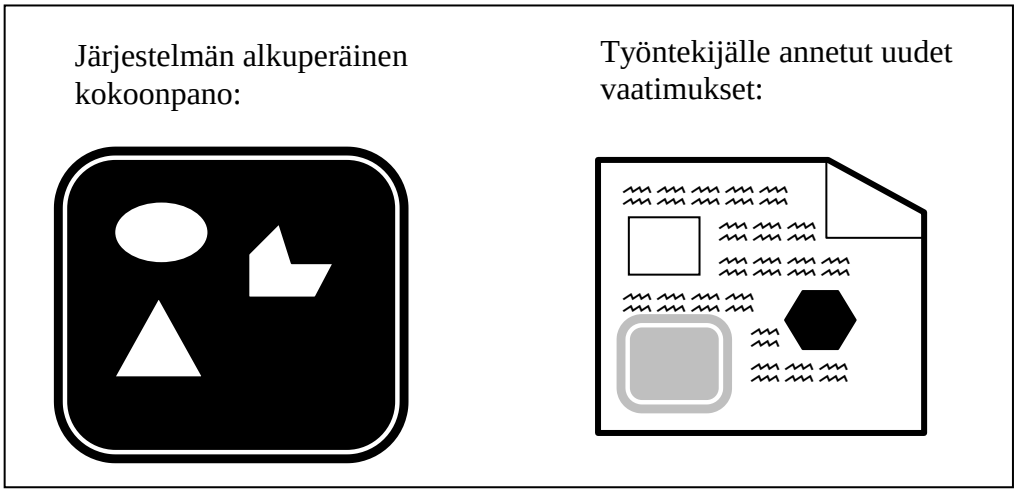
3.2 Konfiguroinnin teoriaa

Konfiguroinnin tavoitteena on siis aikaansaada soveltuva lopputulos tai järjestelmässä haluttu kokoonpano tai tila eli konfiguraatio. Voidaksemme toteuttaa kokoonpanon hallintaa automatisoidusti, on tunnistettava manuaalisen konfiguroinnin toimenpiteet.

3.2.1 Manuaalisen konfiguroinnin vaiheet

Tutkitaanpa alla olevaa esimerkkiä, jossa työntekijän tulee konfiguroida järjestelmä määrityksiä vastaavaksi.

Esimerkki 1 : Työntekijälle annetaan tehtäväksi muuttaa olemassa oleva järjestelmä vastaamaan uusia vaatimuksia. Kuvassa (Kuva 1) on esitettynä järjestelmän kokoonpano alussa sekä työntekijälle annetut uudet vaatimukset järjestelmää koskien.

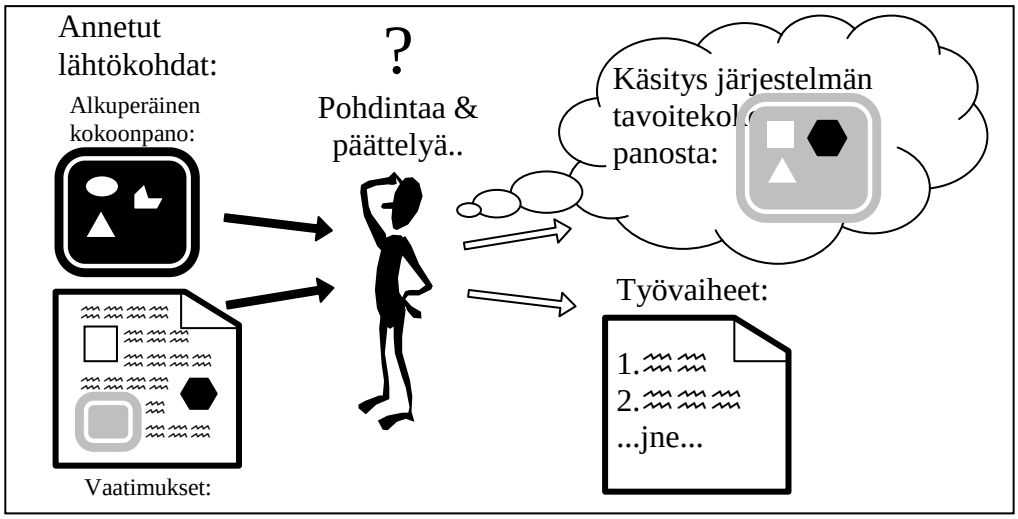


Kuva 1. Alkutilanne

Kuten kuvassa (Kuva 2) on esitetty, työntekijälle annettu tehtävä vaatii työntekijältä pohdintaa ja päättelyä, jonka seurauksena työntekijällä on tarvittavat tiedot järjestelmän konfiguroimiseksi:

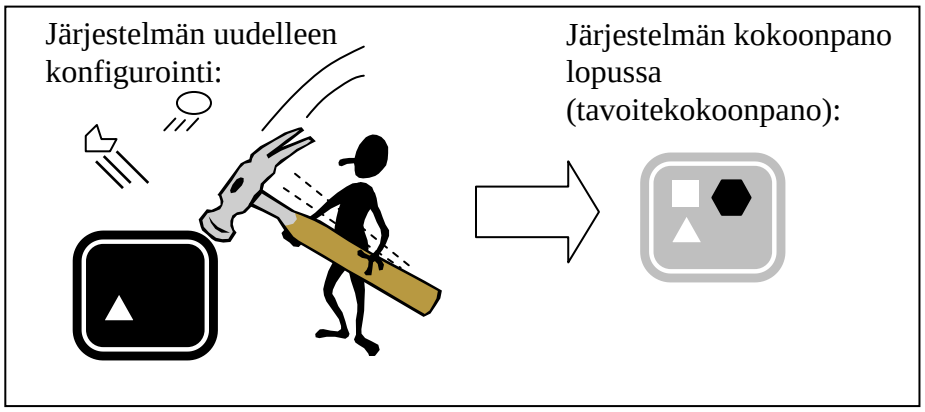
- Käsitys järjestelmän tavoitekokoonpanosta
- Vaadittavat konfiguroinnin työvaiheet tehtävän toteuttamiseksi

Luonnollisesti esivaatimuksina tehtävän toteuttamiseksi on työntekijällä oltava kokemus sekä tietotaito järjestelmästä. Konfigurointijärjestelmässä tästä käytetään termiä ”domain knowledge”. Ilman näitä esivaatimuksia, ei työntekijällä olisi valmiuksia tehtävän suorittamiseksi.



Kuva 2. Pohdinta- ja päättelyvaihe

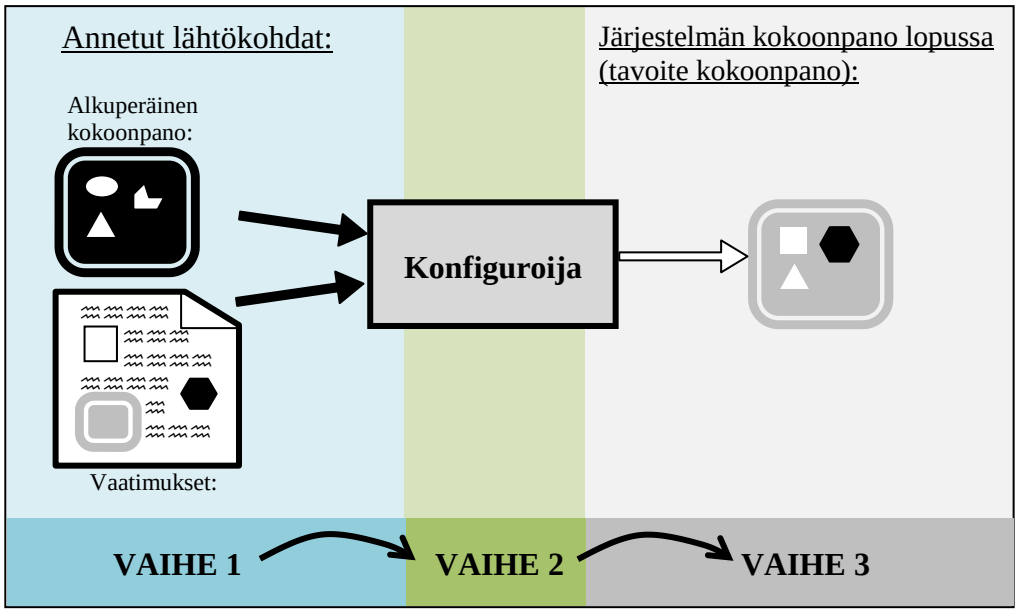
Lopuksi työntekijä voi konfiguroida järjestelmän käyttämällä listaamiaan työvaiheita saavuttaakseen tavoitekokoonpanon (Kuva 3).



Kuva 3. Konfigurointivaihe

3.2.2 Järjestelmän automatisoitu konfigurointi

Luvussa 3.1 todettiin erääksi kokoonpanon hallinnan tavoitteista kokoonpanon hallinnan automatisointi eli konfigurointitoimenpiteiden suorittaminen ilman manuaalisia toimenpiteitä. Tutkin seuraavaksi, kuinka luvussa 3.2.1 esitetty järjestelmän manuaalinen konfigurointi toteutetaan automatisoidusti. Ilman tarkempia tietoja automatisoidusta konfiguroinnista kuvaan aluksi luvussa 3.2.1 esitetyn järjestelmän manuaalisen konfiguroinnin automatisoidusti kuvan (Kuva 4) mukaisesti.



Kuva 4. Järjestelmän automatisoitu konfigurointi: malli 1

Kuten manuaalisessakin konfiguroinnissa, kuvassa (Kuva 4) esitetyssä mallissa, vaiheessa 1 konfiguroijalle annetaan syötteenä alkutiedot, eli järjestelmän alkuperäinen kokoonpano sekä järjestelmän tavoitekokoonpanolle asetetut vaatimukset. Vaiheessa 2 konfiguroija suorittaa konfiguroinnin työvaiheita. Suoritettuaan konfiguroinnin (vaihe 3), tuottaa konfiguroija tavoitekokoonpanon. Kuvassa (Kuva 4) esitetty malli esittää alustavaa tavoitetta, miten luvussa 3.2.1 suoritettava manuaalinen konfigurointi toteutettaisiin automatisoidusti. Automatisoitu konfigurointi vaatii toimiakseen myös muuta lisätietoa järjestelmästä, joten kuvassa (Kuva 4) esitettyä mallia tulee täydentää.

Sabin ja Weigel (1998, 43) vertaavat konfigurointitoimenpidettä mihinkä tahansa ongelmanratkaisutoimenpiteeseen, joka koostuu kahdesta eri vaiheesta:

- **VAIHE 1:** Esittely eli representaatio:

  Ongelman kuvaus

- **VAIHE 2:** Algoritmien käyttö:

  Ongelmaan sovelletaan algoritmeja, jotka tuottavat lopulta ratkaisun ongelman kuvaukseen perustuen

Vaiheessa 1 tehdään tavoiteympäristön esittely tai kuvaus (eng: domain knowledge) sekä määritellään tavoitekokoonpano. (Sabin & Weigel 1998, 43.)

Tavoiteympäristön esittelyn yhteydessä tulee kuvata sovelluskohtaisesti olemassa olevat objektit, objektien tyypit (luokat) sekä objektien ilmentymien (instance) väliset liitokset. Todellisuudessa suurimmat haasteet konfigurointiongelman ratkaisussa liittyvät esittelyvaiheessa tehtävään tavoiteympäristön kuvaamisen sen monimutkaisuudesta johtuen.

Edelleen esittelyvaiheessa objektien välisten erilaisten liitosten sekä liitosten kuvauksien monimutkaisuudesta johtuen suurimmat ongelmat koskevat esityksien ylläpitoa. (Sabin & Weigel 1998, 43.)

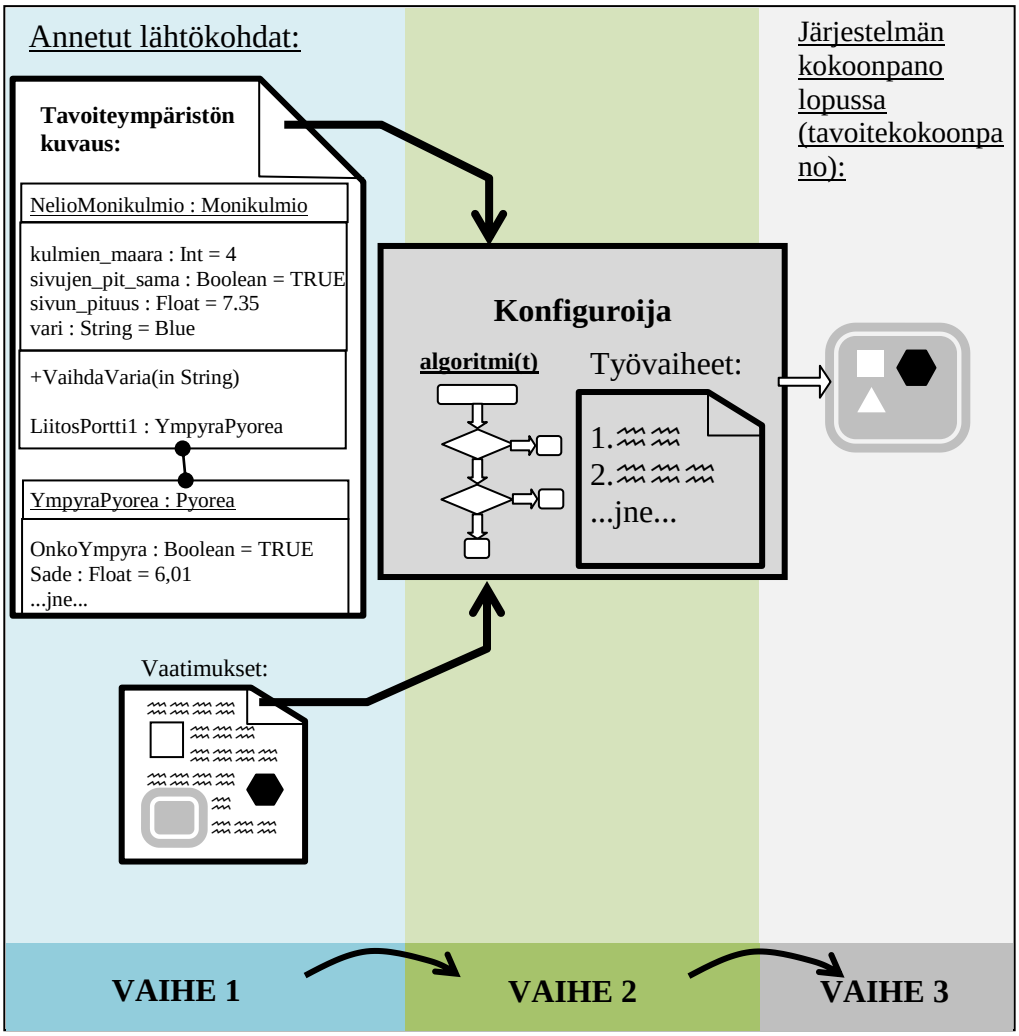
Sabinin ja Weigelin (1998, 43.) mukaan konfiguroijan tulisi kyetä ymmärtämään vähintään seuraavat objektien liitostyypit:

- määritelmä ”on jotakin” : (is-a)
- ryhmitys ”osana jotakin” : (part-of)
- käyttäjän määrittelemät liitokset, mukaan lukien itsenään olevat objektit
- paikalliset rajoitukset (rakenteelliset, laskennalliset, geometriset, perusluvut (kardinaalisuus) jne.
- yleiset (global) rajoitukset (resurssirajoitukset, optimointikriteerit, jne.).

Kokoonpanoa määriteltäessä tulee kuvata ne vaatimukset, jotka lopputuotteen on täytettävä. Lisäksi tulee tehdä kokoonpanon toimintaympäristön kuvaus sisältäen mahdolliset ratkaisun hakua helpottavat optimointikriteerit. (Sabin & Weigel 1998, 43.)

Vaiheessa 2 konfiguroija käy läpi työvaiheet. Konfiguroija ratkoo ongelmaa ennalta määriteltyjä menetelmiä ja ohjeita eli algoritmeja soveltamalla. On mahdollista, että ennen optimaaliseen tulokseen päätymistä, konfiguroija on käynyt läpi useita konfiguraatioratkaisuja. (Sabin & Weigel 1998, 43.)

Edellisen pohjalta voimme muokata kuvaa (Kuva 4) liittämällä siihen puuttuvat osat eli tavoiteympäristön kuvaus sekä algoritmin soveltaminen - ks. kuva (Kuva 5). On tärkeä huomata, että alkuperäinen kokonpano ei sisällä sellaista tietoa, mitä konfiguroija tarvitsee tehtävän suorittamiseksi, joten tämä on poistettu kuvasta. Ympäristön kuvaus sisältää käytännössä tarvittavat tiedot (objektit sekä objektien väliset liitokset) tavoitekokoonpanon lisäksi.



Kuva 5. Järjestelmän automatisoitu konfigurointi: malli 2

Kuvassa (Kuva 5) tavoiteympäristön kuvaus on tehty esimerkin omaisesti hyvin geneerisesti käyttämällä UML:ää. Käytännössä tavoiteympäristön mallinnus tehdään soveltamalla tarkoitukseen soveltuvan paradigman mukaista menetelmää, mutta huomioitavaa on, että tietosisältö on sama kuin kuvassa (Kuva 5) on esitetty. Tämä tarkoittaa vaatimusta myös konfiguroijalle eli myös konfiguroijan on ymmärrettävä valittua mallinnusmenetelmää. Todellisuudessa konfiguroija toteutetaan aina tapauskohtaisesti ymmärtämään vain tarkoitukseen valittua mallinnusmenetelmää (Sabin & Weigel 1998, 43).

Aiemmin luvussa 3.2.1 esitetyssä kuvassa (Kuva 2) esitimme, kuinka käyttäjän päättelyn tuloksena hän saa tarvittavat työvaiheet konfiguroinnin suorittamiseksi. Vastaavalla tavalla konfiguroija tekee päätelmiä perustuen tavoiteympäristön kuvaukseen sekä annettuihin vaatimuksiin saaden näin selville tarvittavat työvaiheet.

Alla on listattuna tunnettuja paradigmoja, joita voidaan käyttää konfiguraatioiden kuvauksiin, ja joihin perustuen konfiguroija voi tehdä päätökset työvaiheista (Sabin & Weigel 1998, 43 - 47).

- Sääntöpohjainen päättely (rule-based reasoning)
- Mallipohjainen päättely (model-based reasoning)
- Ennakkotapauksiin pohjautuva päättely (case-based reasoning)

On huomioitavaa, että paradigmojen soveltuvuus annettuun ongelmaan voi vaihdella ongelmasta riippuen (Sabin & Weigel 1998, 48).

3.2.3 Sääntöpohjainen päättely

Sääntöpohjaiseen päättelyyn (eng. rule-based reasoning) perustuvia järjestelmiä kutsutaan nimellä ”expert systems”. Nämä järjestelmät esittävät sääntöjen avulla ympäristön kuvauksen sekä säännöt, joiden mukaan kyseisen applikaation tapauksessa konfigurointi tulisi suorittaa.

Sääntöpohjaisessa konfiguraationhallinnassa määritellään sääntökuvaus. Sääntökuvaus kertoo konfiguroitavan järjestelmän päämäärän - ei niitä toimenpiteitä, joita tulee suorittaa päämäärän saavuttamiseksi (Burgess & Frisch 2007, 5).

Sääntöpohjaista päättelyä käytetään tekoälyä (eng. AI - artificial intelligence) käyttävissä järjestelmissä päättelyn apukeinona. Tarve sääntöpohjaisen päättelyn käyttöön järjestelmien konfiguroinnissa syntyi proceduraalisten ohjelmointikielien konfigurointitehtävissä ilmenneistä puutteista.

Seuraavassa on esitetty ohjelmakoodi sääntöpohjaisen ohjelman osasta.

```
class apache {

   case $operatingsystem {
      CentOS: {
            $service = "httpd"
            $packagelist = ["$service.$architecture"]
            $destination_file = "/etc/httpd/conf/httpd.conf"
            $source_file = "puppet:///apache/httpd.conf"
            }
     Ubuntu: {
            $service = "apache2"
            $packagelist = ["$service"]
            $destination_file = "/etc/apache2/apache2.conf"
            $source_file = "puppet:///apache/apache2.conf"
            }
    }

    #### The sw-packages installation ###
    package { $packagelist:
               ensure => "installed"
    }

    #### Putting the apache config-file in place ###
    apache::apache_files {

      $destination_file:
            source => $source_file
    }

    #### Setting up the service ###
    service { $service:
           enable     => "true",
           ensure     => "running",
           hasrestart => "true",
           hasstatus  => "true",
           require    => Package["$packagelist"]
    }
}
```

Edellä esitetty ohjelmakoodi on Puppet Labs'in Puppet-ohjelmasta (syntaksi perustuu versioon 0.25.4), jonka avulla voidaan asentaa Apache web -palvelimen ohjelmistopaketit CentOS- sekä Ubuntu Linux-käyttöjärjestelmille. Ohjelmakoodi on hyvin yleisellä tasolla, minkä vuoksi ohjelman kirjoittajan ei tarvitse huolehtia, kuinka varsinainen asennus sekä konfigurointitoimenpiteet tulisi suorittaa.

Sääntöpohjaisen päättelyn rajoitteena voi olla tavoiteympäristön kuvauksen sekä sääntöjen liian tiukat sidokset. Näiden eriyttäminen voi olla monessa sääntöpohjaisessa toteutuksessa vaikeata. Tästä voi aiheutua tavoiteympäristön kuvauksen osalta suuria ylläpidollisia ongelmia. Tätä ongelmaa pyrittiin ratkaisemaan 1980-luvulla kehittämällä tekoälyn uusi ala-tutkimusalue; mallipohjainen päättely (eng. model-based reasoning). (Sabin & Weigel 1998, 44.)

3.2.4 Mallipohjainen päättely

Mallipohjaisen päättelyn (eng. model-based reasoning) päällimmäisenä olettamuksena on ns. järjestelmän malli (eng. system's model). Järjestelmän malli koostuu osakokonaisuuksista sekä näiden välisten elementtien välisistä yhteyksistä.

Konfiguroinnin toteuttamiseksi mallipohjainen päättely tarjoaa useita lähestymistapoja, joista varteenotettavimmat ovat luetteloituna alla.

- Logiikkaan perustuva lähestymistapa.
- Applikaation resursseihin pohjautuva lähestymistapa.
- Applikaation komponenttien rajoitteisiin perustuva lähestymistapa.

Seuraavassa on esitettynä esimerkki CLASSIC-järjestelmässä käytetystä ohjelmointikielestä, jonka avulla voidaan kuvata mallinnettavaa järjestelmää kuvauslogiikan keinoin (eng. description logic). Kuvauslogiikassa käytetään kuvauksien luomiseen muodostussanoja kuten, AND, OR, AT-LEAST sekä ALL.

```
EMPLOYEE ~>
(AND (AT-LEAST 1 HRID-number)
     (ALL HRID-number 7-DIGIT-INTEGER))
```

Ohjelmakoodissa on määritellään, että työntekijällä tulee olla vähintään yksi HRID numero, joka on seitsennumeroinen kokonaisluku. (Brachman & Borgida & McGuinness & Patel-Scheider 2009, 6.)

Kuvauslogiikassa on kuitenkin yksi merkittävä puute, joka liittyy tapahtuvan päättelyn tehokkuuden vs. kuvauksen ilmaisuvoimaisuuden välille. Mikäli ilmaisuvoimaisuutta päätetään vähentää käsiteltävyyden helpottamiseksi, ei käytetyn formalismin avulla kyetä enää kuvaamaan monimutkaisia järjestelmiä, mikä puolestaan on usein välttämätöntä käytännön konfigurointitehtävien kuvauksissa. (Sabin & Weigel 1998, 45.)

3.2.5 Tapauskohtainen päättely

Tapauskohtainen päättely (eng. case-based reasoning) lähtökohta on erilainen verrattuna muihin päättelyyn perustuviin menetelmiin. Tapauskohtainen päättely perustuu oletukseen, jonka mukaan samantyyppisillä ongelmilla on monesti lähes identtiset ratkaisut. Tapauskohtaisessa päättelyssä pyritään ratkaisemaan käsillä oleva konfigurointiongelma löytämällä samantyyppinen aiemmin ratkaistu ongelma, jonka ratkaisua pyritään soveltamaan tai korkeintaan tekemään ratkaisumalliin pieniä muutoksia. (Sabin & Weigel 1998, 47.)

Tapauskohtainen päättely soveltuu käytettäväksi tapauksissa, joissa vaihtoehtoisia konfiguroitavia komponentteja on rajoitetusti.

4 KOKOONPANON HALLINTA IT-ALALLA

Kokoonpanon hallinta IT-alalla voi käsittää pieniä manuaalisia ylläpitotoimenpiteitä tai vastaavasti laajoja yrityksessä käytettävän muutoksenhallinnan mukaisia automatisoituja järjestelmien käyttöönottoja ja päivityksiä. Kokoonpanon hallinnan automatisointi on vielä suhteellisen uutta IT-alalla, mutta siitä huolimatta kokoonpanon hallinnan automatisoimiseksi on saatavilla useita avoimen lähdekoodin ohjelmistoja.

4.1 Kokoonpanon hallinnan historiaa IT-alalla

Digital Equipment Corporation tarvitsi DEC VAX- tietokonejärjestelmän myynnin tueksi keinoja, joiden avulla tietokonejärjestelmän komponenttien tilaus voitaisiin määritellä asiakkaan vaatimusten mukaisesti. Ongelmaa yritettiin ensin ratkaista proceduraalisin keinoin käyttämällä fortran- sekä basic-ohjelmointikieliä. John McDermott Carnegie Mellonin yliopistosta kehitti vuonna 1978 OPS5-ohjelmointikielellä toteutetun sääntöpohjaisen konfigurointijärjestelmän nimeltä R1. Tätä konfigurointijärjestelmää kutsuttiin myös myöhemmin nimellä XCON (e**X**pert **CON**figurer). R1/XCON-järjestelmää pidetään ensimmäisenä merkittävänä konfigurointijärjestelmä. Alla on esitettynä OPS5-ohjelmointikielellä toteutettu ohjelman osa.

```
## http://99-bottles-of-beer.net/language-ops5-2208.html ##
(literalise Count bottles)
(literalise SecondLine)

(startup
    (make Count ^bottles 99)
)

# RULE 1
(p moreBeer
   (Count ^bottles {<beerLeft> > 0}) # Conditional part
  -(SecondLine)
-->
   (writeln <beerLeft>| bottle of beer on the wall, |
<beerLeft> | bottle of beer.|)
   (modify 1 ^bottles (compute <beerLeft>-1)) # Cone-
quence/action part
   (make SecondLine)
)

# RULE 2
(p moreBeerSecondLine
   (Count ^bottles > 1 )# Conditional part
   (SecondLine)
-->
   (writeln |Take one down and pass it around, | <beerLeft>
| bottles of beer on the wall.|)
   (remove 2)
)

...jne...

############################################################
```

AT&T Bellin laboratorioilla vuonna 1990 kehitettiin mallipohjaiseen päättelyyn perustuva konfigurointisovellus käyttämällä 1980-luvun lopulla kehitettyä CLASSIC-järjestelmää, joka puolestaan perustuu kuvauslogiikkaan.

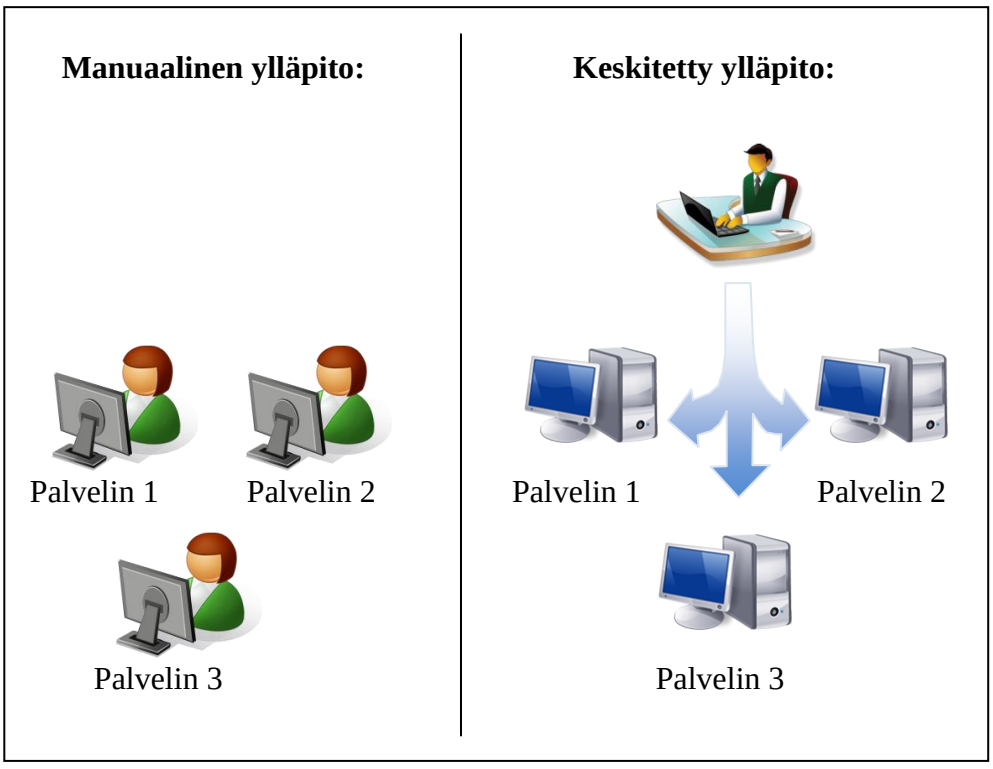
Vuonna 1993 kehitti Oslon yliopiston fysiikan laitoksella työskentelevä tohtori Mark Burgess verkotettujen UNIX-tietokoneiden konfigurointityökalun nimeltä ”Configuration engine 2.0”. Työkalu on sääntöpohjainen, ja se käyttää konfigurointiin suunniteltua kuvaavaa kieltä.

Kokoonpanon hallinta on kehittynyt 1978 vuodesta lähtien merkittävästi. Silti edelleen kokoonpanon hallinta perustuu luvussa 3.2.3, 3.2.4 ja 3.2.5 esitettyihin tunnettuihin paradigmoihin.

4.2 Keskitetty kokoonpanon hallinta IT-alalla

Tämä opinnäytetyö keskittyy tutkimaan soveltuvia menetelmiä, joiden avulla voidaan helpottaa palvelinympäristöjen ylläpitoa. Tavoitteena on löytää soveltuva keskitetty kokoonpanon hallinnan ratkaisu, joka täyttäisi palvelinympäristön ylläpidolle asetetut vaatimukset.

Keskitetyn kokoonpanon hallinnan tavoitteena on keskittää sekä automatisoida palvelinympäristöjen ylläpitotoimenpiteitä. Keskitetty kokoonpanon hallinta vähentää ylläpitoon vaadittavaa työmäärää tehostamalla työtä. Alla olevassa kuvassa (Kuva 6) on esitettynä usean palvelimen ylläpito manuaalisesti vs. keskitetysti.



Kuva 6. Manuaalinen vs. keskitetty ylläpito

Täsmälleen saman toimenpiteen ollessa kyseessä voidaan siis ylläpitotoimenpide suorittaa kuvan (Kuva 6) tapauksessa n. 66,6 % vähemmällä työmäärällä alla olevan kaavan (Kaava 1) mukaisesti.

$$\left(1 - \frac{1}{x}\right) \times 100\% \quad (1)$$

Kaava 1. Keskitetyn ylläpidon vähentämä työmäärä

Lisäksi keskitetyn ylläpidon avulla voidaan vähentää huomattavasti manuaalisesti tehtävän ylläpidon yhteydessä tapahtuvien virheiden määrää. Esimerkkinä lyhyen huoltokatkoksen aikana tehtävä konfigurointi usealle palvelimelle voi aiheuttaa työntekijälle kiireettä, jonka seurauksena työntekijä joutuu työskentelemään paineen alaisena, mikä puolestaan altistaa virheille (Holmen 2010, 13). Toteuttamalla ylläpitotoimenpiteet keskitetysti, työntekijälle jää enemmän aikaa tehtävän konfiguroinnin verifioimiseksi ja näin myös työskentelijän tekemien konfigurointivirheiden määrää voidaan huomattavasti vähentää.

4.3 Kokoonpanon hallinta, CMDB ja ITIL

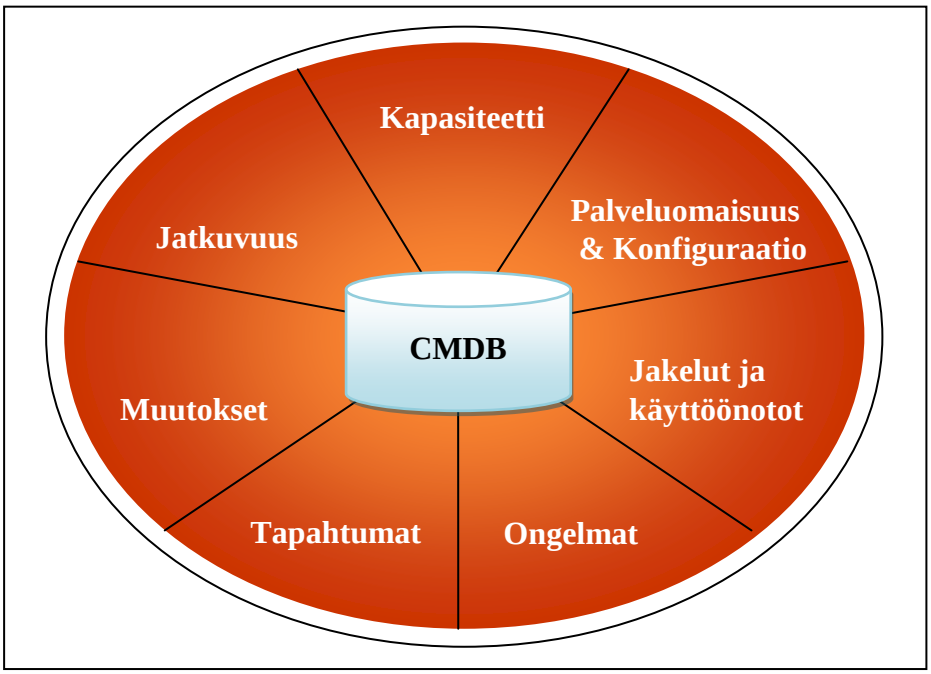
Kokoonpanon hallintaan usein liitetään lyhenne CMDB (eng. configuration management database). CMDB on tietokanta, jonne tallennetaan tietoja keskitetyn kokoonpanon hallintajärjestelmän hallitsemista osista eli CI:stä (eng. configuration item) sekä vastaavista komponenteista eli palveluomaisuudesta (eng. asset).

Palveluomaisuudella on rahallista arvoa, eli assetti voi olla esimerkiksi tietokoneen kovalevy tai ohjelmistolisenssi. CI:llä ei puolestaan ole rahallista arvoa eikä sillä ole elinkaarta. Sen sijaan CI on sidoksissa tietojärjestelmässä käytettyyn konfiguraatioon. CMDB:ssä on erityyppisiä CI:tä, joissa jokaisella on määriteltynä ominaisuuksia eli attribuutteja. Attribuutteina voi olla tieto CI:n tilasta, sijainnista, liitoksista muihin CI:hin, suunnitelluista muutoksista, vastuuhenkilöstä, avoimista ongelmista jne.

CMDB:n avulla tietojärjestelmien muutostenhallinta voidaan toteuttaa suunnitelmallisemmin, sillä CI:n välisten liitosten dokumentoinnin avulla voidaan vähentää ja ennakoida tietojärjestelmään tehtäviä muutoksia. CMDB:n avulla voidaan paremmin hallita tietojärjestelmissä olevia riskejä priorisoimalla esimerkiksi tarvittavien tietoturvapäivitysten asennuksia. CMDB:n avulla voidaan pitää kirjaa CI:hin tehdyistä muutoksista sekä käytössä olevista ohjelmista.

CMDB liittyy vahvasti ITIL:iin ollen yksi sen pääkomponenteista. ITIL on yksi IT-alan referenssi malleista (eng. IT-reference models), jonka omistaa Ison-Britannian hallinnon toimielin nimeltä ”Office of Government Commerce” eli OGC.

ITIL on IT-paveluiden hallintaan ja johtamiseen koottu kokoelma parhaita käytäntöjä. Kuvassa (Kuva 7) on esitettynä osa CMDB:hen liittyvistä ITIL:in prosesseista.



Kuva 7. ITIL prosessit sekä CMDB (OGC www.itil-officialsite.com)

4.4 Soveltuvia avoimen lähdekoodin kokoonpanon hallinnan ohjelmistoja

Tässä opinnäytetyössä on tarkoituksena vertailla avoimeen lähdekoodiin perustuvia keskitetyn kokoonpanon hallinnan ohjelmistoja. Ohjelmien avulla tutkitaan palvelimien (esimerkiksi web-, tietokanta-, tiedosto- ja sähköpostipalvelimien) keskitettyä kokoonpanon hallintaa.

Tiedossa olevia avoimen lähdekoodin kokoonpanon hallinnan ohjelmistoja tämän opinnäytetyön dokumentoinnin aikana (huhtikuussa 2010) oli seitsemäntoista kappaletta. Keskityttäessä kokoonpanon hallinnan ohjelmistoihin, joita edelleen aktiivisesti kehitetään, voidaan vaihtoehtoisten ohjelmien määrä vähentää taulukossa (Taulukko 1) listattuihin vaihtoehtoihin.

Taulukko 1. Aktiivisesti kehitettävät avoimen lähdekoodin kokoonpanon hallinnan ohjelmistot

Ohjelma:	**Viimeisin julkaistu versio:**
Bcfg2	2010-01-21, versio 1.0.1
Cfengine 3	2010-03-05, versio 3.0.4
Chef	2010-03-04, versio 0.8.6
Puppet	2010-01-12, versio 0.25.3

Edelleen tutkittaessa sovellusten käytön laajuutta, käyttäjä yhteisöjen aktiivisuutta, tukea eri käyttöjärjestelmille sekä käytetyn ohjelmointikielen

soveltuvuutta kokoonpanon hallintaan, jää jäljelle vain kaksi soveltuvaa vaihtoehtoa; Cfengine 3 sekä Puppet.

4.4.1 Cfengine 3

Cfengine 3 (community edition) on avoimen lähdekoodin ohjelmisto, jonka avulla voidaan automatisoida verkossa olevien tietokoneiden ylläpitoa ja konfiguraatioita client-server mallin mukaisesti. Cfengine on pitkälti tekijänsä Mark Burgessin sekä hänen omistamansa yrityksensä ”Cfengine AS/Inc” käsialaa. Mark Burgess (tohtori, Ph.D teoreettinen fysiikka) toimii professorina Oslon yliopistossa alueenaan järjestelmän ylläpito (network and system administration).

Mark Burgess toteutti ensimmäisen version Cfengine ohjelmasta vuonna 1993 nimeltä ”Configuration engine 2.0”. Mark on jatkanut Cfenginen kehitystä määrätietoisesti. Cfengine versio 2 julkaistiin vuonna 2002. Mark aloitti vuonna 2003 tutkimustyön täydentääkseen Cfenginen toiminnallista mallia, jonka hän sai valmiiksi vuonna 2008 määriteltyään ns. ”lupaus teorian” (eng. promise theory). Tämä malli toteutettiin Cfengineen sen seuraavan ison muutostyön yhteydessä. Vuonna 2009, viiden vuoden tutkimuksen jälkeen, julkaistiin viimeisin suurin muutos (Cfengine 3), jonka yhteydessä ohjelman käyttämä sääntöpohjainen (rule-based) ohjelmointikieli myös uudistettiin (http://cfengine.com/pages/history).

”Cfengine 3 on geneerinen toteutus lupauksia sisältävästä ohjelmointikielestä, mikä mahdollistaa konfiguroinnin sekä muutosten eri osa-alueiden toteuttamisen saman katon alla” (Burgess, M. 2009, 19).

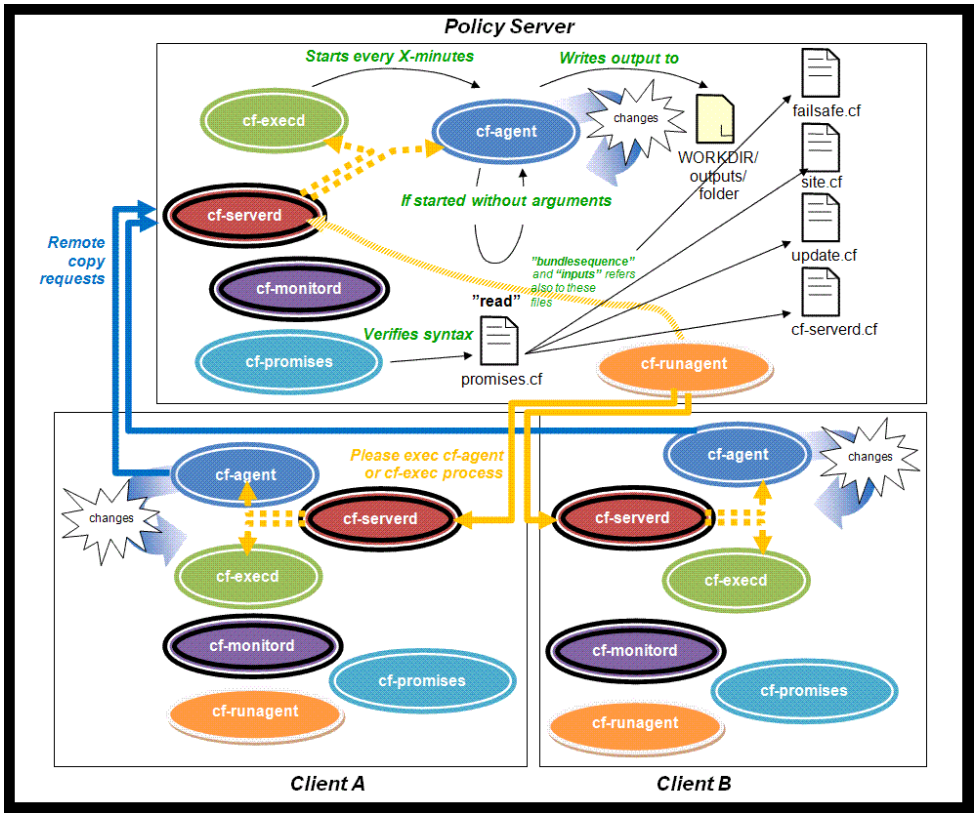
Cfenginen kuvaavan ohjelmointikielen (eng. declarative language) avulla määritellään konfiguroitavan järjestelmän päämäärä, ilman tarvetta määritellä erikseen toimenpiteitä tuon päämäärän saavuttamiseksi. Juuri tässä on ero perinteisiin proceduraalisiin tai käskeviin ohjelmointikieliin verrattuna, kuten esimerkiksi shell- sekä Perl-skriptit, jotka vaativat jok’ikisen vaiheen määrittelyn. (Burgess & Frisch 2007, 5.)

Cfenginen avulla voidaan suorittaa monia tietokoneiden järjestelmän kokoonpanon hallintaan sekä ylläpitoon liittyviä toimenpiteitä. Näitä ominaisuuksia on tarkemmin luetteloituna luvussa 6.2.

Cfengine 3:n ydin on lisensoitu GPLv3:n mukaisesti. Cfengine on toteutettu C-kielellä, joka puolestaan vähentää tarvetta asentaa järjestelmään tulkkaavien ohjelmointikielien komponentteja. Cfengine 3 vaatii toimiakseen yleensä alla luetellut komponentit.

- openssl (OpenSSL component)
- libdb4.6 (Berkeley v4.6 Database Libraries [runtime])
- libdb-dev (Berkeley Database Libraries [development])
- libpcre3 (Perl 5 Compatible Regular Expression Library)

Cfengine toimii client-server mallin mukaisesti. Cfengine 3:n client sekä server ovat itse asiassa sama asennuspaketti. Clientin ja serverin osuudet määritellään käyttöön sovellusta konfiguroitaessa. Alla kuvassa (Kuva 8) on esitettynä client sekä server osuuksien prosessit sekä niiden välinen kommunikointi.



Kuva 8. Cfengine 3:n prosessit

4.4.2 Puppet

Puppet on Ruby-ohjelmointikieleen perustuva avoimen lähdekoodin ohjelmisto, jonka avulla voidaan automatisoida Unix- sekä Linux-pohjaisten tietokoneiden ylläpitotoimenpiteitä. Puppet noudattaa client-server mallia sekä on julkaistu GPLv2:n lisenssin alaisuudessa.

Puppetin alkuperäinen kehittäjä on Luke Kanies. Kanies on tehnyt työtä Unixin sekä järjestelmien ylläpidon alueella vuodesta 1997 lähtien, joihin kokemuksiin nojautuen Puppettia on lähdetty kehittämään. Luke ei ollut tyytyväinen olemassa oleviin keskitettyihin kokoonpanon hallinnan ohjelmistoihin, ja hän alkoi Puppetin kehitystyön vuonna 2001. Vuonna 2005 hän perusti yrityksen nimeltä Reductive Labs. Reductive Labs keskittyy avoimeen lähdekoodin perustuvien automatisointiohjelmistojen kehitykseen. Pian yrityksen perustamisen jälkeen julkaistiin Puppetin ensimmäinen versio.

Puppet käyttää kuvaavaa ohjelmointikieltä (eng. declarative language), jonka avulla voidaan kuvata tarvittava konfiguraatio riippumatta

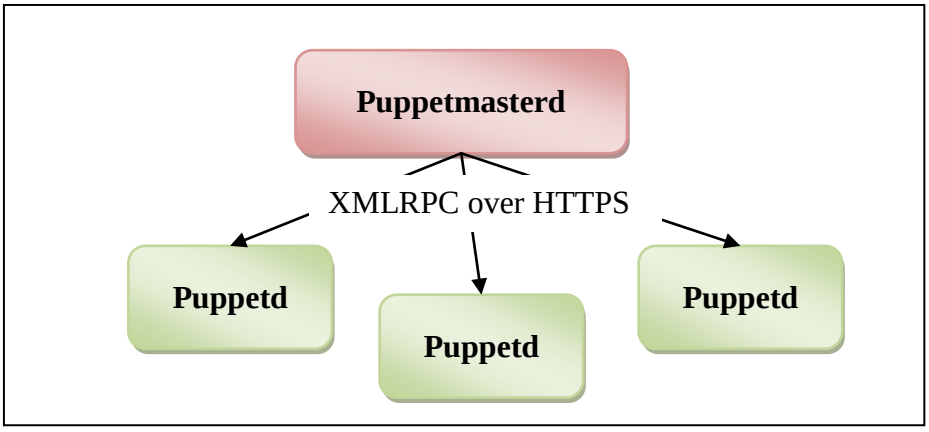
käytetystä käyttöjärjestelmästä (Turnbull 2007, 41). Käytetty ohjelmointikieli on sääntöpohjainen (rule-based).

Puppetin avulla voidaan suorittaa monipuolisesti erilaisia järjestelmän ylläpitoon liittyviä toimenpiteitä. Näistä on enemmän kerrottuna luvussa 6.2.

Koska Puppet on toteutettu Ruby-ohjelmointikielellä, vaatii tämä toimiakseen tarvittavat Ruby-ohjelman komponentit. Puppetin yleisimmin tarvittavat komponentit ovat lueteltuna alla.

- openssl (OpenSSL component)
- ruby - An interpreter of object-oriented scripting
- libopenssl-ruby - OpenSSL interface for Ruby
- libxmlrpc-ruby - transitional dummy package
- rdoc - Generate documentation from ruby source file
- facter - a library for retrieving facts from operating systems
- lsb-release - Linux Standard Base version reporting utility

Puppet toimii client-server mallin mukaisesti ja kummallekin on oma asennuspakettinsa. Server-komponentti tuo palvelimelle ”**puppetmasterd**” palvelun ja sitä vastaavan prosessin. Client-komponentti koostuu ”**puppetd**” palvelusta ja vastaavasta prosessista. Alla kuvassa (Kuva 9) on esitettynä Puppetin client-server -malli.



Kuva 9. Puppetin client-server -malli

5 KOHDEYMPÄRISTÖ

Ixonos tarjoaa asiakkaillensa web-, tietokanta, tiedosto, sähköposti, jne. palvelimien ylläpito- sekä hosting-palveluita. Ixonoksen hosting-palveluiden ylläpidosta vastaa ryhmä asiantuntijoita. Tämä opinnäytetyö tehdään tämän ryhmän toimeksiannosta, tavoitteena löytää soveltuva avoimen lähdekoodin ohjelmisto keskitetyn kokoonpanon hallinnan toteuttamiseksi.

5.1 Ixonos hosting-center

Ixonoksen hosting-centerissä ylläpidettävien palvelimien käyttöjärjestelminä on useita Linux (Red Hat Enterprise Linux, CentOS, Ubuntu, Debian) sekä Windows (Windows 2003 Server, Windows 2003 Server 64-bit) jakeluita. Osa palveluista on myös virtualisoitu.

Palvelimien määrän noustessa myös toistuvien ylläpitotoimenpiteiden määrä kasvaa. Osan haasteesta tekee ympäristöjen heterogeenisuus (eri käyttöjärjestelmä, eri varusohjelmistot, eri käyttäjätilit, eri ryhmät, eri tavalla jaetut oikeudet, jne.). Lähtökohtana opinnäytetyön toimeksiannolle onkin kokoonpanon hallinnan automatisointi sekä mahdollisuus hallita erilaisia kokoonpanoja.

5.2 Kokoonpanon hallinnalle asetetut vaatimukset

Ixonoksen hosting-palveluiden ylläpidosta vastaavan ryhmän kesken pidetyssä ensimmäisessä palaverissa (18.12.2009) käydyn keskustelun pohjalta sovittiin alustavasti keskitetyn kokoonpanon hallinnan ohjelmistolle asetettavista vaatimuksista.

Ohjelmiston avulla tulisi kyetä ryhmittelemään ylläpidettäviä palvelimia tarkoituksenmukaisiin ryhmiin. Toteutettavia konfigurointitoimenpiteitä tulisi kyetä kohdistamaan tehtyihin ryhmityksiin. Ryhmäjako voi perustua käyttöjärjestelmään, asiakkuuteen, palvelimen rooliin (esimerkiksi tuotanto tai testipalvelin) tai omavalintaiseen ryhmittelyyn.

Ohjelmiston avulla on kyettävä suorittamaan palvelimien konfiguraatiotiedostojen päivityksiä, ohjelmistopakettien asennusta, komentorivikomentojen ajamista, asentamaan Windows-palvelimille varmistusohjelmistoja sekä päivittämään Windows-palvelimien varmistusohjelmistoja.

Palvelimille suoritettavissa asennustoiminnallisuuksissa tulisi kyetä määrittelemään erikseen ennen asennusta suoritettavat toimenpiteet sekä asennuksen jälkeen suoritettavat toimenpiteet.

Ohjelmistolle on kyettävä määrittelemään myös konfigurointitoimenpiteet, joita ennen sekä jälkeen on suoritettava ennalta määritettyjä muita konfigurointitoimenpiteitä (esimerkiksi komentorivi komentojen suoritus).

Ixonoksen hosting-palveluiden ylläpidosta vastaavan ryhmän kesken pidetyssä toisessa palaverissa (15.02.2010) päädyttiin kirjaamaan ohjelmistolle asetettavat vaatimukset yleisimpinä testitapauskohtaisina konfiguraatiotoimenpidevaatimuksina. Tämä helpotti huomattavasti testaamisen suorittamista ilman tarvetta sulkea pois aiempia vaatimuksia. Testitapaukset ovat esitettynä luvuissa 5.2.1 – 5.2.3. Kaikki testit ovat Linux-käyttöjärjestelmässä suoritettavia testejä. Lisäksi luvussa 5.2.2 esitetyssä varmistusohjelmistojen testissä esitelty testi suoritetaan myös Windows-ympäristössä.

5.2.1 Tiedostotason testit

Tiedostotason testit jakautuvat konfiguraatiotiedostojen hallinnan lisäksi yleisiin testeihin, joihin kuulu tiedostojen, hakemistojen sekä linkkien hallinta. Konfiguraatiotiedostojen hallinta on käytännössä tekstitiedostojen muokkaamista annettujen ehtojen mukaisesti. Linux palvelimien ylläpidossa on tärkeätä kyetä keskitetysti muokkaamaan hallittavien palvelimien konfiguraatiotiedostoja.

Konfiguraatiotiedostojen konfigurointitestit koostuvat määritellyn konfiguraatio-osan etsimisestä-, lisäämisestä-, korvaamisesta- sekä poistamisesta konfiguraatiotiedostosta.

Yleinen tietostotasolla toteutettava hallinta koostuu:

- tiedoston sekä hakemiston omistajuuden sekä oikeuksien tarkistuksesta ja muutoksesta tarvittaessa
- tiedoston kopioinnista paikallisesti
- tiedoston kopioinnista palvelimelta toiselle
- linkin olemassaolon tarkistuksesta ja luonnista tarvittaessa
- tiedoston tuhoamisesta sekä luonnista (touch)
- määriteltyjen tiedostojen pakkauksesta
- tiedostojen muutosten seurannasta sekä hälytyksestä (trip-wire).

5.2.2 Kokonaisuuksien hallinnan testit

Kokonaisuuksien hallinnan testeissä keskitetyn kokoonpanon hallinnan ohjelmiston avulla pyritään toteuttamaan yksittäisen kokonaisuuden hallinta alusta loppuun. Hallittavat kokonaisuudet ovat Apache web –palvelin, Nagios-client sovellus, paketinhallinta NFS-client sovelluksen näkökulmasta sekä varmistusohjelmiston asennus.

Apache web -palvelimen testin yhteydessä testataan Apache web -palvelimen asennusta, konfigurointia, konfiguroinnin verifiointia sekä Apache web -palvelun käynnistämistä. Apache web –palvelimen käyttö Ixonoksen hosting-centerissä on hyvin runsasta, joten tämän osakokonaisuuden testaaminen on tärkeässä osassa.

Nagios sovelluksen avulla voidaan hallittavien koneiden resursseja (prosessit, palvelut, levytila, jne.) monitoroida keskitetysti. Tämän testin

tarkoituksena on muokata Nagios-client sovelluksen konfiguraatio palvelimella määritellyn mukaiseksi.

Nagios-client sovelluksen hallinnassa testataan:

- nagios-clientin käyttöönottoa ja perusasetusten alustamista
- palvelimen verkkokortin monitoroinnin lisäämistä
- muutetun Nagios-clientin konfiguraation tarkistamista
- palvelimen kuorman sekä omistamattomien prosessien (zombie) tarkistuksen poistamista.

Paketinhallinnan testaus on toteutettu testaamalla käyttöjärjestelmäkohtaisten NFS-tiedostojärjestelmän levyjakoon vaadittavien pakettien (NFS-client) asennusta levyjaon konfiguroinnin yhteydessä. NFS-client pakettien asennuksella saadaan testattua hyvin eri käyttöjärjestelmien pakettienhallintaa keskitetyn kokoonpanon hallinnan ohjelmiston avulla. Paketinhallinnan testin yhteydessä testataan lisäksi levyjaon konfigurointia (/etc/fstab), käyttöönottoa sekä poistamista.

Varmistusohjelmistojen avulla hallittavien palvelimien varmuuskopioinnit voidaan suorittaa päivittäin varmistussuunnitelmien mukaisesti. Tämän testin tarkoituksena on puolestaan varmistaa, että varmistusohjelmistojen asennus sekä päivitys ovat mahdollisia sekä Linux- että Windows-käyttöjärjestelmissä.

5.2.3 Muut testit

Palvelimien ylläpidon muiden osa-alueiden testit koskettavat käyttäjähallintaa, palomuurisääntöjen konfigurointia, komentorivikomentojen suorittamista sekä ehdollisia testejä.

Käyttäjähallinnan testissä hallitaan käyttöjärjestelmätason käyttäjätilejä, käyttäjien ryhmiä sekä käyttäjälle määriteltäviä lisäoikeuksia (/etc/sudoers). Käyttäjähallinnan yhteydessä on kyettävä tarkistamaan käyttäjätilien, ryhmien sekä sudo-oikeuksien olemassaolo sekä lisäys tarvittaessa. Uusien käyttäjien, ryhmien sekä sudo-oikeuksien lisääminen tulisi olla mahdollista. Vastaavasti käyttäjien, ryhmien ja sudo-oikeuksien muokkaus sekä poistaminen tulisi olla mahdollista.

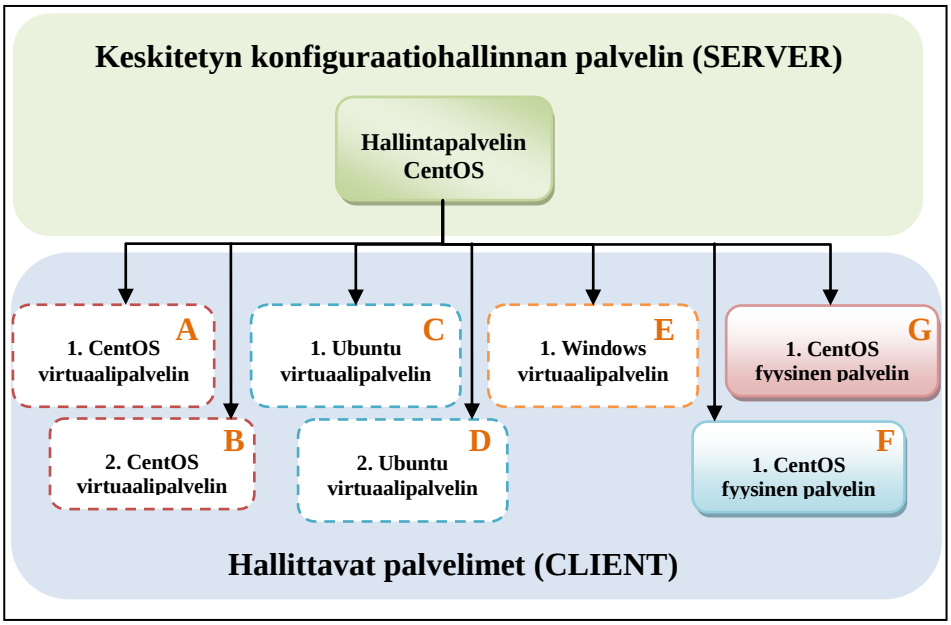
Palomuurisääntöjen hallinta on käytännössä olemassa olevien palomuurisääntöjen muokkaamista annettujen ehtojen mukaisesti. Palomuurisääntöjen hallinnan yhteydessä on voitava toteuttaa palomuurisääntöjen tarkistaminen, lisääminen, muokkaus, poisto sekä palomuurisääntöjen käyttöönotto.

Komentorivikomentojen testauksessa testataan komentorivikomentojen suoritusta, komentojen avulla tehtävää tiedon haun mahdollisuutta sekä komentorivikomentojen virhetilanteiden havaitsemista. Lisäksi aiemman komentorivikomennon avulla haetun tiedon käyttöä toisessa komentorivikomennossa testataan.

Ehdollisen testin tarkoituksena on testata mahdollisuutta ryhmitellä palvelimia tai konfiguraatiotapahtumia tiettyjen ehtojen mukaisesti. Palvelimien määrän kasvaessa, ryhmittelyn avulla voidaan keskitetty kokoonpanon hallinta toteuttaa edelleen hallitusti. Ryhmittely voi perustua omavalintaiseen malliin, palvelimen tyypin (virtuaalinen tai fyysinen) mukaiseen ryhmittelyyn tai palvelimen hostname-tietoon. Ehdollisissa testeissä testataan myös konfiguraatiotoimenpiteiden toteuttamista ajan (kellonaika, viikonpäivä, päiväys, vuosi) perusteella.

5.3 Testiympäristö

Luvuissa 5.2.1 - 5.2.3 esitettyjen testien suorittamiseen sovittiin käytettäväksi luvuissa 0 sekä 5.3.2 esitettyjä palvelinkokoonpanoja. Kuvassa (Kuva 10) on havainnollistettuna testikokoonpano.



Kuva 10. Kuvaus suunnitellusta testikokoonpanosta

Kuvassa (Kuva 10) katkoviivalla esitetyt palvelimet ovat virtuaalipalvelimia. Yhtenäisellä viivalla esitetyt palvelimet ovat fyysisiä palvelimia.

Testiympäristöiden vaatimukset verkkoyhteyksille koskivat ainoastaan keskitetyn kokoonpanon hallinnan ohjelmistojen client- ja server-osuuksien välistä kommunikointia määriteltyihin TCP-portteihin. Puppetin server-osuus palvelee portissa 8140 ja client-osuudet portissa 8139. Cfengine 3:n client- ja server-osuudet kuuntelevat molemmat TCP-porttia 5308.

5.3.1 Keskitetyn kokoonpanon hallintapalvelin

Testattavat keskitetyn kokoonpanon hallintapalvelimet noudattavat client-server -mallia, jossa hallintapalvelimelle (server-osuus) esitettyjä kokoonpanoja voidaan välittää keskitetysti hallituille (client-osuus)

palvelimille. Alla olevassa taulukossa (Taulukko 2) on esitettynä testausta varten rakennetulle hallintapalvelimelle asetetut vaatimukset.

Taulukko 2. Hallintapalvelimen ympäristön vaatimukset

Vaatimus:	**Määritelmä:**
Palvelimen tyyppi	Virtuaalipalvelin
Käyttöjärjestelmä	64-bittinen CentOS
Muistin määrä	512 MB
Kovalevyn määrä	5 GB
Esiasennetut komponentit	Olemassa olevan 64-bittisen CentOS asennuspaketin tuomat komponentit

Hallintapalvelimen ohjelmistoversiot ovat listattuna liitteessä 1.

5.3.2 Hallittavat palvelimet

Hallittavien palvelimien käyttöjärjestelmiksi sovittiin käytettäväksi CentOS- sekä Ubuntu-pohjaisia Linux käyttöjärjestelmiä. CentOS on Red Hat Enterprise Linux (RHEL) –käyttöjärjestelmään perustuva avoimen lähdekoodin käyttöjärjestelmä. CentOS-käyttöjärjestelmän vahvuutena onkin vankka Red Hat tausta. Myös Ubuntu on vapaasti jaossa oleva käyttöjärjestelmä, joka on vakiinnuttanut paikkansa suosituimpana vapaasti jaossa olevana Linux-käyttöjärjestelmänä. (DistroWatch,com.)

Alla olevassa taulukossa (Taulukko 3) on esitettynä testausta varten rakennetuille hallittaville CentOS-palvelimille asetetut vaatimukset.

Taulukko 3. CentOS-pohjaiset palvelimet

Vaatimus:	**Määritelmä:**
Palvelimen tyyppi	Virtuaalipalvelin
Käyttöjärjestelmä	64-bittinen CentOS
Muistin määrä	512 MB
Kovalevyn määrä	5 GB
Esiasennetut komponentit	Olemassa olevan 64-bittisen CentOS asennuspaketin tuomat komponentit

CentOS-pohjaisten palvelimien ohjelmistoversiot ovat listattuna liitteessä 2.

Alla olevassa taulukossa (Taulukko 4) on esitettynä testausta varten rakennetuille hallittaville Ubuntu-palvelimille asetetut vaatimukset.

Taulukko 4. Ubuntu-pohjaiset palvelimet

Vaatimus:	**Määritelmä:**
Palvelimen tyyppi	Virtuaalipalvelin
Käyttöjärjestelmä	64-bittinen Ubuntu
Muistin määrä	512 MB
Kovalevyn määrä	5 GB
Esiasennetut komponentit	Olemassa olevan 64-bittisen Ubuntu-asennuspaketin tuomat komponentit

Ubuntu-pohjaisten palvelimien ohjelmistoversiot ovat listattuna liitteessä 3.

6 VERTAILU

Opinnäytetyön käytännön osassa vertailtiin kahta soveltuvinta avoimen lähdekoodin keskitetyn kokoonpanon hallinnan ohjelmistoa keskenään sovittujen testien avulla. Opinnäytetyön tekijä toteutti testeistä noin 80-prosenttia saaden testausapua noin 20-prosenttiin Ixonoksen hosting-tiimin asiantuntijalta. Lisäksi luvussa 5.3 esitettyjen palvelimien käyttöönoton suoritti Ixonoksen hosting-tiimin asiantuntija, jonka jälkeen opinnäytetyön tekijä asensi palvelimille tarvittavat Cfengine 3- sekä Puppet-komponentit.

Opinnäytetyön alussa opinnäytetyön tekijän kokemus keskitetyn kokoonpanon hallinnan järjestelmistä rajoittui matkapuhelinverkon käytönohjauskeskuksen sekä hajautetun automaatiojärjestelmän avulla tehtäviin keskitettyihin toimenpiteisiin. Nämäkin järjestelmät ovat keskitettyjä hallintajärjestelmiä, tosin ne eivät hallitse alijärjestelmiä sääntöpohjaisin perustein. Sääntöpohjainen kokoonpanon hallinta tapahtuu määriteltyjen sääntöjen mukaisesti. Sekä Cfengine 3, että Puppet toteuttavat kokoonpanon hallinnan tietokoneen konfiguroimiseksi suunnitellun ohjelmointikielen mukaisesti.

Vertailevan tutkimuksen suorittamiseen oli varattuna seitsemän viikon aikajakso. Tänä aikana opinnäytetyön tekijän oli tutustuttava valittujen ohjelmistojen tekniseen toteutukseen, käyttöönottoon, ohjelmointikieleen sekä tuotettava tarvittava dokumentaatio ohjelmien ominaisuuksista, käyttöönotosta sekä itse testeistä.

Puppetin osalta työn suorittamista edesauttoi huomattavasti soveltuvan kirjallisuuden saatavuus. Puppetin osalta aikaa ohjelmiston teknisen toteutuksen opiskeluun sekä käyttöönoton toteutukseen kului yksi viikko. Seuraavan puolentoista viikon aikana ohjelmointikielen tutustumisen ohessa suoritettiin sovitut testit.

Cfengine 3:n osalta ongelmaksi osoittautui kirjallisuuden puute. Cfengine 2:sta oli saatavilla hyvin kirjallisuutta, mutta tämä ei soveltunut Cfengine 3:n tutkimiseen. Cfengine 2:n sekä Cfengine 3:n toimintaperiaatteet vastaavat toisiaan. Cfengine 3:ssa on tehty osittaisia teknisiä muutoksia, mutta suurimmat ongelmat koskivat soveltuvan käyttöönotto ohjeistuksen puutetta sekä kokonaan uudistettua ohjelmointikieltä. Cfengine 3:n teknisen toteutuksen sekä käyttöönoton tutustuminen vaativat yhteensä kaksi ja puoli viikkoa. Jäljellä oleva kahden viikon aika kului täysin Cfengine 3:n ohjelmointikielen tutustumiseen sekä testien suorittamiseen.

Cfengine 3:een verrattuna Puppetin käyttöönottamiseksi vaatiman tietomäärän sisäistämistä voisi verrata loivan mäen ylitykseen. Edellisen sijaan Cfengine 3:n sisäistäminen vastaa huomattavasti jyrkemmän mäen ylitystä.

6.1 Opinnäytetyössä tehdyt rajaukset

Vertailevan tutkimuksen aikana havaittiin käytössä olevan ajan olevan riittämätön vaadittavien testien suorittamiseen. Tämän vuoksi opinnäytetyön aikana, tehtiin sovittuun toteutukseen seuraavia rajauksia.

Luvussa 5.2.2 esitetty Nagios-client sovelluksen konfigurointitesti päätettiin jättää suorittamatta. Perusteluina tälle oli muun muassa se, että testin vaiheet ovat identtisiä jo suoritetun Apache web -palvelintestin kanssa, joten tarvittava tieto tämän testin vertailun tekemiseksi oli jo olemassa. Lisäksi luvussa 5.2.2 esitetty paketinhallinnan testi ehdittiin suorittaa täydellisesti ainoastaan Puppet-työkalun avulla. Luvussa 5.2.2 esitetty varmistusohjelmistojen asennusten testi päätettiin myös jättää suorittamatta. Lisäksi kuvassa (Kuva 10) esitettyjen palvelimien (E, F ja G) osalta testejä ei suoritettu. Varusohjelmistojen asennusten testin rajaaminen pois oli perusteltua, koska varmistusohjelmistojen asennusten testi muistuttaa paketinhallinnaltaan Apache web -palvelimen testiä (luku 5.2.2). Lisäksi, koska Puppet ohjelmisto ei tue Windows-käyttöjärjestelmän hallintaa, oli sen poissulkeminen tässä vaiheessa perusteltua. Tämä testi voidaan suorittaa jälkikäteen, mikäli valinta kohdistuu Cfengine 3:een.

6.2 Muutama sana vertailtavista ohjelmistoista

Verrattavat ohjelmistot sisältävät samankaltaisia toiminnallisuuksia. Eräänä syynä ohjelmistojen samankaltaisuuksille on varmasti se, että Puppetin kehittäjä, Luke Kanies lähti kehittämään Puppettia Cfenginessä havaitsemiensa puutteiden sekä näkemiensä rajoitusten pohjalta.

Molemmissa ohjelmissa hallittavat palvelimet ottavat yhteyden isäntäpalvelimelle esimerkiksi kaksi kertaa tunnissa, josta ne hakevat uusimman konfiguraation, mikäli sellainen on saatavilla. Tätä kutsutaan pull-arkkitehtuuriksi, eli hallittava palvelin käy itsenäisesti hakemassa konfiguraation sen sijaan, että isäntäpalvelin aika-ajoin ottaisi yhteyden hallittaviin palvelimiin niiden konfiguroimiseksi (push-arkkitehtuuri).

Pull-arkkitehtuuri mahdollistaa sammutettujen sekä vielä asentamattomien palvelimien konfiguroinnin, koska hallittava palvelin hakee isäntäpalvelimelta konfigurointiasetukset sekä mahdolliset konfigurointimuutokset automaattisesti kytkeydyttyään verkkoon (Strejcek B. 2009, 22).

Molemmat ohjelmat toteuttavat konfigurointeja ns. idempotent-määritelmän mukaisesi. Tämä tarkoittaa, että palvelimelle suoritettava konfigurointi on voitava suorittaa tarvittaessa useita kertoja ilman vaaraa, järjestelmän rikkoutumisesta tai, että toistuva konfigurointi asettaisi järjestelmän tai konfiguroitavan sovelluksen epämääräiseen tilaan.

Molemmissa ohjelmissa käytetään sääntöpohjaista sekä kuvaa ohjelmointikieltä, jonka periaatteena on kuvata hallittavan järjestelmän

tavoitekokoonpanoa, tavoitekokoonpanoon pääsemiseksi tarvittavien työvaiheiden kuvaamisen sijaan.

Molempien ohjelmien avulla voidaan suorittaa monia tietokoneiden järjestelmän kokoonpanon hallintaan sekä ylläpitoon liittyviä toimenpiteitä, esimerkiksi:

- tiedostojen, hakemistojen, linkkien hallintaa
- tiedostojen jakelua
- tiedostojen muutoksien valvontaa (ns. ”trip-wire”)
- tekstitiedostojen (konfiguraatiotiedostojen) muokkaamista käyttäjien sekä ryhmien hallintaa
- mahdollisuus suorittaa komentoja sekä komentojonoja (scripts) kohdekoneilla
- ohjelmistopakettien hallinta (tarkistus, asennus, poisto, päivitys, jne.)
- hallittavien tietokoneiden hallintaa (ryhmittelyä ja kategorisointia) Cfengine 3:n luokkien mukaisesti.

6.3 Tutkimustuloksien vertailua

Ohjelmistojen vertailu suoritettiin käyttämällä Pughin evaluointimatriisia. Ohjelmistojen suoriutuminen kussakin testissä ilmoitettiin numeroarvolla nollasta kolmeen alla olevan taulukon (Taulukko 5) mukaisten määritysten perusteella.

Taulukko 5. Pughin evaluointimatriisissa käytetty arviointi

Arvosana	Arvosanan määritelmä
0	Ohjelmiston käyttö hankaloittaa nykyistä ylläpitoa
1	Ohjelmisto ei tuo apua nykyiseen ylläpitoon
2	Ohjelmisto tuo jonkin verran lisäarvoa ylläpitotoimenpiteisiin
3	Ohjelmisto tuo merkittävää apua ylläpitotoimenpiteisiin

6.3.1 Tiedostotason testien tulokset

Tiedostotason testauksessa testattiin tiedostojen, hakemistojen sekä linkkien hallintaa. Molemmat ohjelmistot suoriutuivat tehtävästä hyvin.

Tiedostotason testissä testattiin lisäksi mahdollisuutta suorittaa annetuilla parametreilla määriteltyjen tiedostojen pakkausta sekä valvottujen tiedostojen muutoksien seurantaa (”trip-wire”). Molemman ohjelmistot tukivat valvottujen tiedostojen muutoksien seurantaa, mutta eivät mahdollistaneet tiedostojen pakkausta eli kumpikaan ei ansainnut tästä testistä täysiä arvosanoja. Tulokset ovat esitettynä alla olevassa taulukossa (Taulukko 6).

Taulukko 6. Tiedostotason testin tulokset

Testi	Puppet	Cfengine 3
Tiedostotason testi	2	2

Konfiguraatiotiedostojen konfigurointitestissä muokattiin tekstitiedostoja sekä lisäämällä että poistamalla tiedostoista konfigurointitietoja. Lisäksi testissä testattiin mahdollisuutta etsiä määriteltyjä konfiguraatio-osia, sekä korvata olemassa olevia konfiguraatio-osia uusilla konfiguraatio-osilla.

Cfengine 3:n pitkä kehityshistoria heijastuu tässä ominaisuudessa hyvin, sillä Cfengine 3:n avulla konfiguraatio tiedostojen muokkaus voidaan toteuttaa täysin annettujen ehtojen mukaisesti. Lisäksi Cfengine 3:ssa on erillinen tuki rivipohjaisten tiedostojen (eng. flat-file) muokkaamiseksi.

Puppetin konfiguraatiotiedoston konfigurointiominaisuudet puolestaan osoittautuivat hyvin rajoittuneeksi. Puppetin avulla voidaan kopioida kohde palvelimelle ennalta konfiguroituja konfiguraatiotiedostoja. Tämän ominaisuuden käyttökelpoisuus on hyvin rajoittunut esimerkiksi ainoastaan tietyntyyppisten tiedostojen korvaamiseen määritellyllä konfiguraatiotiedostolla. Tätä korvaamaan Puppettiin on lisätty mahdollisuus käyttää Ruby-ohjelmointikielellä määriteltyjä ERB-template tiedostoja, joiden avulla monimutkaisiakin konfiguraatio tiedostoja voidaan luoda dynaamisesti ERB-templaten mukaisesti. Tämä ominaisuus on myös rajoittunut olemassa olevien konfiguraatiotiedostojen muokkauksen osalta. ERB-templaattien avulla olemassa oleviin tiedostoihin lisäykset tapahtuvat ainoastaan tiedostojen loppuosaan, sekä uusien osien lisääminen vaatii aiemman ERB-templaatin mukaisen osan poiston ennen uuden lisäämistä.

Puppet-ohjelmistoa voidaan laajentaa määrittelemällä Ruby-ohjelmointikielellä oma konfiguraatiotiedostokohtainen resurssityyppi (eng. resource type). Tämä ominaisuus on tervetullut, mutta luonnollisesti Puppetin jatkaminen omilla Ruby-osilla kasvattaa sekä pirstaloittaa ylläpidettävän ohjelmakoodia ja näin tekee ylläpidosta työläämpää. Puppetin rajoitteita konfiguraatiotiedostojen muokkauksen osalta voidaan myös kiertää käyttämällä muita avoimen lähdekoodin ohjelmistoja, kuten Augeas (Augeas, configuration file editing tool). Augeasin käytössä on joitain rajoitteita, kuten rajoitettu tuki Debian käyttöjärjestelmille. Lisäksi määrittelemättömien konfiguraatiotiedostojen konfigurointi vaatii Augeasilta ns. ”linssin” (eng. lense) määrittelyn, joka edelleen kasvattaa sekä pirstaloittaa ylläpidettävän koodin määrää.

Konfiguraatiotiedostojen osalta Puppet ohjelmisto on hyvä esimerkki nykyajan avoimen lähdekoodin ohjelmista, sillä Puppetin kohdalla on konfiguraatiotiedostojen konfigurointiominaisuudet selvästi päätetty jättää pois. Laajan konfigurointituen saaminen edellyttää muiden avoimen lähdekoodin ohjelmistojen käyttöä.

Cfengine 3 suoriutui testistä kiitettävin arvosanoin. Puppetin avulla samat ominaisuudet vaatisivat lisäohjelmointia tai lisäohjelmien käyttöönottoa. Tulokset ovat esitettynä alla olevassa taulukossa (Taulukko 7).

Taulukko 7. Konfiguraatiotiedostojen konfigurointitestien tulokset

Testi	Puppet	Cfengine 3
Konfiguraatiotiedostojen konfigurointi	2	3

6.3.2 Kokonaisuuksien hallinnan testien tulokset

Apache web -palvelimen testi koostuu useasta osa-alueesta. Testin aikana suoritetaan ohjelmistopakettien hallintaa, konfiguraatiotiedostojen konfigurointia, komentorivikomentojen suoritusta sekä prosessin hallintaa.

Sekä Cfengine 3 että Puppet suoriutuvat vaadittavista toimenpiteistä. Puppetin puutteet Cfengine 3:een verrattuna tämän testin kohdalla liittyvät jälleen konfiguraatiotiedostojen konfigurointiin. Cfengine 3:n kohdalla puolestaan Puppettiin verrattuna on Apache web -palvelimen testin toteutuksen paloittelemattomuus. Puppetin avulla toteutus tehdään Cfengine 3:een verrattuna elegantisti käyttämällä Puppetin moduulitoiminnallisuutta, jonka avulla pääosa toteutuksesta voidaan pitää sivussa (eng. abstracting) selkiyttäen toteutusta.

Kumpikaan ohjelmisto ei saanut täysiä pisteitä, mutta tarjoavat lisäarvoa perinteiseen ylläpitoon verrattuna. Tulokset ovat esitettynä alla olevassa taulukossa (Taulukko 8).

Taulukko 8. Apache web -palvelintestin tulokset

Testi	Puppet	Cfengine 3
Apache web -palvelimen testi	2	2

Nagios sovelluksen avulla voidaan hallittavien koneiden resursseja (prosessit, palvelut, levytila, jne.) monitoroida keskitetysti. Tämän testin tarkoituksena on muokata Nagios client -sovelluksen konfiguraatio palvelimella määritellyn mukaiseksi.

Nagios clientin konfigurointitesti koostuu myös useasta osa-alueesta. Testin aikana suoritetaan ohjelmistopakettien hallintaa, ennalta määritellyn konfiguraatiotiedostojen asettamista sekä prosessin hallintaa.

Tämä testi on sisällöltään hyvin samankaltainen Apache web -palvelin testin kanssa. Ajanpuutteen vuoksi tämän testin suoritus jätettiin tuonnemmaksi ja lopulta aikaa tämän suorittamiseksi ei jäänyt. Testitulokset tämän ominaisuuden osalta määriteltiin siis Apache web -palvelintestin pohjalta. Tulokset ovat esitettynä alla olevassa taulukossa (Taulukko 9).

Taulukko 9. Nagios clientin konfigurointitestin tulokset

Testi	Puppet	Cfengine 3
Nagios client -sovelluksen konfigurointitesti	2	2

Paketinhallinnan testaus ehdittiin suorittaa täydellisesti ainoastaan Puppet-työkalun avulla. Cfengine 3:n osalta paketinhallinnan testaus perustuu aiempaan Apache web -palvelimen testiin. Näiden tietojen perusteella Cfengine 3 sekä Puppet olisivat suoriutuneet testistä yhtä tasavahvasti.

Opinnäytetyön aikana paketinhallintaan liittyen päätettiin tutkia mahdollisuutta toteuttaa palvelimien keskitettyä ohjelmistopakettien jakelua sekä päivitystoiminnallisuutta. Ohjelmistopakettien jakelu sekä päivitys ei ollut osana alkuperäisiä luvussa 5.2 mainittuja kokoonpanon hallinnan vaatimuksia. Kuitenkin Ixonoksen hosting-palveluiden ylläpidosta vastaavan asiantuntijaryhmän toiveen mukaisesti tämän ominaisuuden tutkiminen asetettiin vertailevan tutkimuksen erääksi testattavaksi ominaisuudeksi.

Palvelimien keskitetty ohjelmistopakettien jakelu sekä päivitysten toteuttaminen ohjelmistojen perusominaisuuksilla osoittautui haastavaksi tehtäväksi. Palvelimien keskitetty ohjelmistopakettien jakelu sekä päivitysten toteuttaminen tulisi kyetä toteuttamaan toimittamalla palvelimille luettelo (esimerkiksi tekstitiedoston kautta), asennettavista paketeista. Asennuksen jälkeen tulisi palvelimelta saada luettelo onnistuneesti asennetuista paketeista. Puppetin osalta näiden ominaisuuksien toteuttamiseen ei ole olemassa olevia keinoja. Cfengine 3:n readstringlist (Burgess M. 2009, /1/ 346.) metodin sekä listan iteroivan läpikäyntiominaisuuden (Burgess M. 2009, /2/ 27.) ansiosta palvelimien keskitetty ohjelmistopakettien jakelu ja päivitys nähdään toteuttamiskelpoiseksi.

Näiden tulosten perusteella Cfengine 3 kykenee hoitamaan pakettienhallinnan Puppettia paremmin. Tulokset ovat esitettynä alla olevassa taulukossa (Taulukko 10).

Taulukko 10. Paketinhallinnan testin tulokset

Testi	Puppet	Cfengine 3
Paketinhallinnan testi	2	3

Varusohjelmistojen testaus jätettiin suorittamatta ajanpuutteen vuoksi. Varusohjelmiston asennuksen aikana suoritettavat toimenpiteet sisältävät ohjelmistopakettien hallintaa, konfiguraatiotiedostojen konfigurointia, komentorivikomentojen suoritusta sekä prosessin hallintaa. Ohjelmistojen osalta voidaan arvioida, kuinka ne suoriutuisivat edellä mainituista toimenpiteistä Linux-käyttöjärjestelmässä aiemmin suoritetun Apache web -palvelimen testin perusteella. Tulokset ovat esitettynä alla olevassa taulukossa (Taulukko 11).

Taulukko 11. Varmistusohjelmistojen asennusten testi

Testi	Puppet	Cfengine 3
Varusohjelmistojen asennusten testi	2	2

6.3.3 Muiden testien tulokset

Käyttäjähallinnan testissä testataan käyttäjätilien, ryhmien sekä Linux-käyttöjärjestelmän käyttäjätilien lisäoikeuksien (/etc/sudoers) tarkistusta, luontia, poistoa sekä muokkausta.

Molemmat vertailtavat ohjelmistot suoriutuivat testistä hyvin. Puppetin osalta käyttäjähallinnassa havaittiin kaksi rajoitusta. Puppetin avulla ei voida verifioida, onko poistettavassa ryhmässä edelleen käyttäjiä vai ei. Lisäksi, koska käyttäjätilien lisäoikeuksien (/etc/sudoers) muokkaaminen on konfiguraatiotiedoston konfigurointitoimenpide, Puppetin osalta tätä toimenpidettä ei voida sellaisenaan suorittaa. Tulokset ovat esitettynä alla olevassa taulukossa (Taulukko 12).

Taulukko 12. Käyttäjähallinnan testitulokset

Testi	Puppet	Cfengine 3
Käyttäjähallinnan testi	2	3

Palomuurisääntöjen hallinta toteutettiin Ubuntu sekä CentOS käyttöjärjestelmissä muokkaamalla ennalta määriteltyä /etc/sysconfig/iptables –tiedostoa sekä ottamalla uudet säännöt käyttöön iptables-restore -komennon avulla.

Tämä testin pääosa on palomuurisääntöjen määrittävän konfiguraatiotiedoston konfigurointia. Testin tulokset vastaavat siis konfigurointitiedostojen konfigurointitestiä. Tulokset ovat esitettynä alla olevassa taulukossa (Taulukko 13).

Taulukko 13. Palomuurisääntöjen testitulokset

Testi	Puppet	Cfengine 3
Palomuurisääntöjen testi	2	3

Kuten palomuurisääntöjen testauksen yhteydessäkin tuli osoitettua, vaativat konfigurointitoimenpiteet usein myös komentojen suorittamista. Komentojen suorittaminen on voitava suorittaa luotettavasti. Tämän vuoksi testattavien ohjelmien on kyettävä havaitsemaan, mikäli komennon suorittaminen onnistui. Sekä Cfengine 3, että Puppet tarjoavat mahdollisuuden, jonka avulla voidaan verifioida, onko suoritetun komennon paluuarvo nolla, eli onnistunut suoritus.

Konfigurointitoimenpiteet myös vaativat usein tiedon dynaamista etsimistä tai hakemista järjestelmästä ennen konfiguraatiotoimenpiteen suorittamista. Esimerkiksi, jos palvelimella on useita virtuaalisia IP-osoitteita, on uuden virtuaalisen IP-osoitteen lisäämiseksi selvitettävä viimeisen virtuaalisen interfacen numero (esimerkiksi eth1:2). Puppetilla komentorivikomennon tuloksen hyödyntäminen ei ole mahdollista, mutta Cfengine 3:lla tämä onnistuu.

Komentorivikomentojen kolmas testattava osa-alue koski mahdollisuutta käyttää aiemman komentorivikomennon tulosta argumenttina uudelle

komentorivikomennolle. Puppetilla tämä ei ollut mahdollista, mutta jälleen Cfengine 3:n avulla tämä voitiin toteuttaa.

Komentorivikomentojen testissä Cfengine 3:n sekä Puppetin välillä näkyi isoin ero Cfengine 3:n hyödyksi. Tulokset ovat esitettynä alla olevassa taulukossa (Taulukko 14).

Taulukko 14. Komentorivikomentojen testin tulokset

Testi	Puppet	Cfengine 3
Komentorivikomentojen testi	1	3

Ehdollisessa testissä tutkitaan ohjelmistojen kykyä ryhmitellä palvelimia esimerkiksi palvelimen nimen mukaisesti (hostname), käyttöjärjestelmän mukaisesti, palvelinkohtaisen ryhmittelyn mukaisesti (testipalvelin, tuotantopalvelin, jne.). Sekä Cfengine 3 että Puppet suoriutuivat testin näiltä osin hyvin.

Lisäksi testissä verrataan ohjelmiston kykyä tutkia kellon aikaa, päiväyksen tunnistamista, viikonpäivän tunnistusta sekä vuoden tunnistamista. Nämä ominaisuudet helpottavat ajoittaisten konfigurointitoimenpiteiden määrittelyä. Cfengine 3 tukee näitä sisäisten luokkamäärittelyidensä kautta, mutta Puppet ei osaa tunnistaa vastaavia ajanhetkiä.

Testin viimeisessä osassa testattiin työkalujen kykyä tunnistaa palvelimen tyyppi (virtuaalinen vai fyysinen), fyysisen palvelimen mallin tunnistusta sekä laitetoimittajan ajuripaketin versiota. Kumpikaan ohjelmisto ei tukenut näitä ominaisuuksia sellaisenaan. Molemmissa näiden ominaisuuksien tukeminen on mahdollista. Tämä kuitenkin vaatii palvelimen asennuksen yhteydessä soveltuvien identifiointitiedostojen tai komentojonojen sijoittamista palvelimelle, joiden avulla tunnistaminen voidaan suorittaa. Tulokset ovat esitettynä alla olevassa taulukossa (Taulukko 15).

Taulukko 15. Ehdollisen testin tulokset

Testi	Puppet	Cfengine 3
Ehdollinen testi	2	2

6.4 Sääntöpohjaisen kokoonpanonhallinnan sääntöjen ylläpidettävyys

Edellä mainittujen testien lisäksi vertailun yhteydessä opinnäytetyön kirjoittaja tutki vertailtavien ohjelmistojen keskitetyn konfiguraatiohallinnan sääntöjen ylläpidettävyyttä. Sääntöjen ylläpidettävyyteen vaikuttavat tekijät ovat pääsääntöisesti samoja kuin minkä tahansa ohjelmakoodin ylläpidettävyyteen vaikuttavat tekijät. Alla on käsitelty ohjelmiston ylläpidettävyyteen vaikuttavia tekijöitä verratuissa ohjelmistoissa.

Ohjelmakoodin kommentointi on ohjelmakoodin ylläpidon kannalta tärkeä ominaisuus. Sekä Cfengine 3 että Puppet tukevat ohjelmakoodin kommentointia.

Ohjelmointikielen selkeys on merkittävä ominaisuus, jota voitaisiin luokitella ohjelmointikielen käytettävyyteen vaikuttavana ominaisuutena. Molempien ohjelmien ohjelmointikieli on suunniteltu nimenomaisesti järjestelmien konfigurointia varten. Tämä on edesauttanut Cfengine 3:n sekä Puppetin ohjelmointikielen luettavuutta.

Ohjelmistojen tavoitteena on tarjota kuvaava ohjelmointikieli, jonka kautta kohdejärjestelmiä voidaan konfiguroida riippumatta konfiguroitavan kohdejärjestelmän käyttöjärjestelmästä. Puppet onnistuu tässä Cfengine 3:a paremmin.

Molemmat ohjelmointikielet tukevat ns. globaalien muuttujien käyttöä, joiden avulla ohjelmakoodissa käytettyjä määrityksiä ei tarvitse uudelleen määritellä. Tämän ominaisuuden ansiosta ylläpidon yhteydessä esimerkiksi hakemiston tai tiedoston nimen muutoksen yhteydessä tarvitsee muutos tehdä vain yhteen paikkaan.

Kokoonpanon hallinnan vaatimusten muuttuessa syntyy tarve myös muuttaa tai lisätä ohjelman sääntöjä. Tämän testaaminen käytännössä vaatisi järjestelmän pidempiaikaista käyttöä, joten tämän osakokonaisuuden testausta ei suoritettu.

Molemmat ohjelmistot tukevat palvelimien ryhmittelyä omavaltaisesti sekä ryhmittelyn muuttaminen on molemmissa ohjelmointikielissä helppo toteuttaa. Alla on esitettynä molempien ohjelmointikielien ryhmittelyesimerkit, jossa prod-ryhmään on liitettynä palvelimet ccm_test_vm03.dmz sekä ccm_test_vm05.dmz, sekä loput qa-ryhmään.

Esimerkki Cfengine 3:n tavasta ryhmitellä palvelimia promises.cf –tiedostossa.

```
# Global bundle
bundle common g {

classes:

"prod_env" or => {"ccm_test_vm03_dmz", "ccm_test_vm05_dmz"};
"qa_env"   or => {"ccm_test_vm02_dmz", "ccm_test_vm04_dmz"};
```

Seuraavassa on puolestaan Puppetin keinoin esitettynä nodes.pp –tiedostossa esitetty ryhmittelytapa.

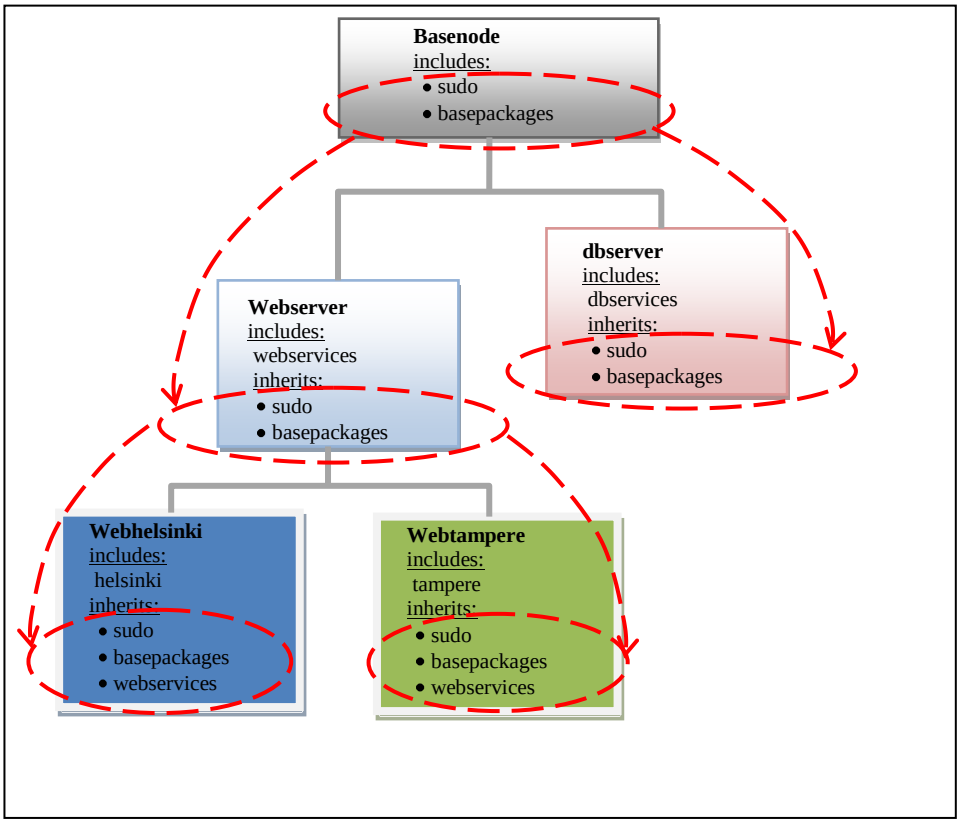
```
# node 'ccm-test-vm02' inherits basenode {
            $nodetype = "QAENV"
}

node 'ccm-test-vm03' inherits basenode {
            $nodetype = "PROD"
}

node 'ccm-test-vm04' inherits basenode {
            $nodetype = "QAENV"
}

node 'ccm-test-vm05' inherits basenode {
            $nodetype = "PROD"
}
```

Esimerkkejä edistyneimmistä ohjelmoinnissa käytettävistä jäsentelymenetelmistä ovat uudelleen käytettävät komponentit (esimerkiksi moduulit tai metodit), tuki olio-ohjelmoinnille, jne. Sekä Cfengine 3, että Puppet tarjoavat molemmat omat versionsa uudelleen käytettävistä komponenteista. Cfengine 3:ssa kyse on metodeista, joiden avulla voidaan kapseloida toistuvasti käytettäviä konfigurointiosia (Burgess M. 2009, /1/ 255). Puppetissa puhutaan käsitteestä nimeltä moduuli, joka on konfigurointiosa, jota voidaan uudelleen käyttää ja jakaa (Turnbull 2007, 109). Näiden uudelleenkäytettävien komponenttien avulla voidaan konfiguroinnin sisäinen toteutus eriyttää käyttämällä määriteltyä rajapintaa.

Kumpikaan ohjelmistojen ohjelmointikielistä ei ole olio-ohjelmoinnin vaatimukset täyttävä. Tosin Puppetissa on olemassa olio-ohjelmointia muistuttavia piirteitä, kuten luokka sekä luokan periyttäminen. Tämän ominaisuuden avulla voidaan proceduraaliseen ohjelmointiin verrattuna tehdä korkeamman tason määrityksiä esimerkiksi hallittavien palvelimien kokoonpanoista, kuten allaolevassa koodiesimerkissä on esitettynä.

```
node basenode {
  include sudo, basepackages
}
node webserver inherits basenode {
  include webservices
}
node dbserver, dbserver_node1, dbserver_node2 inherits ba-
senode {
  include dbservices
}
node webhelsinki inherits webserver {
  include helsinki
}
node webtampere inherits webserver {
  include tampere
}
```

Kuvassa (Kuva 11) on esitettynä esimerkin periytyminen punaisilla katkoviivoilla.



Kuva 11. Palvelimien kokoonpanojen määrittyminen periytymiseen perustuen

Sabin ja Weigel (1998, 44) mainitsivat sääntöpohjaisessa päättelyssä mahdollisen rajoitteen liittyen tavoiteympäristön kuvauksen sekä sääntöjen liian tiukkoihin sidoksiin, josta saattaa aiheutua ylläpidollisia ongelmia. Ylläpidettävien palvelimien määrän sekä palvelinkokoonpanojen erilaisuuden kasvaessa, onkin hyvin todennäköistä, että ilman edellä kuvattuja edistyneempien jäsentelymenetelmien käyttöä, ohjelmiston ylläpito vaatisi liian paljon työtä. Tulokset ovat esitettynä alla olevassa taulukossa (Taulukko 16).

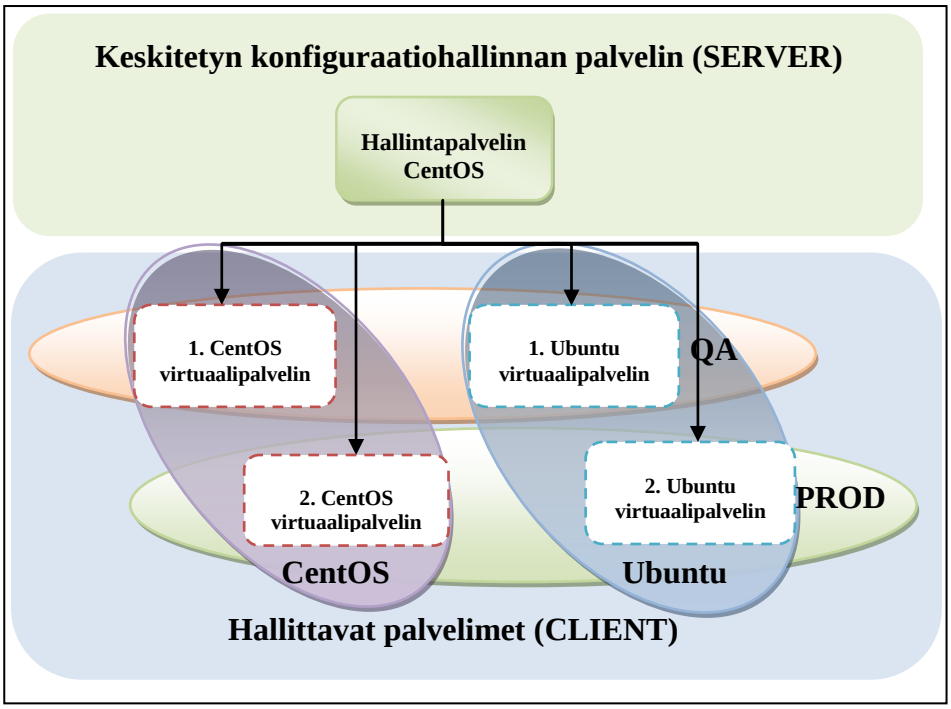
Taulukko 16. Sääntöpohjaisen kokoonpanonhallinnan sääntöjen ylläpidettävyyden arviointi

Testi	Puppet	Cfengine 3
Sääntöpohjaisen kokoonpanonhallinnan sääntöjen ylläpidettävyys	3	2

6.5 Keskitetyn kokoonpanon hallinnan ohjelmistoista opittua

Palvelimien uusien osa-alueiden kokoonpanon hallinnan toteuttaminen havaittiin vaativan runsaasti aikaa, keskimäärin yhdestä kolmeen päivään. Monesti suurimpana haasteena olikin muuttaa omaa ajattelutapaa vastaamaan ohjelmiston vaatimia määrityksiä vastaaviksi.

Testipalvelimissa käytettyjen käyttöjärjestelmien (Ubuntu sekä CentOS) määrä sekä palvelimien ryhmitysten (QA sekä PROD) määrät pysyivät vielä suhteellisen vähäisenä kuten kuvassa (Kuva 12) on esitettynä. Tästä huolimatta kokoonpanon hallinnassa on jo nyt otettava huomioon neljä erilaista variaatiota tai kokoonpanoa, jotka heijastuvat määriteltäviin konfiguraatioihin.



Kuva 12. Testiympäristöjen variaatiot

Alla on esitettynä esimerkit Puppetin sekä Cfengine 3:n ohjelmakoodeista kuinka niillä voidaan määrittää säännöt, joiden perusteella ohjelman tulisi luoda alla olevien määritysten mukainen tiedosto.

Määritykset tiedoston luonnille:

a) Mikäli palvelimen ryhmitys on QA, on tiedoston alkuosa oltava ”TestQA.”
b) Mikäli palvelimen ryhmitys on PROD, on tiedoston alkuosa oltava ”PROD-env.”
c) Mikäli käyttöjärjestelmä on CentOS, on tiedoston loppuosa oltava ”CentOS”
d) Mikäli käyttöjärjestelmä on Ubuntu, on tiedoston loppuosa oltava ”Ubuntu”

Luonnollisesti alla olevat esimerkit voitaisiin toteuttaa helpommin käyttämällä käyttöjärjestelmä sekä ympäristökohtaisia muuttujia. Tämän esimerkkien ideana onkin sen sijaan osoittaa konkreettisesti tarvittavien variaatioiden määrä.

Ohjelmakoodi Puppetin ohjelmointikielellä toteutettuna:

```
# TIEDOSTO A (QA-ympäristölle)
##############################
class qaenv {
 case $operatingsystem {
 CentOS: {
      file {
     "/root/TestQA.CentOS":
     ensure => present;
           }
 }
 Ubuntu: {
      file {
     "/root/TestQA.Ubuntu":
     ensure => present;
           }
}

# TIEDOSTO B (PROD-ympäristölle)
################################
class prodenv {
 case $operatingsystem {
 CentOS: {
      file {

     "/root/PROD-env.CentOS":
     ensure => present
           }
 }
 Ubuntu: {
      file {

     "/root/PROD-env.Ubuntu":
     ensure => present
           }
}
```

Ohjelmakoodi Cfengine 3:n ohjelmointikielellä toteutettuna:

```
files:
  qaenv.ubuntu::
    "/root/TestQA.Ubuntu"
   create => "true";
  prodenv.ubuntu::
    "/root/PROD-env.Ubuntu"
   create => "true";
  qaenv.centos::
    "/root/TestQA.CentOS"
   create => "true";
  prodenv.centos::
    "/root/PROD-env.CentOS"
   create => "true"
```

Havaitsemme, että kukin variaatio vaatii oman määritelmänsä, joka on vastaavasti testattava. Jotta keskitettyyn kokoonpanon hallinnan avulla voitaisiin toteuttaa todellisten tuotantojärjestelmien hallintaa, on selvää, että kokoonpanon hallinnan toimivuus on testattava testiympäristössä ennen tuotantojärjestelmään siirtymistä.

Keskitetty kokoonpanon hallinta sääntöpohjaisella ohjelmointikielellä täyttää hyvin nopeasti ohjelmistoprojektin tunnusmerkit. Tämä asettaa keskitetyn kokoonpanon hallinnalle muun muassa seuraavia suosituksia:

1. Ohjelmiston koodit on versioitava,
2. Ohjelmiston koodeille on määriteltävä testisuunnitelma,
3. Ohjelmisto on testattava testisuunnitelman mukaisesti testiympäristössä,
4. Hyväksyttyjen testien jälkeen myönnetään lupa tuotantojärjestelmään siirtymiseen,
5. Tuotantoon siirto

7 VERTAILUTULOKSET

Opinnäytetyön tutkimusmenetelmänä oli vertaileva tutkimus. Tutkimuksen tulokset perustuvat luvussa 6.3 kuvattuihin vertailuihin.

7.1 Ohjelmistojen laajennettavuus

Sekä Cfengine 3:ssa että Puppetissa havaittuja puutteita voidaan korjata käyttämällä ohjelmien tukemia laajennusosia. Cfengine 3:ssa laajentaminen voidaan toteuttaa käyttämällä Cfenginen moduuleita, jotka ovat yksinkertaista protokollaa tukevia komentoja, joiden kautta voidaan asettaa uusia muuttujia sekä luokkia käyttäjän määrittelemän moduuli-ohjelman mukaisesti (Burgess M. 2009, /1/ 153). Puppetissa laajentaminen voidaan tehdä määrittelemällä konfiguroitavalle asialle oma konfigurointiresurssi määritelmä Ruby-ohjelmointikielen avulla (Turnbull 2007, 158).

7.2 Ehdotettu kokoonpanon hallintajärjestelmä

Opinnäytetyön käytännön osassa tehdyn vertailun aikana havaittiin molemmissa ohjelmistoissa, Cfengine 3:ssa sekä Puppetissa, useita rajoituksia, jotka vaikuttivat kyseisen ohjelmiston kohdalla heikentävästi ohjelmiston vertailun tuloksiin. Luvussa 7.1 esitettyjen keinojen avulla, ohjelmistojen testien yhteydessä havaittuja rajoitteita olisi voitu täydentää soveltuvin keinoin. Tämän opinnäytetyön tarkoituksena on ollut vertailla valittuja ohjelmia Cfengine 3:a sekä Puppettia niiden perusominaisuuksiin perustuen. Ohjelmistojen laajennettavuuden suomat mahdollisuudet on tiedostettava kummankin ohjelmiston osalta, mutta näiden ominaisuuksien testausta ei ollut sisällytetty opinnäytetyön käytännön osuuteen.

Päätös ehdotetusta kokoonpanon hallintajärjestelmästä perustuu luvuissa 5.2.1 – 5.2.3 sekä 6.4 esitettyjen testien perusteella saatuihin kokemuksiin. Jo ennen testien suorittamista oli molempien ohjelmien dokumentoinnista selvinnyt se tosiseikka, että molemmat ohjelmat ovat kykeneviä suorittamaan määritellyt testit. Testien tarkoituksena onkin ollut mitata ohjelmistojen keskinäistä eroa testien toteutettavuuden osalta. Vertailu tehtiin käyttämällä Pughin evaluointimatriisia.

Vertailussa käytettiin myös normaalin pisteytyksen lisäksi painotettua vertailua. Painotuksen avulla haluttiin korostaa tärkeiden testien tuloksia. Painoarvolla kaksi olivat komentorivikomentojen testi, Apache web –palvelimen testi sekä ehdollinen testi. Komentorivikomentojen suorittaminen on osa Linux-palvelimien ylläpitoa, joten sen korostaminen oli perusteltua. Apache web –palvelin ohjelmisto on eräs yleisimmin käytetyistä varusohjelmistoista, joka puolestaan velvoitti Apache web –palvelimen testin korostusta. Ehdollisuuden avulla voidaan helpottaa palvelimien ryhmittelyä, minkä vuoksi ehdollinen testi nähtiin tärkeäksi. Hyvin tärkeiden osa-alueiden testejä painotettiin painoarvolla kolme. Näitä olivat tiedostotason-, konfiguraatiotiedostojen-, paketinhallinnan sekä sääntöpohjaisen kokoonpanonhallinnan sääntöjen ylläpidettävyyden testit. Linux-palvelimien ylläpidossa tiedostojen- sekä paketinhallinta ovat

tärkeässä asemassa esimerkiksi sovellusten päivityksiä hoidettaessa. Vastaavasti konfiguraatiotiedostojen dynaaminen muokkaaminen keskitetysti on ominaisuus, jonka avulla voidaan säästää runsaasti työaikaa ja vähentää ylläpitovirheitä. Koska keskitetty kokoonpanon hallinta itsessään asettaa myös uusia haasteita ylläpidolle esimerkiksi sääntöjen ylläpidettävyyden osalta, katsottiin myös tämän painottaminen tärkeäksi.

Evaluointimatriisin tulokset ovat luettavissa taulukosta (Taulukko 17).

Taulukko 17. Ohjelmistojen evaluointimatriisi

Kriteerit	Painoarvo	Puppet		Cfengine 3	
		Tulos	Painotettu tulos	Tulos	Painotettu tulos
Tiedostotason testit	3	2	6	2	6
Konfiguraatiotiedostojen konfigurointi testi	3	2	6	3	9
Käyttäjähallinnan testi	1	2	2	3	3
Palomuurisääntöjen testi	1	2	2	3	3
Komentorivikomentojen testi	2	1	2	3	6
Apache web -palvelimen testi	2	2	4	2	4
Ehdollinen testi	2	2	4	2	4
Nagios-client sovelluksen konfigurointi testi	1	2	2	2	2
Paketinhallinnan testi	3	2	6	3	9
Varmistusohjelmiston asennusten testi	1	2	2	2	2
Sääntöpohjaisen kokoonpanonhallinnan sääntöjen ylläpidettävyys	3	3	9	2	6
TULOKSET:		**22**	**45**	**27**	**54**

Tulosten perusteella Cfengine 3 suoriutui testeistä selkeästi paremmin arvosanoin. Normaalissa pisteytyksessä Cfengine 3 sai 27 pistettä eli 80 % täysistä pisteistä Puppetin 22 pisteen ja 73 %:n sijaan. Painotetussa pisteytyksessä Cfengine 3 menestyi vieläkin paremmin saaden 54 pistettä (82 %) sekä Puppet sen sijaan 45 pistettä (68 %).

Ixonoksen käyttöön ehdotettu keskitetyn kokoonpanon hallintajärjestelmä kahdesta vertaillusta ohjelmistosta kohdistui siis Cfengine 3:een.

8 YHTEENVETO

Opinnäytetyössä selvitettiin Ixonos Oy:n asettamat vaatimukset keskitetylle kokoonpanon hallinnalle. Asetettujen vaatimusten toteuttamiskelpoisuutta lähdettiin vertaamaan usean keskitetyn kokoonpanon hallinnan avoimen lähdekoodin sovelluksen joukosta valituilla Cfengine 3:lla sekä Puppetilla. Sekä Cfengine 3 että Puppet osoittautuivat varsin hyviksi ohjelmistoiksi keskitetyn kokoonpanon hallintaan.

Opinnäytetyön käytännön osuudessa suoritettujen testien perusteella opinnäytetyön tekijän keräämä kokemus keskitetystä kokoonpanon hallinnasta tulee varmasti olemaan arvokasta tulevaisuuden haasteita varten. Suoritettujen testien perusteella voidaan nähdä useita ylläpidon toimenpiteitä, kuten ohjelmistopakettien asennus, konfigurointitiedostojen muokkaus, komentojen suorittaminen, jne., joita keskitetyn kokoonpanon hallinnan ohjelmistot voivat tehostaa.

Cfengine 3 menestyi paremmin konfiguraatiotiedostojen muokkauksessa sekä komentojen suorittamisessa, paketinhallinnassa. Näiden Ixonoksen organisaatiolle tärkeiden ominaisuuksien vuoksi Cfengine 3:n valinta ehdotettavaksi keskitetyn kokoonpanon hallinnan ohjelmistoksi oli lopulta helppo.

Cfengine 3:een tutustuminen vei yli kaksi viikkoa Puppettiin verrattuna. Osasyynä tähän oli soveltuvan Cfenginen versio kolmen kirjallisuuden puute. Ehkäpä isompana vaikuttavana tekijänä oli kuitenkin Cfengine 3:n tarjoamien ominaisuuksien valtaisa kirjo, josta osoituksena on Cfengine 3:n lähes neljäsataasivuinen referenssimanuaali. Cfengine 3:n sekä Puppetin eroja kuvastaa hyvin seuraava lausahdus.

> ”*Puppet allows you to get on board quicker, but after a while you’ll hit a wall which you get through with Cfengine 3*”
>
> Asko Oukka

LÄHTEET

Augeas, configuration file editing tool. Luettu 24.2.2010. http://augeas.net/page/Creating_a_lens_step_by_step

Borwick, J. 2006. Automate System Configurations and Changes with cfengine. Sys Admin (the journal for UNIX and Linux system administrators) – tammikuu 2006, vol. 15 issue 01, pages 39-43.

Brachman, R.J. & Borgida, A & McGuinness D.L. & Patel-Scheider P.F, 2009. "Reducing" CLASSIC to Practice: Knowledge Representation Theory Meets Reality, Bell labs - http://ect.bell-labs.com/ (http://ect.bell-labs.com/who/pfps/publications/reducing-classic.pdf).

Burgess, M. 2009. /1/ Cfengine Reference Manual. http://cfengine.com, cfengine AS.

Burgess, M. 2009. /2/ Cfengine 3 Concept Guide. http://cfengine.com, cfengine AS.

Burgess, M. & Frisch, Æ, 2007. A System Engineer's Guide to Host Configuration and Maintenance Using Cfengine. Berkley, CA USA: USENIX Association.

Campi, N. & Bauer, K. 2009. Automating Linux and Unix System Administration – Second edition. New York, USA: Apress

DistroWatch.com. Luettu 5.6.2010.
http://distrowatch.com/dwres.php?resource=major

Holmen, A. 2010. Turvallinen lääkehoito – kansainvälinen kirjallisuuskatsaus. Opinnäytetyö. Lohja: Laurea-ammattikorkeakoulu, hoitotyön koulutusohjelma.

Pressman, R.S. 2001, Software engineering: a practitioner's approach 5th ed. New York, USA: McGraw-Hill

Sabin, D, Weigel, R. 1998. Product configuration frameworks - a survey. IEEE Intelligent Systems &their applications, 42–49.

Strejcek, B. 2009. Big engine. http://www.linuxpromagazine.com - Issue 101 huhtikuu 2009, pages 22-28

Swanson, E.B. The Dimension of Maintenance. Proc 2nd Int. Conf. on Software Eng., Oct. 1976, pages 492-497.

Turnbull, J. 2007. Pulling Strings with Puppet: Configuration Management Made Easy. New York, USA: Apress.

HALLINTAPALVELIMEN OHJELMISTOVERSIOT

Taulukko 18. Hallintapalvelimen ohjelmistoversiot

Ohjelmiston kuvaus:	**Versio:**
Käyttöjärjestelmän versio (kernel)	2.6.18-164.11.1.el5
Openssl	openssl-0.9.8e-12.el5_4.1
Puppet-ohjelmisto https://reductivelabs.com/downloads/puppet/puppet-latest.tgz	0.25.4
Ruby (ruby.x86_64) (sisältäen tarvittavat libopenssl-ruby sekä libxmlrpc-ruby paketit)	1.8.5-5.el5_4.8
Ruby Standard Documentation System (ruby-rdoc.x86_64)	1.8.5-5.el5_4.8
Ruby library (ruby-libs.x86_64) (sisältäen openssl sekä rxmlrpc/client komponentit)	1.8.5-5.el5_4.8
Facter	1.5.7
lsb-release	redhat-lsb-3.1-12.3.EL.el5.centos
Cfengine (community version) https://cfengine.com/inside/myspace	3.0.3-1
Berkeley DB database library (compat-db.x86_64)	4.2.52-5.1
Perl Compatible Regular Expression Library (pcre.x86_64)	pcre-6.6-2.el5_1.7

CENTOS-POHJAISTEN PALVELIMIEN OHJELMISTOVERSIOT

Taulukko 19. CentOS-pohjaisten palvelimien ohjelmistoversiot

Ohjelmiston kuvaus:	**Versio:**
Käyttöjärjestelmän versio (kernel)	2.6.18-164.11.1.el5
Openssl	openssl-0.9.8e-12.el5_4.1
Puppet ohjelmisto https://reductivelabs.com/downloads/puppet/puppet-latest.tgz	0.25.4
Ruby (ruby.x86_64) (sisältäen tarvittavat libopenssl-ruby sekä libxmlrpc-ruby paketit)	1.8.5-5.el5_4.8
Ruby Standard Documentation System (ruby-rdoc.x86_64)	1.8.5-5.el5_4.8
Ruby library (ruby-libs.x86_64) (sisältäen openssl sekä rxmlrpc/client komponentit)	1.8.5-5.el5_4.8
Facter	1.5.7
lsb-release	redhat-lsb-3.1-12.3.EL.el5.centos
Cfengine (community version) https://cfengine.com/inside/myspace	3.0.3-1
Berkeley DB database library (compat-db.x86_64)	4.2.52-5.1
Perl Compatible Regular Expression Library (pcre.x86_64)	pcre-6.6-2.el5_1.7

UBUNTU-POHJAISTEN PALVELIMIEN OHJELMISTOVERSIOT

Taulukko 20. Ubuntu-pohjaisten palvelimien ohjelmistoversiot

Ohjelmiston kuvaus:	**Versio:**
Käyttöjärjestelmän versio (kernel)	2.6.31-19-server
Openssl	0.9.8g-16ubuntu3.1
Puppet-ohjelmisto https://reductivelabs.com/downloads/puppet/puppet-latest.tgz	0.25.4
Ruby (ruby.x86_64) (sisältäen tarvittavat libopenssl-ruby sekä libxmlrpc-ruby paketit)	1.8.7.174-1ubuntu1
Ruby-irb	1.8.7.174-1ubuntu1
Ruby Standard Documentation System	1.8.7.174-1ubuntu1
Ruby library	1.8.7.174-1ubuntu1
Ruby OpenSSL library (libopenssl-ruby) (sisältäen tarvittavat 'rxmlrpc/client' kirjastot)	1.8.7.174-1ubuntu1
Ruby readline interface library (libreadline-ruby)	1.8.7.174-1ubuntu1
Facter	1.5.4
lsb-release	4.0-0ubuntu5
Cfengine (community version) https://cfengine.com/inside/myspace	3.0.3-1
Berkeley DB database library (libdb4.7)	4.7.25-7ubuntu2
Perl Compatible Regular Expression Library (libpcre3)	7.8-3
Lisäksi vaadittiin kaksi tokyocabinet komponenttia: libtokyocabinet-dbg libtokyocabinet8	1.4.23-1

CFENGINE 3:N TOTEUTUS APACHE WEB -PALVELIMEN ASENNUKSESTA

promises.cf tiedosto:

```
 body common control {

   bundlesequence => {
      "update",
      "executor",
      "webserver_pre_phase",
      "webserver_install",
       };

   inputs => {
      "update.cf",
      "cf-serverd.cf",
      "cf-execd.cf",
      "library.cf",
      "webserver_pre_phase.cf",
      "webserver_install.cf",
      "common_webserver.cf",
      };
```

common_webserver.cf tiedosto:

```
# --- COMMON SETTINGS FOR WEBSERVER INSTALLATION ############################
bundle common common_webserver {

vars:
   centos::
        "version"               string => "httpd-2.2.3-31";
            "server"            string => "/usr/sbin/httpd";
            "apachectl"                    string => "/usr/sbin/apachectl";
            "conf_file_destination"        string => "/etc/httpd/conf/httpd_conf";
            "conf_file_source"    string =>
"/var/cfengine/inputs/webserver/httpd.conf";
            "paketit"           slist => { "httpd" };
            "webserver_process"   string => "httpd";
            "dir_for_virthost_file"        string => "/etc/httpd/conf.d";
            "docroot_dir"                  string => "/var/www/html";

   ubuntu::
            "version"           string => "2.2.12-1ubuntu2.1";
            "server"            string => "/usr/lib/apache2/mpm-worker/apache2";
            "apachectl"                    string => "/usr/sbin/apache2ctl";
            "conf_file_destination"        string => "/etc/apache2/apache2_conf";
            "conf_file_source"    string =>
"/var/cfengine/inputs/webserver/apache2.conf";
            "paketit"           slist => { "apache2" };
            "webserver_process"   string => "apache2";
            "dir_for_virthost_file"        string => "/etc/apache2/sites-available";
            "docroot_dir"                  string => "/var/www";

   any::
         "path_to_cfengine_controls_dir" string => "/root/cfengine_controls";
         "file_add_virtual_hosts"        string =>
                                "$(path_to_cfengine_controls_dir)/add.virtual_hosts";
         "file_remove_virtual_hosts"     string =>
                                "$(path_to_cfengine_controls_dir)/remove.virtual_host
                                s";
         "virtual_hosts_file_fp"         string =>
                                "$(dir_for_virthost_file)/virtualhosts.conf";

classes:
     "add_virtual_hosts"    expression => fileexists("$(file_add_virtual_hosts)");
     "remove_virtual_hosts" expression => fileexists("$(file_remove_virtual_hosts)");
}
```

webserver_pre_phase.cf tiedosto:

```
# --- PRE-STEPS FOR WEBSERVER INSTALLATION ########################################
bundle agent webserver_pre_phase
{
vars:
  any::
    "touch_bin_fp"           string => execresult("/usr/bin/which touch", "noshell");
    "apt_last_exec_flagfile" string => "apt_get_update_last_exec";
    "apt_ok2exec_flagfile"   string => "ok_to_exec_apt_get_update";

  ubuntu::
    "this_day"        string => execresult("/bin/date '+%Y-%m-%d'","useshell");
   "prev_apt_exec_day" string => execresult("/bin/ls -l $(com-
        mon_webserver.path_to_cfengine_controls_dir)/$(apt_last_exec_flagfile) | awk
        '{print $6}'","useshell");

classes:
   ubuntu::
     "executed_already_today"    expression =>
                                    strcmp("$(this_day)","$(prev_apt_exec_day)");
     "ok_to_exec_apt_get_update" expression => fileex-
        ists("$(common_webserver.path_to_cfengine_controls_dir)/$(apt_ok2exec_flagfil
        e)");

   # ------------------------------------------------------------------------
   # The below ensures that the 'apt-get update' is only run if:
   #  a) we're on Ubuntu node, and
   #  b) it's allowed (i.e. the $(apt_ok2exec_flagfile) file exists), and
   #  c) the 'apt-get update' has not been executed today:
   # ------------------------------------------------------------------------
   ubuntu.ok_to_exec_apt_get_update.!executed_already_today::
     "apt_get_failed" not => returnszero("/usr/bin/apt-get update","noshell");
     "apt_get_update_file_exists_already" expression => fileex-
        ists("$(common_webserver.path_to_cfengine_controls_dir)/$(apt_last_exec_flagf
        ile)");

commands:
  # Update the timestamp on the file so the "apt-get update" won't be executed again
   ubuntu.!apt_get_failed.apt_get_update_file_exists_already::
   "$(touch_bin_fp) $(common_webserver.path_to_cfengine_controls_dir)/
   $(apt_last_exec_flagfile)";

files:
   # For Ubuntu, we'll create the $(apt_last_exec_flagfile) file if the "apt-get
update" did not fail
   ubuntu.!apt_get_failed.!apt_get_update_file_exists_already::
      "$(common_webserver.path_to_cfengine_controls_dir)/$(apt_last_exec_flagfile)"
             create => "true";
```

webserver_install.cf tiedosto:

```
# --- INSTALLING NECESSARY PACKAGES
#######################################################
bundle agent webserver_install
{
vars:
  any::
    "packages_to_install" slist => { @(common_webserver.paketit) };

packages:
  centos::
    "$(packages_to_install)"
           package_policy        => "add",
           package_method        => yum,
           package_architectures => { "$(sys.arch)" },
           package_version       => "$(common_webserver.version)",
           classes               => if_ok("webserver_sw_ok"),

  ubuntu.!apt_get_failed::
    "$(packages_to_install)"
           package_policy        => "add",
|          package_method        => apt,
           classes => if_ok("webserver_sw_ok"),

# --- CONFIGURING INSTALLED WEBSERVER
#######################################################
add_virtual_hosts::
   "virtual_hosts_to_be_added"   slist =>
      {readstringlist("$(common_webserver.file_add_virtual_hosts)",
       "#.*","[\n]",5,400) };

classes:
  linux::
     "virtual_host_file_exists" expression =>
        fileexists("$(common_webserver.virtual_hosts_file_fp)");

# Let's place the prepared webserver configuration file in place:
files:
  "$(common_webserver.conf_file_destination)"
    copy_from => local_cp("$(common_webserver.conf_file_source)");

  # Create the necessary directory for virtual host config if not there:
  "$(common_webserver.conf_file_destination)/."
    create => "true";

  # Create the DocRoot directories:
  "$(common_webserver.docroot_dir)/$(virtual_hosts_to_be_added)/."
    create => "true";

  !virtual_host_file_exists::
    # Create the necessary virtual host file if not there:
    "$(common_webserver.virtual_hosts_file_fp)"
      create => "true";

  any::
    # Now edit the same file:
    "$(common_webserver.virtual_hosts_file_fp)"
       edit_line =>
         add_virtual_host("$(common_webserver.path_to_cfengine_controls_dir)",


# --- RESTARTING THE NECESSARY WEBSERVER PROCESS
#########################################
processes:
  webserver_sw_ok::
    "$(common_webserver.webserver_process)"
      restart_class => "start_webserver";

commands:
  start_webserver::
    "/etc/init.d/$(common_webserver.webserver_process) start";
}
#############################################################################################
#################
bundle edit_line add_virtual_host(sourcedir,sourcefile)
{
insert_lines:
           "$(sourcedir)/$(sourcefile)"
           insert_type    => "file",
           expand_scalars => "true";
   }
```

PUPPETIN TOTEUTUS APACHE WEB -PALVELIMEN ASENNUKSESTA

/etc/puppet/manifests/nodes.pp tiedosto:

```
node 'ccm-test-vm03' inherits basenode {
     include apache
     apache::virtual_host { "vm03a.testing.com":
          ip => "30.03,.33.13",
          order => "eka", # just to make setup unique
          apply => "true"
     }
     apache::virtual_host { "vm03b.testing.com":
          ip => "30.03.33.23",
          order => "toka",  # just to make setup unique
          apply => "true"
     }
}
```

/etc/puppet/manifests/site.pp tiedosto:

```
  import "templates.pp"
  import "nodes.pp"
  import "classes/*"
  import "os/*"
  import "apache"
```

/etc/puppet/modules/apache/manifests/init.pp tiedosto:

```
class apache {

                case $operatingsystem {
                     CentOS: {
                                $service = "httpd"
                                $packagelist = ["$service.$architecture"]
                                $destination_file = "/etc/httpd/conf/httpd.conf"
                                $source_file = "puppet:///apache/httpd.conf"
                                }
                     Ubuntu: {
                                $service = "apache2"
                                $packagelist = ["$service"]
                                $destination_file = "/etc/apache2/apache2.conf"
                                $source_file = "puppet:///apache/apache2.conf"
                                }
                }

                #### The sw-packages installation ################################
                package { $packagelist:
                    ensure => "installed"
                }

                #### Putting the apache config-file in place ####################
                apache::apache_files {

                    $destination_file:
                                source => $source_file
                }

                #### Setting up the service #####################################
                service { $service:
                    enable     => "true",
                    ensure     => "running",
                    hasrestart => "true",
                    hasstatus  => "true",
                    require    => Package["$packagelist"]
                }
}
```

/etc/puppet/modules/apache/manifests/apache_files.pp tiedosto:

```
define apache::apache_files($owner = root, $group = root, $mode = 644, $source,
$backup  = false, $recurse = false, $ensure = file) {

                case $operatingsystem {
                      CentOS: {
                                $service = "httpd"
                                $packagelist = ["$service.$architecture"]
                      }
                      Ubuntu: {
                                $service = "apache2"
                                $packagelist = ["$service"]
                      }
                }

                file { $name:
                                mode            => $mode,
                                owner           => $owner,
                                group           => $group,
                                backup          => $backup,
                                recurse         => $recurse,
                                ensure          => $ensure,
                                require => Package["$packagelist"],
                                source          => $source,
                }
}
```

/etc/puppet/modules/apache/manifests/virtual_host.pp tiedosto:

```
define apache::virtual_host($ip, $order, $apply, $ensure = "enabled") {

        #### Setting the virtual host conf file destination & DocumentRoot variable #
           case $operatingsystem {
                      CentOS: {
                         $directory = "/etc/httpd/conf.d"
                         $filu = "$directory/$name.conf"
                         $document_root = "/var/www/html/$name/"
                         $service = "httpd"
                      }
                      Ubuntu: {
                         $directory = "/etc/apache2/sites-available"
                         $filu = "$directory/$name.conf"
                         $document_root = "/var/www/$name/"
                         $service = "apache2"
                      }
           }

           #### Creating necessary directory ######################################
           Exec { path => "/usr/bin:/bin" }
           exec { $order:
                      command => "mkdir -p $directory",
                      before => File["$filu"]
           }

           #### Placing the actual virtual host file in place #####################
           file { $filu:
                      ensure => $apply ? {
                                  "true"  => "present",
                                  "false" => "absent",
                      },
                      content => template("apache/virtual_host.erb"),
                      notify => Service["$service"],
                      require => Exec["$order"]
           }

           #### Creating the actual DocumentRoot directory ########################
           file { $document_root:
                      ensure => $apply ? {
                                  "true"  => "directory",
                                  "false" => "absent",
                      },
                      ensure => directory,
                      require => File["$filu"]
           }
}
```

/etc/puppet/modules/apache/templates/virtual_host.erb tiedosto:

```
<VirtualHost <%= ip %>>
        DocumentRoot <%= document_root %>
        ServerName <%= name %>
</VirtualHost>
```