jamk.fi

Tuotekehitysympäristön monistaminen ja automatisointi

Case: Contriboard

Henri Tervakoski

Opinnäytetyö
Toukokuu 2015

Ohjelmistotekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Tervakoski, Henri	Opinnäytetyö	27.5.2015
	Sivumäärä 55 + 5	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Tuotekehitysympäristön monistaminen ja automatisointi
Case: Contriboard

Koulutusohjelma
Ohjelmistotekniikka

Työn ohjaaja(t)
Matti Mieskolainen

Toimeksiantaja(t)
N4S@JAMK, Marko Rintamäki

Tiivistelmä

Työn tarkoituksena oli tutkia, kuinka sovelluskehitystä voidaan tehostaa eri menetelmin. Työssä käydään lyhyesti läpi perinteiset ohjelmistotuotantomallit ja uudempi DevOps toimintatapa. Lisäksi työ käsittelee muutamia oleellisia sovelluskehityksessä käytettyjä työkaluja ja pilvipalveluita. Tutkittuja kohteita pyritään automaation avulla ketjuttamaan niin, että sovellukselle voidaan luoda tuotanto- tai kehitysympäristö mahdollisimman pienellä vaivalla.

Työn toimeksiantajana toimi N4S@JAMK-projekti, jossa kehitetään Contriboard-nimistä palvelua. Contriboard on toistaiseksi avoimen lähdekoodin tuote, jonka ydintoiminnallisuus on tarjota web-pohjainen, jaettava valkotaulu esimerkiksi muistiinpanoille ja tehtäville. Contriboard on arkkitehtuuriltaan hajautettu järjestelmä, jonka toimintaan saanti voi olla erittäin aikaa vievää. Työssä tutkittuja menetelmiä hyödyntämällä pyritään luomaan työnkulku, jolla Contriboardin kehitykseen voi osallistua helposti luomalla kehitysympäristö automaation avulla. Kehitysympäristöt ovat osa Contriboardin Corolla-työkaluketjua.

Lopuksi pohditaan ketjun soveltuvuutta muihin ohjelmistoprojekteihin, kehitysideoita ja yleisesti saavutettua hyötyä.

Avainsanat (asiasanat)
Automaatio, DevOps

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Description

Author(s)	Type of publication	Date
Tervakoski, Henri	Bachelor's thesis	27.5.2015
	Number of pages 55 + 5	Language of publication Finnish
		Permission for web publication: x

Title of publication

Automation and Duplication of Software Product Development Platform
Case: Contriboard

Degree programme
Software Engineering

Tutor(s)
Mieskolainen, Matti

Assigned by
N4S@JAMK, Marko Rintamäki

Abstract

The purpose of this thesis was to research procedures to make applications development more efficient. The work consists of basics of traditional software engineering methodologies and takes a peek at more recent DevOps method. In addition, some of the most essential software development tools and cloud service providers are covered. The ultimate goal is to find a way to make fully functional production or development environments for any product with minimal effort.

The thesis was assigned by N4S@JAMK project, in which a service named Contriboard is being developed. For now, Contriboard is an open source project, the core feature of which is to act as shareable, web-based whiteboard for memos and other tasks. As the Contriboard is a quite complex and distributed system by design, it can be a very overwhelming task to get it up and running. By using the researched procedures, the objective is to make a workflow for Contriboard developers who can easily deploy the new Contriboard development environments, which are a part of Corolla, an automated development toolchain for Contriboard.

Finally, the thesis discusses the usability of the toolchain for other projects, future development and the generally gained benefits.

Keywords/tags (subjects)
Automation, DevOps

Miscellaneous

SISÄLTÖ

KUVIOT

KÄSITTEET

Build – Ohjelmakoodin käännösvaihe tai viittaus tiettyyn käännösversioon

Distro, Distribuutio – Linux jakelu, jonka ohjelmisto on toimittajakohtainen

Fork – Ohjelmistokehityksessä alkuperäisestä poikkeava, ohjelmakoodin toinen kehityshaara

Host, hosting – Tietotekniikassa palvelimen tai palvelutilan ylläpito

Inkrementti – Vähittäin kasvava tai kehittyvä

Instanssi – Yksittäinen esiintymä, esim. virtuaalikone pilvipalvelussa

Iteraatio – Toistuva prosessi, jossa tehdään samat vaiheet

Patch - Ohjelmaan tehty korjaus

QA – Quality Assurance, laadunvarmistus

Revisio – Tarkastettu versio, vedos

Rollback – Palautus edeltävään tilanteeseen

SSH – Secure Shell, etäkäyttöön luotu salattu tietoliikenneprotokolla

UAT – User Acceptance Testing, käyttäjän hyväksymistestaus

VM – Virtual Machine, virtuaalikone

Wrapper – Sovitin, kehys. Esimerkiksi kahden rajapinnan yhdistäminen tai piilottaminen uuden kerroksen alle

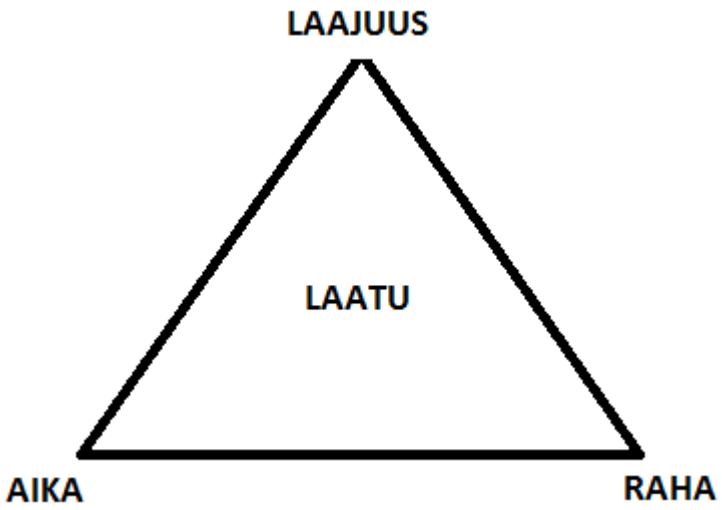
1 Työn lähtökohdat

1.1 Johdanto

Ohjelman tai ohjelmiston toimittaminen asiakkaalle ei ole koskaan erityisen yksinkertainen prosessi, varsinkaan yritysmaailmassa. Yksinkertaisesti ajateltuna helpoin kuviteltavissa oleva skenaario on, että ohjelman toimittajalla on olemassa jo valmis palvelu tai ratkaisu, joka vastaa asiakkaan tarpeisiin. Vaikka yritysten hankinnoissa on aina byrokratiaa taustalla, palvelun käyttöönotto voi lopulta olla hyvin nopea: maksetaan käyttöoikeudesta ja otetaan tuote käyttöön. Skenaarion toinen ääripää voi olla usein vuosikausia kestävä prosessi, jossa lähdetään rakentamaan täysin uutta tuotetta. Osapuolet joutuvat tekemään tiivistä yhteistyötä, että päästään molemmin puolin miellyttävään lopputulokseen. Koska sekä ohjelman toimittaja että asiakas haluavat luonnollisesti tehdä kannattavaa liiketoimintaa, on molempien etu, että toimitusprosessi etenee ongelmitta ja pysyy sovitussa aikataulussa molempien tahojen puolesta.

Jos asiaa tarkastelee ohjelman toimittajan silmin, mitkä käytännöt täytyvät olla kunnossa, että vältetään pahimmat sudenkuopat? Ohjelmistokehitys kuten projektit yleensä on tasapainottelua muutaman resurssin ehdolla (ks. kuvio 1). Vaakalaudalla ovat laajuus, aika, laatu ja kustannukset. Kun kaikki resurssit on pyrittävä pitämään tasapainossa, on tunnistettava ja minimoitava kaikki ylimääräinen. Jo alkuvaiheessa tehtävillä päätöksillä ja toimintatavoilla on ehkäistävissä monia potentiaalisia myöhemmin esiintyviä ongelmia.

Erilaisia ohjelmistotuotannon tapoja on kehitetty yhtä kauan kuin itse ohjelmistojakin. Vaikka ei ole olemassa valmista mallia joka toimii täydellisesti kaikissa tapauksissa, on olemassa erittäin hyviä lähtökohtia oman toimintamenetelmän rakentamiseksi.



Kuvio 1. Projektikolmio havainnollistaa resurssijaon

Ohjelmistokehityksessä on paljon toistuvia prosesseja, joihin kuluu huomattava määrä aikaa. Kehityksen aikana sovellusta käännetään, testataan ja ajetaan jatkuvasti. Eri vaiheita automatisoimalla voi paremmin keskittyä tuottamiseen ja säästää aikaa ja siten epäsuorasti rahaa. Samalla vähennetään inhimillisiä virheitä.

Ohjelmistojen koko voi kasvaa massiivisiin mittoihin, ja kokonaisuuden hallitseminen käy vaikeaksi. Ohjelmistokehyksiä (Framework) käyttämällä säästää aikaa ja parantaa hallittavuutta, mutta kehyksen valinta on syytä tehdä huolella. Järjestelmät voivat olla hajautettuja, joilla on kehittäjiä ja käyttäjiä ympäri maailmaa. Uusia teknologioita tulee jatkuvasti vanhojen rinnalle. Tässä työssä tutkittiin alan viimeisimpiä trendejä ja samalla etsittiin parhaita työvälineitä ja menetelmiä ohjelmistokehityksen saralla.

1.2 N4S-tutkimusohjelma

Need 4 Speed (N4S) on Digilen vuonna 2014 käynnistynyt, enintään nelivuotinen tutkimusohjelma jonka osittain rahoittaa Tekes. Tutkimusohjelman tarkoituksena on rakentaa perusta suomalaisille ohjelmistoyrityksille uudessa digitaalisessa taloudessa ja parantaa kilpailukykyä kansainvälisillä markkinoilla. Ohjelmassa kokeillaan reaaliaikaisia liiketoimintamalleja, jotka perustuvat syvään asiakastuntemukseen, joiden avulla päästään reaaliaikaiseen arvon tuottamiseen. Ohjelmassa on mukana yli 30 organisaatiota, joihin kuuluu Suomessa johtavia teollisuusyrityksiä, PK-yrityksiä ja oppilaitoksia. (N4S-ohjelma 2014.)

Ohjelmalla on kolme painopistettä:

***Arvon tuottaminen reaaliajassa**: Suomalainen ohjelmistointensiivinen teollisuus on uudistanut liiketoimintansa ja organisaationsa kohti arvopohjaista ja mukautuvaa reaaliaikaista liiketoiminnan mallia. Tämän muutoksen tukemiseksi on luotu tekninen infrastruktuuri ja sille vaadittavat ominaisuudet.*

***Syvällinen asiakastuntemus – parempi tuotto**: Ohjelmistointensiiviset teollisuudenalat Suomessa hyödyntävät uutta teknistä infrastruktuuria ja sen mahdollisuuksia, samoin kuin moninaisia data- ja tietolähteitä hankkiakseen syvällistä tietämystä asiakkaiden tarpeista ja käyttäytymisestä. Tämän tiedon avulla ala pystyy parantamaan myyntiä ja saa huomattavaa tuottoa panostuksistaan tuotteidensa ja palveluidensa kehitystyöhön.*

***"Elohopeabisnes"** – Uutta rahaa etsimässä: Tämä kokonaisuus keskittyy siihen, miten yritykset ja yhteisöt voivat käyttäytyä nestemäisen elohopean tavoin, löytäen ja vallaten uusia uria. "Elohopeabisnes" merkitsee kykyä mukautua uusiin liiketoiminnan olosuhteisiin ja etsiä aggressiivisesti uusia liiketoimintamahdollisuuksia uusilla markkinoilla vähäisellä vaivalla. Tämän uuden lähestymistavan liiketoiminnan kasvuun mahdollistaa jatkuva ja aktiivinen strateginen painotus, uusi tapa johtaa.*

(N4S-ohjelma 2014.)

Jokainen painopiste on samalla ns. työpaketti, jota ohjelmassa mukana olevat yritykset ratkovat.

N4S@JAMK-projekti

Varsinainen toimeksiantaja työlle oli N4S@JAMK, joka keskittyy N4S ohjelman ensimmäiseen työpakettiin - Arvon tuottamiseen reaaliajassa. N4S@JAMK kehittää ohjelman aikana Contriboardia, jota käytetään referenssituotteena nopeamman sovelluskehityksen tutkimuksessa. Contriboard esitellään tarkemmin luvussa 4.

1.3 Työn tavoitteet ja rajaus

Työn tavoitteena oli tutkia menetelmiä, joilla Contriboardin kehittämisestä saataisiin mahdollisimman ketterää ja vaivatonta. Contriboard on useasta komponentista koostuva hajautettu palvelu, joka voi skaalautua tuhansille samanaikaisille käyttäjille. Siksi se on vaikeasti asennettava ja ylläpidettävä. Samalla kehittäjän näkökulmasta on työlästä ja aikaa vievää saada toimiva kehitysympäristö Contriboardille. Tähän ongelmaan haettiin ratkaisua DevOps periaatteista, erityisesti automaatiosta. Työssä kokeiltiin muutamia automaatiotyökaluja ja niillä toteutettiin tapoja jotka vähentävät asennuksiin ja konfigurointeihin menevää aikaa. Samoja menetelmiä käyttäen mahdollistettiin jatkuvan julkaisun ketju, jolloin Contriboardista on aina viimeisin versio saatavilla. Työ rajattiin Contriboardin kehittämisessä käytettäviin ympäristöihin, eikä sen lopulliseen tuotantoympäristöön, vaikka tutkitut tekniikat lähtökohtaisesti tukevat myös tuotantoa. Työn tavoitteet kiteytettynä ovat

- Millä tavalla voidaan nopeuttaa ja yksinkertaistaa Contriboard tuotteen koekäyttö/evaluointitilanteita?
- Mitä tekniikkaa soveltaen voidaan luoda nopeasti uusi kehityshaara (ominaisuus) Contriboard-tuotteeseen?
- Soveltuvatko löydetyt teknologiat testikohteen rakentamiseen ja millä ehdoilla?

2 Ohjelmistotuotantomallit

2.1 Yleistä

N4S-ohjelman sekä erityisesti N4S@JAMK-projektin luonteen ja toiminnan vuoksi on hyvä tuntea ohjelmistotuotantomallin perusta, jotta voi ymmärtää, mitä ongelmia nykyaikaisilla ketterillä kehitysmenetelmillä ratkotaan. Olipa kyse sitten ohjelmistokehityksestä tai mistä tahansa projektista, onnistumisen kannalta on välttämätöntä hallinnoida ja organisoida tehtäviä. Tässä työssä ei kuitenkaan käsitellä varsinaisia projektinhallintasovelluksia, vaikka sellaisia vaaditaankin menetelmien avuksi.

Noin 50 vuotta sitten kohdattiin ohjelmistokriisi, jossa systemaattisten työmenetelmien puuttuminen ja ohjelmistojen kasvaminen johti tehottomaan työskentelyyn ja huonosti toimiviin ohjelmiin. Tuolloin alettiin kehittämään ohjelmistotuotantomalleja ja niiden tukitoimintoja. Vaikka ohjelmistokriisistä on kulunut jo pitkä tovi, samoja kompastuskiviä on havaittavissa vielä tänä päivänäkin. Vielä vuonna 2010 Forresterin tekemän tutkimuksen mukaan yli 30 % vastanneista IT-ammattilaisista ei käyttänyt mitään muodollista prosessimenetelmää. (West & Grant 2010.).

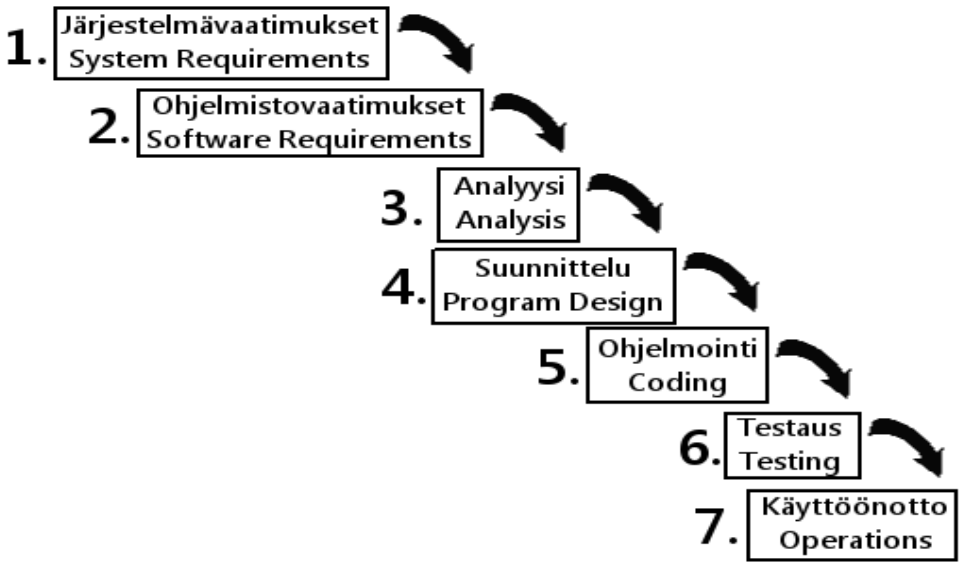
Ohjelmistotuotannon mallit voidaan jakaa perinteisiin ja ketteriin menetelmiin ja usein jopa näiden yhdistelmäksi. Perinteiset mallit sisältävät yleensä hyvin tarkasti ennalta määrätyt vaiheet. Perinteisessä mallissa jopa koko ohjelman elinkaari pyritään suunnittelemaan etukäteen, jolloin muutoksiin reagointi on vaikeaa.

Ketterissä menetelmissä on myös selkeät vaiheet, mutta ne ovat pienempiä ja kehitys tapahtuu pienissä iteraatioissa. Ketterät menetelmät ovat hyvin asiakasläheisiä, joiden avulla saadaan palautetta hyvin nopeasti ja kehityssuunta pidetään oikeana. Luvuissa 2.2.1-2.2.3 käydään läpi muutamia perinteisiä ohjelmistokehitysmalleja yleisellä tasolla.

2.2 Perinteiset mallit

2.2.1 Vesiputousmalli

Perinteistä vesiputousmallia pidetään yhtenä ensimmäisistä ohjelmistotuotannon malleista. Sen esitteli amerikkalainen Winston W. Royce julkaisemassaan artikkelissa vuonna 1970. Yksinkertaistetussa vesiputousmallissa eri vaiheista edetään nimensä mukaisesti vain alaspäin. Royce itse kuitenkin piti iterointimahdollisuutta eri vaiheiden välillä tärkeänä. (Royce 1970.). Kuviossa 2 on esitetty Roycen alkuperäisen vesiputousmallin vaiheet. (Vesiputousmalli n.d.).



Kuvio 2. Roycen vesiputousmalli

Perinteisellä vesiputousmallilla tehdään varmasti vieläkin sovelluksia, sillä se on yksinkertainen ja helppo tapa. Vesiputousmalli sellaisenaan on kuitenkin joustamaton ja huono, jos projektin laajuudessa tai vaatimusmäärittelyssä on epävarmuutta. Vesiputousmallin perusajatus on kuitenkin usein havaittavissa muissakin ohjelmistotuotannon malleissa.

2.2.2 Rational Unified Process

Rational Unified Process (RUP, yhtenäistetty ohjelmistokehitysprosessi) on ohjelmistokehitystä varten kehitetty prosessimalli. Se on johdettu ja yksinkertaistettu Unified Process (UP) prosessikehyksestä, joka on hyvin laaja ja esittelee mallin yleisemmällä tasolla. Mallin on kehittänyt alun perin Rational Software, josta tuli IBM:n jaosto vuonna 2003. RUP on iteratiivinen menetelmä, jossa eri vaiheet viedään läpi usealla vesiputousiteraatiolla. RUP määrittää projektille neljä perusvaihetta, jotka ovat:

1. Aloitus, perustaminen
2. Kehittely, tarkentaminen
3. Valmistus, rakentaminen
4. Muutos, siirtyminen

RUP on menetelmä, jossa sovelluksen toiminnollisuus johdetaan käyttötapauksista (Use Case). Käyttötapaus on yksinkertainen kuvaus käyttäjän ja järjestelmän välisestä vuorovaikutuksesta. Tällä varmistetaan, että vain arvoa tuovat ominaisuudet kehitetään. (RUP summary 2008.)

2.2.3 V-malli ja prototyyppimalli

V-malli on kuin vesiputousmallin laajennus, jossa jokaisessa vaiheessa tehdään vielä erityinen testaus. Tällä pyritään varmistamaan, että vaiheet tehdään huolella, eikä seuraavaan vaiheeseen siirrytä ilman kunnollista tarkistamista ja katselmointia.

Prototyyppimallin toiminta perustuu siihen, että tehtyjen vaatimusten perusteelta tehdään karu versio kuvitellusta lopputuotteesta, joka toimitetaan asiakkaalle. Näin saadaan palautetta heti ja prototyypin jatkokehitys etenee asiakkaan toiveiden mukaisesti.

2.3 Ketterät menetelmät

Ketterän ohjelmistokehityksen määritelmästä on varmasti useita näkemyksiä, mutta virallisen julistuksen (Julistuksen takana olevat periaatteet 2001.) 12 periaatetta ovat seuraavat:

- *Tärkein tavoitteemme on tyydyttää asiakas toimittamalla tämän tarpeet täyttäviä versioita ohjelmistosta aikaisessa vaiheessa ja säännöllisesti.*
- *Otamme vastaan muuttuvat vaatimukset myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi.*
- *Toimitamme versioita toimivasta ohjelmistosta säännöllisesti, parin viikon tai kuukauden välein, ja suosimme lyhyempää aikaväliä.*
- *Liiketoiminnan edustajien ja ohjelmistokehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.*
- *Rakennamme projektit motivoituneiden yksilöiden ympärille. Annamme heille puitteet ja tuen, jonka he tarvitsevat ja luotamme siihen, että he saavat työn tehtyä.*
- *Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.*
- *Toimiva ohjelmisto on edistymisen ensisijainen mittari.*
- *Ketterät menetelmät kannustavat kestävään toimintatapaan. Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.*

- *Teknisen laadun ja ohjelmiston hyvän rakenteen jatkuva huomiointi edesauttaa ketteryyttä.*
- *Yksinkertaisuus - tekemättä jätettävän työn maksimointi - on oleellista.*
- *Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä.*
- *Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan, ja mukauttaa toimintaansa sen mukaisesti.*

2.3.1 Extreme Programming

Varhaisista ketteristä menetelmistä erityisen suosion saavuttanut Extreme Programming (XP), sai alkunsa vuonna 1996. Se painottaa asiakastyytyväisyyttä ja perustuu arvoihin yksinkertaisuus, kommunikointi, palaute, kunnioitus ja rohkeus. Extreme Programming pitää sisällään sääntömanifestin, jota tarkasti noudattamalla työntekoa tehostetaan ja varmistetaan tulos. (Wells n.d). Extreme Programming painottaa erityisesti tiimityöskentelyn ja kommunikoinnin tärkeyttä. Yksi erikoinen käytäntö on pariohjelmointi, jolloin kaksi ohjelmoijaa tekevät työtä yhdellä päätteellä. Myös asiakas tulisi pitää tiiviisti kehityksessä mukana ja sovelluksesta toimittaa pienempiä julkaisuja lyhyellä syklillä.

Kuten muutkin menetelmät, aivan sellaisenaan Extreme Programming voi olla mahdotonta toteuttaa. Manifestista löytyy kuitenkin ketterän kehittämisen ydinajatus ja ominaispiirteet ovat toimineet ketterien menetelmien perustana.

2.3.2 Scrum

Scrum on Japanista lähtöisin oleva ketterä projektinhallinnan viitekehys, jonka periaate kuvattiin ensimmäisen kerran vuonna 1986. (Takecuchi &

Nonaka 1986.). Scrum on tapa tehdä asioita, jota myöhemmin kehitettiin ohjelmistokehitysmenetelmäksi Jeff Sutherlandin ja Ken Schwaberin toimesta. (What is Scrum? n.d.).

Scrum-viitekehyksessä sovellusta kehitetään lyhyissä, tyypillisesti noin kahden viikon mittaisissa sprinteissä. Scrumin voi ajatella sarjana vesiputousmallin toistoa, ja yhden sprintin aikana tehdään yksi vesiputous. Scrumissa on tiimejä, joille on osoitettu jokin rooli. Lisäksi jokaista tiimiä koskevat tietyt tapahtumat, tuotokset ja säännöt. Tiimit ovat

1. Tuotteen omistaja
2. Kehittäjät
3. Scrummaster

Tuotteen omistaja päättää, mitä sprintin aikana tehdään. Hänen tehtävänään on ylläpitää tuotteen kehitysjonoa (Product backlog), joka on lista asioista ja ominaisuuksista, jotka tuottavat varsinaisen arvon.

Kehittäjät ovat ryhmä asiantuntijoita, jotka ratkaisevat annetut tehtävät sprintin aikana. Kun tuotteen omistaja on päättänyt sprintin aikana toteutettavan asian, se pilkotaan vielä sopivan kokoisiksi tehtäviksi. Tehtävien työmääriä on tärkeää arvioida esimerkiksi pelaamalla suunnittelupokeria. Tehtävät muodostavat sprint kehitysjonon (Sprint backlog) Sprintin päätteeksi tulokset esitellään tuotteen omistajalle.

Scrummasterin vastuulla on, että tiimit noudattavat ja ymmärtävät Scrumin sääntöjä. Hänen on myös tehostettava ja autettava tiimien työskentelyä. Joka päivä tulisi käydä heti aamulla lyhyt palaveri, jossa tiimin jäsenet kertovat, mitä ovat tekemässä ja onko ollut ongelmia.

Vuonna 2009 tehdyn tutkimuksen mukaan Scrum on selkeästi eniten käytetty ketterä menetelmä. (West & Grant 2010.). Usein jopa ketterällä kehittämisellä

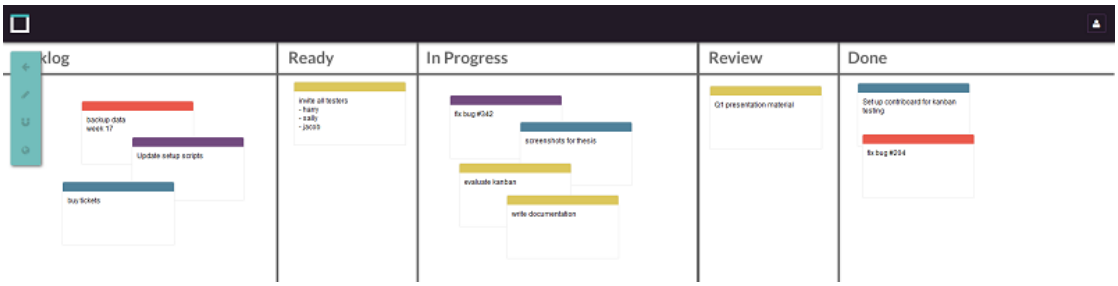
uskotaan tarkoittavan juuri Scrumia. Scrumia kuvaillaan helposti omaksuttavaksi, mutta vaikeasti taidettavaksi.

2.3.3 Kanban ja Lean

Japanilainen yhtiö Toyota kehitti 1940-luvun lopulla yksinkertaisen tavan tehostaa omaa tuotantoprosessiaan. Työntekijät laitettiin ilmaisemaan visuaalisesti toisilleen työn etenemisestä käyttämällä Kanban-taulua, josta sen hetkisten tehtävien tilanteen näki nopealla vilkaisulla. Tauluun kiinnitettyjä tehtäväkortteja liikutettiin tilanteen mukaan. Tämä nopeutti ja helpotti kommunikointia tiimien välillä. (What is Kanban? n.d.).

Kanbanin juuret juontavat itsensä Toyotan vieläkin varhaisemmasta Lean-johtamisfilosofiasta ja ajattelutavasta. Leanin keskeisin periaate oli arvon tuottaminen asiakkaalle mahdollisimman kustannustehokkaasti. (A Brief History of Lean n.d.). Kanban ei ole siis uusi keksintö, mutta ohjelmistotuotannossa sitä on sovellettu vasta 2000 luvun alusta, kun David Anderson laati varsinaisen Kanban menetelmän. (Kanban (development) n.d.). Lisäksi Lean periaatteet on sittemmin muutettu vielä ketterän ohjelmistokehityksen käytänteiksi (Poppendieck 2003.).

Kanbanin toimintaperiaate on yksinkertainen: taulu jaetaan sarakkeisiin, joista kolme yleisintä ovat tehtävät (to do), työn alla (in progress) ja tehty (done). Uudet tehtävät lisätään vasemmalle ensimmäiseen sarakkeeseen odottamaan. Jokainen sarake tulisi myös rajoittaa, kuinka monta lappua siihen saa korkeintaan laittaa. Tämä on erittäin tärkeää työn tehokkuuden kannalta. Tehtävän edetessä lappuja siirretään seuraavaan sarakkeeseen. Kanban-taulun voi toteuttaa esimerkiksi valkotaululla ja perinteisillä muistilapuilla tai modernimmin vaikka Contriboardilla (ks. Kuvio 3).



Kuvio 3. Contriboardin kehitysversiolla toteutettu Kanban-taulu

2.3.4 Scrum-ban

Scrum-ban on ketterä ohjelmistotuotannon menetelmä, joka yhdistää Scrumin ja Kanbanin. Se soveltuu pelkän Scrumin sijasta paremmin ylläpitotehtäviin, joissa sprintin aikana tulevia tehtäviä ei tiedetä etukäteen. Scrum-ban siis löysää Scrumin iteraatioita ja lisää Kanbanin visualisoinnin tehtäville. Muuten Scrum-ban ei juuri eroa normaalista Scrumista.

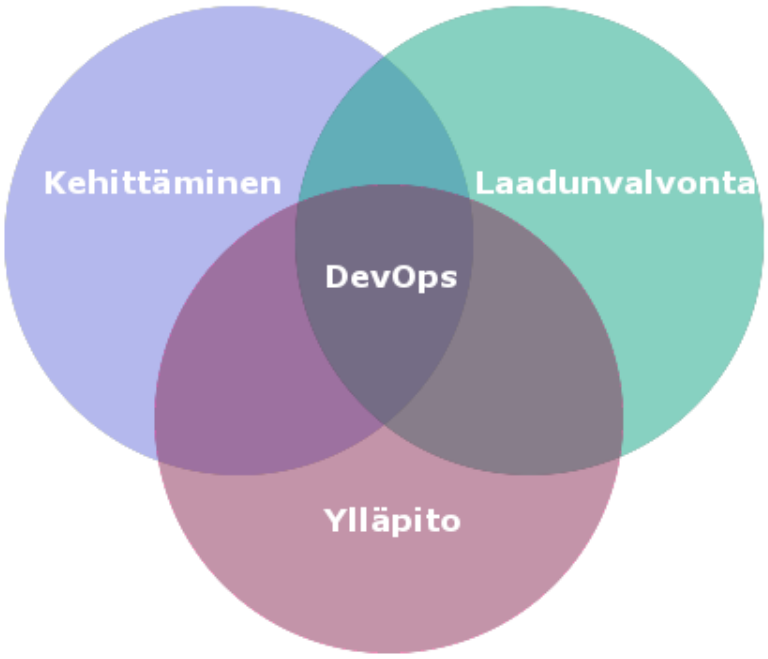
N4S@JAMK-projektissa on käytössä lähinnä Scrum-bania muistuttava työskentelytapa, joka on aikaisempien kokemusten perusteella havaittu toimivaksi.

3 Moderni palveluntuotanto pilvessä

3.1 DevOps

Englanninkielisisistä sanoista Development & Operations tuleva DevOps on alan viimeisimpiä termejä jolle ei vielä ole virallista tai tarkkaa määritelmää. DevOpsia voidaan pitää viimeisimpänä ketterän sovelluskehityksen evoluutiona, toimintatapana tehdä asioita käyttämällä automaatiotyökaluja ja minimoimalla turhuudet.

Yleisesti ottaen DevOps sotkee järjestelmäylläpitäjien ja sovelluskehittäjien rooleja (ks. kuvio 4). Tällä pyritään poistamaan kommunikaatio-ongelmia ja viivettä kehitystiimin ja ylläpitäjien välillä. Osana DevOps tiimiä onkin todennäköistä, että roolit menevät hieman päällekkäin. Sovelluskehittäjät joutuvat ylläpitotehtäviin, ja ylläpitäjätkään eivät välttämättä säästy ohjelmakoodin lukemiselta. Kun ryhmä toimii tiiviisti yhteen, kaikki tietävät tilanteen ja mihin ollaan menossa. (Wilinski 2014.).



Kuvio 4. DevOps Venn-diagrammi

3.2 Palvelutuotanto

Palvelutuotanto on järjestelmällistä toimintaa palvelun tuottamiseksi ja velvoitteiden täyttämiseksi. Se on kanssakäymistä palvelun tuottajan ja asiakkaan välillä. Tuottajan roolissa on luonnollisesti palvelun ylläpitäminen ja tuen antaminen asiakkaalle.

Kun organisaatiossa laitetaan uusi tuote tuotantoon, voidaan joskus havaita niin sanottu kuolleen kissan syndrooma. Tällä tarkoitetaan tilannetta, jossa esimerkiksi toimintamenetelmissä tai tukipalveluissa on puutteita. Tuote jätetään ikään kuin oman onnensa nojaan, ja tilanteen korjaamiseksi vastuuta pompotellaan edestakaisin. Tuotetta ei pitäisi laittaa tuotantoon, jos vaikka omistusoikeuksissa tai takuussa on epäselvyyksiä.

Kuten ohjelmistotuotantomallejakin on kehitetty, myös palvelun hallintaan ja johtamiseen on olemassa viitekehyksiä. Tunnetuimpana ITIL (Information Technology Infrastructure Library) on kattava kokoelma parhaita käytänteitä IT-palvelunhallintaan. Se on maailmanlaajuisesti tunnustettu ja soveltuu kaiken kokoisille yrityksille. ITIL:n viimeisin versio 3 ottaa kantaa erityisesti palvelunäkökulmaan viidellä eri alueella:

- Palvelustrategia (Service Strategy)
- Palvelumuotoilu (Service Design)
- Palveluntransitio (Service Transition)
- Palvelutuotanto (Service Operation)
- Palvelun jatkuva kehittäminen (Continual Service Improvement)

ITIL on hyvä lähtökohta palvelustrategioiden kehittämisessä. Palvelumuotoilu ja ITIL ovat kuitenkin sen verran laajoja kokonaisuuksia, että tässä työssä niihin ei enempää mennä.

IBM:n tekemässä julkaisussa pidetään palvelunhallintaa välttämättömänä pilvipalveluissa, koska palvelu on tehtävä hyvin hallittavaksi ja näkymättömäksi asiakkaalle. (Integrated service management and cloud computing 2010.).

3.3 Virtualisointi, pilvipalvelun perusta

3.3.1 Yleistä

Virtualisointi on sinänsä laaja käsite ja se voidaan toteuttaa monella tapaa. Ominaista on kuitenkin luoda kuvitteellinen laite, laitteisto tai ympäristö, joka käyttäytyy kuin oikea vastaava. Esimerkiksi virtualisoinnin avulla voidaan yksi tietokone jakaa useammaksi, toisistaan riippumattomiksi tietokoneiksi. Yleisesti ottaen tällöin vaaditaan laitteistolta tehokas suoritin virtualisointiominaisuuksilla ja riittävä määrä keskusmuistia. Virtualisointi ei rajoitu vain samaan arkkitehtuuriin, vaan on mahdollista emuloida täysin erilaisia, kokonaisia laitearkkitehtuureja. Tällöin toteutus on vain yleensä merkittävästi oikeaa laitetta hitaampi.

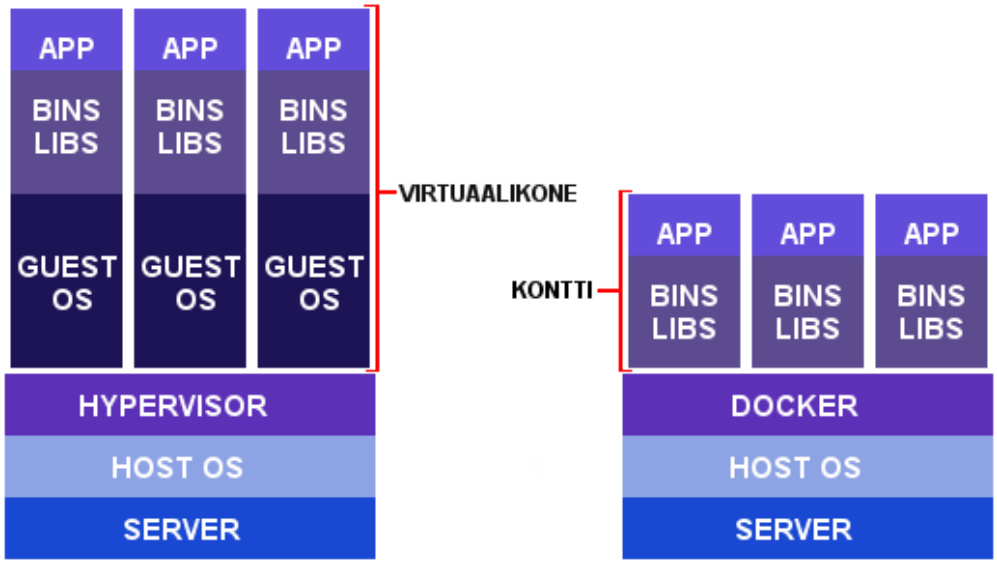
Virtualisointi on myös äärimmäisen kätevä tapa kokeilla uusia ohjelmia ja käyttöjärjestelmiä ilman erillisiä laitteita. Ei tarvitse pelätä ensisijaisen käyttöjärjestelmän sotkeutumista, kun voi vapaasti testata erillisessä, suljetussa ympäristössä. Tällöin puhutaan englanninkielisestä käsitteestä Sandboxing. Virtualisoinnilla onkin tärkeä osa sovelluskehityksessä. Sovelluksesta voi olla useita eri versioita testattavana, ja kehittäjät voivat luoda hetkessä puhtaan, sovelluksen kehitykseen vaaditun ympäristön työkaluineen. Yleinen käyttötapaus on esimerkiksi Linux käyttöjärjestelmän virtualisointi Windows käyttöjärjestelmässä, kun työkaluja ei ole natiivisti saatavilla.

Windows 8 sisältää natiivin tuen virtualisoinnille (Hyper-V), ja samoin useimmiten Linux jakelut kykenevät virtualisointiin (KVM). Puhtaasti kokeilumielessä näillä menetelmillä virtualisointi vaatii kuitenkin enemmän asiaan perehtymistä ja joissain tapauksissa erityisiä toimenpiteitä. Virtualisointi on helpompi toteuttaa käyttöjärjestelmästä riippumatta erillisellä ohjelmalla, joita on

saatavilla ilmaiseksi. Helppokäyttöisempiä ja suositumpia virtualisointisovelluksia ovat VirtualBox ja VMware. N4S@JAMK-projektissa käytettiin VirtualBoxia Contriboardin kokeilemiseen ja sen kehitysympäristön pystyttämiseen.

3.3.2 Konttiteknologia ja Docker

Konttiteknologia ja kontit ovat yksi virtualisointimenetelmä, jolla ei virtualisoida kokonaisia käyttöjärjestelmiä, vaan käyttöympäristöjä (ks. kuvio 5.). Sovellus voidaan eristää omaan virtuaaliseen käyttöympäristöön joka sisältää vain sen ajamiseen vaaditut kirjastot ja käskyt. Tämä on kokonaisen virtuaalikoneen sijasta huomattavasti kevyempi ja tehokkaampi tapa. Konttiin rakennettuja sovelluksia on helpompi jakaa ja ottaa käyttöön, kunhan isäntäkäyttöjärjestelmä ei eroa merkittävästi.



Kuvio 5. Virtuaalikoneen ja kontin ero

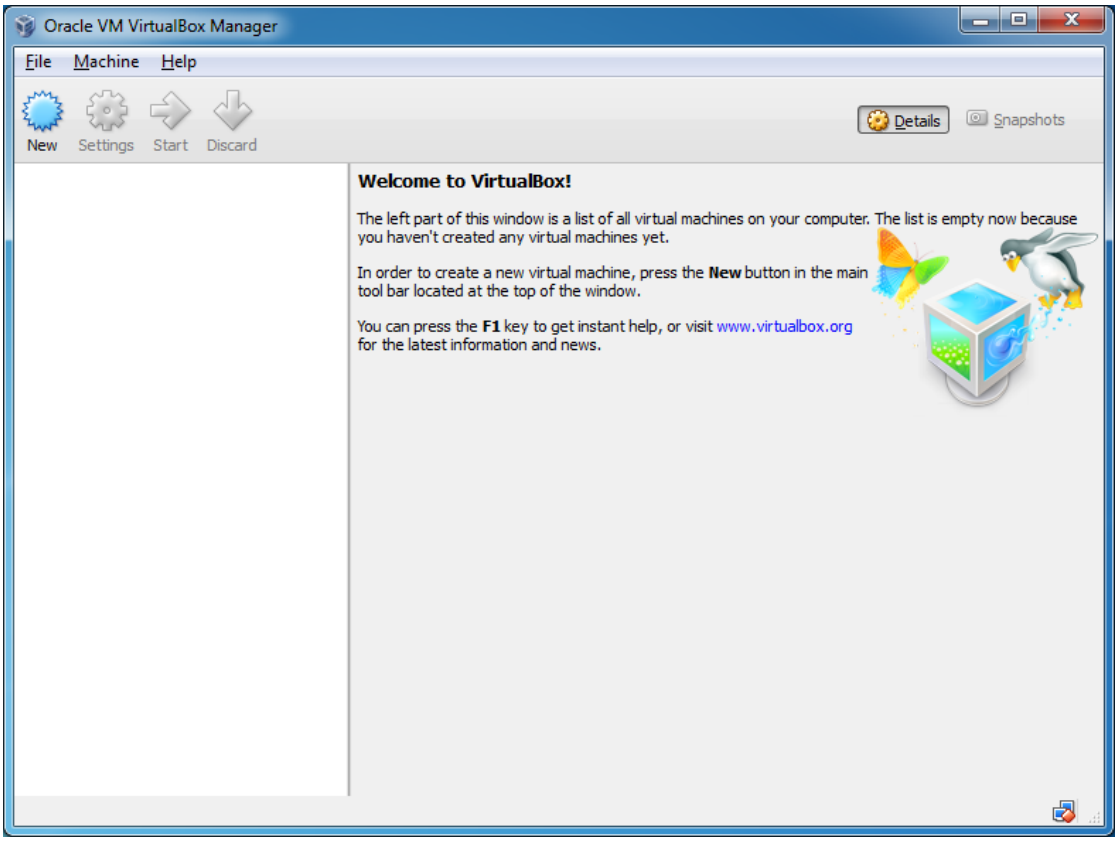
Linuxissa on ollut konttivirtualisoinnin mahdollistavia tekniikoita kuten OpenVZ ja LXC (Linux Containers) jo vuosikaudet, mutta ne ovat ehkä jääneet hieman täysvirtualisoinnin varjoon eivätkä ole saavuttaneet kovin laajaa

suosiota. Syinä on voinut olla rajalliset käyttömahdollisuudet ja epäkypsyys tuotantokäyttöön. Kehitystä on kuitenkin tapahtunut ja tilanne muuttunut niin merkittävästi, että konttiteknologiat ovat alkaneet jo kyseenalaistaa virtualisointia pilvessä. (Wheatley 2014.).

Viimeisimpänä tulokkaana konttiteknologioiden joukossa on vuonna 2013 julkaistu Docker. Sen toiminta perustui alun perin LXC:hen, joka on sittemmin korvattu Dockerin omalla libcontainer teknologialla. Libcontainerin etuna on konttien parempi standardointi ja siirrettävyys järjestelmien välillä. (Yegulalp 2014.). Docker toimii korkeamman tason työkaluna, joka tarjoaa lisäominaisuuksia eristetyn ympäristön lisäksi. Konttien sisältö määritellään erillisellä Dockerfilellä, jonka avulla rakennetaan varsinainen kuvatiedosto (image). Kuvatiedosto on ”dockeroitu” ohjelma joka sitten on ajettavissa Dockerilla. Kuvatiedostoja voi jakaa Docker Hub Registryssä tai muulla tavalla.

3.3.3 VirtualBox

VirtualBox on vuodesta 2010 eteenpäin ollut Oraclen kehittämä virtualisointialusta, joka on saatavilla Windowsille, Linuxille ja OSX:lle. Se on todella helppokäyttöinen ja monipuolinen ohjelma, jonka käyttö ei maksa mitään. Kuviossa 6 on VirtualBoxin selkeä graafinen käyttöliittymä. Uuden virtuaalikoneen luonti onnistuu helposti, koska toiminto on hyvin opastava.



Kuvio 6. Oracle VM VirtualBox Manager

3.4 Pilvipalvelut

3.4.1 Yleistä

Viime vuosina pilvipalvelut ja pilvilaskenta ovat olleet yksi IT-alaa vallanneista trendeistä. Kuluttajat ovat voineet jo käyttää pilvipalveluita pitkään kuin huomaamatta, esimerkiksi lukuisia tiedostonjakopalveluita tai kokonaan verkossa toimivia toimisto-ohjelmia. Pilvipalveluilla on kolme pääluokkaa, jotka ovat:

- SaaS – Software as a Service
- PaaS – Platform as a Service
- IaaS – Infrastructure as a Service

SaaS on kuluttajan näkökulmasta juuri näkymättömin pilvipalvelu. Asiakas hankkii käyttöoikeuden pilvessä toimivaan palveluun, jota yleensä käytetään kevyellä asiakaspääteellä, kuten selaimella. Tällöin kyseessä on todennäköisesti palveluna toimivasta sovelluksesta, jonka yhdellä instanssilla on useita käyttäjiä (eng. Multi-tenancy arkkitehtuuri). Käyttäjillä voi olla rajalliset muokkausominaisuudet esimerkiksi käyttöliittymän personointiin.

PaaS on enemmän kehittäjille tarkoitettu, pilvessä toimiva kehitysalusta. Se voi olla valmiiksi asennettu ja konfiguroitu käyttöympäristö. Tämä käyttöympäristö on asiakkaan käytettävissä ja hyödynnettävissä tarpeen mukaan.

IaaS on näistä tasoista kaikkein alimpana, sillä se kattaa varsinaisen alustan, kuten virtuaalikoneet, levytilan ja verkotuksen. Asiakas voi vapaasti rakentaa tarvittavan ympäristön pilveen. Enää ei ole välttämätöntä hankkia omaa kallista fyysistä laitteistoa, jonka ylläpito voi olla myös hyvin kallista ja aikaa vievää. IaaS pilvipalvelusta voidaan hankkia tarvittava määrä koneita ja maksaa niistä vain käytön mukaan. Laiterikot ovat toki mahdollisia myös pilvipalveluissa, joten toimintojen redundanssi ja tietojen varmuuskopiointi on huolehdittava.

Eri pilvipalvelun tarjoajia on markkinoilla lukematon määrä, mutta tämän työn puitteissa tarkastellaan kahta tunnettua - DigitalOceania ja Amazon EC2:sta. Näitä kahta pilvipalvelua on käytetty onnistuneesti Contriboardin- ja Corolla kehitysympäristön hostaamiseen. Pilvipalveluissa on kuitenkin selviä eroja, osittain hyviä ja huonoja puolia.

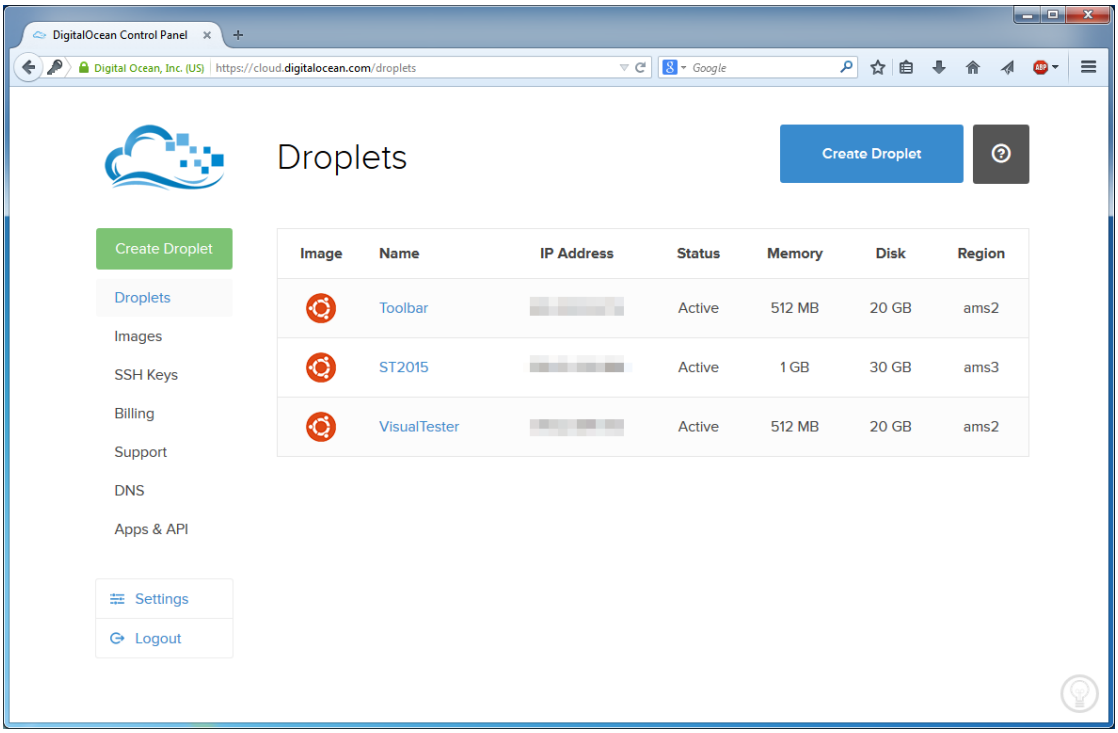
3.4.2 DigitalOcean

DigitalOcean on vuonna 2011 perustettu IaaS pilvipalveluntarjoaja, jonka pääkonttori sijaitsee New Yorkissa. Se tarjoaa koneita datakeskuksista, jotka sijait-

sevat New Yorkissa, San Franciscossa, Lontoossa ja Singaporessa. DigitalOcean tarjoaa edullisimmillaan yhden koneinstanssin n. viiden dollarin kuukausihintaan. DigitalOceanin palvelininstansseja kutsutaan Dropleteiksi. Dropletti on virtualisoitu Linux, jonka käyttöjärjestelmän voi valita muutamasta eri vaihtoehdosta. Valittavana on mm. Ubuntu, CentOS ja Fedora. Luomisvaiheessa on myös mahdollista valita joitain yleisiä ohjelmistoja esiasennettuina, kuten GitLab, Docker, LAMP ja useita muita. Windows palvelimia ei ole mahdollista tehdä DigitalOceanissa. Dropleteissa on oletuksena nopea SSD kiintolevy ja vähintään 512 Mt muistia. Web-hallintapaneeli on selkeä ja helppokäyttöinen, joten virhevalintoja ei pääse juurikaan syntymään. DigitalOcean tarjoaa koneiden luontiin ja hallintaan API rajapinnan, jota voi hyödyntää kun koneita pitää tehdä automatisoidusti.

DigitalOcean mahdollistaa myös virtuaalisen lähiverkon koneiden välille, jota mainostetaan yksityisenä verkkona, mutta tehtyjen lisätutkimusten ja tietoturvakartoituksen mukaan näin ei kuitenkaan ole. Sillä yksityiseen verkkoon liitetyt koneet voivat vapaasti keskustella toisten samassa datakeskuksessa olevien koneiden kanssa, jotka ovat jonkun toisen asiakkaan käytössä. Tämä on huomioitava tietoturvassa ja säädettävä palomuuriasetukset niin, että sisäverkon liikenne on varmasti vain haluttujen koneiden välillä.

Kuviossa 7 on DigitalOceanin yksinkertainen käyttöliittymä.



Kuvio 7. DigitalOceanin hallintapaneeli

3.4.3 Amazon EC2

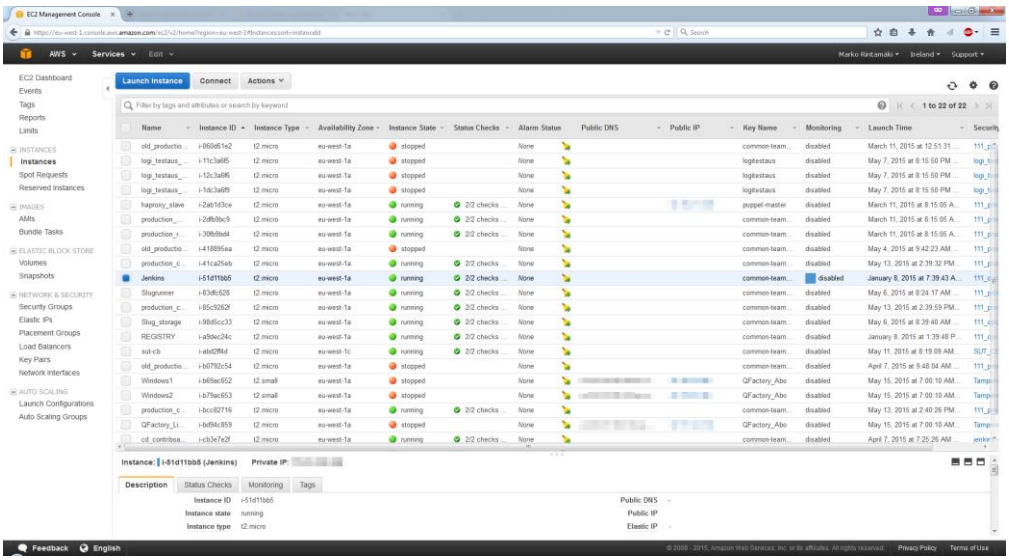
Amazon on kiistatta yksi suurimmista pilvipalvelun tarjoajista. Vuonna 2006 lanseerattu Amazon Web Services (AWS) ja siihen kuuluva Amazon Elastic Compute Cloud (EC2) on monella tapaa pilvipalveluiden pioneeri ja edelläkävijä. Maantieteellisesti eri alueille sijoitetut konesalit toimivat toisistaan riippumattomasti, ja ne on vielä jaettu pienempiin yksiköihin. Euroopassa konesaleja on Frankfurtissa ja Irlannissa.

Kuten DigitalOcean, EC2 on IaaS-tyyppinen palvelu, joka tarjoaa todella monipuoliset ominaisuudet ja mahdollisuudet oman infrastruktuurin suunnitteluun ja toteuttamiseen. Virtuaalikoneiden käyttöjärjestelmäksi voi valita yleisimpien Linux distrojen lisäksi myös Windows Serverin tai jopa kokonaan kustomoituja konfiguraatiopaketteja (Amazon Machine Image, AMI). EC2:n hinnoittelumalli on käyttöveloitteinen, joten lasku kertyy virtuaalikoneen tyy-

pin ja sen käytön mukaan. Amazon tarjoaa myös rajapinnan virtuaalikoneiden hallintaan, mikä on tietysti hallittavuuden ja automaation kannalta tärkeää.

EC2 mahdollistaa perusteellisen verkkokonfiguraation ja jopa kehittyneen kuormantasaajan, joka osaa ohjata liikennettä eri instansseille. Kuviossa 8 näkyvä Amazon EC2:sen käyttöliittymä onkin jo huomattavasti DigitalOceania informatiivisempi ja kattavampi. EC2 tarjoaa sen verran kattavat ominaisuudet, että niiden puolesta se on hyvin soveltuva alusta Contriboardille.

Amazon EC2:sta ei juuri löydä puutteita, mutta hinnoittelumalliin on syytä tutustua hyvin. Pelkkien koneiden lisäksi jotkin ominaisuudet lisäävät kustannuksia melkein huomaamatta, vaikkakaan ei merkittävästi.



Kuvio 8. Amazon EC2 hallintapaneeli

3.4.4 Rackspace

Rackspace eroaa IaaS palveluntarjoajista siinä mielessä, että se tarjoaa ensisijaisesti kokonaan itsehallittavan palvelun sijaan erityisiä palvelutasoja, jotka vähentävät käyttäjälle jääviä huolenaiheita. Tasot ovat seuraavat:

Managed Infrastructure: Rackspacen insinöörit konsultoivat oikeanlaisen infrastruktuurin rakentamisessa, tietoturvassa ja käyttöönotossa.

Managed Operations (SysOps): Edellisen tason lisäksi tukea saa järjestelmien seurantaan ja ylläpitotehtäviin kuten päivityksiin.

Managed Operations (DevOps): Rackspace tekee yhteistyötä kehittäjien kanssa ja auttaa palvelun tuotannossa. Automaatiolla tehdään konfiguraationhallinta ja nopeutetaan käyttöönottoa.

(Rackspace Managed Cloud Services—More than just infrastructure n.d).

Rackspacen soveltuvuus Contriboardin kehittämiseen ja osaksi Corolla tuotekehitysketjua on hieman kyseenalaista, sillä ylläolevat palvelutasot koostuvat juuri niistä asioista, mitä N4S@JAMK-projektissa on tutkittu ja ratkaistu, mutta puhtaasti omassa hallinnassa olevalla infrastruktuurilla.

3.5 Jatkuva integraatio - Continuous Integration

Jatkuva integraatio on käytäntö, jossa kehittäjien tekemä työ yhdistetään. Kun ohjelmakoodiin on tehty muutokset, se on ohjelmointikielestä riippuen vielä erikseen buildattava, eli käännettävä ajettavaksi binääriksi. Prosessiin kuuluu myös esimerkiksi yksikkötestaus ja koodin katselmointi. Jos testauksessa ei ilmene ongelmia, voidaan lopuksi tehdä käyttöönottoa valmistelevia vaiheita, kuten asennuspaketin tekeminen ja asennus haluttuun ympäristöön. Tämä on pitkä prosessi, johon on olemassa työkaluja.

Automaatio tehdään Continuous Integration–työkalulla, joka voi olla itse asennettu ja hostattu, tai hankittu palvelu. Ketju voi toimia niin, että kun ohjelmakoodia muutetaan ja se tallennetaan versionhallintajärjestelmään, siitä lähtee tieto CI-työkalulle, joka alkaa tekemään määritettyjä toimenpiteitä.

3.5.1 Versionhallinta

Ohjelmistokehityksessä työn tuloksena syntyy digitaalista materiaalia, tiedostoja joihin lukeutuvat mm. varsinaiset lähdekoodit, dokumentaatio, grafiikka, testit ja muut riippuvuudet. Eli siis kaikki projektiin kuuluvat tiedostot sähköisessä muodossa. Kun samojen tiedostojen kanssa tekee töitä vielä useampi henkilö, on tiedostojen jakaminen hoidettava jotenkin ja estettävä mahdollinen ylikirjoitus. Lisäksi usein on pidettävä erikseen tuotanto- ja kehitysversiot, joihin voi kuulua useita korjaus- ja kehityshaaroja. Kaikkeen edeltävään on ratkaisuna versionhallintatyökalu, joka tallentaa tiedostojen muutoshistoriaa ja revisioita.

Versionhallinta on välttämätön, sillä sen tuomat edut ovat äärettömän hyödyllisiä ja tärkeitä vaikka kehittäjiä ei olisikaan kuin yksi. Työn edistyessä tuotos tallennetaan repositoryyn (suom. tietovarastoon), jossa se on varmennettu ja muiden saatavilla. Versionhallinta voi olla keskitettyä tai hajautettua. Keskitetty versionhallinta (Centralized Version Control System, CVCS) kattaa yhden repositoryn, josta kaikki lataavat työkopion, tiedostojen sen hetkisen revision. Hajautetussa versionhallinnassa (Distributed Version Control System, DVCS) taas jokainen voi kloonata itselleen kokonaisen repositoryn historiatietoineen, joka voi toimia itsenäisesti.

Erilaisia versionhallintatyökaluja on paljon, mutta kaksi selvästi suosituinta ovat Git ja SVN (Subversion). Molemmat työkalut ovat avointa lähdekoodia ja tuettuina useilla käyttöjärjestelmillä. SVN on vanhempi, vuonna 2000 julkaistu versionhallinta, joka on tilastojen mukaan vielä toistaiseksi eniten käytetty versionhallinta. (O'Grady, 2013.). Noin viisi vuotta nuorempi Git on kuitenkin ollut todella vahvassa nosteessa ja kaventanut vuosi vuodelta SVN:n

johtoasemaa. Luvussa 3.6.2 tutustutaan tarkemmin Git:iin, koska se on N4S@JAMK-projektin käyttämä versionhallinta.

3.5.2 Git

Git on alkujaan Linuxinkin isänä pidetyn Linus Torvaldsin käsialaa. Alun perin Linux ytimeen tehtyjen korjausten (eng. patch) hallintaan luotu Git julkaistiin vuonna 2005. (Getting Started – A Short History of Git 2014.). Se on tyypiltään hajautettu versionhallintajärjestelmä ja sittemmin noussut yhdeksi suosituimmista ja monipuolisimmista versionhallintatyökaluista. Se on saatavilla ilmaiseksi Windows, Linux ja OSX käyttöjärjestelmille.

Git-versionhallinnalla on vahva sidos GitHub-verkkopalveluun, joka on vuonna 2008 avattu julkinen Git repositorioiden hostauspalvelu. Repositorioiden lisäksi GitHubissa on projektin hallintaa helpottavia ominaisuuksia kuten ongelmien ja vikojen seurantajärjestelmä (Issue Tracker) Kehittäjien suosimalla GitHubilla on jo miljoonia käyttäjiä ja repositoryjä, mikä on varmasti vaikuttanut Gitin yleistymiseen. GitHubin käyttö on ilmaista julkisille ja avoimen lähdekoodin projekteille, jolloin repository on kenen tahansa kloonattavissa. Jos kuitenkin projektin repository halutaan yksityiseksi, GitHub tarjoaa kuukausimaksullisen tilausmallin, jonka hinta määräytyy yksityisten repositoryjen määrän mukaan.

GitHubia ei ole kuitenkaan pakko käyttää, sillä Git-repositoryt toimivat mainiosti myös vain paikallisesti käytettynä. Koska yleensä kuitenkin repositoryyn pitää olla useammalla käyttäjällä pääsy, sen hostaaminen ja jakaminen on mahdollista toteuttaa itse. Riittää, kun palvelinkoneeseen on asennettu Git-ohjelmisto, SSH-palvelin ja käyttäjillä on käyttöoikeus repositoryyn SSH-yhteyden kautta. Jos haetaan vielä enemmän GitHubin kaltaista ratkaisua, on saatavilla GitLab repositorynhallintasovellus. GitLabia käytettiin aluksi

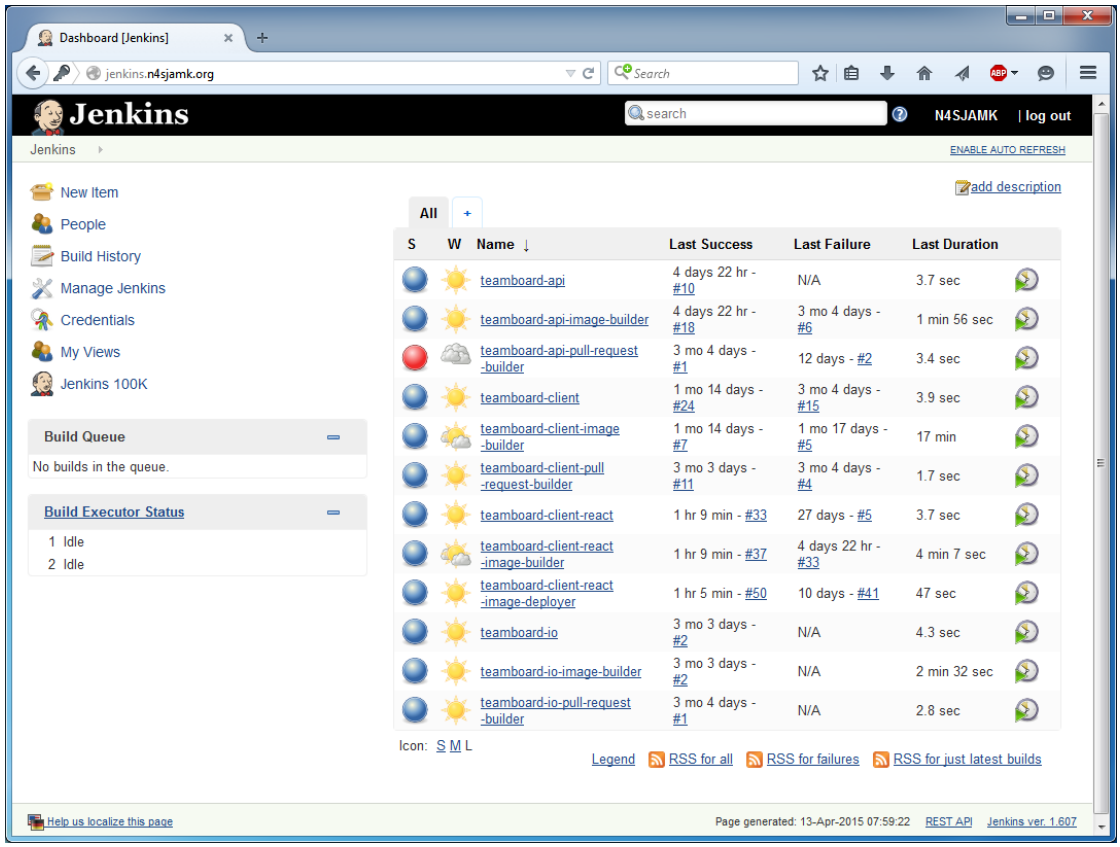
N4S@JAMK projektissa, mutta myöhemmin siirryttiin GitHubiin paremman julkisen näkyvyyden takia. Lisäksi itse toteutetuissa järjestelmissä on huolehdittava ylläpidosta ja tietojen varmennuksesta. Koska Contriboardin Corolla tuotantoketjussa käytetään GitHub palvelua, sinne pitää olla käyttäjätili rekisteröitynä. Myös Gitin peruskomennot pitää hallita.

3.5.3 Jenkins

Jenkins on yksi suosituimmista ja eniten käytetyistä Continuous Integration työkaluista. Se on avointa lähdekoodia, toteutettu Javalla ja toimii sen avulla hyvin laajalla laitekannalla. Jenkinsin perusta on lähtöisin Hudson–nimisestä CI-ohjelmasta, josta sen kehitys haarautettiin (fork) erilleen silloisen omistajan, Oraclen ja kehittäjien erimielisyyksien takia. (Who's driving this thing? 2010.).

Jenkins on konfiguroitavissa erittäin laajamittaisesti. Sillä voi tehdä helposti useasti toistuvia toimintoja, jotka usein koostuvat ajettavista shell komennoista. Jenkinsin toimintoja on mahdollista lisätä asentamalla laajennuksia (plugin), joita on saatavilla erittäin paljon. Yksi oleellinen laajennus on GitHub-plugin, joka mahdollistaa Jenkins-jobien automaattisen käynnistymisen kun GitHub-repositoriossa tapahtuu muutos. (GitHub Plugin n.d.).

Jenkinsillä on helppokäyttöinen WEB-käyttöliittymä (ks. Kuvio 9), josta voidaan säätää jenkinsin asetuksia ja hallita jobeja. Jobien tilan voi nopeasti tarkistaa etusivulta, josta näkee heti, jos jossain jobissa on tapahtunut virhe. Jenkinsiä on mahdollista hallita myös komentoriviltä.



Kuvio 9. Jenkinsin WEB-Käyttöliittymä

3.6 Konfiguraationhallintatyökaluja

3.6.1 Orkestrointi ja provisiointi

Orkestrointi (Orchestration) ja provisiointi (Provisioning) eivät ole kovin vakiintuneita sanoja, mutta niihin törmää helposti kun alkaa tutkimaan konfiguraationhallintatyökaluja. Tietotekniikassa sanalla provisiointi tarkoitetaan esimerkiksi virtuaalikoneen valmistelua, johon kuuluu sen käynnistäminen ja asetusten alustaminen määritellyin arvoihin. Orkestrointi puolestaan on automatisoitu toimenpide, jolla ohjataan useita koneita hallitusti. Määritelty konfi-

guraatio ajetaan kohteisiin samanaikaisesti, jotka ovat jo provisioitu käyttövaliiksi. Koska konfiguraatiotyökalut lähes poikkeuksetta hoitavat molempia asioita, niiden raja hieman hämärtyy.

Konfiguraatiotyökalut ovat korvaamaton apuväline, kun on hallittava laajaa konekantaa pilvipalvelussa. Työkalut voi vielä jakaa toimintatavaltaan kahteen eri tyyppiin; push ja pull. Push tyyppisessä työkalussa on yksi tai useampi pääkone, joka käskyttää määrättyjä koneita (nodeja) ja ajaa halutun konfiguraation niihin. Pull tyyppinen työkalu toimii toisinpäin, eli hallinnan alaiset koneet hakevat konfiguraationsa pääkoneelta.

Työkalusta riippuen toimenpide voidaan suorittaa joko käsin tarvittaessa, ajastetusti tai muuten automatisoidusti. Luvuissa 3.6.2-3.6.4 on tarkempi katsaus muutamaan työkaluun, jotka esitutkimuksen mukaan soveltuvat osaksi Contriboardin kehitystä.

3.6.2 Fabric

Fabric on Pythonilla toteutettu kirjasto ja komentorivipohjainen etähallintatyökalu. Se käyttää SSH:ta, jonka avulla se pystyy ajamaan komentoja kohdekoneessa ilman lisäasennuksia. Se on siis puhtaasti push tyyppinen työkalu ja kätevä vaikka toisen konfiguraatiotyökalun esiasentamiseen.

Fabric käyttää ns. fabfilejä, jotka määrittävät mitä komentoja ja missäkin kohteessa ajetaan. Fabfile on syntaksiltaan helposti omaksuttavaa Pythonia.

3.6.3 Puppet

Puppet on noin 10 vuoden ikään ennättänyt konfiguraatiotyökalu, jolla on laaja käyttäjäkunta. Se on yksi vanhimmista automaatiokehyksistä. Puppet on tehty Ruby ohjelmointikielellä, ja varsinaiset konfiguraatiot (moduulit) määritellään käyttämällä Puppetin omaa DSL:ää (Domain Specific Language).

Puppetin toiminta vaatii, että se on asennettuna kaikkiin hallittaviin koneisiin, joka puolestaan edellyttää Ruby ympäristöä ja Puppet agentin. Nämä lisäävät muistin käyttöä koneessa merkittävästi, joka voi olla ongelma pienitehoisissa virtuaalikoneissa. Lisäksi koneiden välinen verkotus täytyy tehdä niin, että koneet löytävät toisensa isäntänimen (Hostname) perusteella. Jos pilvipalveluntarjoaja ei aseta koneille täydellistä verkkonimeä (Fully Qualified Domain Name, FQDN), on käytettävä omaa nimipalvelinta.

Puppet voi olla sekä push että pull tyyppinen automaatiotyökalu. Oletusarvoisesti Puppet agentit kysyvät pääkoneesta (master) konfiguraatiotietoja 30 minuutin välein. Jos masterin lähettämä konfiguraatio eroaa viimeksi ajetusta, agent ajaa uuden konfiguraation. On myös mahdollista käskeä Puppet agentteja hakemaan konfiguraatio pyynnöstä, mutta se vaatii mcollectiven ja activemqn asentamisen, jotka lisäävät entisestään Puppetin korkeahkoja laiteresurssivaatimuksia.

3.6.4 Ansible

Ansible on helppokäyttöinen automaatiotyökalu, joka julkaistiin 2012. Se on vielä suhteellisen uusi, mutta jo erittäin tuotantokelpoinen. Ansible on toteutettu pythonilla, joka vaaditaan sekä isäntä että kohdekoneessa. Jos pythonia ei ole valmiina kohdekoneessa, Ansible tarjoaa raw-moduulin jonka avulla pythonin asennus onnistuu pelkällä SSH:lla. Muutoinkin Ansible toimii SSH:n yli push periaattella, eikä kohdekoneeseen tarvita erillistä agent ohjelmistoa kuten Puppetin tapauksessa. (Ansible Installation n.d.).

Ansiblelle kirjoitetaan ns. playbookeja, jotka sisältävät rooleja ja taskeja. Kun playbook ajetaan ansiblella, se ajaa yhteen tai useampaan kohdekoneeseen vain niille määrätyt roolit. Rooli koostuu eri tehtävistä (task), jotka voivat olla esimerkiksi paketin asennus ja palvelun käynnistäminen. Kohdekoneet voi

alustavasti nimetä ja ryhmitellä erillisessä konfiguraatiotiedostossa /etc/ansible/hosts

Ansiblella on kattava valikoima ydin moduuleja (Core modules), jotka sisältävät kaikki perustoiminnot. Lisäksi Ansiblea voidaan laajentaa omilla moduuleilla.

3.7 Continuous delivery

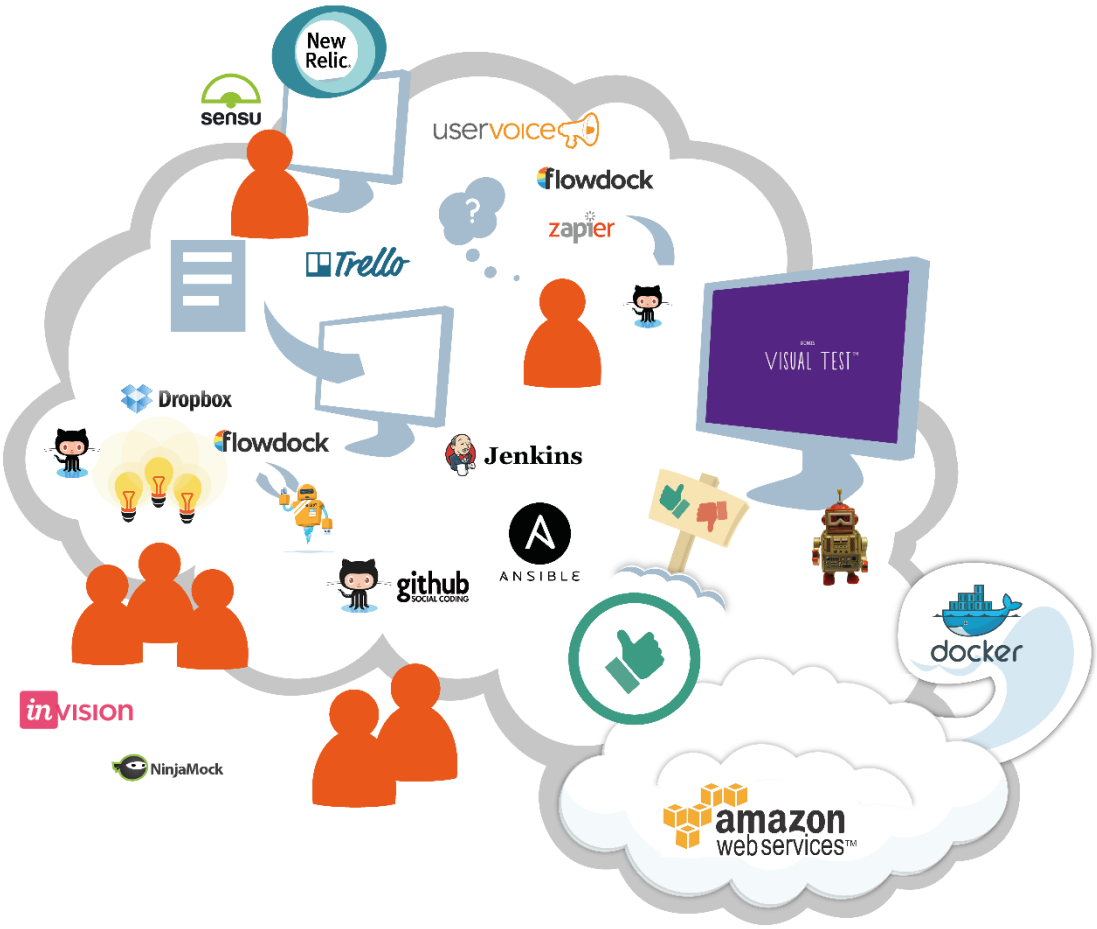
Continuous delivery (CD) on käytäntö, jossa ohjelmisto on toimitettavissa käyttäjälle milloin tahansa. Toimitusta ei kuitenkaan tehdä automaattisesti, vaan haluttaessa asiakkaan pyynnöstä. Tuote voidaan laittaa esimerkiksi suoraan tuotantoon tai ns. staging vaiheeseen, jossa tuotantokelpoisuus varmistetaan. Tässä vaiheessa testaus on enemmänkin laadun varmistamista tai hyväksyntätestausta. Kun sovellusta testaa oikea käyttäjä, havaitaan paremmin bisnes logiikassa olevia ongelmia ja muita suunnitteluvirheitä, joita yksikkötestauksella ei löydetä. (Fowler 2013.).

3.8 Continuous deployment

Continuous deploymentin ja continuous deliveryn ero on hieman epäselvä, mutta yleisesti ottaen Continuous deploymentissä tuote julkaistaan tuotantoonkin automaattisesti. Tuotteesta tehdään siis jatkuvasti julkaisuja ja asiakkaalla on aina viimeisin versio käytettävissä. Koko ketjussa on testein varmistettava, että tuotantoon ei mene täysin toimimatonta versiota. Continuous deployment ja delivery nojaa continuous integrationiin, jossa buildaus ja testaus tapahtuu kehitysympäristössä ennen tuotantoon siirtoa. (Fowler 2013.). Mikäli tuotantoon menneessä versiossa kuitenkin havaitaan ongelmia, on hyvä pystyä tekemään palautus aiempaan versioon helposti (rollback).

3.9 Tuotantoketju Corolla

Corolla on nimitys Contriboardin tuotantoketjulle ja siihen kuuluvalle kehitysympäristöille (ks. Kuvio 10). Vaikka se on suunniteltu Contriboardin kehittämistä varten, siinä on käytetty hyvin yleisiä työkaluja ja palveluita, joita voi käyttää muihinkin tuotekehitystarkoituksiin. Suurin osa Corollan työkaluista on ulkopuolisia palveluita, joilla tehdään vaikkapa tehtävien suunnittelua tai kerätään käyttäjätilastoja. Kaikki työkalut ovat pitkälti valinnaisia ja vaihdettavissa toisiin halutessa.



Kuvio 10. Corolla-ketjun sisältö

Corolla tuotantoketju käsittää myös muutamia itse asennettuja työkaluja, kuten Jenkins ja Ansible. Nämä työkalut ovat tärkeässä roolissa automaation

kannalta ja ne on asennettava käsin tai esimerkiksi Fabricilla. Corolla tuotantoketjuun liittyy vielä paikalliset kehitysympäristöt, joiden luontia käsitellään luvussa 5.2.6.

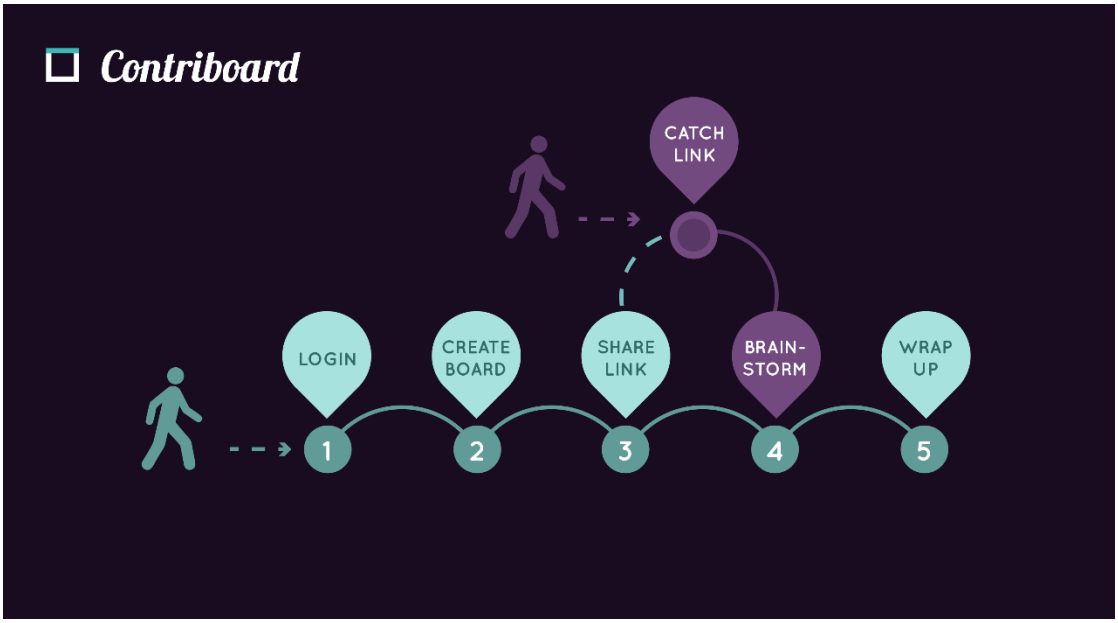
4 Contriboard

4.1 Yleistä

Contriboard tunnettiin aiemmin Teamboardina. Teamboard oli osa FreeNest tuotealustaa, joka on kokoelma avoimen lähdekoodin projektinhallinta ja tuotekehitystyökaluja. Teamboardia ja FreeNestiä kehitettiin vuosina 2012-2013 Jyväskylän Ammattikorkeakoulun järjestämissä kesätehtaissa, joka kuului Digilen Cloud Software Finland ohjelmaan. FreeNestin kehitys oli vilkkainta kesätehtaissa, joissa opiskelijaryhmät toteuttivat vaadittuja toimintoja. FreeNestin kehitys on ohjelman päätyttyä pysähtynyt käytännössä kokonaan, mutta Teamboardille nähtiin yhä tarvetta. FreeNestiin kuulunut Teamboard haluttiin irrottaa itsenäiseksi tuotteeksi, mutta sen tekninen toteutus ei vastannut nykyisiä standardeja.

Need 4 Speed ohjelman alkaessa N4S@JAMK-projektissa alettiin kehittämään Teamboardia kokonaan alusta. Aluksi työnimenä oli Teamboard 2.0, mutta koska Teamboard oli sekoitettavissa jo olemassa olevaan tuotteeseen, nimi vaihdettiin Contriboardiksi. Contriboardissa on samat toiminot kuin TeamBoardissa, mutta sen käytöstä pyritään tekemään suoraviivaisempaa ja helpompaa. Contriboard on web-pohjainen työkalu, jonka ydintoiminnallisuus on digitaalisten muistilappujen eli ns. tikettien tekeminen ja jakaminen muiden käyttäjien kesken. Contriboardia voi käyttää apuna esimerkiksi ideointipalavereissa, tehtävien suunnittelussa tai jonkin tapahtuman ohjauksessa.

Teknologiavalintoihin on kiinnitetty huomiota sen mukaan, että se toimisi hyvin useilla eri laitteilla. Kuviossa 11 esitetään Contriboardin käyttövaiheet brainstorming-työkaluna.



Kuvio 11. Contriboardin käyttövaiheet

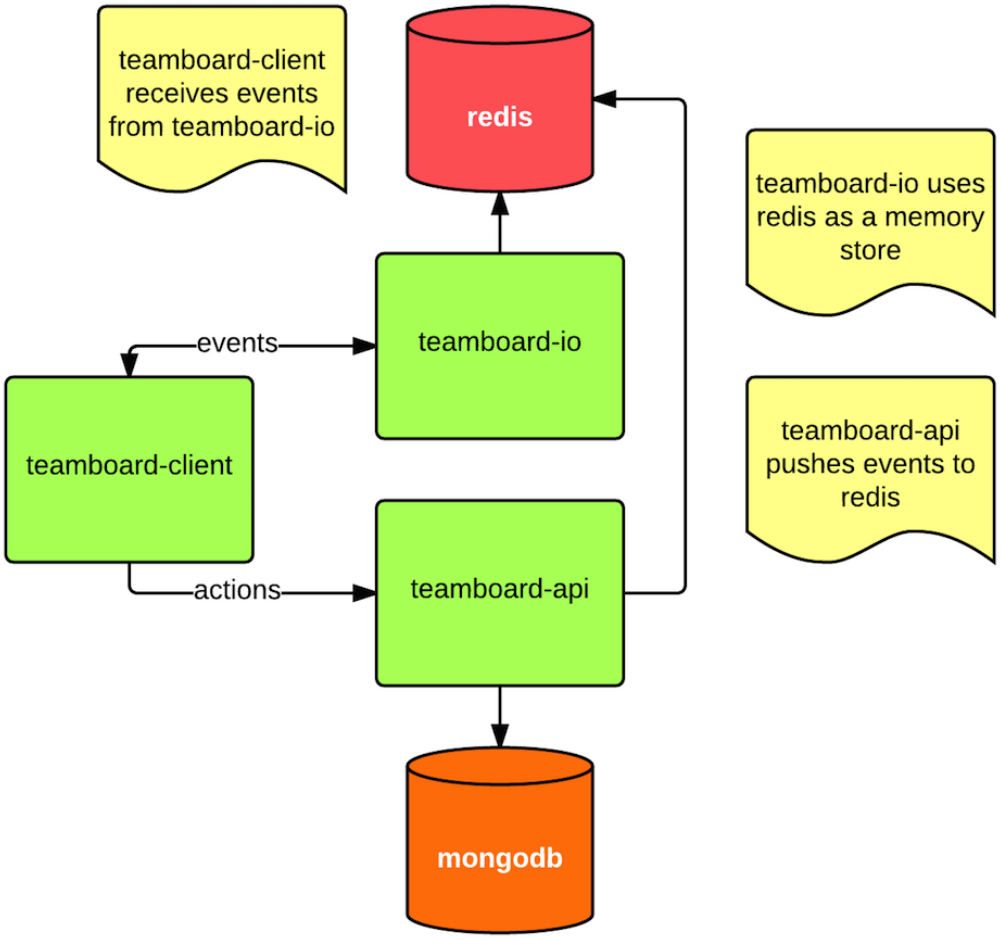
Contriboard on avointa lähdekoodia, joka on saatavilla GitHubista. Contriboardin virallinen tuotantoversio on toistaiseksi käytettävissä ilmaisena palveluna. On kuitenkin mahdollista asentaa ja hostata Contriboard kokonaan itse.

4.2 Arkkitehtuuri

Contriboard on suunniteltu alusta asti skaalautuvaksi ja hyvin tehokkaaksi. Yksi Contriboardin vaatimuskriteereistä oli, että palvelun tulee kestää tuhansia samanaikaisia käyttäjiä. Tämä on pyritty tekemään mahdolliseksi jakamalla Contriboardin toiminnollisuutta eri komponentteihin, joiden määrää voidaan muuttaa dynaamisesti käyttötarpeen mukaan. (ks. kuvio 12).

Jokainen komponentti on asennettavissa hajautetusti omiin virtuaalikoneisiinsa, tai yhdistetysti samaan koneeseen. Laitevaatimukset ovat testikäyttöä

ja kehittämistä varten maltilliset, jotka minimissään yksiytiminen suoritin, 2GB muistia ja Ubuntu 14.04. Käyttäjämäärien lisääntyessä vaatimukset kasvavat ja silloin on otettava käyttöön kuormantasausmenetelmät. Contriboardin kuormantasausta käsitellään tarkemmin työssä Kuormantasauksen soveltaminen Teamboard 2.0 –palvelussa. (Viinikanoja 2014.).



Kuvio 12. Contriboardin arkkitehtuuri

4.2.1 API

Contriboard API (Application Program Interface) on rajapinta koko Contriboardin toimintojen ytimeen. Kaikki käyttäjätunnistautuminen eteenpäin tapahtuu API:n kautta, joten se on kriittisin komponentti. Se on toteutettu Express.js frameworkilla. Sillä on riippuvuudet MongoDB tietokantaan ja Redikseen jotka täytyy olla toiminnassa ennen APIa. Contriboard API:iin kohdistuu eniten rasitusta, ja käynnistämällä lisää API-instansseja voidaan kasvattaa yhtäaikaisten käyttäjien maksimimäärää.

4.2.2 IO

IO on vastuussa Contriboardissa tapahtuvasta socket liikenteestä. Socketien avulla Contriboardista on voitu tehdä reaaliaikainen ja tehokas. Käyttäjät näkevät heti, kun Contriboardissa tapahtuu jotain, kuten tikettien liikkuminen. IO on riippuvainen API:sta ja Redis välimuistista.

4.2.3 Client

Client on Contriboardin asiakaspää, eli selainnäkymä. Se on toteutettu käyttäen viimeisimpiä web-standardeja ja kirjastoja. Se on tarkoitettu käytettäväksi nykyaikaisilla selaimilla ja laitteilla. Client liikennöi asiakkaan selaimen lisäksi myös API:n ja IO:n välillä.

4.2.4 MongoDB

MongoDB on nosteessa oleva NoSQL-tietokanta, jota käytetään usein Node.JS sovellusten kanssa. MongoDB:ssä ei ole tauluja, vaan kokoelmia joiden tietorakenne on dynaaminen.

4.2.5 REDIS

Redis on avain-arvo tyyppinen välimuisti jota käytetään Contriboardin socket yhteyksissä. Redis on avainasemassa skaalautuvuuden kannalta.

5 Toteutus

5.1 Pilvipalvelun ja työkalujen valinta

Osa työssä käytetyistä ohjelmista oli valittu jo ennakkoon aiempien kokemusten perusteella, eikä niille katsottu olevan tarpeellista harkita vaihtoehtoja. Esimerkiksi Git ja Jenkins ovat hyvin vakiintuneita ohjelmia. Lisäksi Contriboard on kehitetty toimivaksi Ubuntu 14.04 käyttöjärjestelmällä, joten sitä suosittiin.

IaaS pilvipalveluntarjoajista Contriboardille oli valittavana joko DigitalOcean tai Amazon EC2. Molempia palveluita käytettiin, mutta Amazon EC2 osoittautui paremmaksi. Syitä tähän olivat monipuolisempi ja yksityinen verkotusmahdollisuus koneiden välille.

5.2 Contriboardin asennus

5.2.1 Yleistä

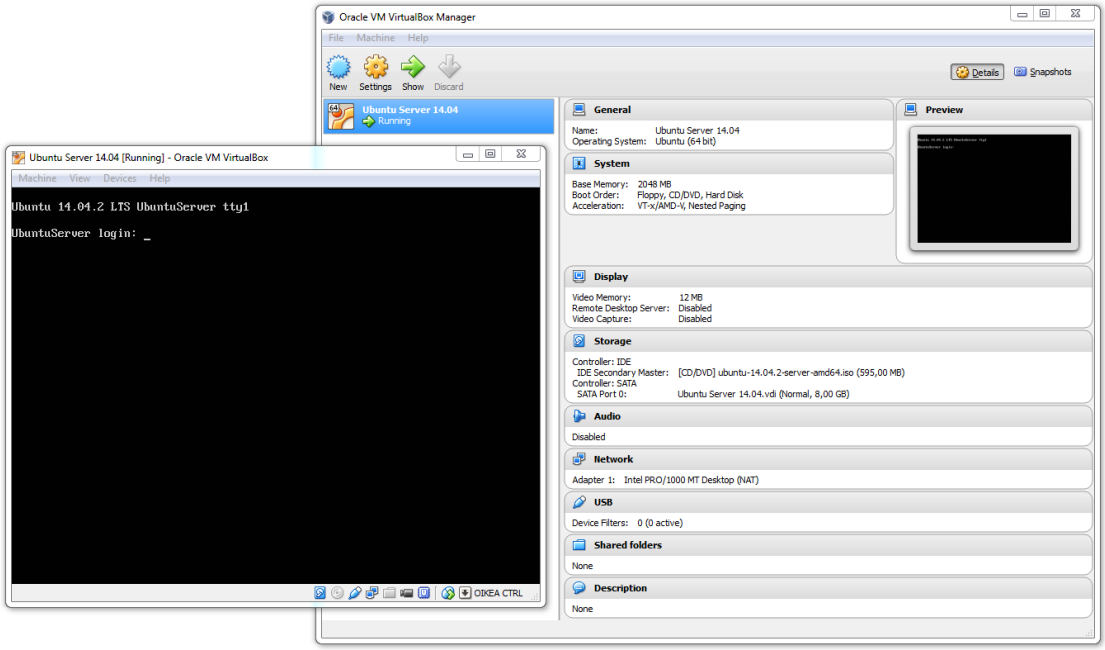
Yksi selvityksen kohde oli kuinka Contriboardin asennus tehdään helpoiten. Toimiva Contriboardin asennus paikallisesti on sinänsä myös esivaatimuksena kehitysympäristölle. Asentaminen käsin on mahdollista, mutta perusteellista asennusohjetta ei ole. Lisäksi jokaisen komponentin asentaminen käsin on työlästä. Käsin tehty asennus on sikäli kuitenkin varteen otettava vaihtoehto, jos ei halua tai voi käyttää lisätyökaluja. Lisäksi se auttaa ymmärtämään

komponenttien toimintaa ja suhteita toisiinsa. Automatisoitua ratkaisua oli haettava konfiguraatiotyökaluista. Etusijalla oli Puppet, koska siitä oli aiemmin tehty kattava esitutkimus ja lisäkokemukset katsottiin eduksi myös toisiin N4S@JAMKin projekteihin.

Lisäksi Contriboardin komponentit on muutettava helpommin jaettavaan muotoon. Ensisijaisesti ehdotettiin uutta Docker konttiteknologiaa. Tämän avulla Contriboard pitäisi olla helposti asennettavissa Linuxeihin, joissa Docker on saatavilla. Ei siis ollut tarvetta tehdä erikseen distro spesifisiä asennuspaketteja.

5.2.2 Manuaalinen tapa

Tässä luvussa Contriboard asennetaan kokonaisuudessaan yhteen koneeseen, suoraan GitHubista ilman erityisiä työkaluja. Ensimmäisenä on tehtävä virtuaalikone valmiiksi joko pilveen tai paikallisesti VirtualBoxilla. Virtuaalikoneen valmistelua, eli koneen määrittelyä ja käyttöjärjestelmän asennusta ei käydä tarkemmin. Käyttöjärjestelmäksi valitaan Ubuntu Server 14.04, ja loput speksit laitetaan käytössä olevien resurssien mukaan. Jotta Contriboardiin pääsee selaimella, täytyy koneelle olla julkinen IP-osoite. Virtualboxissa virtuaalikoneen saa helpoiten verkkoon asettamalla verkkokortin bridged-tilaan. Jos tarkoituksena on tehdä hajautettu asennus, koneiden on pystyttävä liikennöimään toistensa kanssa estotta. Valmiista virtuaalikoneesta kannattaa ottaa välillä kopioita, jotta monistaminen onnistuu nopeasti. Kuviossa 13 on tilanne, jolloin virtuaalikone on valmiina Contriboardin asennukseen.



Kuvio 13. VirtualBoxilla tehty Ubuntu Server

Virtuaalikoneen luonnin ja käyttöjärjestelmän asennuksen jälkeen on asennettava muutama paketti Contriboardia varten. Alla olevista komennoista ensimmäinen lisää Ubuntuun kolmannen osapuolen pakettirepositorion, josta asennetaan uudempi Node.js versio.

```
$ sudo add-apt-repository ppa:chris-lea/node.js
$ sudo apt-get update
$ sudo apt-get install build-essential mongodb redis-server git nodejs
$ npm install gulp
```

Tämän jälkeen voidaan asentaa API ja IO

```
$ npm install N4SJAMK/teamboard-api
$ npm install N4SJAMK/teamboard-io
```

Contriboardin clientistä on kaksi toteutusta, joista uudempi vaatii repositoryn kloonauksen erikseen

```
$ git clone https://github.com/N4SJAMK/teamboard-client-react.git

$ cd teamboard-client-react

$ npm install
```

Kun kaikki komponentit on asennettu, ne voidaan käynnistää järjestyksessä. Komennoissa on parametrejä, joihin on laitettava virtuaalikoneen IP-osoite.

```
$ cd ~/node_modules/teamboard-api && REDIS_HOST=localhost PORT=9002 npm start &

$ cd ~/node_modules/teamboard-io/ && REDIS_HOST=localhost API_URL=http://<ip-osoite>:9002 npm start &

$ cd ~/teamboard-client-react/ && API_URL=http://<ip-osoite>:9002 IO_URL=http://<ip-osoite>:9001 ~/node_modules/gulp/bin/gulp.js &
```

Jos ylläolevat komennot on ajettu onnistuneesti, Contriboardiin pitäisi päästä menemällä selaimessa virtuaalikoneen IP-osoitteeseen portissa 8000.

Vaikka asennusvaiheita ei kovin paljoa olekaan, tässä luvussa käyty asennustapa on epäilemättä erittäin karu ja huono. Se on kuitenkin ainut tapa, jos ei halua tai voi käyttää esimerkiksi Docker kontteja. Lisäksi tällä tavalla asennetut komponentit voivat olla versioiltaan sen verran eriävät, että toimintaa ei voi taata. Käyttötarkoituksia tällä tavalla tehdylle asennukselle ei juuri ole kuin ehkä kokeilumielessä.

5.2.3 Case Puppet

Contriboardin helpompaa asennusta ja konfigurointia lähdettiin toteuttamaan ensin Puppetilla. Tämä vaati aluksi hieman tutustumista Puppetin toimintaan. Ensinnäkin selvisi, että Puppet pitää olla asennettuna jokaiseen hallittavaan koneeseen ja DigitalOceanin tapauksessa verkkoon piti tehdä myös nimipalvelin koneita varten. Tämä ratkaistiin käyttämällä Fabricia Puppetin esiasennukseen. Liitteessä 1 on esimerkki fabfile, joka esiasentaa kohdekoneeseen Puppet-agentin ja tekee tarvittavat säädöt. Kun Puppet on asennettu, se alkaa hoitamaan kyseisen koneen konfiguraatiota eikä Fabricia tarvita enempää.

Puppetissa on moduuleita, jotka sisältävät manifesteja, esimerkiksi vaiheet jonkin ohjelman asentamiseen ja konfigurointiin. Aluksi oli siis kirjoitettava moduuli jokaiselle komponentille. Ohessa on näyte Contriboardia varten tehdystä Redis moduulista ja sen manifestista:

init.pp

```
class teamboard_redis($bind_address="") {
       anchor {"teamboard_redis::begin":} ->
       class {"::teamboard_redis::install":} ->
       class {"::teamboard_redis::configure":} ->
       anchor {"teamboard_redis::end":}
}
```

install.pp

```
class teamboard_redis::install inherits teamboard_redis {
       package { 'tb_redis_package':
       name   => "redis-server",
       ensure => present,
       }
}
```

configure.pp

```
class teamboard_redis::configure inherits teamboard_redis {
       if $bind_address == "" {
       file_line { 'redis_bind_address':
        notify => Service["tb_redis_service"],
        path => '/etc/redis/redis.conf',
        line => "#bind ",
        match => '^#?bind.*',
        } }
       else {
         file_line { 'redis_bind_address':
         notify => Service["tb_redis_service"],
         path => '/etc/redis/redis.conf',
         line => "bind $bind_address",
         match => '^#?bind.*',
         }

       }
       service { 'tb_redis_service':
       name    => "redis-server",
       ensure  => running,
       enable  => true,
       }

}
```

Tämä hyvin yksinkertainen manifesti käskee Puppettia asentamaan käyttöjärjestelmään paketin ”redis-server”. Kannattaa huomioida että kyseinen paketti löytyy Ubuntusta, muttei välttämättä muista distroista samalla nimellä. Asentamisen lisäksi moduuli tekee pienen muutoksen Rediksen konfiguraatioon ja käynnistää sen uudestaan.

Loput manifestit löytyvät vielä toistaiseksi N4S@JAMK GitHub repositoriosta https://github.com/N4SJAMK/teamboard-puppet

Kun kaikki moduulit ovat valmiina, on tehtävä päämanifesti joka sisältää node koneiden konfiguraatiot. Esimerkkinä päämanifesti, jossa koneeseen tb-redis.in.n4sjamk.org asentuu Redis:

site.pp

```
node 'tb-redis.in.n4sjamk.org' {
	class { "teamboard_redis":
	}
}
```

Vaikka Contriboard saatiinkiin asennettua Puppetilla ja tilanne parani käsi-asennuksesta, siinä oli myös paljon huonoksi koettuja seikkoja:

- Resurssisyöppö, paljon lisäasennettavaa
- Moduulien hankala ja taipumaton syntaksi
- Erillisen nimipalvelimen ja sertifikaattien hallinta

5.2.4 Case Ansible / Docker

Parempia vaihtoehtoja etsiessä Ansible vaikutti Puppettiin verrattuna varsin yksinkertaiselta. Se otettiin kokeiluun, koska asennus oli helppo ja nopea. Fabricin kaltoin Ansible täytyy asentaa vain koneeseen, josta hallinta tehdään. Tämän jälkeen alettiin tehdä Contriboardin asentamiseen vaadittua playbookia. Viralliset Contriboardin asennukseen käytetyt Ansible playbookit eivät toistaiseksi ole julkisia.

Samalla kuitenkin Docker tuli vahvasti kuvioihin, koska sillä saatiin vielä Contriboardin komponentit kätevästi eristettyä omiin ympäristöihinsä. Tällä parannettiin hallittavuutta entisestään.

Jokaiselle Contriboardin komponentille oli tehtävä dockerfile, jossa määritellään mille pohjalle kontin sovellus rakentuu. Alla on esimerkki Contriboardin API:lle tehdystä dockerfilestä.

```
FROM library/ubuntu:14.04

MAINTAINER n4sjamk

RUN apt-get update && apt-get install -y software-properties-common build-essential python
RUN add-apt-repository ppa:chris-lea/node.js
RUN apt-get update && apt-get install -y nodejs

RUN ["useradd", "-m", "teamboard", "-u", "23456"]

ADD . /home/teamboard/teamboard-api

RUN chown -R teamboard:teamboard /home/teamboard/teamboard-api

USER teamboard

RUN cd /home/teamboard/teamboard-api && \
        npm install

RUN ["mkdir", "/home/teamboard/logs"]

CMD /usr/bin/node /home/teamboard/teamboard-api/index.js \
        2>> /home/teamboard/logs/teamboard-api.err \
        1>> /home/teamboard/logs/teamboard-api.log
```

Dockerfilen luonnin jälkeen kontti on vielä buildattava joko itse tai esimerkiksi ilmaisessa Docker Hub Registry-palvelussa. (Docker Hub Registry n.d.). Docker Hub Registry on myös mahdollista liittää GitHubiin, jolloin kontti buildataan automaattisesti muutosten tapahtuessa.

Contriboardin asennuksessa vaadittavat komponentit ovat saatavilla docker kontteina, ja ne ovat asennettavissa Docker Hub Registrystä osoitteessa https://hub.docker.com/u/n4sjamk/ . Liitteessä 2 on esimerkki dockerfilestä, jolla kaikki Contriboardin komponentit laitettiin yhteen konttiin.

5.2.5 Vagrant

Vagrant on työkalu, joka toimii wrapperina muille ohjelmille kuten VirtualBoxille ja Ansiblelle. Vagrantille luodaan erityinen tiedosto Vagrantfile, mikä määrittelee esimerkiksi virtuaalikoneen luomistavan ja sille tehtävän konfiguraation. Vagrant löydettiin työn kannalta hieman liian myöhään, eikä siitä ehtinyt muodostua kunnollista näkemystä tai saatu aivan kaikkea irti. Se vaikutti kuitenkin olevan juuri oikea työkalu kehitysympäristöjen monistamiseen, eikä vastaavia tuntunut olevan.

5.2.6 Kehitysympäristön monistaminen

Työasemassa täytyy olla valmiina seuraavat asiat:

- Windows 7 x64 tai Ubuntu 14.04 64-bit (suositus)
- VirtualBox
- Vagrant
- Git

Erityishuomiona Windows käyttöjärjestelmä ja Contriboardin tapauksessa npm moduulit voivat aiheuttaa ongelmatilanteen. Windows sallii maksimissaan 255 merkkiä pitkät tiedostopolut, joka helposti ylittyy node moduulien tiedostorakenteen takia. Tämä ongelma ilmenee Vagrantin ja VirtualBoxin kanssa, koska isäntäkäyttöjärjestelmästä liitetään kansio virtuaalikoneeseen, jolloin Windowsin rajoitukset tulevat voimaan. Ongelmaa voi koittaa kiertää tekemällä virtuaalikone aivan C: aseman juureen, mutta täydellistä ratkaisua ei toistaiseksi ole löytynyt.

Ensimmäinenä on haettava Contriboardin kehitysympäristöjä varten luodut skriptit GitHubista. Tämä tapahtuu antamalla terminaalissa komento:

```
$ git clone https://github.com/N4SJAMK/teamboard
```

Tämän jälkeen on valittava tapa, jolla Contriboardin komponenttien repositoryt kloonataan. Repositoriot voi myös linkittää omaan GitHub-käyttäjätunnukseen.

```
$ cd teamboard
```

```
$ sh initialize-ssh.sh <github käyttäjänimi>
$ sh initialize-https.sh <github käyttäjänimi>
$ sh initialize-https.sh
```

Kun repositoryt on kloonattu, voidaan käynnistää virtuaalikone, joka sisältää kehitysympäristön.

```
$ vagrant up
```

Virtuaalikoneen käynnistyminen ja asentuminen kestää joitain minuutteja. Kun asennus on päättynyt, koneeseen pitää mennä SSH:lla sisään.

```
$ vagrant ssh
```

Contriboard on vielä asennettava ja käynnistettävä. Seuraavat askeleet riippuvat kehittäjän käyttöjärjestelmästä. Jos käytössä on Linux, voi Contriboardin todennäköisesti käynnistää automaattisesti antamalla komennon:

```
$ sh ~/teamboard-start/install_and_start.sh
```

Jos käyttöjärjestelmänä on Windows, suositeltavaa on tehdä asennus käsin komennoilla:

```
$ cd ~/teamboard-api
$ npm install –no-bin-links
$ npm start &
$ cd ~/teamboard-io
$ npm install –no-bin-links
$ npm start &
$ cd ~/teamboard-client-react
$ npm install –no-bin-links
$ gulp
```

Tässä vaiheessa kaikki Contriboardin komponentit pitäisi olla käynnissä, ja siihen pääsee menemällä selaimella osoitteeseen http://localhost:8000

Contriboardia voi nyt kehittää isäntäkoneessa haluamillaan työkaluilla, sillä lähdekoodit ovat virtuaalikoneen kanssa jaetussa kansiossa. Samalla kansiot ovat versionhallinnan alaisia ja yhteydessä Corolla tuotantoketjuun.

(Contriboard development environment setup using Vagrant 2015.).

6 Tulokset ja yhteenveto

Työn tuloksena löydettiin menetelmiä, joilla kehittämistä voi todellakin nopeuttaa. Automaatiotyökalujen edut korostuvat mitä monimutkaisemmassa järjestelmässä niitä sovelletaan. Contriboardin tapauksessa saatu hyöty on yhtenäinen kehitysympäristö jokaiselle kehittäjälle minimiajassa, ilman ylimääräistä säätöä. Jokaisella työssä käydyllä konfiguraatiotyökalulla olisi teoriassa mahdollista saavuttaa sama tulos kuin Vagrantilla, mutta se ei olisi paikallisen kehitysympäristön pystyttämiseen millään tavoin järkevää. Sen sijaan konfiguraatiotyökaluilla on tärkeä rooli kuitenkin koko ketjun myöhemmässä vaiheessa, kun Contriboard asennetaan pilvipalvelussa oleviin virtuaalikoneisiin. Luvussa 1.3 asetetut tavoitteet toteutuivat seuraavasti:

Contriboardin koekäytön tai evaluoinnin edistäminen

Contriboard on helpoiten koekäytettävissä käyttämällä valmiiksi tehtyjä Docker kontteja, joita käsiteltiin luvussa 5.2.4. Asentaminen vaatii Linuxin ja Dockerin, ja kontit asennetaan Docker Hub Registrystä. Jos Dockeria ei ole mahdollista käyttää, on mahdollista tehdä virtuaalikone ja automaattinen asennus Vagrantin avulla. Minimalistinen asennus on mahdollista käsin kuten luvussa 5.2.2 tehtiin.

Contriboardin kehittämisen haarauttaminen

Contriboardin kehittäminen ja kehitysympäristön monistaminen on helpointa Vagrantin ja VirtualBoxin avulla, joka on käsitelty luvussa 5.2.6. Kehitysympäristö on versionhallinnan alainen ja kytköksissä GitHubiin.

Contriboardin testaaminen

Contriboardin kehityksen aikainen testaus on helpointa tehdä samalla tavalla, kuin kehitysympäristön monistaminen Vagrantin avulla. Osa testaamisesta tapahtuu osana continuous integration ketjua.

7 Pohdinta

Contriboardin kehityksessä käytettiin hyvin yleisiä työkaluja ja menetelmiä, mitä yleensäkin voisi kuvitella käytettävän sovelluskehityksessä. Samoin Corolla tuotantoketjuun oli valikoitu hyviä palveluita, eikä suurempia ongelmia havaittu. Kaikki teknologiat ovat sovitettavissa hyvin muihinkin kehitystarkoituksiin. Teknologiat kuitenkin kehittyvät, ja välillä on vaikeaa pysyä kehityksen tahdissa. Se oli myös ongelmana työn rajauksessa, sillä oli vaikeuksia pysäyttää tilanne johonkin pisteeseen ja dokumentoida se kattavasti. Esimerkiksi Contriboardin asentaminen Puppetilla on jo vanhentunut menetelmä

eikä enää toimi. Moduuleja ei ole päivitetty ja Puppetkin on saanut merkittävän päivitysversion 4, joka toisaalta voisi olla kokeilemisen arvoinen.

Yksi aivan uutena tulleista asioista oli konttiteknologia ja Docker, jonka ympärillä vaikuttikin olevan vilinää. Konttiteknologia ei ehkä aivan vielä ole lyönyt läpi täysillä, mutta sellaiset merkit toki ovat ilmassa. Microsoft on tuomassa konttiteknologian myös Windows alustalle.

N4S@JAMK-projektissa DevOps toimintatapa tuntui toimivan ainakin pienryhmässä erittäin hyvin. Automaatiolla saavutettua nopeutta ei voi korostaa tarpeeksi, mutta on myös olemassa asioita jotka yksinkertaisesti vaativat hyväksynnän muutoin kuin koneellisesti.

Jatkokehitettävää ei tarvitse kaukaa etsiä, sillä työn loppuvaiheilla ilmaantui uusia tutkimisen arvoisia asioita kuten Docker compose ja CoreOS/Rocket. Lisäksi jäljellä olevia ongelmia pitäisi ratkaista, kuten jaettujen kansioiden rajoitukset Windowsissa. Myös pilvipalvelun hyödyntäminen testauksessa jäi puuttumaan työstä, ja olisi selvitettävä kuinka testaajat saavat pieniä testiympäristöjä nopeasti.

LÄHTEET

A Brief History of Lean. N.d. Artikkeli Lean Enterprise Instituten sivulla. Viitattu 11.5.2015. https://www.lean.org/WhatsLean/History.cfm.

Ansible Installation. N.d. Ansiblen dokumentaatio. Viitattu 24.5.2015. http://docs.ansible.com/intro_installation.html.

Contriboard development environment setup using Vagrant. 2015. Teamboard repository GitHubissa. Viitattu 24.5.2015. https://github.com/N4SJAMK/teamboard.

Docker Hub Registry. N.d. Docker Hub Registry palvelu verkossa. Viitattu 24.5.2015. https://registry.hub.docker.com/.

Fowler, M. 2013. ContinuousDelivery. Viitattu 23.5.2015. http://martinfowler.com/bliki/ContinuousDelivery.html.

Getting Started – A Short History of Git. 2014. Viitattu 11.5.2015. https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git.

GitHub Plugin. N.d. Jenkins Wiki. Viitattu 24.5.2015. https://wiki.jenkins-ci.org/display/JENKINS/GitHub+Plugin.

Integrated service management and cloud computing. 2010. Viitattu 16.5.2015. http://www.ibm.com/ibm/files/E955200R99025N70/5Integrated_service_management_and_cloud_computing_644KB.pdf.

Julistuksen takana olevat periaatteet. 2001. Agile Manifeston verkkosivu. Viitattu 11.5.2015. http://agilemanifesto.org/iso/fi/principles.html.

Kanban (development). N.d. Wikipedian artikkeli. Viitattu 23.5.2015. http://en.wikipedia.org/wiki/Kanban_%28development%29.

N4S-ohjelma. 2014. Digilen N4S-verkkosivut. Viitattu 23.5.2015. http://www.n4s.fi/fi/.

O'Grady, S. 2013. DVCS and Git Usage in 2013. Viitattu 11.5.2015. http://redmonk.com/sograady/2013/12/19/dvcs-and-git-2013/.

Poppendieck, M. 2003. Lean Software Development: An Agile Toolkit.

Rackspace Managed Cloud Services—More than just infrastructure. N.d. Rackspace Managed Cloud esittelysivu. Viitattu 24.5.2015. http://www.rackspace.com/cloud.

Royce, Winston W. 1970. Managing the Development of Large Software Systems. Viitattu 7.2.2015. http://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf.

RUP summary. 2008. SUNIWE verkkosivut. Viitattu 17.5.2015. http://projects.staffs.ac.uk/suniwe/project/rup.html.

Takecuchi, H. & Nonaka, I. 1986. The new new product development game. http://scrum.kaverjody.com/wp-content/uploads/2013/01/9-The-new-new-product-development-game.pdf.

Vesiputousmalli. N.d. Wikipedian artikkeli. Viitattu 23.5.2015. http://fi.wikipedia.org/wiki/Vesiputousmalli.

Viinikanoja, J. 2014. Kuormantasauksen soveltaminen Teamboard 2.0 –palvelussa. Opinnäytetyö. Jyväskylän Ammattikorkeakoulu. Viitattu 17.5.2015. http://urn.fi/URN:NBN:fi:amk-2014092314146.

Wells, D. N.d. Extreme Programming: A gentle introduction. Viitattu 7.2.2015. http://www.extremeprogramming.org/.

West, D. & Grant Tom. 2010. Agile Development: Mainstream Adoption Has Changed Agility. Viitattu 7.2.2015. http://pmshow2012.programmedevelopment.com/public/uploads/files/forrester_agile_development_mainstream_adoption_has_changed_agility.pdf.

What is Kanban?. N.d. Artikkeli LeanKit verkkosivulla. Viitattu 7.2.2015. http://leankit.com/kanban/what-is-kanban/.

What is Scrum?. N.d. Scrum Guides verkkosivu. Viitattu 23.5.2015. http://www.scrumguides.org/.

Wheatley, M. 2014. Deadly Docker: Why containers are a threat to cloud virtualization. Viitattu 16.5.2015. http://siliconangle.com/blog/2014/08/22/deadly-docker-why-containers-are-a-threat-to-cloud-virtualization/.

Who's driving this thing? 2010. Viitattu 13.4.2015. http://jenkins-ci.org/content/whos-driving-thing.

Wilinski, E. 2014. 6 DevOps Best Practices: Bridging the Culture Gap. Viitattu 16.5.2015. https://blog.newrelic.com/2014/06/02/devopsculture/.

Yegulalp, S. 2014. 4 reasons why Docker's libcontainer is a big deal. Viitattu 16.5.2015. http://www.infoworld.com/article/2607966/application-virtualization/4-reasons-why-docker-s-libcontainer-is-a-big-deal.html.

LIITTEET

Liite 1: Esimerkki fabfilestä

```
from fabric.api import cd, lcd, local, run, env, hosts, sudo, execute
from fabric.contrib.files import contains
import ConfigParser

config = ConfigParser.ConfigParser()
config.read('fabsettings.ini')

def install_puppet(ubuntu_version = config.get('agent', 'ubuntu_version')):
   with cd('/tmp'):
      run('wget https://apt.puppetlabs.com/puppetlabs-release-{0}.deb'.format(ubuntu_version))
      sudo('dpkg -i puppetlabs-release-{0}.deb'.format(ubuntu_version))
      sudo('apt-get update')
      sudo('apt-get -y install puppet')



def change_puppet_conf(
      puppetmaster     = config.get('master', 'puppetmaster'),
      run_interval       = config.get('agent', 'run_interval')):
   with cd('/etc/puppet'):
      if not contains('/etc/puppet/puppet.conf', r'^server=.+$', escape = False):
         sed_command = 'sed -i "/\[main\]/a server={0}" puppet.conf'.format(puppetmaster)
         sudo(sed_command)
      else:
         sed_command = 'sed -i "s/^server=.*$/server={0}/g" puppet.conf'.format(puppetmaster)
         sudo(sed_command)
      if int(run_interval):
         if not contains('/etc/puppet/puppet.conf', r'^\[agent\]$', escape = False):
            sudo("echo '[agent]' >> puppet.conf")
```

```
        if not contains('/etc/puppet/puppet.conf', r'^runinterval=.+$', escape = False):
            sed_command = 'sed -i "/\[agent\]/a runinterval={0}" puppet.conf'.format(run_interval)
            sudo(sed_command)
        else:
            sed_command = 'sed -i "s/^runinterval=.*$/runinterval={0}/g" puppet.conf'.format(run_interval)
            sudo(sed_command)


def set_puppet_start(start = config.get('agent', 'start')):
    with cd('/etc/default'):
        sed_command = 'sed -i "s/^START=.*$/START={0}/" puppet'.format(start)
        sudo(sed_command, shell_escape = False)


def restart_puppet():
    sudo("service puppet restart")


def change_dns(
        dns_server_ip   = config.get('dns', 'dns_server_ip'),
        domain          = config.get('agent', 'domain')):
    with cd('/etc/resolvconf/resolv.conf.d'):
        sudo('echo "search {0}" > head'.format(domain))
        sudo('echo "nameserver {0}" >> head'.format(dns_server_ip))
        sudo('resolvconf -u')


def update_dns(
        dns_server_ip   = config.get('dns', 'dns_server_ip'),
        domain          = config.get('agent', 'domain'),
        dnssec_key      = config.get('dns', 'dnssec_key'),
        interface       = config.get('agent', 'interface')):
    ip = run("ifconfig {0} | grep 'inet addr:' | cut -d: -f2 | awk '{{ print $1}}'".format(interface))
    ip_octets = ip.split('.')
    revzone = ip_octets[1] + "." + ip_octets[0]                                     #works for our configuration, using /16 mask
    revip = ip_octets[3] + "." + ip_octets[2] + "." + revzone
```

```
    hostname = run("hostname")
    update_string = (
        '"server {0}\n"
        "zone {1}\n"
        "update delete {2}.{1}. A\n"
        "update add {2}.{1}. 86400 A {3}\n"
        "send\n"
        "zone {4}.in-addr.arpa\n"
        "update delete {5}.in-addr.arpa.\n"
        "update add {5}.in-addr.arpa 86400 PTR {2}.{1}.\n"
        "send"').format(dns_server_ip, domain, hostname, ip, revzone, revip)
    print(update_string)
    run("echo {0} | nsupdate -y '{1}'".format(update_string, dnssec_key), shell_escape = False)

def install_openntpd():
    sudo("apt-get update")
    sudo("apt-get install openntpd")



def configure_openntpd():
    with cd('/etc/opentpd'):
                                    sudo("sed 's/^servers/#servers/g' ntpd.conf'")
                                    sudo("echo 'servers time1.mikes.fi\nserverstime2.mikes.fi' >
ntpd.conf'")
                                    sudo("service openntpd restart")

def setup_puppet():
    execute(install_puppet)
    execute(change_puppet_conf)
    execute(set_puppet_start)
    execute(change_dns)
    execute(update_dns)
    execute(restart_puppet)
```

Liite 2: Esimerkki dockerfile, jossa Contriboard yhdessä kontissa

```
FROM phusion/passenger-nodejs

MAINTAINER N4S@JAMK

RUN apt-get update && apt-get install -y mongodb-server redis-server ruby-sass node-gyp

RUN mkdir /etc/service/mongodb /etc/service/redis /etc/service/teamboard-api /etc/service/teamboard-io

ADD mongodb.sh /etc/service/mongodb/run

RUN sed -i -e 's/^journal=true/#journal=true/' /etc/mongodb.conf

RUN echo "smallfiles=true\nnojournal=true\nnohttpinterface=true" >> /etc/mongodb.conf

ADD redis.sh /etc/service/redis/run

RUN npm install --global gulp bower

RUN useradd -m teamboard

RUN su -l teamboard

RUN mkdir ~api ~io ~client

WORKDIR /home/teamboard/api

RUN npm install N4SJAMK/teamboard-api#dev

ADD teamboard-api.sh /etc/service/teamboard-api/run

WORKDIR /home/teamboard/io

RUN npm install N4SJAMK/teamboard-io#dev

ADD teamboard-io.sh /etc/service/teamboard-io/run

WORKDIR /home/teamboard/client

RUN npm install N4SJAMK/teamboard-client#dev

WORKDIR /home/teamboard/client/node_modules/teamboard

RUN bower install --allow-root

ENV NODE_ENV production

ENV API_PORT 9002

ENV IO_PORT 9001

ENV TOKEN_SECRET topkek

ENV MONGODB_URL mongodb://localhost:27017/teamboard

ENV REDIS_HOST localhost

ENV REDIS_PORT 6379
```

```
RUN API_URL='/api' IO_URL='/socket.io' gulp build
ENV API_URL http://localhost:9002/api
ENV IO_URL http://localhost:9001
USER root
RUN chown -R teamboard:teamboard /home/teamboard/
RUN rm -f /etc/service/nginx/down /etc/nginx/sites-enabled/default
ADD teamboard.nginx /etc/nginx/sites-enabled/teamboard
RUN apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
ADD docker.key.pub /root/.ssh/authorized_keys
EXPOSE 22 80 9001 9002
```