Tampereen ammattikorkeakoulu

CI/CD-työkalut web-ohjelmiston jakelussa

Joonas Salojärvi

OPINNÄYTETYÖ
Toukokuu 2021

Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Ohjelmistotuotanto

SALOJÄRVI, JOONAS:
CI/CD-työkalut web-ohjelmiston jakelussa

Opinnäytetyö 64 sivua, joista liitteitä 8 sivua
Toukokuu 2021

Ohjelmiston elinkaari käsittää monia eri vaiheita. Se kulkee ideaalitilanteessa projektinhallinnasta ja koodin tuottamisesta laadunvarmistuksen kautta aina asiakkaiden käsiin. Näitä eri työvaiheita on aikaisemmin hoitanut niihin erikoistuneet henkilöt, ja niihin käytetyt työtunnit ovat tulleet kalliiksi. Tähän ongelmaan DevOps-filosofia ja menetelmät pyrkivät tuomaan ratkaisun yhdessä automaation kanssa.

Opinnäytetyön toimeksiantaja Oscar Software Oy oli automatisoinut web-käyttöliittymäpohjaisen toiminnanohjausjärjestelmän jakelun Jenkins-työkalulla. Tavoitteena oli selvittää, voidaanko tuotekehitysprosessia parantaa siirtymällä toiseen automaatiotyökaluun. Työkalujen vertailussa pääpaino oli jakelun automatisoinnissa. Tarkoituksena oli asentaa ja ottaa käyttöön eri automaatiotyökaluja sekä toteuttaa näille työjonot toiminnanohjausjärjestelmän käyttöliittymän ja taustapalvelimen jakeluun. Vertailtaviksi työkaluiksi valittiin Jenkinsin ohella GitLab sekä Bamboo.

Tuloksena todettiin, että jokainen vertailtu työkalu suoriutui annetusta tehtävästä, eikä huomattavia eroavaisuuksia jakelun kannalta huomattu. Tuloksen pohjalta arvioitiin, että työkalun vaihtamisesta ei juurikaan saada hyötyä itse jakeluprosessiin. Vertailtavista työkaluista erityisesti GitLab tarjosi paljon muuta automaatiopalvelun lisäksi, ja jatkokehitysehdotuksena on vertailla eri työkaluja tarkemmin myös muilta näkökannoilta pelkän jakelun sijaan ottaen huomioon koko ohjelmiston elinkaaren.

Asiasanat: devops, jenkins, bamboo, gitlab, automaatio

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Software Development

SALOJÄRVI, JOONAS:
CI/CD Tools in Web Application Delivery

Bachelor's thesis 64 pages, appendices 8 pages
May 2021

The client for this thesis, Oscar Software, had automated their software delivery process with Jenkins automation server. The purpose of this study was to find out if there was any benefit in switching from one automation server to another, and what those benefits were. The main emphasis was on the delivery of a web application. The compared software were Jenkins, GitLab and Bamboo.

All the automation tools were installed on different servers. Pipeline scripts were created in order to deliver Oscar Software's web-based ERP software to the target server. Other features were compared by studying the official documentation of each application.

No significant differences were observed while creating and running the pipelines. All the tools were able to deliver the web application to the target environment.

Switching an automation server could not be recommended for software delivery only. More research on other features offered by the tools should be done, to determine if a tool could also be used to replace other software.

Key words: devops, jenkins, bamboo, gitlab, automation

SISÄLLYS

1 JOHDANTO 7
2 DevOps 9
2.1 Määritelmä 9
2.2 Menetelmät ja hyödyt 10
3 CI/CD-työkalut 12
3.1.1 Määritelmä 12
3.2 Jenkins 13
3.3 GitLab 13
3.4 Bamboo 14
4 Oscar Cloud ja cERP 15
5 Virtualisointi 16
6 Docker 17
7 Palvelinympäristö 18
7.1 Virtuaalipalvelinten perusta 18
7.2 Palvelimet ja asennusprosessi 19
7.2.1 Tietovarastopalvelin 20
7.2.2 Kohdepalvelin 21
7.2.3 Jenkins-palvelin 23
7.2.4 GitLab 25
7.2.5 Bamboo 27
8 Työjonojen luonti 32
8.1 Oscar Cloud 32
8.1.1 Oscar Cloudin asennus Jenkinsin kautta 33
8.1.2 Oscar Cloudin asennus GitLabin kautta 34
8.1.3 Oscar Cloudin jakelu Bamboon kautta 36
8.2 cERP 42
8.2.1 cERPin jakelu Jenkinsin kautta 42
8.2.2 cERPin jakelu GitLabin kautta 43
8.2.3 cERPin jakelu Bamboon kautta 44
9 Työkalujen vertailu 45
9.1 Asennus ja ympäristö 45
9.2 Hinnoittelu 45
9.3 Työjonojen luonti ja muokkaaminen 46
9.4 Työjonojen suorittaminen ja lokit 47
9.5 Versionhallintaintegraatio ja työjonojen ajastaminen 47
9.6 Laajennettavuus 48

10 POHDINTA 49
LÄHTEET 55
LIITTEET 57
Liite 1. Oscar Cloudin työjono Jenkinsissä 57
Liite 2. Oscar Cloudiin liittyvien kirjastojen työjono Gitlabissa 59
Liite 3. Oscar Cloudin pääkirjaston työjonoskripti GitLabissa 60
Liite 4. Oscar cERP-kirjaston työjonoskripti Jenkinsissä 61
Liite 5. Oscar cERP-kirjaston työjonoskripti GitLabissa 62
Liite 6. Työjonojen suoritusajat 63

LYHENTEET JA TERMIT

CD	Continuous delivery, jatkuva jakelu
CI	Continuous integration, jatkuva integraatio
DevOps	Development plus Operations
shell	komentorivi
skripti	komentosarja
SSH	Secure Shell, salatun tietoliikenteen protokolla
VPN	Virtual Private Network

1 JOHDANTO

Opinnäytetyön toimeksiantajana toimii Oscar Software Oy, joka on yritysten liiketoimintaan erikoistuva yhtiö. Sen päätuotteita ovat erilaiset toiminnanohjausjärjestelmät eri käyttötarkoituksiin, joista käytetään yhteistä nimitystä liiketoimintaalusta. Se on perustettu vuonna 2005, ja sen pääkonttori sijaitsee Tampereella.

Oscar Software Oy on soveltanut DevOps-menetelmiä käyttämällä Jenkins-automaatiopalvelinta tuotekehityksen apuna. Jenkinsin kautta toteutettu automaatio hoitaa tietyillä Oscar Softwaren ohjelmistoilla niin ohjelmiston rakentamisen, kuin tuotantoon saattamisen.

Käytössä olevaa Jenkins-automaatiopalvelinta ei ole aiemmin vertailtu muihin tarjolla oleviin vaihtoehtoihin nykyisessä käyttötarkoituksessa. Tämän opinnäytetyön tavoitteena on selvittää, voidaanko tuotekehitysprosessia ja DevOps-työskentelyä parantaa siirtymällä toisen automaatiopalvelimen piiriin. Lisäksi on olennaista selvittää, kuinka ison työn takana se on.

Tarkoituksena on asentaa ja ottaa käyttöön eri automaatiotyökaluja, sekä luoda kyseisille työkaluille tarvittavat työjonot Oscar Cloud- ja Oscar cERP-ohjelmistojen jakeluun. Jakeluprosessit suunnitellaan vastaamaan nykyistä tuotantokäytössä olevaa mahdollisimman hyvin. Eroavaisuuksia saattaa syntyä, koska vertailtavat ohjelmat ovat luonteeltaan hieman erilaisia. Pääpaino on jakeluprosessien vertailussa, mutta on myös otettava huomioon, mitä muuta vertailtavat työkalut tarjoavat.

Kokeiltaviksi uusiksi vaihtoehdoiksi sovittiin esimiehen kanssa GitLab ja Bamboo. Jenkins toimii näiden kahden vertailukohtana. Aikaisempaa kokemusta opinnäytetyön tekijällä on ainoastaan hieman Jenkins-ohjelmistosta.

Opinnäytetyön menetelminä toimii aiheeseen liittyvään kirjallisuuteen tutustuminen, sekä vertailtaviin työkaluin tutustuminen niin itsenäisesti, kuin viralliseen dokumentaatioon tukeutumalla.

Opinnäytetyön tärkeimpänä tuotoksena on tämä raportti, joka antaa arvion nykyisen jakeluprosessin tilasta sekä mahdolliset kehitysehdotukset. Sen ohella syntyy työjonot jokaiselle työkalulle, joita on mahdollista käyttää pohjana työkalun vaihdon tapahtuessa. Muut huomiot ja ongelmatilanteet, joita on mahdollista kohdata vaihdon yhteydessä, sisällytetään raporttiin.

2 DevOps

2.1 Määritelmä

DevOps (Development plus Operations) on viime aikoina tullut keskeiseksi osaksi ohjelmistokehityksen elinkaarta. DevOps tarjoaa prosessikehykset ja työkalut integroida kaikki ohjelmistokehityksen elinkaaren vaiheet yhdeksi yhtenäiseksi kokonaisuudeksi. Se auttaa ohjaamaan ja automatisoimaan prosessia kehityksen, testaamiseen, jakelun ja tuen vaiheiden välillä. Se sisältää parhaat käytänteet, kuten koodiarkistot, rakentamisen automatisointi, jatkuva jakelu ja muita. (Vadapalli 2018.)

DevOps on määritelmänä hyvin laaja, ja se käsittää paljon niin konkreettisia, kuin abstraktejakin asioita. Sitä voisi luonnehtia eräänlaiseksi filosofiaksi pelkän toimintamallin sijaan.

DevOps-liikkeen alkutaipaleet alkoivat Patrick Deboisista, joka oli ohjelmistokehittäjänä kyllästynyt kehittäjien ja IT-osaston kahtiajakoon. Hän piti O'Reilly Velocity-konferenssissa esityksen nimeltä *10+ Deploys per Day: Dev and Ops Cooperation at Flickr*. Esityksen aiheena oli kehittäjien ja IT-osaston yhteistyö ja mitä sillä voidaan saavuttaa, ja se otettiin hyvin vastaan. Tämän seurauksena Patrick järjesti Ghentissä, Belgiassa tapahtuman nimeltä DevOpsDays, josta nykyisin käytetty termi DevOps juontaa juurensa. (Verona 2018.)

DevOpsin voidaan sanoa noudattavan ketterän ohjelmistokehityksen ensimmäistä periaatetta, joka kuuluu ”yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja”. Vaikka DevOps on tuonut mukanaan paljon työkaluja, ideana kuitenkin on, että nimenomaan näiden työkalujen avulla voidaan rikkoa seiniä yrityksien sisällä, mahdollistaen paremman kanssakäymisen tiimien välillä. (Verona 2018.)

2.2 Menetelmät ja hyödyt

Kokoonpanon hallinta (configuration management) on käytännön määritelmä ohjelmiston hallintaan ja muutosten tekemiseen. Se koostuu käytetystä versionhallintaohjelmasta, sekä sen sisällä käytetyistä käytänteistä. (Blogumas 2020).

Continuous integration (CI) eli jatkuva integraatio on tapa yhdistää ohjelmoijien tuotokset ja automaattisesti testata tämän jälkeen ohjelman toimivuutta (Pittet n.d.). Kun ohjelmaa testataan jo pienten muutoksien jälkeen, korjaukset voidaan tehdä ennen kuin ne pääsevät vaikuttamaan muiden tuottamaan koodiin. Jatkuva integraatio vaatii CI-työkalun käyttämistä, joka automaattisesti rakentaa ohjelman ja ajaa sille yksikkötestit (Blogumas 2020).

Automaatiotestaus on yleensä osana jatkuvaa integraatiota, ja sen testit tulisi määrittää ohjelman tyypin, sekä valmiusasteen perusteella (Blogumas 2020). Testin tyypistä riippuen, automaatiosta huolimatta, testit voivat viedä paljon aikaa, ja siksi niiden määrittely oikeisiin vaiheisiin ohjelmiston elinkaarta on tärkeää.

Continuous delivery (CD) eli jatkuva jakelu on prosessi, jolla toimivaksi testattu ohjelma saatetaan jakeluun (Pittet, n.d.). Jakelu voi tapahtua joko testi- tai tuotantoympäristöön tarpeen mukaan. Jatkuvassa jakelussa julkaisujen valmistelu vie vähemmän aikaa ja niitä voidaan tehdä nopeammalla syklillä, mikä johtaa myös nopeampaan palautteen saantiin asiakkailta (Pittet n.d.).

Continuous deployment eli jatkuva käyttöönotto tarkoittaa koko prosessin automatisoimista koodin tuottamisesta tuotantoympäristöön jakamiseen. Jatkuva käyttöönotto vaatii suurta luottoa testausprosessiin, sillä jokainen muutos koodin, joka läpäisee testit, viedään julkiseksi. (Blogumas 2020.)

Jatkuva monitorointi on vaihe, joka voidaan toteuttaa, kun ohjelmisto on viety tuotantoon. Jatkuvan monitoroinnin tarkoituksena on automaattisesti huomauttaa kehittäjiä ja laadunvarmistajia mahdollisista ongelmatilanteista. Infrastruktuuritason monitorointi kertoo tietoja jakeluun käytettävistä laitteista, esimerkiksi palve-

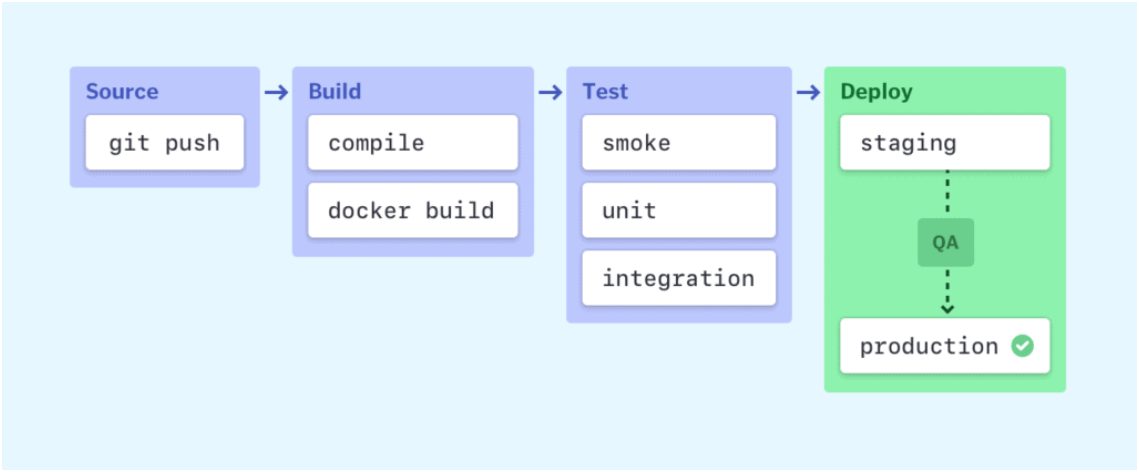
limista, tallennustilasta, verkosta tai muusta vastaavasta. Ohjelmistotason monitoroinnin tavoitteena on mitata itse ohjelmiston suorituskykyä. Tähän kuuluu esimerkiksi päätepisteiden kuormitus ja käyttöaste. Verkkotasolla monitoroinnissa tarkkaillaan verkon saatavuutta sekä palomuurien toimintaa. (Bose 2020.)

3 CI/CD-työkalut

3.1.1 Määritelmä

CI/CD on lyhenne DevOps-menetelmien vaiheista continuous integration ja continuous delivery. CI/CD-työkaluilla tarkoitetaan ohjelmistoja, jotka suorittavat nämä vaiheet automatisoidusta.

CI/CD-pipelinen eli työjonon vaiheet (kuvio 1) alkavat koodin tuottamisesta ja saattamisesta yhteiseen tietovarastoon (repository). Työjono voidaan käynnistää manuaalisesti tai ajastetusti, tai se voi aloittaa toimintansa itsenäisesti uuden lähdekoodin tullessa saataville. Build- eli rakennusvaiheessa ohjelman lähdekoodista muodostetaan ajettava ohjelma. Testausvaiheessa ajetaan automatisoidut testit koodin ja itse ohjelman toimivuuden varmistamiseksi. Deploy-vaiheessa ohjelma saatetaan oikeaan testi- tai tuotantoympäristöön ajettavaksi. (Anastasov 2019.)



KUVIO 1. CI/CD-työjonon vaiheet (Anastasov 2019)

Kokeiltaviksi ohjelmiksi valittiin esimiehen toiveesta GitLab ja Bamboo. Myös Jenkins otetaan mukaan vertailukohteeksi. Näissä työkaluissa keskitytään erityisesti ohjelmiston jakeluun eli deploy-vaiheeseen.

3.2 Jenkins

Jenkins on itsenäinen avoimen lähdekoodin automaatiopalvelin, jota voidaan käyttää kaikenlaisten tehtävien automatisointiin liittyen ohjelmiston rakentamiseen, testaamiseen ja jakeluun (Jenkins n.d.).

Jenkins oli alun perin Kohsuke Kawaguchin harrastelijaprojekti vuonna 2004 nimeltä Hudson. Kawaguchi työskenteli Sun Microsystemsillä, ja Hudsonin kehittyessä yhä useampi tiimi Sunilla alkoi ottaa sitä käyttöönsä. Vuonna 2008 yritys tajusi kyseisen työkalun laadun ja arvon, ja pyysi Kawaguchia työskentelemään sen parissa täysipäiväisesti. (Smart 2011.)

Vuonna 2009 Oraclen ostaessa Sun Microsystemsin, Hudsonin kehitystiimin ja Oraclen välille alkoi syntyä jännitteitä tavaramerkin omistajuudesta. Vuonna 2011 Hudsonille äänestettiin uudeksi nimeksi Jenkins, ja sen lähdekoodi siirrettiin GitHubissa omaksi projektikseen. Suurin osa Hudsonin käyttäjistä vaihtoi Jenkinsiin. (Smart 2011.)

3.3 GitLab

GitLabia kuvaillaan kokonaisena avoimen lähdekoodin DevOps-alustana GitLabin esittelyvideolla What is GitLab? Speed. Efficiency. Trust. (2020). Videolta käy ilmi GitLabin tarjoamat oleellisimmat ominaisuudet, joita ovat ketterän projektinhallinnan mukainen suunnittelu, versionhallinta, koodin vertaisarviointi, jatkuva integrointi ja jakelu, tietoturva ja monitorointi.

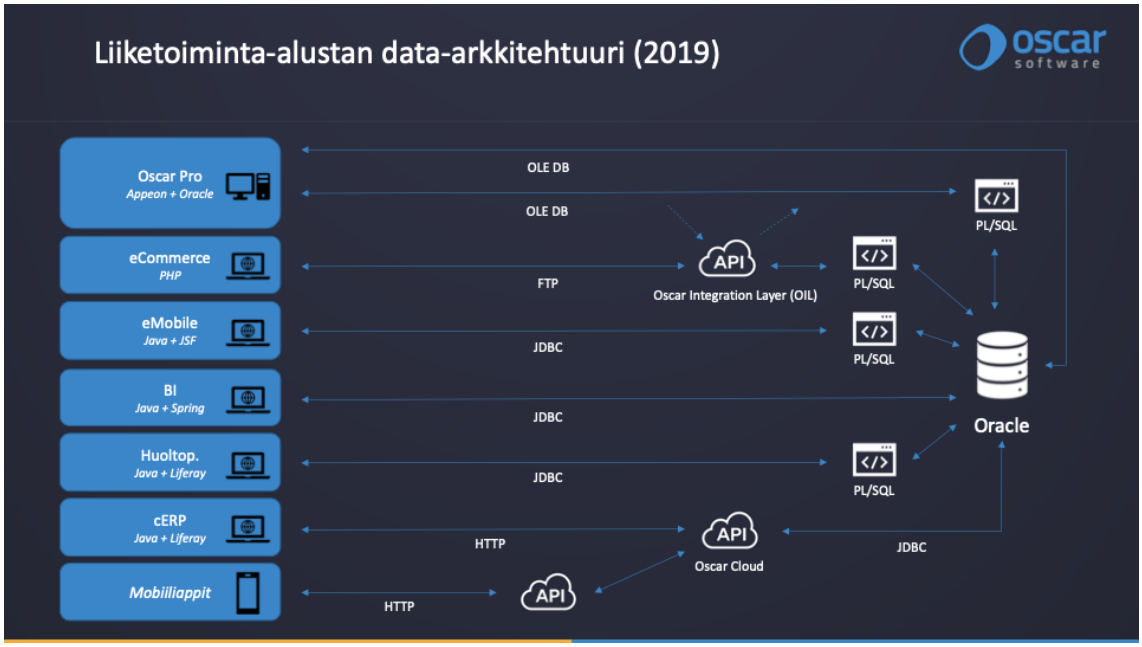
Yli 100 000 organisaatiota käyttää GitLabia ohjelmistokehityksen elinkaaressaan, ja sillä on yli 30 miljoonaa rekisteröityä käyttäjää. GitLab Inc. työllistää yli 1300 työntekijää 67:stä eri maasta. GitLabiin on tarjolla erilaisia maksullisia tilauksia, jotka tarjoavat enemmän ominaisuuksia avoimen lähdekoodin versioon verrattuna. (GitLab n.d.)

3.4 Bamboo

Bamboo on Atlassianin kehittämä jatkuvan integraation, jakelun ja julkaisun ohjelmisto. Sen ominaisuuksia ovat ohjelmiston automaattinen rakentaminen, testaaminen ja jakelu. Se tarjoaa lukuisia integraatioita muihin ohjelmiin, mukaan lukien esimerkiksi Jira-tehtävienhallintatyökaluun ja BitBucket-versionhallintatyökaluun. Bamboo tarjoaa siirtymätyökalun Jenkinsin käyttäjille, jolla Jenkinsiin luodut työjonot voidaan siirtää automaattisesti Bamboo-ohjelmistoon. (Atlassian n.d.)

4 Oscar Cloud ja cERP

CI/CD-työkalujen kautta jaeltavat ohjelmat olivat Oscar Cloud ja Oscar cERP. Ne ovat osa Oscar Softwaren liiketoiminta-alustaa, joka koostuu useasta eri ohjelmistosta muodostaen yhden ison kokonaisuuden yrityksen toiminnanohjaamiseen. Kuviossa 2 on esitelty osa liiketoiminta-alustan arkkitehtuuria sekä käytettyjä tekniikoita vuodelta 2019.



KUVIO 2. Liiketoiminta-alustan ohjelmistot ja niiden väliset kytkökset (Oscar Software 2019)

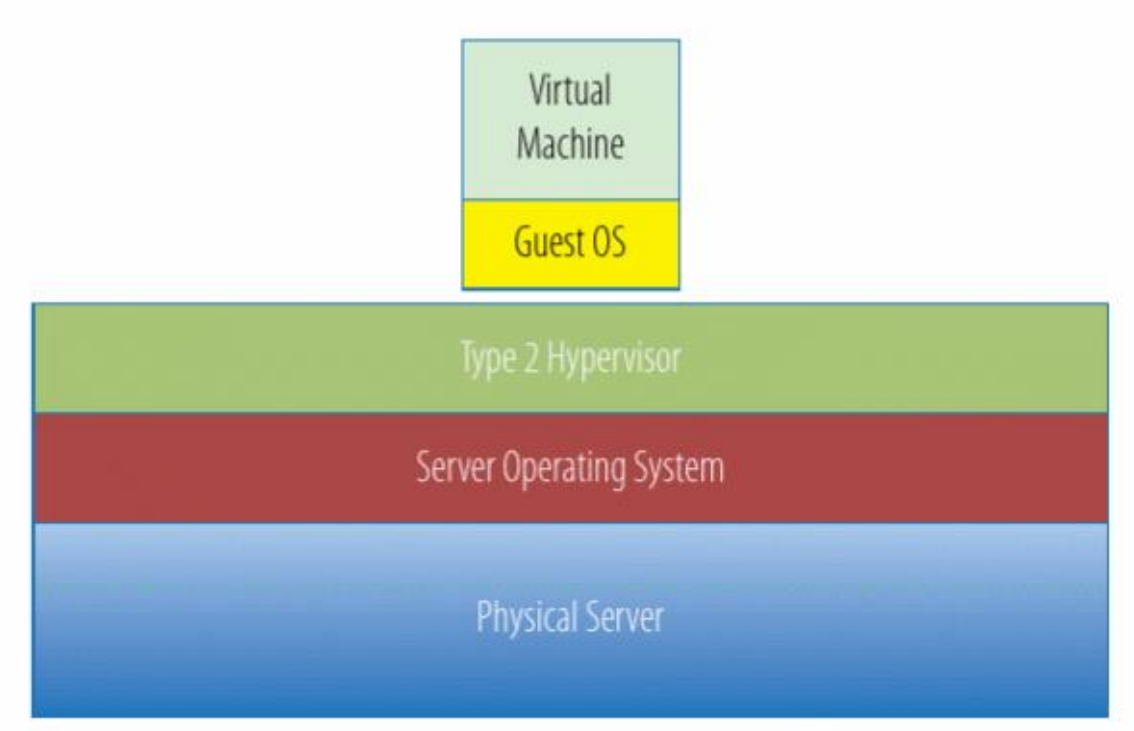
Oscar Cloud on liiketoiminta-alustan taustapalvelin, jossa käsitellään liiketoimintalogiikka sekä tiedon hakeminen ja tallentaminen. Se on rakennettu Spring Framework-kehystä käyttäen Java-ohjelmointikielellä ja sen taustalla toimii Oracle Database-tietokanta.

Oscar cERP on web-käyttöliittymäpohjainen ERP- eli toiminnanohjausjärjestelmä. Sen pohjana toimii Liferay-portaali, johon eri toiminnallisuudet on rakennettu portletteina. Portletit ovat pieniä, muokattavissa ja liikuteltavissa olevia osia Liferay-portaalissa. Liferay käyttää tietokantanaan MySQL:ää, ja Oscar cERP -portletit ovat yhteydessä Oracle Database -tietokantaan Oscar Cloudin välityksellä.

5 Virtualisointi

Virtualisointi mahdollistaa useiden käyttöjärjestelmien ajamisen samalla palvelimella samaan aikaan, pitäen ne kuitenkin erillään toisistaan. Tietokoneiden kehittymisen ja aiemmin käytetyn ”yksi palvelin, yksi ohjelma”-mallin myötä palvelinten kuorma alkoi laskea. Virtualisointi tarjosi tähän ratkaisun, jakamalla yhden fyysisen tietokoneen resurssit useammalle palvelimelle, jolloin tarvittavien palvelinten määrä pieneni. IDC:n mukaan vuonna 2009 virtuaalisia palvelimia asennettiin enemmän kuin fyysisiä. (Portnoy 2016, 9-12.)

Virtuaalisia ympäristöjä voidaan ajaa virtualisointiohjelman eli hypervisorin kautta. Hypervisori on linkki virtuaalisen tietokoneen ja fyysisen tietokoneen resurssien välillä (Portnoy 2016, 21). Hypervisoreita on kahta eri tyyppiä, joista toinen on ilman käyttöjärjestelmää ajettava, ja toinen vaatii käyttöjärjestelmän allensa (Portnoy 2016, 23, 25). Vertailuun valittu Oracle VM VirtualBox hypervisori oli jälkimmäistä tyyppiä, jonka toiminta on havainnollistettu kuviossa 3.

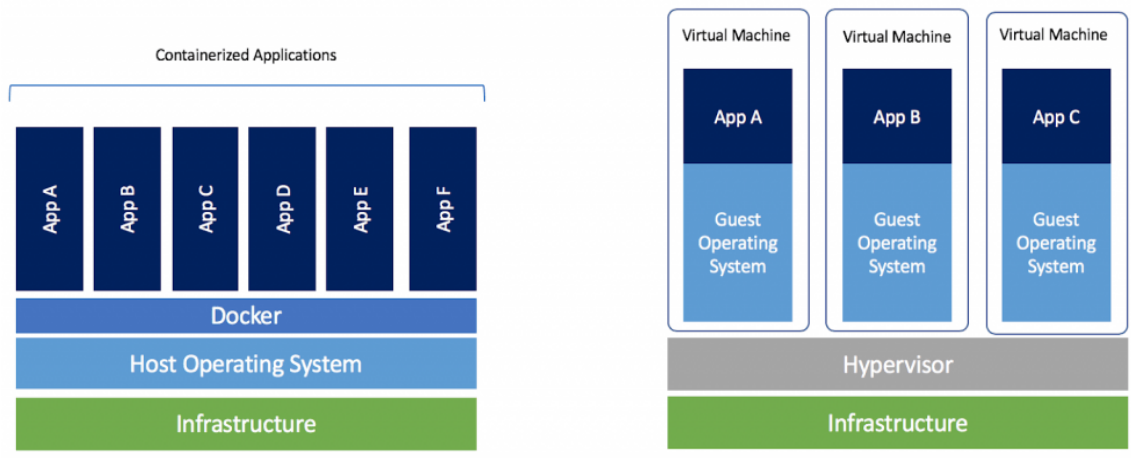


KUVIO 3. Käyttöjärjestelmän päällä ajettava hypervisori (Portnoy 2016, 25)

6 Docker

Docker on alusta siirrettävien ohjelmistojen kehittämiseen, pakkaamiseen ja ajamiseen. Docker yksinkertaistaa ohjelman kehitystä ja ajamista pakkaamalla kaiken ohjelman ajamiseen tarvittavan yhden image- eli näköistiedoston sisälle. Se eroaa muista virtuaalisista ympäristöistä siten, että näköistiedosto sisältää kokonaisen käyttöjärjestelmän sijaan pelkästään ajettavan ohjelmiston. Docker-ohjelmat ajetaan omissa konteissaan (container), jotka ovat eristetty toisistaan sekä Dockeria ajavasta käyttöjärjestelmästä (kuvio 4). (Vohra 2016.)

Docker käyttää Dockerfile-nimistä tiedostoa toiminnassaan. Dockerfile sisältää kaikki ohjeet ohjelman koostamiselle, kuten mitkä tiedostot ladataan, mitkä komennot ajetaan, mitkä portit avataan, mitkä kansiot lisätään tiedostojärjestelmään ja mitkä ympäristömuuttujat asetetaan. Dockerfilessä voidaan määritellä käynnistyskomento, jolloin muodostuvasta näköistiedostosta tulee ajettava ohjelma. (Vohra 2016.)



KUVIO 4. Dockerin toimintaperiaate verrattuna perinteiseen virtualisointiin (Fong 2018)

Dockeria on hyödynnetty Oscar Cloudin jakelussa ja sen käyttöä on suunniteltu cERPissä, joten se otettiin mukaan vertailussa luotaviin työjonoihin.

7 Palvelinympäristö

Vertailtavat työkalut ja muut tarvittavat ohjelmistot asennettiin virtuaalikoneille. Virtuaalisia palvelimia, joilla oli tarkoitus simuloida tuotantoympäristöä, asennettiin viisi kappaletta. Virtualisointiohjelmana käytettiin Oracle VM VirtualBoxia. Kohdetietokoneena toimi pöytäkone, joka on varusteltu 8-ytimisellä Ryzen 7 5800X-prosessorilla ja 32:lla gigatavulla RAM-muistia.

7.1 Virtuaalipalvelinten perusta

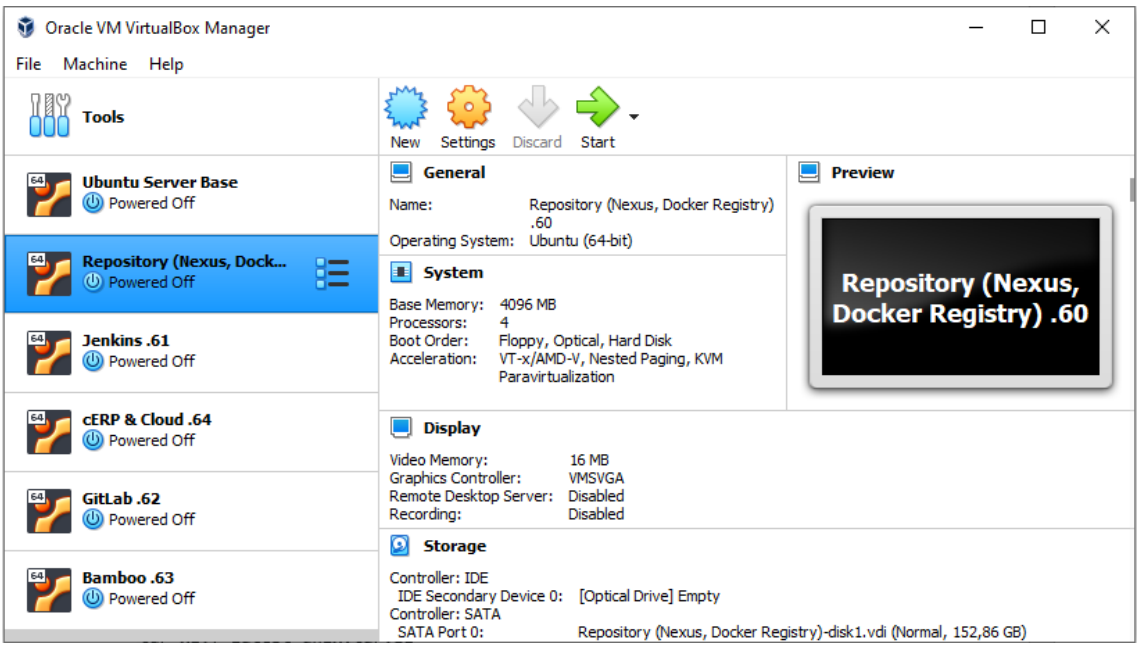
Käyttöjärjestelmä ja perusohjelmat asennettiin yhdelle virtuaalikoneelle, jonka jälkeen virtuaalikonetta monistettiin tarvittava määrä (kuva 1).

Käyttöjärjestelmäksi valittiin Ubuntu Desktop 20.04.2.0. Käyttöjärjestelmän kriteerinä oli olla UNIX-pohjainen, jotta se vastaa tuotantoympäristöä mahdollisimman hyvin. Desktop-versioon päädyttiin, jotta koneisiin saadaan asennettua tarvittava FortiClient VPN. Virtuaalikoneen tarkemmat tiedot esitelty taulukossa 1.

VPN- eli Virtual Private Network -ohjelma mahdollistaa yhteyden muodostamisen yrityksen sisäverkkoon, jolloin saadaan yhteys palvelimiin, jotka on suljettu ulkoverkolta. Tämänlaisia sisäverkossa olevia palvelimia ovat esimerkiksi tietokantapalvelimet, joten VPN on tarpeellinen Oscar Cloud- ja cERP-ohjelmistojen ajamiseen etänäpalvemilla.

Lisäksi virtuaalikoneelle asennettiin OpenSSH-palvelin, joka mahdollistaa komentoriviyhteyden toiselta laitteelta virtuaalikoneeseen.

Edellä kuvailtu kokoonpano toimi pohjana jokaiselle käytetylle virtuaaliselle palvelimelle.



KUVA 1. Kuvankaappaus Oracle VM VirtualBox Manager-ohjelmasta

TAULUKKO 1. Suunnitellut yksittäisen virtuaalisen koneen tekniset tiedot perustettavaan palvelinympäristössä

Käyttöjärjestelmä	Ubuntu Desktop 20.04.2.0. 64-bit
Varattujen ytimien lukumäärä	4
Varattu keskusmuisti	4096 MB
Varattu tallennustila	Skaalautuva, maksimissaan 50 GB
Internet-yhteys	Sillattu, suoraan yhteydessä reitittimeen

7.2 Palvelimet ja asennusprosessi

Palvelinympäristöllä kuvataan järjestelmää, joka koostuu monesta palvelimesta. Palvelimet jaoteltiin käyttötarkoitusten ja asennettavien ohjelmien mukaan (taulukko 2). Jokaiselle palvelimelle määrättiin staattinen eli muuttumaton IP-osoite. IP-osoite on tietokoneen tai muun internet-päätelaitteen osoite, jolla siihen voidaan olla ulko- tai sisäverkosta yhteydessä.

Ohjelmistojen asennukset suoritettiin SSH- eli Secure Shell-yhteyden avulla. SSH mahdollistaa suojatun komentoriviyhteyden kohdepalvelimelle. Tämä tapa

valittiin, koska palvelimet harvoin sisältävät työpöytäympäristöä, ja sillä saatiin parempi kuva asennusprosessin kulusta oikealle palvelimelle.

TAULUKKO 2. Suunnitelma asennetuista palvelimista

Sisäverkon IP-osoite	Tyyppi	Asennettavat ohjelmistot
192.168.1.60	Tietovarastopalvelin	Nexus Repository OSS
192.168.1.61	CI/CD	Jenkins
192.168.1.62	CI/CD	GitLab
192.168.1.63	CI/CD	Bamboo
192.168.1.64	Kohdepalvelin	Docker, Oscar Cloud, cERP, Liferay, MySQL

7.2.1 Tietovarastopalvelin

Ensimmäinen asennettava palvelin oli repository- eli tietovarastopalvelin. Tämä tietovarastopalvelin toimi välityspalvelimena julkisen keskustietovarastopalvelimen ja CI/CD-ohjelmien välillä, säilöen haettavat riippuvuudet itseensä. Näin riippuvuudet haettiin Internetistä ainoastaan ensimmäisen rakennusvaiheen yhteydessä, ja jatkossa ne löytyivät paikallisesta ympäristöstä, säästäen aikaa. Tietovarastopalvelin toimi myös säilytyspaikkana omille kirjastoille, joita työjonoissa luotiin. Lisäksi tietovarastopalvelimelle säilöttiin Docker-näennäistiedostot omaan rekisteriinsä.

Tietovarasto-ohjelmistoksi valittiin Nexus Repository OSS. Sen asennus aloitettiin lataamalla ohjelman paketti palvelimelle. Paketti purettiin omaan hakemistoonsa, jonka jälkeen ohjelma linkitettiin palveluhakemistoon, joka mahdollisti ohjelman ajamisen service-komennon avulla. Lopuksi ohjelmalle lisättiin käynnistysskripti, jolloin ohjelma käynnistyy automaattisesti tietokoneen käynnistyessä.

```
oscar@oscar-VirtualBox:/opt$ cd /opt
oscar@oscar-VirtualBox:/opt$ sudo wget -q https://download.sonatype.com/nexus/3/latest
-unix.tar.gz
oscar@oscar-VirtualBox:/opt$ sudo tar xzf latest-unix.tar.gz
oscar@oscar-VirtualBox:/opt$ sudo ln -s \
> /opt/nexus-3.30.0-01/bin/nexus /etc/init.d/nexus
oscar@oscar-VirtualBox:/opt$ cd /etc/init.d
oscar@oscar-VirtualBox:/etc/init.d$ sudo update-rc.d nexus defaults
```

KUVIO 5. Nexus Repositoryn asentamisessa käytetyt komennot

Asennuksen jälkeen ohjelman käyttöliittymä oli käytettävissä selaimen avulla oletusportin 8081 kautta.

Tietovarastopalvelin sisälsi automaattisesti välityspalvelimen ja rekisterin Maven-kirjastojen ylläpitoon. Maven on työkalu Java-pohjaisten projektien rakentamiseen ja hallintaan (Apache Maven Project n.d.).

Näköistiedostoja varten tietovarastopalvelimelle luotiin Docker-rekisteri. Rekisterin luominen onnistui Nexus-palvelun käyttöliittymästä Repositories-kohdan alta. Rekisterin luomisen vaihtoehtoja olivat group, proxy ja hosted. Group yhdistää alleen monta rekisteriä ja hakee näköistiedostoja kaikista niistä, kun taas proxy toimii välityspalvelimena säilöen keskustietovarastosta haetut näköistiedostot. Rekisterin tyypiksi valittiin hosted eli isännöity, jolloin rekisteriä käytetään ainoastaan omien näköistiedostojen säilömiseen.

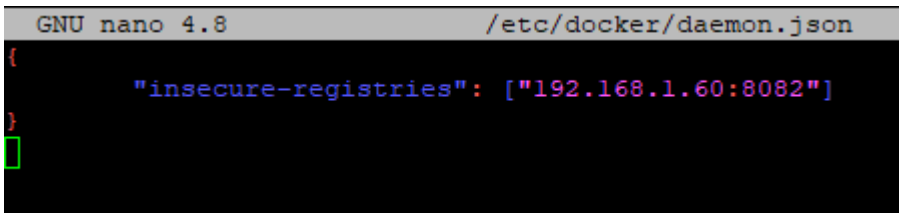
7.2.2 Kohdepalvelin

Kohdepalvelimelle asennettiin Docker, jonka kautta Oscar Cloud- ja cERP-ohjelmia ajetaan. Lisäksi Docker asennettiin myös CI/CD-palvelimille, koska se vaaditaan Docker-näköistiedostojen luontiin. Asentamisessa käytettiin Dockerin virallista asennusskriptiä, joka ladattiin *curl*-komentoa käyttämällä ja ajettiin *sh*-komentoa käyttämällä kuviossa 6 esitellyllä tavalla.

```
oscar@oscar-VirtualBox:~$ curl -fsSL https://get.docker.com -o get-docker.sh
oscar@oscar-VirtualBox:~$ sudo sh get-docker.sh
# Executing docker install script, commit: 7cae5f8b0decc17d6571f9f52eb840fbc13b2
737
```

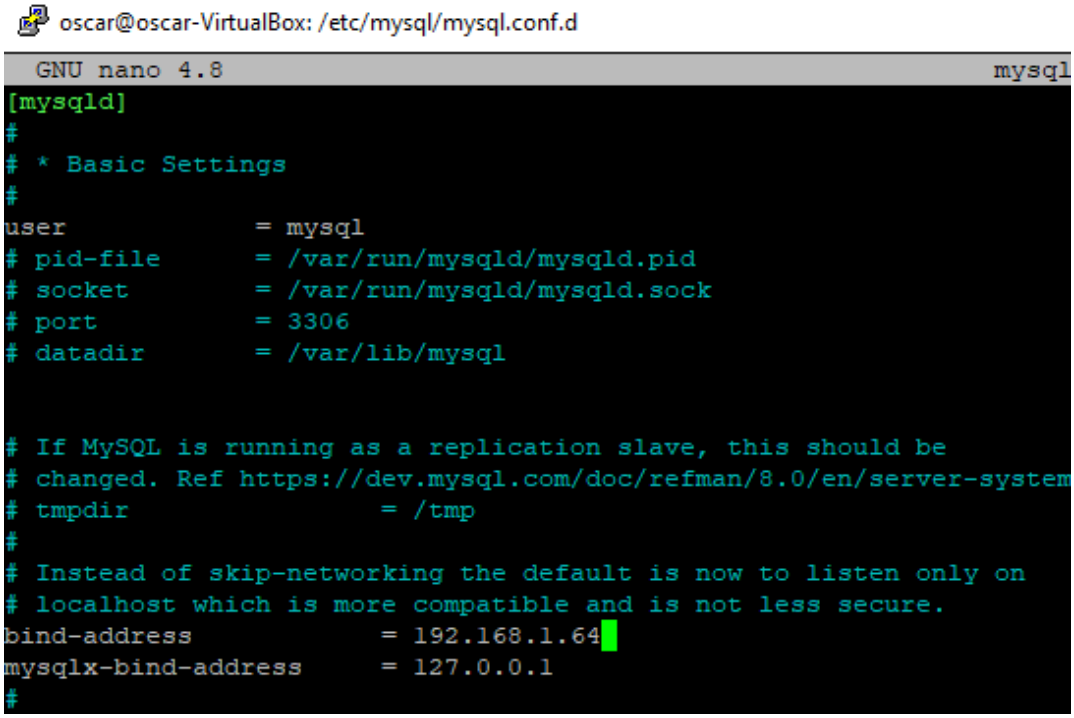
KUVIO 6. Dockerin asennuksessa käytetyt komennot

Tietolähdepalvelin asennettiin ilman salaussertifikaatteja, joten Dockerin asetuksiin täytyi tehdä poikkeus, jotta salaamatonta yhteyttä voidaan käyttää pakettien lataamisessa palvelimelle ja palvelimelta. Hakemistoon */etc/docker/* luotiin tiedosto nimeltä *daemon.json*, jolle annettiin kuvan 2 mukainen sisältö.



KUVA 2. Kuvankaappaus nano-tekstinkäsittelyohjelmasta *daemon.json*-tiedosto avattuna

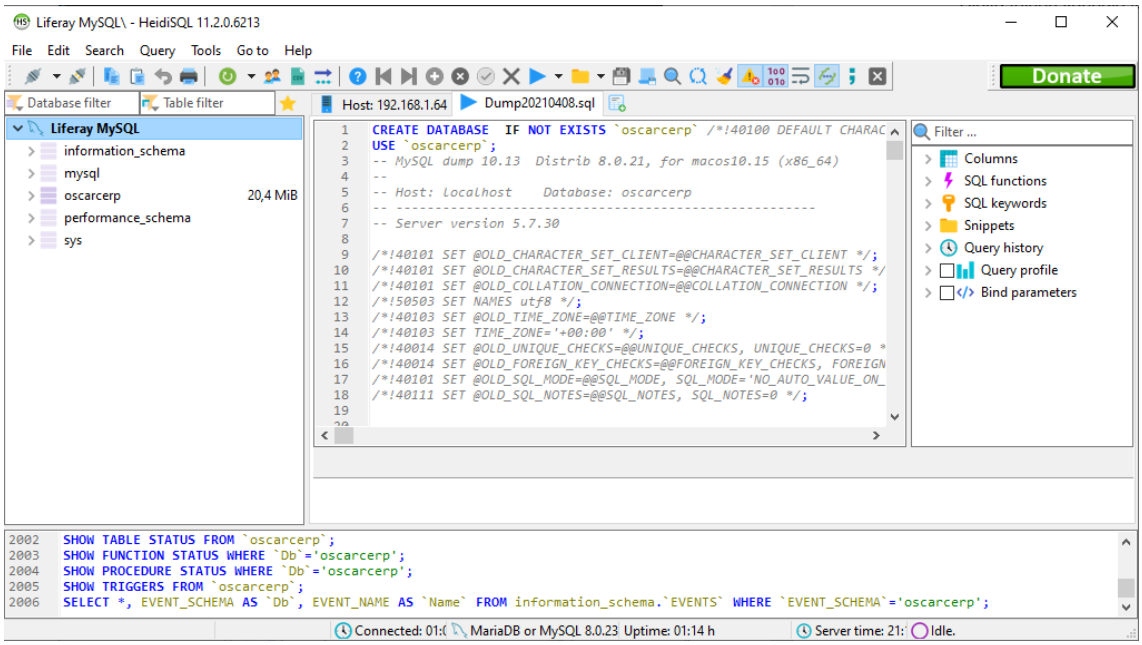
Liferay-portaali vaatii toimiakseen MySQL-tietokantapalvelimen, joten se asennettiin kohdepalvelimelle komennolla *sudo apt install mysql-server*. MySQL-palvelimelle tehtiin asetukset ajamalla komento *sudo mysql_secure_installation*, jonka kautta asetettiin salasana pääkäyttäjälle. MySQL-serverin asetustiedostoon käytiin vaihtamassa bind-address IP-osoite. Kyseinen kohta määrittää, mihin osoitteeseen yhteyden muodostus on mahdollista. Asettamalla se virtuaalikoneen lähiverkon IP-osoitteeksi, mahdollistettiin yhteys MySQL-kantaan kaikilta lähiverkon tietokoneilta. Asetusten muokkaamisen jälkeen palvelin käynnistettiin uudestaan kommenolla *sudo systemctl restart mysql*.



KUVA 3. MySQL-palvelimen asetustiedosto Nano-tekstinkäsittelyohjelmassa

Oscar cERPin toimintaa varten Liferay-portaaliin täytyy tehdä lukuisia asetuksia, joten asennusprosessin helpottamiseksi koko Liferayn tietokanta kopioitiin työtietokoneen paikallisesta kehitysympäristöstä MySQL Workbench-ohjelman Data

Export-työkalua käyttäen. Uuteen tietokantapalvelimeen otettiin yhteys HeidiSQL-nimisellä ohjelmalla ja export-työkalusta saatu tietokantatiedosto avattiin ja suoritettiin, luoden uuden taulun kohdepalvelimen tietokantaan Oscar cERP-ohjelmaa varten (kuva 4).



KUVA 4. Kuvankaappaus HeidiSQL-ohjelmasta taulun luonnin jälkeen

7.2.3 Jenkins-palvelin

Jenkinsin asennus suoritettiin lisäämällä Jenkinsin tietolähde Linuxin paketinhallintajärjestelmän tietolähteisiin ja asentamalla se apt-get-paketinhallintatyökalua käyttäen (kuvio 7).

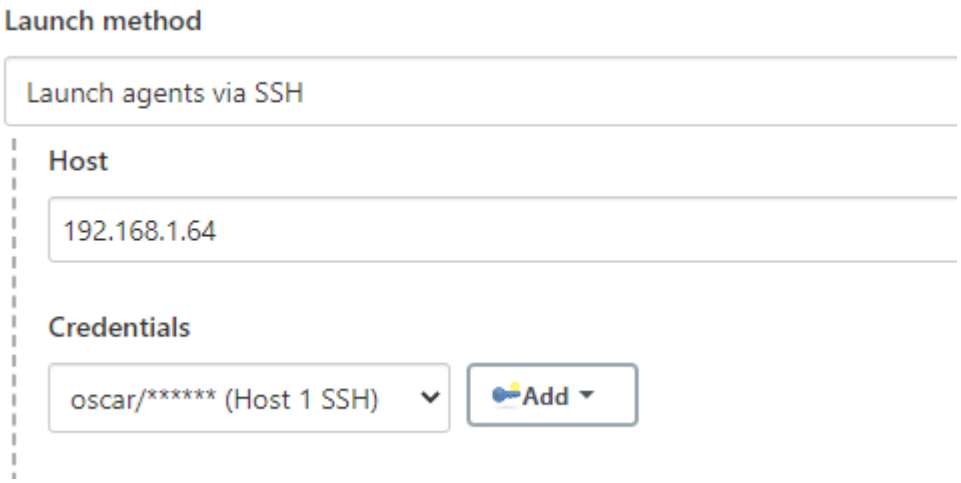
```
oscar@oscar-VirtualBox:~$ wget -q -O - \
> https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
OK
oscar@oscar-VirtualBox:~$ sudo sh -c 'echo deb \
> https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
oscar@oscar-VirtualBox:~$ sudo apt-get -y -qq update
oscar@oscar-VirtualBox:~$ sudo apt-get -y install jenkins
```

KUVIO 7. Jenkinsin asentamiseen käytetyt komennot

Asennuksen jälkeen Jenkins käynnistyi automaattisesti. Ohjattuun asennukseen siirryttiin selainta käyttäen. Ohjattu asennus oli suojattu salasanalla, joka noudet-

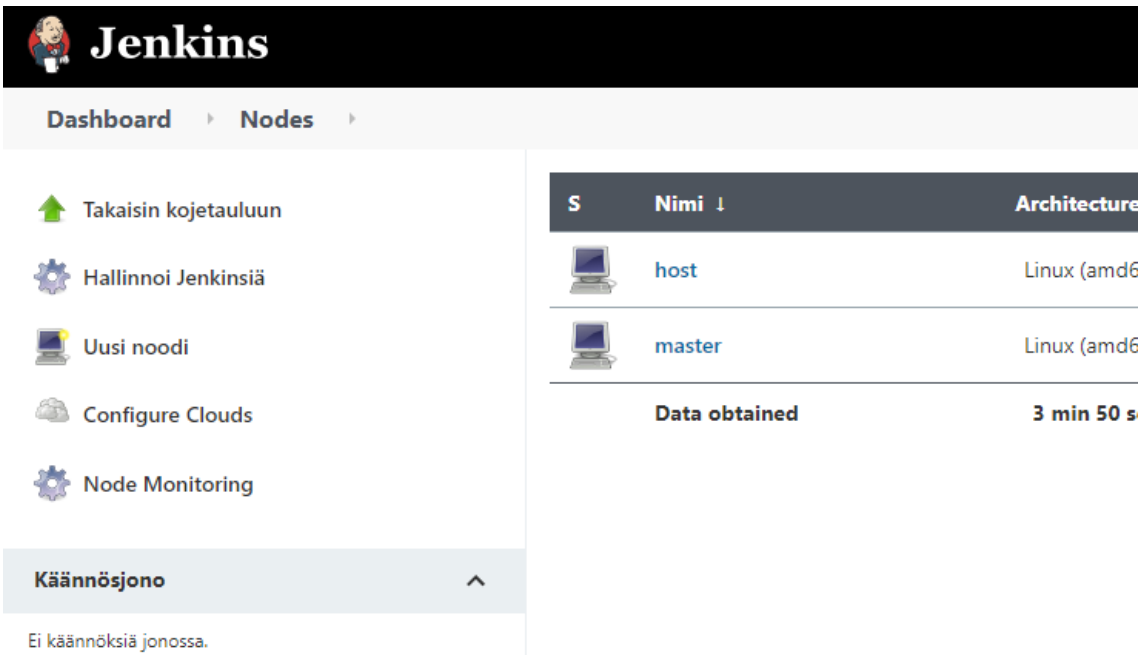
tiin palvelintietokoneen */var/lib/jenkins/secrets/*-hakemistosta asennuksen jatkamiseksi. Ohjatussa asennuksessa asennettiin suositellut lisäosat ja luotiin ensimmäinen ylläpitäjä.

Jenkinsiin luotiin uusi suorittaja, jotta Jenkinsin kautta komentojen ajaminen kohdepalvelimella onnistuu. Suorittajan käynnistystyypiksi valittiin SSH (kuva 5).



KUVA 5. Kuvankaappaus suorittajan luomisesta.

Suorittajan asetuksiin annettiin myös muita tietoja, kuten mm. labelit eli tunnukset ja suorittajien määrä kohdetietokoneella. Tunnuksien avulla Jenkins-komentosarjoja suoritettaessa voidaan vaikuttaa, millä tietokoneella komennot halutaan suorittaa. Tässä tapauksessa uudelle suorittajalle annettiin tunnukseksi host1. Asetukset tallennettiin ja Jenkins yhdisti kohdetietokoneeseen. Jenkins yritti asentaa varsinaisen suorittajaohjelman kohdetietokoneelle automaattisesti, mutta kaatui tarvittavien käyttöoikeuksien puuttuessa. Kohdetietokoneen */var*-hakemistoon tehtiin kansio nimeltä *jenkins*, ja sen omistajuus siirrettiin oscar-käyttäjälle *chown*-komentoa käyttäen, jonka jälkeen suorittaja käynnistyi onnistuneesti.



KUVA 6. Kuvankaappaus Jenkinsin Nodes-välilehdeltä näkyvistä suorittajista

Jenkinsiin lisättiin GitHub-tunnukset valmiiksi, jotta niitä voitiin käyttää työjonoissa. Hallintapaneelissa tunnuksia hallittiin Credentials-sivun alla. Tunnuksia voi olla joko yksilöllisiä, tai yleisiä, kaikkien käytössä olevia. Tunnukset voivat olla käyttäjänimi ja salasana yhdistelmiä, tiedostoja, sertifikaatteja tai generoituja avaimia. Asettamalla tunnukset Jenkinsin tunnustenhallintaan voidaan parantaa tietoturvaa jättämällä ne pois itse työjonoista sellaisenaan. GitHub-tunnukset lisättiin yleisenä käyttäjätunnus ja salasana yhdistelmänä, mahdollistaen niiden käytön työjonoissa käyttäjästä riippumatta.

7.2.4 GitLab

GitLab-ohjelmasta on saatavilla maksullinen Enterprise Edition-versio, ja ilmainen Community Edition-versio, joista jälkimmäinen valittiin vertailuun. GitLab asennettiin hakemalla riippuvuudet paketinhallintaohjelman. GitLabin tietolähteen lisäämistä varten GitLabin sivuilla on skriptitiedosto, joka ladattiin ja ajettiin. GitLab asennettiin *apt-get*-komentoa käyttäen. Käytetyt komennot kuvattu kuvioissa 8, 9 ja 10.

```
oscar@oscar-VirtualBox:~$ sudo apt-get install curl openssh-server
ca-certificates tzdata perl
```

KUVIO 8. Riippuvuuksien hakuun käytetty komento.

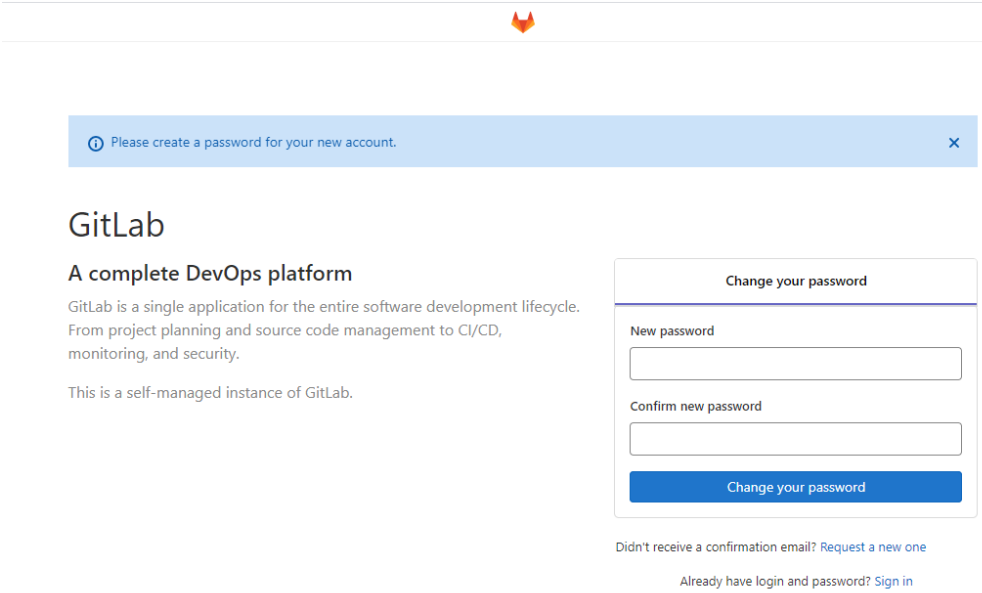
```
oscar@oscar-VirtualBox:~$ curl -sS https://packages.gitlab.com/install/
repositories/gitlab/gitlab-ce/script.deb.sh | sudo bash
```

KUVIO 9. GitLabin tietolähdeskriptin lataamiseen ja asentamiseen käytetty komento.

```
oscar@oscar-VirtualBox:~$ sudo EXTERNAL_URL="http://192.168.1.62" apt-get
 install gitlab-ce
```

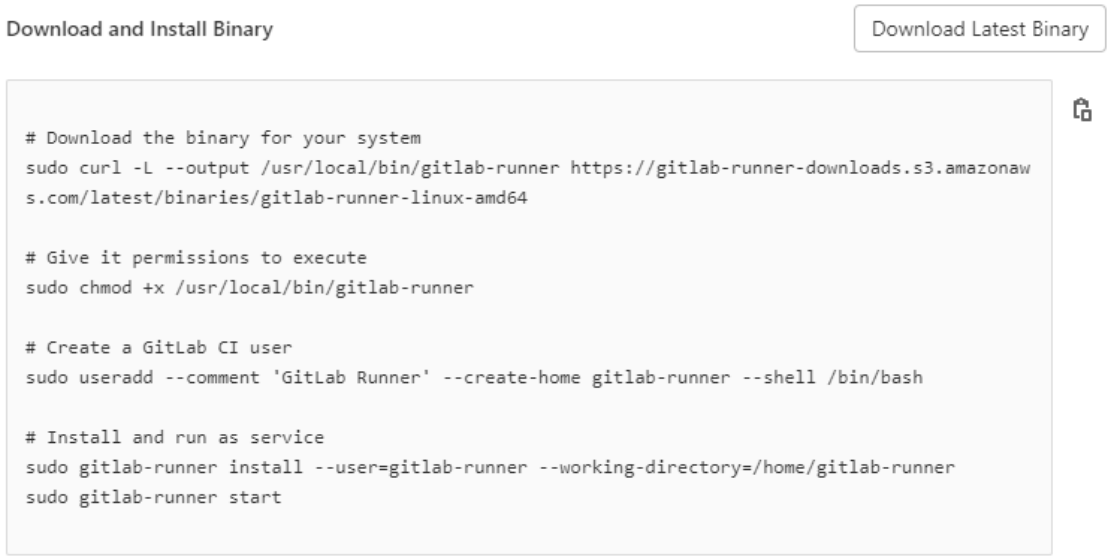
KUVIO 10. Komento GitLabin asennuksen aloittamiseksi.

Navigoimalla selaimella GitLab-palvelimen osoitteeseen päästiin asettamaan pääkäyttäjän salasana ja kirjautumaan sisään.



KUVA 7. Kuvankaappaus GitLabin käyttöliittymästä ensimmäisellä käyttökerralla.

CI/CD-työjonoja varten asennettiin suorittajat GitLab- sekä kohdetietokoneelle. Suorittajien asennuksessa ja rekisteröinnissä käytettiin apuna GitLab-ohjelman tarjoamia ohjeita, jotka on esitelty kuvissa 8 ja 9. Ohjeiden mukaiset komennot ajettiin komentorivin kautta. Rekisteröinnin jälkeen suorittajat tulivat saataville työjonoihin.



KUVA 8. Kuvankaappaus GitLab-ohjelmasta suorittajan asennusohjeista.

KUVA 9. Kuvankaappaus GitLab-ohjelmasta suorittajan rekisteröinnin ohjeista.

Rekisteröinnin yhteydessä kysyttiin, millä asetuksilla suorittajat haluttiin rekisteröidä. Molemmille suorittajille tyypiksi valittiin shell eli komentorivi, joka mahdollistaa komentojen ajamisen suorittajatietokoneessa niillä oikeuksilla, jotka suorittajakäyttäjälle on asetettu. Niille asetettiin selosteet ja tunnukset, jotta ne voidaan yksilöidä niin käyttöliittymässä, kuin työjonoskriptissä.

GitLab-palvelimelle asennettiin myös Docker-näköistiedostojen luontia varten seuraten samoja vaiheita, kuin kohdepalvelimelle asentaessa.

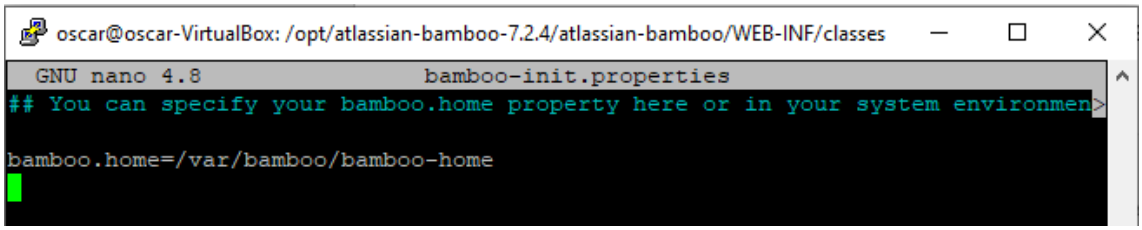
7.2.5 Bamboo

Bamboosta ei ole ilmaista versiota saatavilla, joten vertailuun otettiin 30 päivän kokeilujaksolla Bamboon uusin versio 7.2. Asennus aloitettiin lataamalla Bamboo *tar.gz*-pakettina tietokoneelle sen virallisilta sivuilta (kuvio 11). Paketti ladattiin suoraan */opt*-hakemistoon, jonka alle se haluttiin asentaa.

```
oscar@oscar-VirtualBox:/opt$ sudo curl --location -o /opt/bamboo.tar.gz \
> https://product-downloads.atlassian.com/software/bamboo/downloads/atlassian
-bamboo-7.2.4.tar.gz
```

KUVIO 11. Bamboo-paketin lataamiseen käytetty komento.

Paketti purettiin *tar*-komennolla ja Bamboon sijaitsi tämän jälkeen */opt/atlassian-bamboo-7.2.4*-kansiossa. Bamboolle luotiin kotikansio *mkdir*-komennolla polkuun */var/bamboo/bamboo-home*, johon Bamboo tallentaa käyttämänsä datan. Bamboon asennuspolun alla *atlassian-bamboo/ WEB-INF/classes/*-hakemistossa sijaitsi *bamboo-init.properties*-asetustiedosto, johon asetettiin Bamboolle luotu kotihakemisto kuvan 10 osoittamalla tavalla.



KUVA 10. Kuvankaappaus Bamboon asetustiedostosta Nano-tekstinkäsittelyohjelmassa.

Bamboon asennuskansion omistajuus muutettiin oscar käyttäjälle kuvion 12 mukaisilla komennoilla. Samat komennot ajettiin myös Bamboon kotihakemistoon.

```
oscar@oscar-VirtualBox:/opt$ sudo chown -R oscar atlassian-bamboo-7.2.4/
oscar@oscar-VirtualBox:/opt$ sudo chmod 0755 atlassian-bamboo-7.2.4/
```

KUVIO 12. Omistajuuden ja oikeuksien muuttaminen.

Bamboo käynnistettiin ajamalla *start-bamboo.sh*-skriptitiedosto asennuskansion *bin/*-kansion alta kuviossa 13 kuvaillulla tavalla.

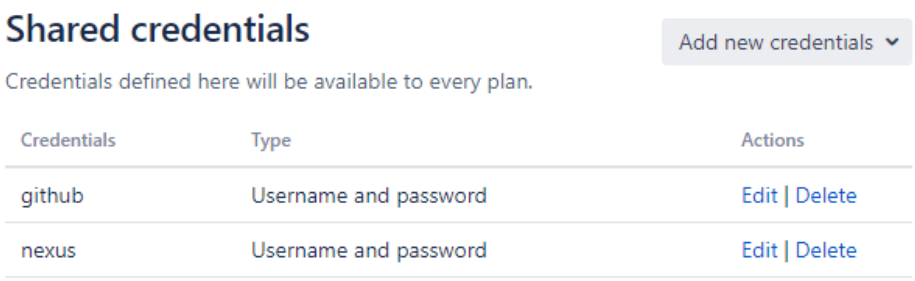
```
oscar@oscar-VirtualBox:~$ cd /opt/atlassian-bamboo-7.2.4/
oscar@oscar-VirtualBox:/opt/atlassian-bamboo-7.2.4$ ./bin/start-bamboo.sh
```

KUVIO 13. Bamboo-palvelun käynnistäminen.

Palvelun käynnistyttyä selaimella navigoitiin palvelimen porttiin 8085 ja eteen aukesi Bamboon asennusnäkymä. Asennusnäkymässä ensimmäisenä kysyttiin

tuotteen lisenssiä, ja tätä varten käytiin generoimassa 30:en päivän kokeilulisenssi Atlassianin sivuilta. Asennukseen tarjottiin kahta vaihtoehtoa, express eli pika-asennus ja custom eli mukautettu asennus, joista valittiin ensimmäinen. Asennuksen aikana luotiin pääkäyttäjä.

Bamboon ylläpitopaneelista käytiin valmiiksi lisäämässä työjonoissa käytettävät kirjautumistiedot. Kirjautumistietojen lisäys tapahtui Shared credentials-välilehden alta (kuva 11), jonne lisätyt kirjautumistiedot ovat saatavilla kaikilla työjonoilla.



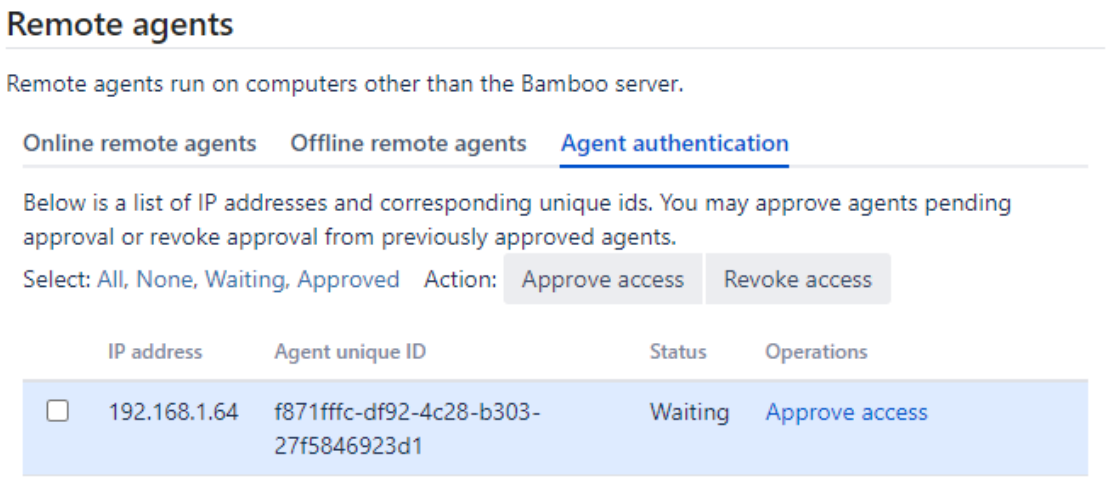
KUVA 11. Kuvankaappaus Bamboon hallintapaneelista, jossa näkyy lisätyt kirjautumistiedot

Kohdepalvelimelle asennettiin agentti eli Bamboon suorittaja. Bamboo-palvelin sisälsi automaattisesti asennuksen jälkeen paikallisen agentin, mutta etäagentti tarvittiin komentojen suorittamiseksi kohdepalvelimella. Agentti asennettiin lataamalla agentin jar-paketti ja suorittamalla se kohdetietokoneella kuviossa 14 esitellyllä tavalla. Jar-pakettia suorittaessa sille annettiin parametriksi Bamboo-palvelimen osoite.

```
oscar@oscar-VirtualBox:~$ curl -q -o bamboo-agent.jar \
> http://192.168.1.63:8085/agentServer/agentInstaller/atlassian-bamboo-agent-installer-7.2.4
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  131M  100  131M    0     0   304M      0 --:--:-- --:--:-- --:--:--  304M
oscar@oscar-VirtualBox:~$ java -jar bamboo-agent.jar http://192.168.1.63:8085/agentServer/
```

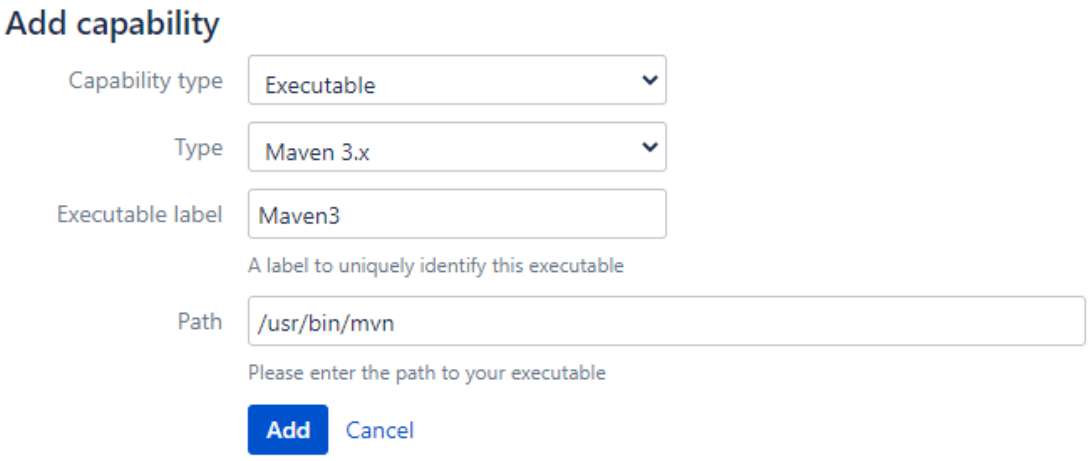
KUVIO 14. Bamboon agentin asentaminen.

Ajon yhteydessä lokiin tuli URL-osoite, jonka kautta agentti oli vielä hyväksyttävä Bamboo ohjelmassa (kuva 12).



KUVA 12. Kuvankaappaus Bamboo-ohjelman hallintapaneelista Agents-välilehdeltä

Bamboo-palvelimelle asennettiin Maven-työkalu pakettien rakentamista varten työjonoissa ja sen asentaminen suoritettiin *apt install maven*-komentoa käyttämällä. Bamboossa palvelimen käytössä olevat ohjelmat käyttävät nimeä capability eli kyky, kuvaten, mitä kaikkea palvelimella voi suorittaa. Bamboon hallintapaneelista Maven käytiin lisäämässä palvelimen kyvyksi Server capabilities-välilehdeltä kuvan 13 osoittamalla tavalla.



KUVA 13. Kuvankaappaus palvelimelle kyvyn lisäämisestä

Palvelimelle ladattiin Mavenin asetustiedosto *settings.xml*, jossa määriteltiin käytettävä välityspalvelin riippuvuuksien hakuun. Asetustiedosto saatiin ocresource-kirjaston tietolähteestä, ja siihen muutettiin paikallisen ympäristön Nexus-palvelimen tiedot. Tiedosto voidaan antaa Mavenille argumenttina sitä ajettaessa, se voidaan sijoittaa käyttäjäkohtaiseen paikalliseen tietolähdehakemistoon, tai se

voidaan sijoittaa Mavenin asennuskansioon, jolloin asetukset ovat voimassa jokaisella ajolla, ellei toisin määritetä. Asetustiedosto siirrettiin palvelimelle Mavenin käyttämään */etc/maven/*-hakemistoon WinSCP-ohjelmaa käyttäen, asettaen asetukset jokaiselle ajolle.

Git-ohjelmalle tehtiin samat toimenpiteet kuin Mavenille kyvyn lisäämiseksi.

8 Työjonojen luonti

8.1 Oscar Cloud

Oscar Cloud koostuu neljästä eri kirjastosta. Oscarcloud-entities kirjastoon on määritelty jokainen entiteetti eli tietue, joka on saatavilla Oscar Cloudin kautta. Ocresource on itse Oscar Cloud-palvelin, jonka kautta tietueita voi hakea. Se on myös riippuvainen ocmailserverclient-kirjastosta, joka hallinnoi sähköpostien lähettämistä. Ocresourceclient on asiakaskirjasto Oscar Cloudia varten, joka helpottaa kutsujen tekemistä Oscar Cloudiin muista ohjelmista. Kaikki kirjastot ovat Java-pohjaisia ja niiden rakentamisessa käytetään Maven-työkalua.

Maven rakentaa ohjelman käyttäen apunaan projektin objektimallia (POM) ja joukkoa laajennuksia (Apache Maven Project n.d.). Projektin objektimalli on tallennettu projektissa tiedostoon *pom.xml*, johon on määritelty kaikki yksittäisen kirjaston tiedot ja riippuvuudet, sekä konfiguraatioita Mavenin lisäosille ja skripteille. Oscar Cloudin kirjastojen tapauksessa Maven hoitaa rakentamisen lisäksi myös jakelun Nexus-tietolähteeseen.

Oscar Cloudin asennus valmisteltiin tekemällä jokaisen paketin tietolähteeseen oma haara tämän opinnäytetyön prosessia varten. Jokaiseen pakettiin on määritelty Maven-lisäosa helpottamaan jakelua, joka hoitaa ohjelmiston rakentamisen ja jakelun Nexus-palvelimelle yhdellä komennolla *mvn deploy*. Jokaiseen pakettiin muutettiin *pom.xml*-tiedostoon jakelulisäosan asetuksiin Nexus-palvelimen osoite vastaamaan prosessissa käytettävää Nexus-palvelimen osoitetta.

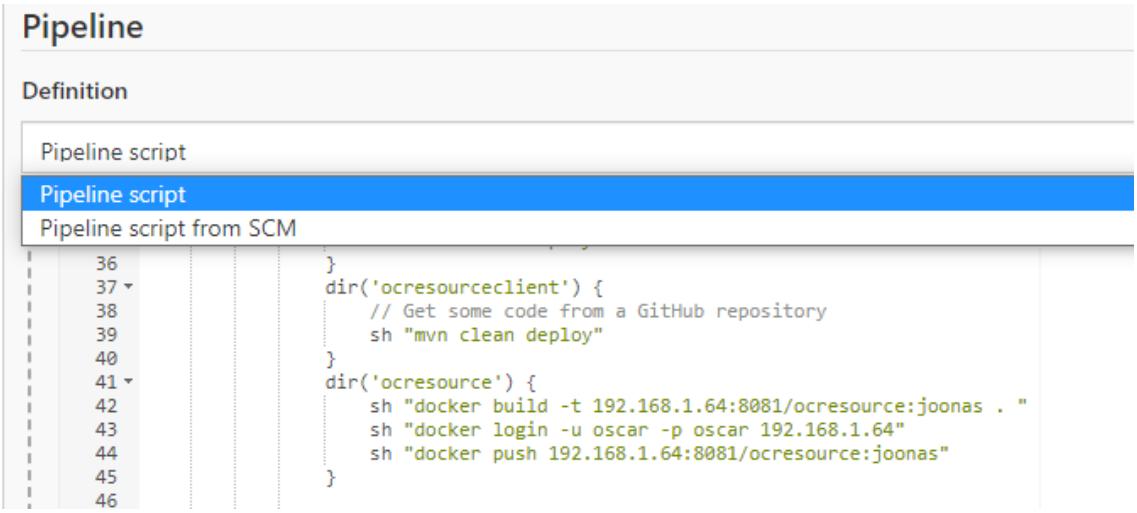
Ocresource-paketin Maven-asetustiedostoon *settings.xml* muutettiin oikea Nexus-palvelin välityspalvelimeksi ja siinä käytettävät tunnukset. Asetusten muuttaminen pelkästään suorittavan tietokoneen Maven-asetuksiin ei riittänyt, koska ocresource-paketin rakennus toteutetaan Docker-kontissa, joka on eristetty isäntäkoneesta.

Jo olemassa olevaa Dockerfile-tiedostoa käytettiin ocresource-paketin osalta. Dockerfile rakensi ocresource-paketin Maven-näköistiedoston sisällä *mvn clean*

package-komennolla, jonka jälkeen rakennettu jar-paketti ja kirjaston tarvitsemat asetustiedostot siirrettiin Java-näköistiedoston sisään. Tästä muodostui uusi näköistiedosto, joka sisälsi kaiken Oscar Cloudin ajamiseen. Se ladattiin Nexus-palvelimelle.

8.1.1 Oscar Cloudin asennus Jenkinsin kautta

Oscar Cloud-kokonaisuutta varten luotiin Jenkinsiin uusi pipeline-tyylinen projekti. Pipeline-projektissa työjono voidaan hakea suoraan tietolähteestä Jenkinsfile-nimisestä tiedostosta, tai kirjoittaa oma työjono itse projektiin. Koska Oscar Cloud on jaettu moneen tietolähteeseen, valittiin tavaksi kirjoittaa oma skripti (Pipeline script kuvassa 14), mahdollistaen kaikkien kirjastojen rakentamisen ja julkaisemisen saman skriptin sisällä.



KUVA 14. Kuvankaappaus Jenkinsin projektieditorista

Työjonossa ainoa käytetty lisäosa oli Git. Sillä haettiin lähdekoodit tietolähteestä, ja kirjastojen rakentaminen toteutettiin ajamalla komento *mvn clean deploy* Jenkinsfilen sh-direktiiviä käyttäen. Kirjastojen rakennusjärjestyksessä huomioitiin keskinäiset riippuvuudet. Pääkirjasto ocresource rakennettiin Dockerin komentoja käyttäen, ja muut kirjastot Mavenilla. Rakennetut kirjastot ladattiin Nexus-tietovarastopalvelimelle.

Git-lisäosa haki tunnukset automaattisesti Jenkinsin tunnuksienhallinnasta annetulla nimellä. Docker-näköistiedostopalvelin vaati tunnistautumisen, joten myös

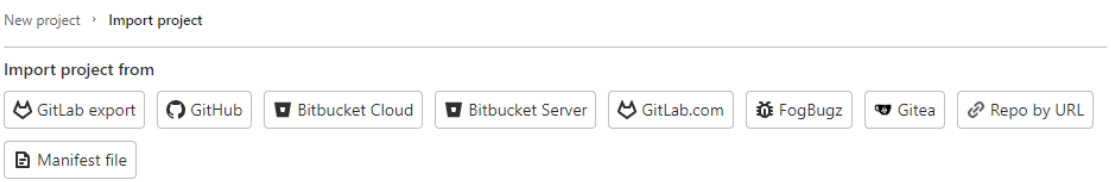
sille luotiin Jenkinsin tunnuksienhallinnassa tunnukset. Tunnukset tuotiin credentials-metodin avulla ympäristömuuttujina työjonoon, jolloin niitä oli mahdollista hyödyntää yksittäisissä komennoissa.

Työjonoa ajettaessa kaikkia riippuvuuksia ei löydetty Mavenin keskustietolähteestä, joten osa riippuvuuksista ladattiin manuaalisesti Nexus-palvelimelle. Manuaalinen lataus tapahtui Nexus-palvelun käyttöliittymän kautta. Nämä puuttuvat riippuvuudet saatiin työkoneen paikallisesta kehitysympäristöstä.

Skriptin viimeinen osuus suoritettiin kohdepalvelimella. Siinä haettiin tietovarastosta juuri luotu Docker-näköistiedosto, joka ajettiin halutuilla parametreillä Docker-kontiksi. Liitteessä 1 esitelty lopullinen Jenkinsin työjonoskripti kommentoituna. Docker-kontin luonnin jälkeen yhteys Oscar Cloudiin testattiin ja todettiin toimivaksi.

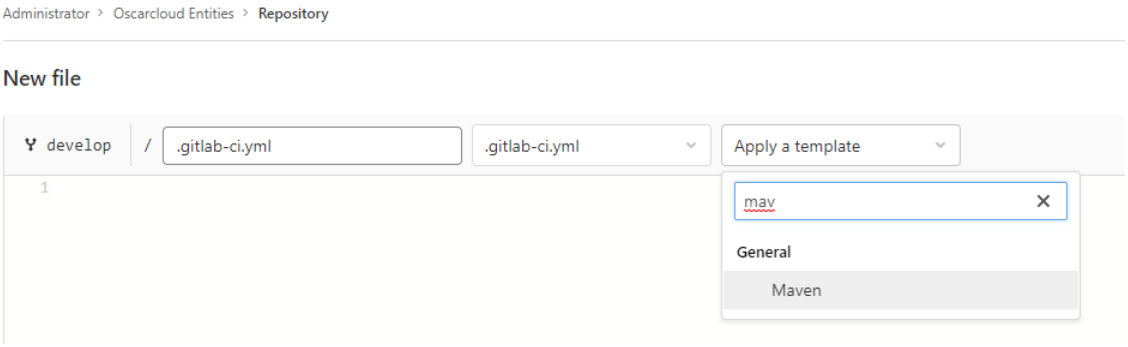
8.1.2 Oscar Cloudin asennus GitLabin kautta

GitLabin CI/CD-työjonot on sidottu projekteihin, toisin kuin Jenkinsissä, joten kaikki Oscar Cloudin kirjastot tuotiin GitLabiin. Tämä onnistui uutta projektia luodessa valitsemalla ”Import project” eli projektin tuonti, joka tarjosi useita vaihtoehtoja siihen, mistä projekti halutaan tuoda (kuva 15). Vaihtoehto GitHub osasi listata ainoastaan henkilökohtaiset projektit GitHubissa, eikä yrityksen projekteja, joten projektit tuotiin Repo by URL -vaihtoehdolla. Tähän vaadittiin kirjaston URL-osoitteen lisäksi myös käyttäjätunnukset, joilla oli oikeus lukea haettavaa kirjastoa. Tuonnin jälkeen jokaisen projektin asetuksista otettiin ”Auto DevOps”-asetus pois päältä, koska työjonot oli tarkoitus luoda itse.



KUVA 15. Kuvankaappaus GitLab-ohjelmasta, projektin tuonti GitLabiin

Työjonojen luonti GitLabiin toteutettiin luomalla jokaiseen projektiin *.gitlab-ci.yml*-tiedosto. GitLabissa kyseisen tiedoston luominen onnistui helposti projektin sisällä Set up CI/CD -painikkeesta, josta päästiin siirtymään suoraan editoriin (kuva 16).



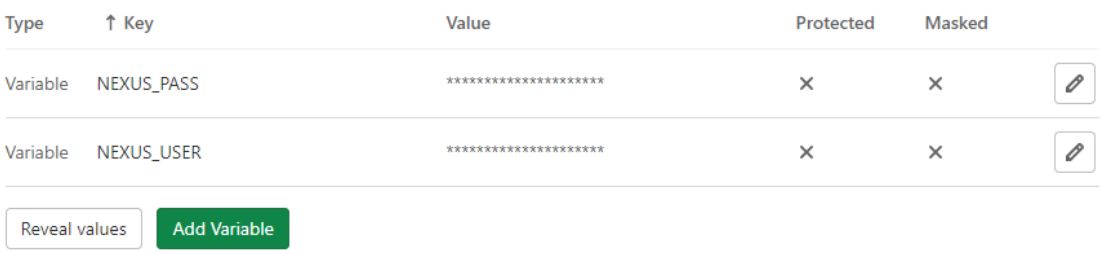
KUVA 16. Kuvankaappaus GitLabin editorista *.gitlab-ci.yml*-tiedoston luonnista

Ocmailserverclient, Oscarcloud Entities ja Ocresourceclient projekteihin luotiin työjonot, jotka olivat identtisiä. Pohjaksi valittiin Maven, jonka kautta saatiin automaattisesti skriptitiedosto sisältäen deploy-vaiheen. Tähän pohjaan jokaisella työlle lisättiin tags-attribuutti, jolla määriteltiin, millä suorittajalla skripti ajetaan. Kirjastot rakennettiin *mvn clean build*-komennolla, kuten liitteessä 2 on esitelty. Työjono määriteltiin suorittamaan automaattisesti aina, kun oletushaaraan tuli muutoksia kussakin projektissa.

Valitulla build-suorittajalla pipeline kaatui virheeseen, koska Maven puuttui kyseiseltä koneelta. Maven asennettiin apt-paketinhallintaohjelman kautta ja jokaiseen projektiin siirrettiin Mavenin käyttämä *settings.xml*-tiedosto, jossa määriteltiin käytettävä Nexus-tietovarastopalvelin kirjastoille. Vastaan tuli myös yleinen shell-tyyppisen suorittajan virhe, jossa *.bash-logout*-tiedostossa sijaitseva komentosarja yrittää tyhjentää konsolin istunnon päätteeksi, aiheuttaen virheen työjonon alustuksessa. Kyseinen tiedosto poistettiin gitlab-runner-käyttäjän kotihakemistosta sekä GitLab-, että kohdepalvelimelta. Korjausten jälkeen työjonoa testattiin tekemällä pieniä muutoksia projektien koodeihin ja työjono todettiin toimivaksi.

Seuraavia työjonoja varten luotiin ympäristömuuttujat GitLabiin, joita Docker käyttää kirjautuessa näköistiedostorekisteriin Nexus-palvelimella. Ympäristömuuttu-

jien avulla kirjautumistiedot saadaan piilotettua itse projektista, sekä työjonon lokista, parantaen tietoturvaa. GitLabissa ympäristömuuttujia pystyy lisäämään projektin, ryhmän tai GitLab-instanssin tasolla. Ympäristömuuttujat päätettiin lisätä instanssin tasolla, jolloin ne ovat käytössä kaikissa kyseisen GitLab-asennuksen projekteissa. Ympäristömuuttujien lisäys tapahtui ylläpitopaneelin asetuksista CI/CD-osiosta.

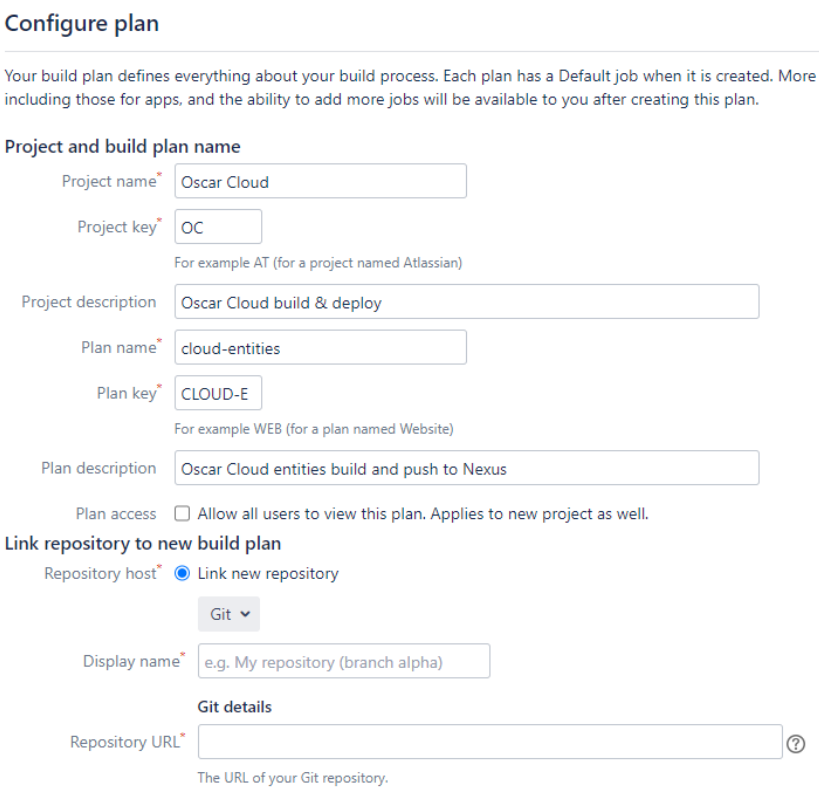


KUVA 17. Kuvankaappaus GitLab-ohjelmasta, ympäristömuuttujien lisäys

Oscar Cloud-projektille tehtiin oma työjono (liite 3), jossa käytettiin Dockeria näköistiedoston luomiseen. Työjonolle tehtiin 3 eri vaihetta, jotka nimettiin build, push ja deploy. Build- ja push-vaiheet suoritettiin GitLab-palvelimella ja deploy kohdepalvelimella. Build-vaiheessa ocresource-kirjasto rakennettiin Dockerin build-komentoa käyttäen. Push-vaiheessa kirjauduttiin sisään näköistiedostorekisteriin käyttäen aikaisemmin luotuja ympäristömuuttujia kirjautumistietoina ja ladattiin luotu näköistiedosto rekisteriin. Deploy-vaiheessa kohdetietokoneella kirjauduttiin Docker-rekisteriin, ladattiin juuri luotu näköistiedosto ja ajettiin se Docker-konttina.

8.1.3 Oscar Cloudin jakelu Bamboon kautta

Bamboossa työjonot ovat nimellä plan eli suunnitelma, ja ne ovat niputettu projektien alle. Ensimmäistä suunnitelmaa luodessa on sille myös luotava projekti (kuva 18).



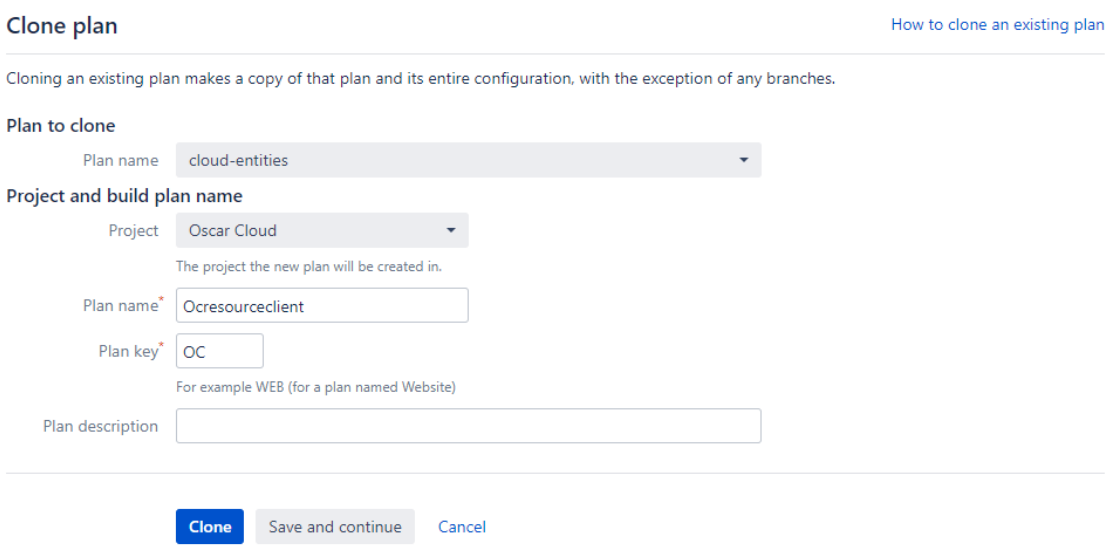
KUVA 18. Kuvankaappaus projektin ja suunnitelman luonnista Bamboossa

Projektille annettiin nimeksi Oscar Cloud, ja sen alle tehtiin suunnitelmat jokaiselle erilliselle kirjastolle. Kuvassa 18 luodaan suunnitelmaa Oscar Cloudin entiteettipaketille. Suunnitelmat linkitettiin jokainen omaan GitHub-tietolähteensä, mikä loi jokaiselle suunnitelmalle automaattisesti ensimmäisen työn (job), jossa haettiin lähdekoodi suunnitelman tietolähteestä.

Vaiheiden alla olevat työt voidaan ajaa rinnan, joten toisistaan riippuvat tehtävät on sijoitettava joko saman työn alle oikeassa järjestyksessä, tai eri vaiheisiin.

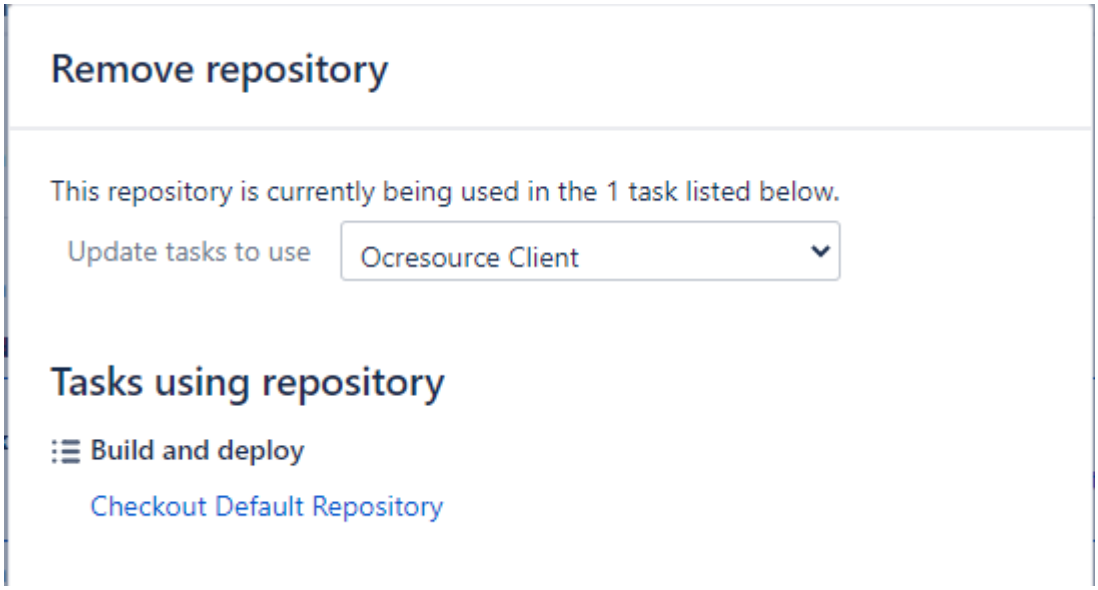
Oscarcloud-entities, ocresourceclient ja ocmailserverclient paketeille tehtiin identtiset suunnitelmat. Suunnitelmiin lisättiin yksi stage eli vaihe, joka nimettiin Build and deploy. Vaiheen ainoa työ sisälsi 2 eri tehtävää, joista ensimmäisessä lähdekoodi haettiin tietolähteestä ja toisessa suoritettiin komento *mvn clean deploy*, joka rakensi paketin ja latasi sen Nexus-tietovarastoon.

Suunnitelma kloonattiin ensimmäisenä luodusta suunnitelmasta muille kirjastoille (kuva 19). Kloonaus tuotti uuden, täysin identtisen suunnitelman.



KUVA 19. Kuvankaappaus suunnitelman kloonaamisesta

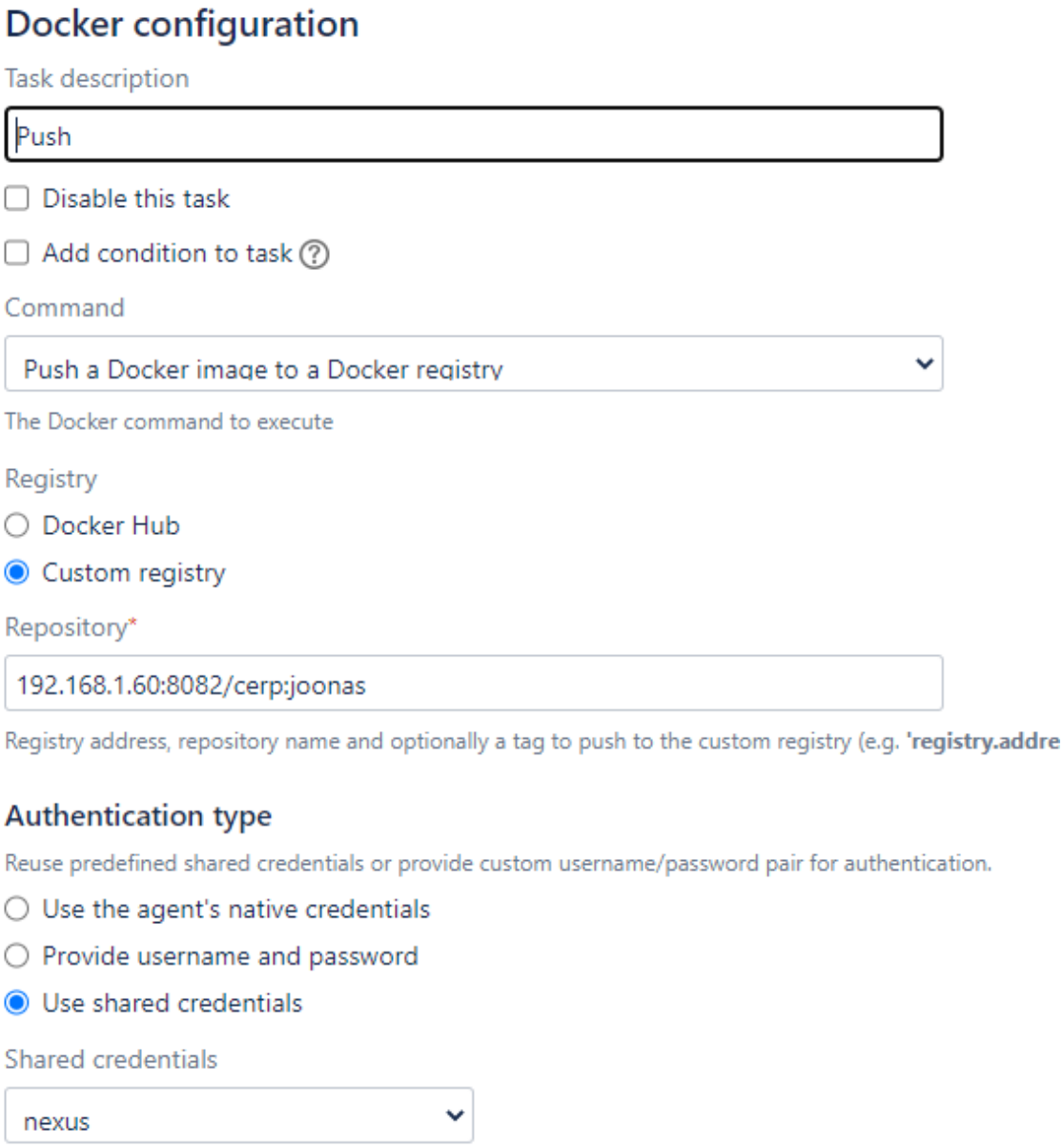
Kloonaamisen jälkeen uusille suunnitelmille vaihdettiin käytettävä tietolähde. Suunnitelman asetuksista Repositories-välilehdeltä lisättiin uusi tietolähde, ja poistettaessa vanhaa tietolähdettä Bamboo huomautti, että vanha tietolähde on käytössä suunnitelman tehtävillä. Tehtäville tarjottiin automaattisesti päivitystä uuteen tietolähteeseen (kuva 20).



KUVA 20. Kuvankaappaus tietolähteen poistamisesta suunnitelmalta

Ocresource-kirjastolle luotiin oma suunnitelma, joka nimettiin Oscar Cloudiksi. Tähän suunnitelmaan luotiin yksi vaihe, jonka nimeksi annettiin Build and push.

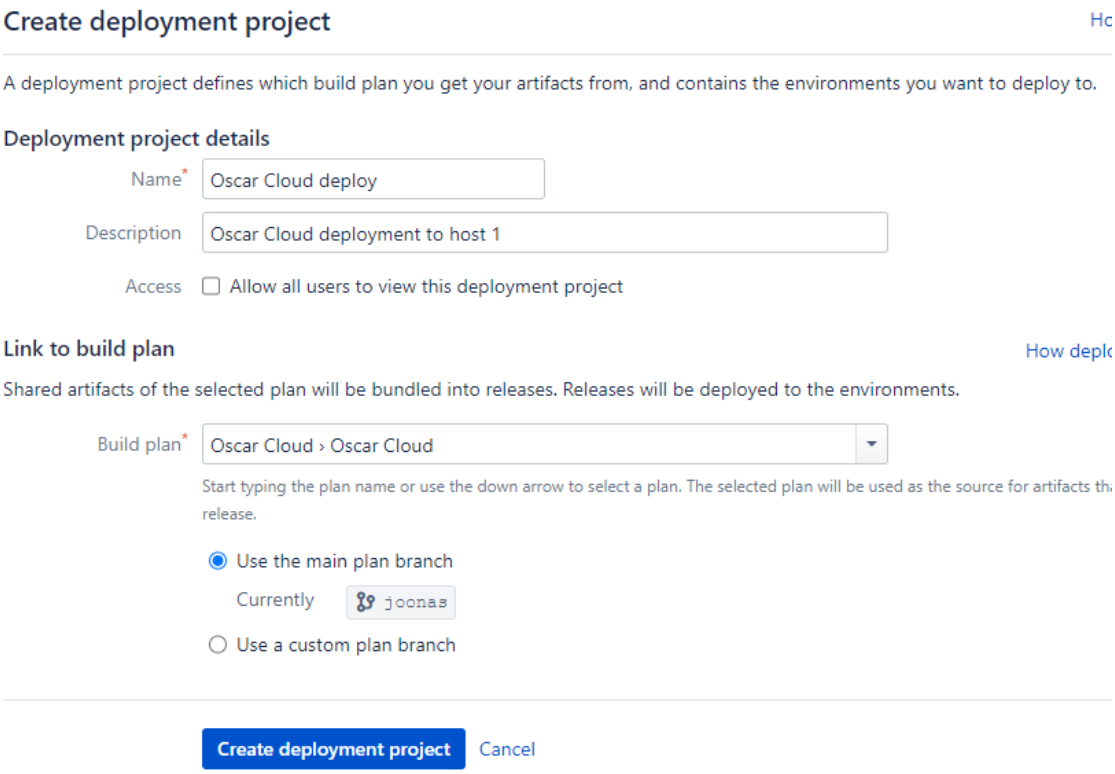
Vaiheen ensimmäisessä tehtävässä haettiin lähdekoodi tietolähteestä tehtävän käyttöön. Toisessa tehtävässä rakennettiin Docker-näköistiedosto ja kolmannessa tehtävässä luotu näköistiedosto ladattiin Docker-rekisteriin. Tehtävät määriteltiin käyttöliittymän kautta. Kuvassa 21 esimerkkinä tehtävä ladata näköistiedosto rekisteriin.



KUVA 21. Kuvankaappaus push-tehtävän asetuksista

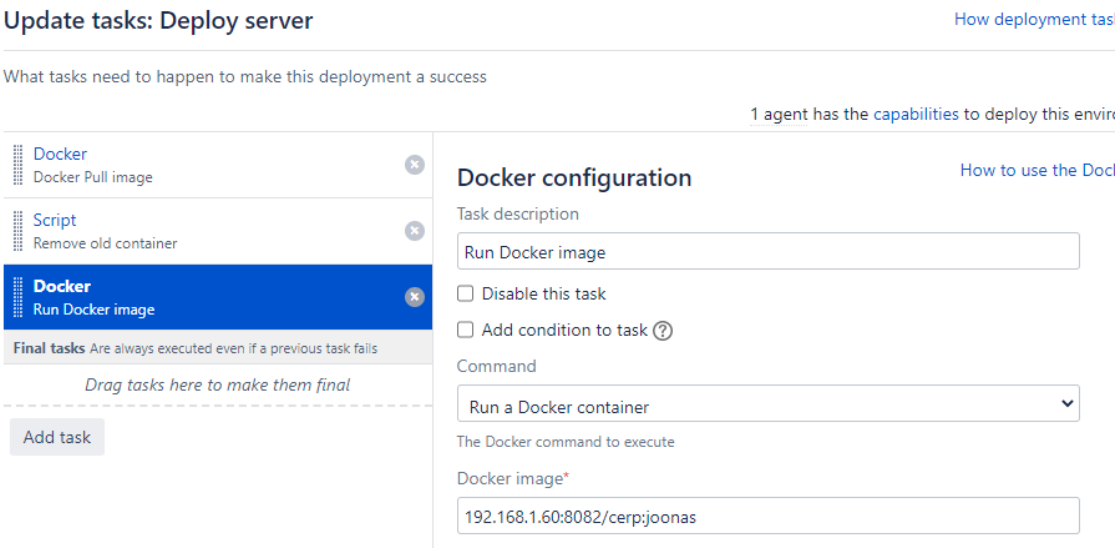
Oscar Cloudin jakelulle luotiin oma deployment- eli jakeluprojektinsa. Jakeluprojektin erona aiemmin käytettyyn rakennusprojektiin on se, että siihen voidaan määrittää eri ympäristöjä, mihin haluttu ohjelma voidaan jakaa. Eri ympäristöille

on myös mahdollista asettaa omat vaiheensa, jolloin esimerkiksi jakelu eri käyttöjärjestelmille on mahdollista toteuttaa yhden jakeluprojektin sisällä. Lisäksi jakeluprojekti voi käyttää rakennusprojektin tuotoksia, mutta koska tuotokset olivat tallennettu Nexus-palvelimelle, sitä ominaisuutta ei tässä käytetty.



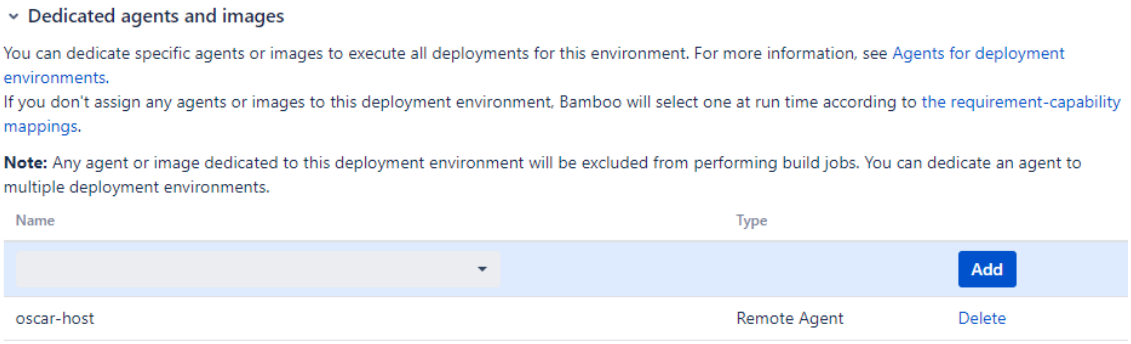
KUVA 22. Kuvankaappaus jakeluprojektin luomisesta

Projektille luotiin uusi ympäristö, jolle annettiin nimeksi Deploy server. Ympäristölle luotiin yksi työvaihe. Vaiheelle luotiin uudet työt, jotka vastasivat tehtäviltään aiemmin luotuja jakeluskriptejä, joissa ensin haettiin Docker-näköistiedosto rekisteristä, pysäytettiin vanha Docker-kontti ja ajettiin uusi kontti halutuilla parametreilla. Tehtävien luonti suoritettiin käyttöliittymän kautta (kuva 23), antamalla tarvittavat tiedot tehtävien suorittamiseen. Docker-tehtävä ei tukenut kontin pysäyttämistä, joten se suoritettiin skriptitehtävällä, joka suorittaa halutun komennon komentorivin kautta.



KUVA 23. Kuvankaappaus jakeluvaiheen tehtävistä

Projektille määritettiin vielä agentti, jolla juuri luotu jakeluvaihe ajetaan. Agentin määrittelyllä varmistettiin, että Docker-kontti ajetaan oikealla palvelimelle. Jakeluprojektin hallintasivulla Agent assignment-painikkeen kautta päästiin agenttien hallintaan, josta avattiin Dedicated agents and images-osio eli projektille omistetut agentit ja näköistiedostot. Haluttu agentti valittiin kuvassa 24 näkyvän Add-painikkeen kautta. Samalla lisätty agentti poistui käytettävistä rakennusprojekteista.



KUVA 24. Agentin valinta jakeluprojektille

Lopuksi palattiin jakeluprojektin päänäkymään, jossa Deploy-painikkeen pudotusvalikosta valittiin Deploy server-ympäristö. Seuraavassa näkymässä nimettiin jakelun nimi ja aloitettiin jakelu. Bamboo ilmoitti jakelun onnistuneeksi, ja se käytiin vielä varmistamassa kohdepalvelimen lokitiedoista.

8.2 cERP

Vertailtaville ohjelmille luotiin komentosarjat cERP-ympäristön asentamiseen halutulle Docker-palvelimelle. Tätä ympäristöä varten oli olemassa valmis Dockerfile, jota käytettiin skriptien sisällä Docker-näköistiedoston luomiseksi. Käytetty Dockerfile sisälsi vaiheet yksittäisten portlet-pakettien luomiseen, Liferay-portaalin hakemiseen ja asetusten asettamiseen, sekä portlet-pakettien siirron Liferay-portaaliin.

Liferay-portaalin asetukset määriteltiin *dev-portal.properties*-tiedostoon. Asetustiedosto sisälsi tietokantayhteyden määrittelyn kohdepalvelimen MySQL-tietokantapalveluun.

8.2.1 cERPin jakelu Jenkinsin kautta

Jenkins ohjelmaan luotiin uusi projekti, jolle annettiin nimeksi Liferay and cERP. Projektin tyypiksi valittiin pipeline. Käytetty työjonoskripti esitelty liitteessä 4.

Työjonon ensimmäisessä vaiheessa projekti haettiin tietolähteestä. Koska ohjelman rakentaminen oli kuvattu kokonaisuudessaan Dockerfile-tiedostossa, haettiin toista vaihetta varten Jenkinsiin Docker Pipeline-liitännäinen. Tämä liitännäinen mahdollisti Docker-näköistiedoston koostamisen ja hallinnan skriptinä komentorivikomentojen sijaan. Kaikki liitännäisen komennot ajettiin *withRegistry*-komennon sisällä, joka asetti kaikille sen sisällä ajettaville komennoille käytettävän tietolähteen niin hakiessa, kuin säilöttäessä Docker-näköistiedostoja. Näköistiedosto luotiin toisessa vaiheessa build-komentoa käyttämällä ja varastoitiin tietolähteeseen push-komennolla.

Työjonon kolmannessa vaiheessa komennot suoritettiin kohdepalvelimella. Komennot ajettiin *withRegistry*-komennon sisällä, mutta *image*-komento ei suoraan osannut tätä huomioida, vaan loi uuden tyhjän näköistiedoston. Tämä korjattiin ajamalla Dockerin pull-komento komentorivin kautta sh-direktiiviä käyttäen. Lopuksi näköistiedosto suoritettiin kontissa *run*-komentoa käyttäen halutuilla parametreillä.

Ongelmaksi tässä kohtaa muodostui Liferayn toimimattomuus, vaikka työjono itsessään oli toimiva. Liferay ei suostunut käynnistymään, koska kyseinen Liferayn versio ei ollut yhteensopiva Java-version kanssa, joka näköistiedostoon ladattiin automaattisesti. Tämä korjattiin muokkaamalla käytettyä Dockerfile-tiedostoa niin, että se haki Javasta yhteensopivan version *apk add*-komennolla, jolloin se automaattisesti korvasi version, joka ei ollut yhteensopiva.

Liferay käynnistyi, mutta kohdattiin uusi ongelma. Liferay ei osannut hakea tietokannasta jo olemassa olevia tietoja, vaan loi uudet tyhjät taulut tietokantaan. Tietokantaa tarkasteltiin, ja huomattiin, että tauluista löytyi duplikaatteja, joista toiset olivat kirjoitettu kokonaan pienillä kirjaimilla ja toiset eivät. Ongelma korjattiin muuttamalla MySQL-palvelun asetuksia niin, että se käytti tauluja käsiteltäessä ainoastaan pieniä kirjaimia. Tämän jälkeen Liferay käynnistettiin uudestaan ja päästiin onnistuneesti käyttämään cERP-ohjelmistoa.

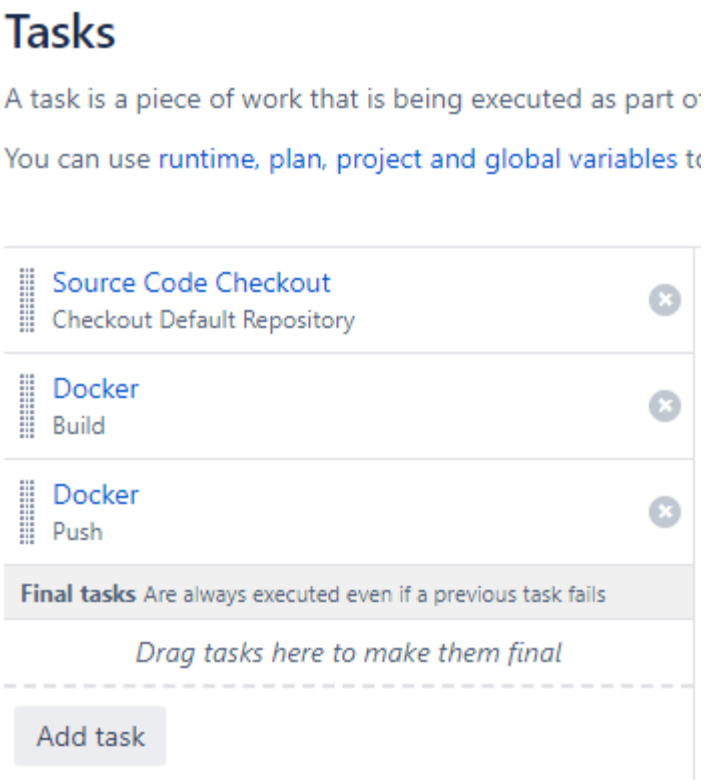
8.2.2 cERPin jakelu GitLabin kautta

Oscar cERP-ohjelman jakeluun vaadittiin samat työvaiheet, kuin Oscar Cloudilla. Oscar Cloudista kopioitiin työjonotiedosto *.gitlab-ci.yml* cERP-projektiin pohjaksi. Työjonoon muutettiin ainoastaan näköistiedoston nimi tarvittaviin kohtiin ja vaihdettiin Docker-kontin ajamiseen käytettävät parametrit. Lopullinen käytetty työjono liitteessä 5.

Työjonoa ajettaessa saatiin virheilmoitus build-vaiheessa, jossa rakennettiin Docker-näköistiedostoa jakelua varten. Virheilmoituksen syyksi selvisi muistin loppuminen. Huomattiin, että pelkästään GitLab vie huomattavan osan muistia ja kyseisen koneen muistin käyttö oli 4,6 gigatavua saatavilla olevasta 6 gigatavusta ennen työjonon suorittamista. Ongelman korjaamiseksi virtuaalikone sammutettiin ja sen muistin määrää nostettiin 8 gigatavuun. Kone käynnistettiin uudelleen ja työjono saatiin ajettua onnistuneesti.

8.2.3 cERPin jakelu Bamboon kautta

Oscar cERP-ohjelmalle luotiin oma rakennusprojekti, jonka nimeksi annettiin cERP. Projektille luotiin uusi suunnitelma nimeltä cERP Deploy, jonka alle automaattisesti luotuun vaiheeseen ensimmäiseen työhön lisättiin tarvittavat tehtävät (kuva 25). Tehtävien järjestys ja rakenne noudatti samaa kaavaa, kuin Oscar Cloud-projektissa.



KUVA 25. Kuvankaappaus cERP Deploy-suunnitelman ensimmäisen työn tehtävistä

Oscar cERPille luotiin uusi jakeluprojekti, joka vastasi Oscar Cloud-paketin jakeluprojektia, ainoana erona käytettävä Docker-näköistiedosto ja Docker-kontin käynnistykseen käytettävät parametrit. Jakeluprojektille vaihdettiin suorittavaksi agentiksi kohdepalvelin.

9 Työkalujen vertailu

9.1 Asennus ja ympäristö

Jenkinsiä tarjotaan ainoastaan asennettava pakettina, eikä siitä ole tarjolla virallista pilvipalvelua. Epävirallisia palveluntarjoajia kuitenkin löytyy Jenkinsille vaihtelevalla hinnoittelulla. Jenkins voidaan asentaa Windows-, Linux- ja macOS-käyttöjärjestelmille, sekä siitä on olemassa virallinen Docker-näköistiedosto Docker-konttina ajamista varten.

GitLab on tarjolla sekä itseasennettavana pakettina, että SaaS- eli Software as a Service-palvelumallina, jolloin hyötynä on säästöt ylläpito- ja päivitystöistä aiheutuvista kustannuksista. GitLab on tuettu Unix-pohjaisilla käyttöjärjestelmillä, mukaan lukien macOS. (O'Grady, 2018)

Bamboo on tarjolla itseasennettavana pakettina, ja se on tuettu Windows-, Linux- ja macOS-käyttöjärjestelmillä. Atlassian tarjoaa SaaS-palvelumallilla kuitenkin toista automaatiopalvelua nimeltä BitBucket Pipelines (Bitbucket, n.d.).

Ajallisesti mitattuna ohjelmistojen asentamiseen meni lähes saman verran aikaa, ja jokaisessa asennusprosessi kohdattiin ongelmia, jotka saatiin kuitenkin ratkaistua.

9.2 Hinnoittelu

Jenkins on täysin ilmainen työkalu asentaa ja käyttää. Jenkins vaatii kuitenkin oman palvelimen, joten todellinen kustannus tulee palvelimen ylläpidosta ja huollosta, sekä päivityksistä.

GitLab-ohjelmasta löytyy ilmaisen version lisäksi 2 eri maksullista versiota, Premium ja Ultimate. Premium-versio kustansaa 19 dollaria kuukaudessa per käyttäjä ja Ultimate versio 99 dollari kuukaudessa per käyttäjä. Premium-versio on suunnattu pienemmille tiimeille, kun taas Ultimate isommille organisaatioille.

Bamboon hinnoittelu menee agenttien määrän mukaan. Edullisin vaihtoehto maksaa 10 dollaria, joka sisältää vain paikalliset agentit ja maksimissaan 10 työtä. Yhdellä etäagentilla varustettu lisenssi kustantaa 1500 dollaria, viidellä 4000 dollaria ja loput siitä ylöspäin. Maksu on lisenssistä, eli se on kertaluontoinen.

9.3 Työjonojen luonti ja muokkaaminen

Jenkins ohjelmassa käytettiin Jenkinsfile-skriptitiedosta työjonojen luontiin, mutta tarjolla on myös muita menetelmiä. Esimerkkinä Jenkinsin virallinen lisäosa Blue Ocean, joka on yksi monista lisäosista, jotka tarjoavat käyttöliittymän työjonojen luomiseen ja muokkaamiseen. Jenkinsfile kuitenkin tarjoaa eniten säätövaraa, ja voi siten olla tehokäyttäjälle parempi vaihtoehto.

GitLabissa työjonot ovat täysin *.gitlab-ci.yml*-tiedoston varassa, joka muistuttaa hyvin paljon Jenkinsissä käytettyä Jenkinsfileä. Käyttöliittymäpohjaisen työjonomuokkaimen puuttuminen voi aiheuttaa haasteita kokemattomille käyttäjille, sillä on turvauduttava jatkuvasti dokumentaatioon. Työjonot ovat projekteihin sidonnaisia.

Bamboon työjonojen luonti ja muokkaaminen tapahtui kokonaan käyttöliittymän kautta. Yksinkertaisten tehtävien lisäämistä helpotti editorin valmiit valinnat ja syötekentät, mikä poisti tarpeen dokumentaatioon tukeutumiselle. Heikkoutena GitLabiin ja Jenkinsiin verrattuna on se, että koko työjonon tarkastelu on haasteellisempaa, vaatien hyppimistä paikasta toiseen.

Ajallisesti mitattuna nopein ohjelma työjonon luontiin tässä opinnäytetyössä oli GitLab. On kuitenkin huomioitava, että ensimmäisten työjonojen luomiseen Jenkinsiin ei ollut kunnollista pohjaa, kun taas GitLabille ja Bamboolle tehdessä vaiheet olivat jo selvillä.

9.4 Työjonojen suorittaminen ja lokit

Työjonojen suorittamiseen kuluneet ajat eivät ole suoraan verrannollisia, koska lopulliset työjonot olivat hieman erilaisia. Työjonojen vaiheet ja lopputulos oli sama. Kaikki lopulliset työjonot ajettiin 5 kertaa ja niiden ajat otettiin ylös lokitiedostoista. Liitteessä 6 on esitelty työjonoihin kuluneet ajat. Jenkins suoriutui kaikista nopeimmin. GitLab oli keskimäärin 10,7 % hitaampi ja Bamboo 15,4 %.

Erot suoritusajoissa mahdollisesti selittyvät työjonojen lukumäärällä. Jenkinsin kautta koko ohjelmisto ajettiin kahdessa, GitLabin kautta viidessä ja Bamboon kautta seitsemässä eri työjonossa.

Jokaisessa työkalussa työjonojen kulkua oli mahdollista seurata reaaliaikaisesti lokin kautta. Jokaisen työkalun lokista oli erotettavissa eri työvaiheet.

9.5 Versionhallintaintegraatio ja työjonojen ajastaminen

GitLab sisältää itsessään versionhallinnan, joten integraatio oli automaattisesti tehty. Muista ohjelmista poiketen GitLabin oletus oli, että työjono ajetaan aina, kun uutta lähdekoodia tulee saataville. Muilla ohjelmilla asetukset sitä varten oli itse tehtävä.

Jenkinsissä ja Bamboossa oli molemmissa tuki GitHubin webhookeille, eli eräänlaisille herätteille. Näitä herätteitä voi laukaista eri tapahtumien yhteydessä, kuten uuden lähdekoodin tai vetopyynnön (pull request) tultua saataville (GitHub Docs, n.d.). Nämä herätteet taas voivat laukaista erilaisia toimintoja Jenkinsissä tai Bamboossa.

Jenkinsiin GitHub-laajennus on vastuussa herätteiden vastaanottamisesta ja toimintojen laukaisemisesta. Laajennuksen kautta voi herätteet itse määritellä, tai sen voi antaa automaattisesti hallita niitä.

Bamboossa herätteet täytyy manuaalisesti asettaa. Bamboo tarjoaa kuitenkin helpomman poll-vaihtoehdon, jossa Bamboo tietyn väliajoin tarkastaa tietolähteen muutoksien varalta ja sen perusteella ajaa halutut toiminnot.

Herätteiden sijaan työjonoja voidaan jokaisella ohjelmalla ajaa myös ajastetusti. Ajastaminen on hyvä vaihtoehto esimerkiksi silloin, kun työjonoilla on tarve ajaa normaalin työajan ulkopuolelle.

9.6 Laajennettavuus

Jenkinsille oli tarjolla eniten laajennuksia vertailluista työkaluista. Sen laajennuskannasta löytyi 1500 ilmaista laajennusta. Ne ovat asennettavissa Jenkinsin käyttöliittymän kautta.

GitLabille ei ole tarjolla laajennuksia, eikä sitä ole suunnitteilla. Tätä päätöstä on perusteltu mm. huonon tuen ja tieturvaongelmien avulla (Buchanan, 2019).

Bamboolle löytyi kirjoittamisen hetkellä Atlassian Marketplacen (n.d.) kautta 114 ilmaista laajennusta.

10 POHDINTA

Opinnäytetyön tavoitteena oli selvittää, voidaanko tuotekehitysprosessia ja DevOps-työskentelyä parantaa eri automaatiopalveluun siirtymällä. Tähän tarkoitukseen asennettiin ja otettiin käyttöön kolme eri CI/CD-työkalua, ja niille tehtiin tarvittavat työjonot Oscar Cloud ja cERP-ohjelmien jakeluun. Verrattavat työkalut olivat Jenkins, joka oli jo käytössä yrityksen sisällä, sekä GitLab ja Bamboo.

Jokaisella työkalulla luotiin onnistuneesti työjonot web-ohjelmiston jakeluun. Vertailtavat työkalut olivat luonteeltaan hieman erilaisia, joten luotujen työjonojen välille tuli pieniä eroavaisuuksia. Ne sisälsivät kuitenkin samat vaiheet ja lopputulos oli niillä sama. Jenkinsillä työjonoja oli kaksi, GitLabilla viisi ja Bamboolla seitsemän.

Jenkins suoriutui tehtävästä noin viidessä minuutissa, GitLabin aika oli keskimäärin 32 sekuntia hitaampi ja Bamboon 46 sekuntia. Eroa Jenkinsiin prosentteina oli GitLabilla 10,7 % ja Bamboolla 15,4 %. Eroavaisuuksia voidaan selittää osittain työjonojen käynnistykseen ja lopetukseen vaaditulla ajalla. Jos työjonot olisi luotu täysin identtisiksi, ajat olisivat luultavasti olleet lähempänä toisiaan. Työjonoissa haluttiin kuitenkin huomioida ohjelmien luonne ja parhaat käytänteet kunkin ohjelman kannalta.

Viisi minuuttia kestävässä työjonossa alle minuutin heitto suuntaan tai toiseen ei ole kovin merkittävä. Pidemmissä työjonoissa tulisi ajat verrata ja arvioida uudestaan. Oletettavasti ajat olisivat tässäkin tapauksessa lähempänä toisiaan, koska työjonon käynnistykseen ja lopetukseen kuluva aika suhteessa työjonon suorittamiseen jäisi pienemmäksi.

Vertailussa pääpaino oli jakelussa, ja sen kannalta työkalun merkitys todettiin vähäiseksi. Yksi syy tähän on se, että Docker-ohjelman käyttäminen yksinkertaistaa jakelua huomattavasti, sillä se ajetaan omassa ympäristössään. Näin ollen kohdeympäristöä ei tarvitse erikseen valmistella, esimerkiksi poistamalla vanhoja asennuksia pois tieltä. Tämän takia kaikki työjonot koostuivat vain muutamista

tehtävistä tai komennoista, eikä aivan kaikkia työkalujen ominaisuuksia päästy kokeilemaan.

Työjonoissa onnistuttiin DevOps-käytänteiden noudattamisessa jakelussa, sillä prosessi oli täysin automatisoitu ja testiympäristöön vietiin vain toimivat ohjelmat. Työjonoissa suoritettiin myös yksinkertaisia testejä, sillä ne löytyivät valmiina osasta kirjastoista, mikä havainnollisti osittain täydellistä CI/CD-työjonoa.

Jenkins ja Bamboo ovat molemmat puhtaasti automaatiotyökaluja. Suurimmat eroavaisuudet näiden kahden välillä havaittiin työjonojen luonnissa, sillä Jenkinsissä työjono toteutettiin skriptitiedoston avulla ja Bamboossa käyttöliittymän kautta. Samat asiat oli mahdollista toteuttaa molempien kautta, joten kyseessä on enemmänkin makuasia, kumpi tapa tuntuu kullekin luontevalta. GitLabissa työjonon luonti muistutti paljon Jenkinsiä, joten siirtymä mahdollisesti onnistuu vähemmällä totuttelulla verrattuna Bamboohon.

Ottaen huomioon sekä ohjelmistojen kustannukset, että vaihdosta aiheutuvat kustannukset, ei CI/CD-työkalun vaihtoa näiden tuloksien valossa nähdä kannattavana pelkästään jakeluprosessin parantamiseen.

Toteutuksessa isoin kohdattu haaste oli ajettavien ohjelmien toimiminen Docker-ympäristössä. Käytettyihin Dockerfileihin tehtiin pieniä muutoksia, jotta ohjelmat saatiin toimimaan. Niihin ei tätä työtä tehdessä juurikaan panostettu ja niitä tulisi miettiä uudestaan, ennen kuin ajaminen Docker-ympäristöön aloitetaan. Varsinkin Oscar cERP-projektin Docker-näköistiedoston rakentaminen vei tarpeettoman kauan aikaa, ja sen jakaminen pienempiin kokonaisuuksiin olisi hyödyllistä, jotta vain tietyt palaset voitaisiin rakentaa tarvittaessa. Näin myös virheet saataisiin nopeammin esille.

Pienempiin kirjastoihin jakaminen auttaisi paremmin toteuttamaan jatkuvan jakelun, sillä rakentaminen voitaisiin toteuttaa aina, kun uutta lähdekoodia tulee saataville. Liian isolla aktiivisesti kehityksessä olevalla kirjastolla työjono ei mahdollisesti ehdi loppuun asti, ennen kuin uutta koodia tulee saataville, mikä muodostaa pullonkaulan prosessiin.

Aiheen käytännön osuus oli hyvin laaja, minkä johdosta aikaa itse ajettavien ohjelmien säätämiselle ei juurikaan jäänyt. Kirjastojen rakennetta ja Dockerfilejä muuttamalla olisi Oscar cERP saatu vielä paremmin vastaamaan DevOpsin jatkuvan jakelun käytänteitä. Oscar Cloudissa tämä toteutui jo kohtalaisen hyvin.

Oscar Cloudin kirjastojen työjonojen luonti onnistui kirjoittajalta helpoiten GitLabissa. Koska GitLabin työjonot ajettiin automaattisesti uuden koodin tullessa saataville, toteutti se automaattisesti jatkuvan jakelun menetelmän ja kirjastojen uusimmat versiot olivat koko ajan saatavilla myös Nexus-tietovarastopalvelimelta.

GitLabia olisi hyvä myös harkita yhdessä esimerkiksi Rancher-ohjelman kanssa. Rancher on käyttöliittymäpohjainen ohjelma Docker-konttien hallintaan, jota Oscar Softwarella käytetään pienessä mittakaavassa. GitLabilla olisi mahdollista pelkästään rakentaa kirjastot ja saattaa Docker-näköistiedostot yhteiselle tietovarastopalvelimelle, ja varsinaisen deploy-osuuden eli palvelimelle viemisen suorittaa Rancherin kautta. Rancherissa on kattavat monitoroinnit ja hyvät rollback eli peruutusominaisuudet virhetilanteiden varalle, ja se voisi paremmin tähän tarkoitukseen kuin GitLab.

Vertailu ja tuloksien pohdinta objektiivisesti oli haastavaa, koska työkaluihin liittyvät eroavaisuudet olivat paljolti makuasioita. Projektin luonne vaikuttaa paljon siihen, mikä jakelutyökalu toimii sen kanssa parhaiten.

Tuloksia ei voi käyttää yleispätevästi kaikkien ohjelmistojen jakelun suunnitteluun, vaan se toimii ainoastaan suuntaa antavana. Vertailu ja työjonot suunniteltiin nimenomaan Oscar Cloud- ja cERP-ohjelmistoja varten Docker-ympäristöä hyödyntäen. Docker oli isossa osassa, ja koska Dockerissa ajettavat ohjelmat noudattavat samanlaista kaavaa jakelussa, voidaan tuloksia käyttää pohjana Dockerissa ajettavien ohjelmien työjonojen suunnitteluun ja toteutukseen.

Tuloksia Oscar Software Oy voi hyödyntää jatkotoimenpiteiden suunnitteluun, sekä mahdollisesti jatkossa työjonojen suunnitteluun. Muille tämä raportti toimii lyhyenä katsauksena eri CI/CD-ohjelmien toimintaperiaatteeseen ja käyttöön.

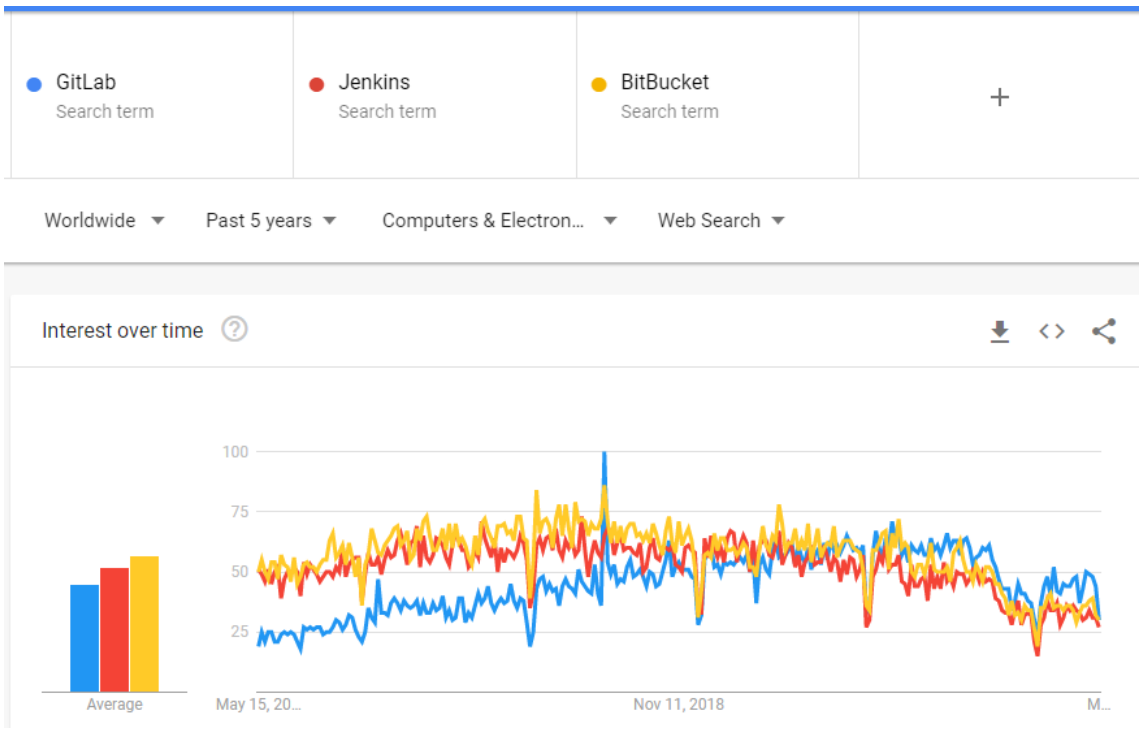
Tuloksien pohjalta jatkokehitysehdotuksena on syvällisemmän tarkastella koko ohjelmistokehityksen elinkaarta, ja siinä käytettäviä ohjelmia. Erityisenä nostona GitLab, joka pelkästään jakelutyökaluna ei tuo paljoa lisää, mutta tuo mukanaan monia muita ominaisuuksia, jotka voivat olla projektinhallinnan kannalta hyödyllisiä. GitLabiin tai vastaavaan ohjelmaan siirtyminen voisi olla hyvä tapa yhtenäistää tuotekehitysprosessin eri vaiheita saman katon alle, vähentäen tarvetta ylimääräisille integraatioille.

Tuomalla koko tuotekehitysprosessin eri vaiheet yhden työkalun alle, toimintatapojen ja menetelmien yhtenäistäminen helpottuu. Lisäksi sillä voidaan kaventaa kuilua eri tiimien välillä yrityksen sisällä DevOps-ajattelutavan mukaisesti.

Yhden työkalun alla toimiminen vähentää tarvetta siirtyä jatkuvasti ohjelmasta toiseen tietoa etsiessä, sekä se olisi kaikille saatavilla. Tunnuksien ja oikeuksien hallinta helpottuisi, mikä jo itsessään helpottaisi DevOps-tiimin taakkaa, sekä tietoturva parantuisi asennettaessa valittu ohjelma yrityksen sisäverkkoon.

Toisena esimerkkinä yksi vastaava ohjelma on BitBucket, josta löytyy valmiina integraatiot Jira-projektinhallintatyökaluun. BitBucket on Atlassianin tuottama ohjelmisto ja se sisältää paljon samoja ominaisuuksia, kuin GitLab.

Trendejä seuraamalla eri lähteistä, saadaan kuva siitä, mihin suuntaan alalla ollaan menossa CI/CD-työkalujen osalta. Koko maailman osalta viimeisen viiden vuoden aikana GitLabiin liittyvät haut ovat olleet suuremmassa nousussa, kuin Jenkinsin ja BitBucketin. Tämä voidaan havaita kuvion 15 graafista.



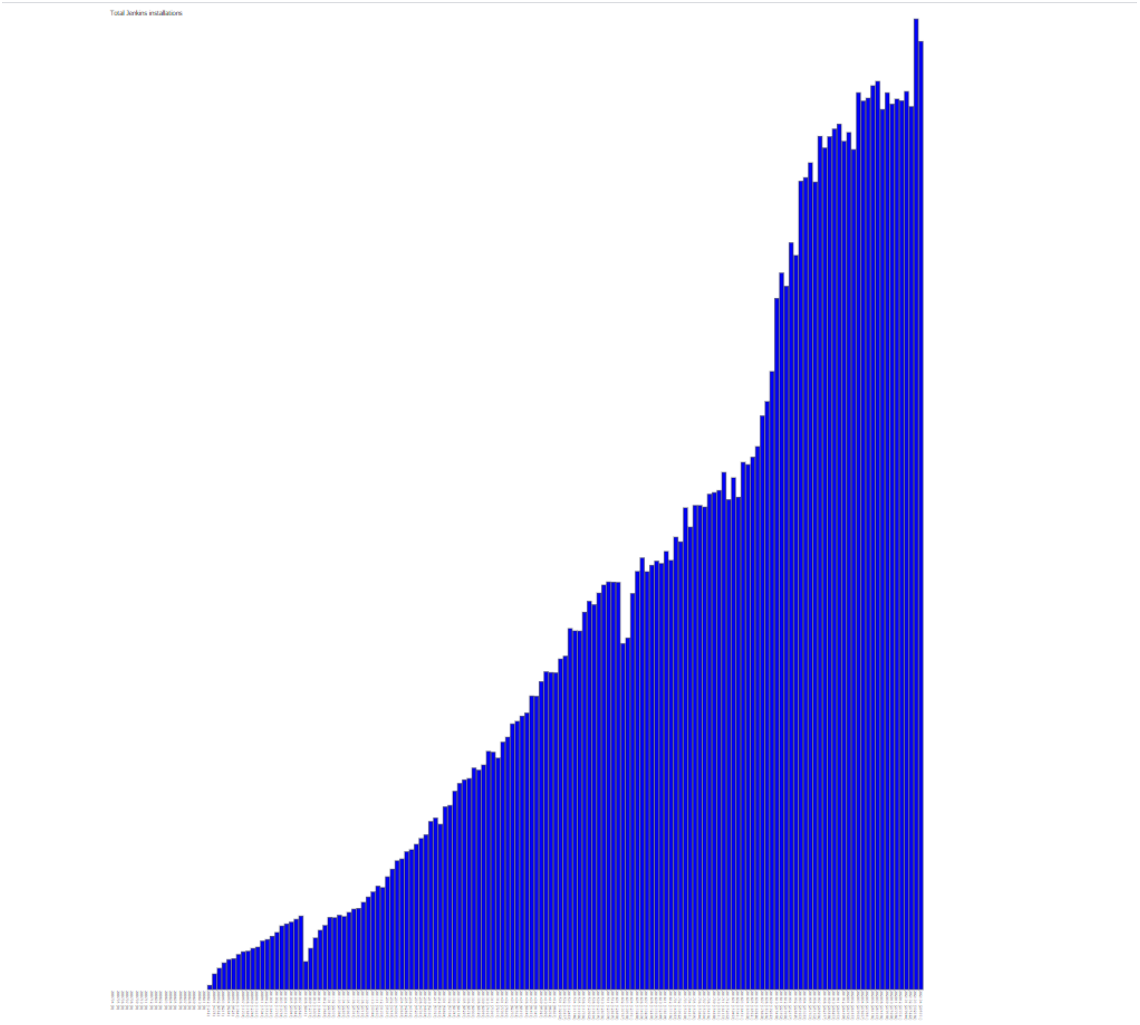
KUVIO 15. Hakusanojen esiintyminen Googlen hauissa tarkasteltuna Google Trendsin kautta

Koronapandemian vaikutukset ovat hyvin havaittavissa, sillä hakujen määrä jokaisen termin osalta lähti laskuun vuosien 2019 ja 2020 vaihteessa. Lomautukset iskivät myös IT-alalle ja erityisesti startup-yrityksiin, mitkä ovat toimineet suurena innovaatioiden lähteenä alalla (Loten, A 2020). Tämä pandemiaan mukautuminen on hidastanut kehitystä IT-alalla.

Agnus Lotenin (2020) artikkelissa haastateltu Jonathan Simnett toteaa, että Covid-19 kriisin myötä tarve yhteistyö- ja etätyöskentelytyökalujen innovaatioille on kasvanut. Kuviota 15 tarkastellessa, voidaan huomata, että GitLab ohittaa hauissa muut työkalut pandemian alun jälkeen, ja syynä tähän on se, että GitLab on pyrkinyt tuomaan näitä innovaatioita jo kauan ennen pandemian alkua.

GitLab on alkujaan kehitetty kamppailemaan etätyöskentelyn tuomia haasteita vastaan. Vuodesta 2014 asti GitLab on ollut täysin etänä toimiva yritys (GitLab 2020, 24). GitLab ylläpitää The Remote Playbook e-kirjaa, jossa kerrotaan, miten GitLab onnistunut täysin etänä toimimaan niin menestyksekkäästi. Se toimii erinomaisena ohjeena muille yrityksille tämän kriisin aikana ja GitLab ohjelmana tarjoaa työkalut näiden ohjeiden noudattamiseen.

Jenkinsin tapauksessa suosiota voidaan erikseen seurata Jenkinsin tilastosivulta, josta nähdään esimerkiksi Jenkins-asennukset kuukausittain (kuvio 16).



KUVIO 16. Jenkins-asennukset kuukausittain vuodesta 2007 lähtien (Jenkins Stats n.d.)

Asennusten määrästä voidaan päätellä, että trendeistä huolimatta Jenkins ei ole kuitenkaan katoamassa markkinoilta ja Jenkinsissä pitäytyminen voidaan nähdä edelleen turvallisena vaihtoehtona.

LÄHTEET

Anastasov, M. 2019. CI/CD Pipeline: A Gentle Introduction. Semaphore. Luettu 23.3.2021. https://semaphoreci.com/blog/cicd-pipeline

Apache Maven Project, n.d. What is Maven? Luettu 26.04.2021. https://maven.apache.org/what-is-maven.html

Atlassian Matkerplace. n.d. Bamboo apps. Luettu 6.5.2021. https://marketplace.atlassian.com/search?moreFilters=free&product=bamboo

Atlassian, n.d. Build, test, deploy. Luettu 4.4.2021. https://www.atlassian.com/software/bamboo

Bitbucket, n.d. Bitbucket Pipelines & Deployments. Luettu 29.4.2021. https://bitbucket.org/product/features/pipelines

Blogumas, T. 2020. Digestible DevOps: The 7 DevOps Practices. Luettu 3.5.2021. https://levelup.gitconnected.com/digestible-devops-the-7-devops-practices-8bd8b34e1418

Bose, S. 2020. What is Continuous Monitoring in DevOps? BrowserStack. Luettu 3.5.2021. https://www.browserstack.com/guide/continuous-monitoring-in-devops

Buchanan, C. 2019. The problem with plugins. GitHub Blog. Luettu 6.5.2021. https://about.gitlab.com/blog/2019/09/27/plugin-instability/

Docker docs, n.d. Dockerfile Reference. Luettu 10.4.2021. https://docs.docker.com/engine/reference/builder/

Fong, J. 2018. Are Containers Replacing Virtual Machines? Docker Blog. Luettu 3.5.2021. https://www.docker.com/blog/containers-replacing-virtual-machines/

GitHub Docs. n.d. About webhooks. Luettu 6.5.2021. https://docs.github.com/en/github/extending-github/about-webhooks

GitLab. 2021. The Remote Playbook. Luettu 13.5.2021. https://about.gitlab.com/resources/downloads/ebook-remote-playbook.pdf

GitLab. n.d. About GitLab. Luettu 4.4.2021. https://about.gitlab.com/company/

Jenkins. n.d. Jenkins User Documentation. Luettu 23.3.2021. https://www.jenkins.io/doc/

Jenkins Plugins. n.d. Plugins Index. Luettu 6.5.2021. https://plugins.jenkins.io/

Jenkins Stats. n.d. Some statistics on the usage of Jenkins. Luettu 13.5.2021. https://stats.jenkins.io/jenkins-stats/svg/svgs.html

Loten, A. 2020. Nearly 70,000 Tech Startup Employees Have Lost Their Jobs Since March. Wall Street Journal. Luettu 13.5.2021. https://www.wsj.com/articles/nearly-70-000-tech-startup-employees-have-lost-their-jobs-since-march-11594167238
O'Grady, A. 2018. GitLab Quick Start Guide. Birmingham, UK: Packt Publishing. Luettu 6.5.2021. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/gitlab-quick-start/9781789534344/

Oscar Software. 2019. Liiketoiminta-alustan data-arkkitehtuuri (2019). Julkaisematon. Opinnäytetyön tekijän hallussa.

Pittet, S., n.d. Continuous integration vs. continuous delivery vs. continuous deployment. Atlassian. Luettu 16.3.2021. https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

Portnoy, M. 2016. Virtualization Essentials. Newark: John Wiley & Sons, Incorporated.

Smart, J. 2011. Jenkins: The Definitive Guide. Sebastopol: O'Reilly Media, Incorporated. Luettu 6.5.2021. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/jenkins-the-definitive/9781449311155/

Vadapalli, S. 2018. DevOps: continuous delivery, integration, and deployment with DevOps. Birmingham, UK: Packt Publishing. Luettu 23.3.2021. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/devops-continuous-delivery/9781789132991/

Vohra, D. 2016. Pro Docker. Berkeley, CA: Apress. Luettu 6.5.2021. Vaatii käyttöoikeuden. https://masterworkshop.skillport.com/skillportfe/assetSummaryPage.action?assetid=RW$6219:_ss_book:112055#summary/BOOKS/RW$6219:_ss_book:112055

What Is GitLab? Speed. Efficiency. Trust. 2020. Video. GitLab. Katsottu 4.4.2021. https://www.youtube.com/watch?v=-_CDU6NFw7U

Verona, J. 2018. Practical DevOps. Birmingham, UK: Packt Publishing. Luettu 6.5.2021. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/practical-devops/9781788392570/

LIITTEET

Liite 1. Oscar Cloudin työjono Jenkinsissä

```
pipeline {
    agent { label "master" }
    environment {
       // Asetetaan Nexus-palvelimen kirjautumistunnukset
ympäristömuuttujiksi
        NEXUS_CREDS = credentials('nexus')
    }

    stages {
        stage('Clone') {
            steps {

                dir('ocresource') {
                    // Haetaan lähdekoodi tietolähteestä
                    git branch: 'joonas', changelog: false, creden-
tialsId: 'gitti', poll: false, url: 'https://github.com/OscarSoft-
ware/ocresource/'

                    // Asetetaan Mavenin asetukset suorittavalle käyt-
täjälle
                    sh 'cp ./docker/settings.xml ~/.m2/settings.xml'
                }

                dir('ocmailserverclient') {
                    git branch: 'joonas', changelog: false, creden-
tialsId: 'gitti', poll: false, url: 'https://github.com/OscarSoft-
ware/ocmailserverclient/'
                }

                dir('ocresourceclient') {
                    git branch: 'joonas', changelog: false, creden-
tialsId: 'gitti', poll: false, url: 'https://github.com/OscarSoft-
ware/ocresourceclient/'
                }

                dir('oscarcloud-entities') {
                    git branch: 'joonas', changelog: false, creden-
tialsId: 'gitti', poll: false, url: 'https://github.com/OscarSoft-
ware/oscarcloud-entities/'
                }

            }
        }

        stage('Build and deploy') {
            agent { label "master" }
            steps {

                dir('oscarcloud-entities') {
                    // Rakennetaan kirjasto ja ladataan tietovarasto-
palvelimelle
                    sh "mvn clean deploy"
                }
                dir('ocmailserverclient') {
                    sh "mvn clean deploy"
                }
```

```
                dir('ocresourceclient') {
                    sh "mvn clean deploy"
                }

                dir('ocresource') {
                   // Kirjaudutaan sisään Docker-rekisteriin
                    sh "docker login -u ${NEXUS_CREDS_USR} -p
${NEXUS_CREDS_PSW} 192.168.1.60:8082"

                    // Rakennetaan Docker-näköistiedosto
                    sh "docker build -t 192.168.1.60:8082/ocre-
source:joonas . "

                    // Ladataan Docker-näköistiedosto rekisteriin
                    sh "docker push 192.168.1.60:8082/ocre-
source:joonas"

                }

            }
        }

        stage("Publish") {
           // Valitaan suorittajaksi kohdepalvelin
            agent { label "host1" }

            environment {
               // Asetetaan käynnistysparametrit Cloudille
                OCRESOURCE_ARGS = """ \
                --name 'cloud1' \
                -d \
                -p 8180:8180 \
                -e BACKEND_NAME='develop_joonass' \
                """
            }

            steps {
                sh "docker login -u ${NEXUS_CREDS_USR} -p
${NEXUS_CREDS_PSW} 192.168.1.60:8082"

                // Haetaan Docker-näköistiedosto rekisteristä
                sh "docker pull 192.168.1.60:8082/ocresource:joonas"

                // Pysäytetään edellinen kontti, mikäli se on
käynnissä
                sh 'docker stop cloud1 || true && docker rm cloud1 ||
true'

                // Käynnistetään Cloud annetuilla parametreillä
Docker-konttina
                sh "docker run ${OCRESOURCE_ARGS}
192.168.1.60:8082/ocresource:joonas"
            }
        }

    }
}
```

Liite 2. Oscar Cloudiin liittyvien kirjastojen työjono Gitlabissa

```
## ocmailserver, ocresourceclient ja oscarcloud-entities CI/CD-työjon-
oskripti

variables:
  MAVEN_CLI_OPTS: "--batch-mode --errors --fail-at-end --show-version
-DinstallAtEnd=true -DdeployAtEnd=true"

cache:
  paths:
    - .m2/repository

deploy:jdk8:
  stage: deploy
  tags:
      - build
  script:
      # Ilmoitetaan, jos asetukset Mavenin asetukset puuttuvat
    - if [ ! -f settings.xml ];
        then echo "CI settings missing\! ";
      fi
      # Rakennetaan kirjasto ja ladataan se tietovarastopalvelimelle
    - 'mvn $MAVEN_CLI_OPTS clean deploy -s settings.xml'
  only:
    variables:
        # Ajetaan vaihe vain oletushaarassa
      - $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

Liite 3. Oscar Cloudin pääkirjaston työjonoskripti GitLabissa

```
## ocresource CI/CD-työjonoskripti

variables:
    MAVEN_CLI_OPTS: "--batch-mode --errors --fail-at-end --show-ver-
sion -DinstallAtEnd=true -DdeployAtEnd=true"
    REGISTRY: "192.168.1.60:8082"

build:
    stage: build
    tags:
        - build
    script:
        # Rakennetaan Docker-näköistiedosto
        - 'docker build -t $REGISTRY/ocresource:joonas . '
    only:
        variables:
            # Ajetaan vaihe vain oletushaarassa
            - $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

push:
    stage: build
    tags:
        - build
    script:
        # Kirjaudutaan Docker-rekisteriin
        - 'docker login -u $NEXUS_USER -p $NEXUS_PASS $REGISTRY'
        # Ladataan näköistiedosto rekisteriin
        - 'docker push $REGISTRY/ocresource:joonas'
    only:
        variables:
            # Ajetaan vaihe vain oletushaarassa
            - $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

deploy:
    stage: deploy
    variables:
        DOCKER_RUN_OPTS: "--name cloud1 -d -p 8180:8180 -e
BACKEND_NAME=develop_joonass"
    tags:
        # Suoritetaan kohdetietokoneella
        - target
    script:
        - 'docker login -u $NEXUS_USER -p $NEXUS_PASS $REGISTRY'
        # Haetaan näköistiedosto rekisteristä
        - 'docker pull $REGISTRY/ocresource:joonas'
        # Pysäytetään edellinen kontti, mikäli se on olemassa
        - 'docker stop cloud1 || true && docker rm cloud1 || true'
        # Ajetaan uusi kontti halutuilla parametreillä
        - "docker run $DOCKER_RUN_OPTS $REGISTRY/ocresource:joonas"
    only:
        variables:
            # Ajetaan vaihe vain oletushaarassa
            - $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

Liite 4. Oscar cERP-kirjaston työjonoskripti Jenkinsissä

```
pipeline {
    agent { label "master" }

    environment {
        registry = "http://192.168.1.60:8082/"
    }

    stages {
        stage('Pull') {
            steps {
                // Haetaan lähdekoodi tietolähteestä
                git branch: 'joonas', changelog: false, credentialsId:
'gitti', poll: false, url: 'https://github.com/OscarSoftware/cerp/'
            }

        }
        stage ('Build image') {
            steps {
                script {
                    // Aloitetaan Docker-laajennuksen käyttö
                    // käyttäen annettua rekisteriä ja kirjau-
tumistietoja
                    docker.withRegistry(registry, 'nexus') {
                        // Rakennetaan Docker-näköistiedosto
                        def customImage = docker.build("cerp:joonas")

                        //
                        customImage.push()
                    }
                }
            }

        }

        stage('Publish') {
            // Valitaan kohdepalvelin suorittamaan tämä osio
            agent { label "host1" }

            steps {
                // Pysäytetään edellinen kontti, jos se on olemassa
                sh 'docker stop cerp1 || true && docker rm cerp1 ||
true'
                script {
                    docker.withRegistry(registry, 'nexus') {

                        // Luodaan uusi tyhjä Docker-näköistiedosto
                        def cerp = docker.image("cerp:joonas")

                        // Haetaan Docker-näköistiedosto rekisteristä
                        sh "docker pull ${cerp.imageName()}"

                        // Ajetaan Docker-näköistiedosto halutuilla
parametreillä
                        docker.image(cerp.imageName()).run("--name
cerp1 -d -p 8080:8080")
                    }
                }
            }
        }
    }
}
```

Liite 5. Oscar cERP-kirjaston työjonoskripti GitLabissa

```
## Cerp CI/CD-työjonoskripti

variables:
  MAVEN_CLI_OPTS: "--batch-mode --errors --fail-at-end --show-version
-DinstallAtEnd=true -DdeployAtEnd=true"
  REGISTRY: "192.168.1.60:8082"

build:
  stage: build
  tags:
    - build
  script:
    # Luodaan Docker-näköistiedosto
    - 'docker build -t $REGISTRY/cerp:joonas . '
  only:
    variables:
      # Ajetaan vaihe vain oletushaarassa
      - $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

push:
  stage: build
  tags:
    - build
  script:
    # Kirjaudutaan Docker-rekisteriin
    - 'docker login -u $NEXUS_USER -p $NEXUS_PASS $REGISTRY'
    # Ladataan Docker-näköistiedosto rekisteriin
    - 'docker push $REGISTRY/cerp:joonas'
  only:
    variables:
      # Ajetaan vaihe vain oletushaarassa
      - $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

deploy:
  stage: deploy
  variables:
    DOCKER_RUN_OPTS: "--name cerp1 -d -p 8080:8080"
  tags:
    # Valitaan suorittajaksi kohdepalvelin
    - target
  script:
    - 'docker login -u $NEXUS_USER -p $NEXUS_PASS $REGISTRY'
    # Haetaan näköistiedosto rekisteristä
    - 'docker pull $REGISTRY/cerp:joonas'
    # Pysäytetään edellinen kontti, jos se on olemassa
    - 'docker stop cerp1 || true && docker rm cerp1 || true'
    # Ajetaan uusi kontti halutuilla parametreillä
    - "docker run $DOCKER_RUN_OPTS $REGISTRY/cerp:joonas"
  only:
    variables:
      # Ajetaan vaihe vain oletushaarassa
      - $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

Liite 6. Työjonojen suoritusajat

TAULUKKO 1. Jenkinsin työjonojen suoritusajat sekunteina

	Oscar Cloud	Oscar cERP	Yht.
1. ajo	26	272	298
2. ajo	24	265	289
3. ajo	24	276	300
4. ajo	25	279	304
5. ajo	22	286	308
Keskiarvo	24,2	275,6	299,8

TAULUKKO 2. GitLabin työjonojen suoritusajat sekunteina

	Oscarcloud-entities	Ocmailserver-client	Ocresour-ceclient	Ocre-source	Cerp	Yht.
1. ajo	7	3	6	45	272	333
2. ajo	7	4	6	45	262	324
3. ajo	6	4	6	45	273	334
4. ajo	6	3	6	45	276	336
5. ajo	6	4	6	45	272	333
Keskiarvo	6,4	3,6	6	45	271	332

TAULUKKO 3. Bamboon työjonojen suoritusajat sekunteina

	Oscarcloud-entities	Ocmailserver-client	Ocresour-ceclient	Ocresource
1. ajo	6	4	6	45
2. ajo	6	3	6	43
3. ajo	7	3	6	40
4. ajo	6	3	6	40
5. ajo	5	4	6	40
Keskiarvo	6	3,4	6	41,6
	Cerp	Cloud deploy	Cerp deploy	Kaikki vaiheet yht.
1. ajo	265	6	11	343
2. ajo	279	1	11	349
3. ajo	278	1	11	346
4. ajo	279	1	11	346
5. ajo	280	1	10	346
Keskiarvo	276,2	2	10,8	346