

Jarkko Nieminen

# Jatkuvan integraation ympäristö pilvipalvelussa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

15.4.2018

Tekijä Otsikko	Jarkko Nieminen Jatkuvan integraation ympäristö pilvipalvelussa
Sivumäärä Aika	32 sivua 15.4.2018
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikka
Ammatillinen pääaine	Tietoverkot
Ohjaajat	Osaamisaluepäällikkö Janne Salonen
<p>Tämän insinööriyön tarkoitus oli tutkia kuvitteellisen jatkuvan integraation ympäristön siirtämistä julkiseen pilvipalveluun sekä tutkia miten ympäristö voitaisiin automatisoida niin, ettei ympäristöä käytettäessä tarvita manuaalista työtä. Myöskin tutkitaan vaihtoehtona mahdollisuutta ajaa testiajoja dynaamisilla suorittajilla staattisten sijaan.</p> <p>Työ itsessään toteutettiin käyttämällä jatkuvan integraation ohjelmistona avoimen lähdekoodin ohjelmisto Jenkinsiä sekä julkisen pilvipalveluntarjoajaksi valittiin Amazon Web Services. Tiedostovarastoksi valikoitui julkinen tiedostovarastopalvelu GitHub.</p> <p>Tavoitteeseen päästiin toteuttamalla ympäristö halutuilla työkaluilla ja liitännäisillä, joita toiminnallisuus vaati sekä dynaamiset tehtävän suorittajat ajettiin virtuaalikoneen kuvilta, jotka työn tullessa tiedostovarastoon, automaattisesti luotiin määritellystä virtuaalikoneen kuvasta suoritusta varten Amazon Web Services pilvipalveluun sekä suorituksen jälkeen automaattisesti tuhottiin.</p>	
Avainsanat	VCS, CI, Jenkins, AWS

Author Title	Jarkko Nieminen Continuous integration environment in the cloud
Number of Pages Date	32 pages 15 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Data Networks
Instructors	Janne Salonen, Director of Expertise
<p>The purpose of this Bachelor's thesis was to examine moving an imaginary continuous integration environment to public cloud and to examine how the environment could be automated so, that when using the environment, no manual interaction was required. Also it was examined that as an alternative, the work loads could be ran automatically on dynamic agents instead of static ones.</p> <p>The work itself was implemented using an open source tool, Jenkins, for the continuous integration and Amazon Web Services was chosen for the cloud service provider. For the repository services, GitHub was chosen for this.</p> <p>The goal was reached by implementing the environment with the tools chosen for the task and with the plugins required for the functionality. Also the dynamic agents were ran from virtual machine images, that when a new commit was made to the repository, a new instance was spawned to Amazon Web Services cloud service based on the defined virtual machine image. Also the instance spawned for the job was terminated after the execution was complete.</p>	
Keywords	VCS, CI, Jenkins, AWS

# Sisällys

## Lyhenteet

1	Johdanto	1
2	Versionhallinta	2
2.1	Toimintaperiaate	2
2.2	Versionhallintajärjestelmät	3
2.2.1	Keskitetty versionhallintajärjestelmä	4
2.2.2	Hajautettu versionhallintajärjestelmä	5
3	Jatkuva integraatio	7
3.1	Toimintaperiaate	7
3.2	Jenkins	9
4	Virtualisointi	9
4.1	Hypervisor	11
5	Pilvipalvelut	12
5.1	Pilvipalvelun keskeiset piirteet	13
5.2	Pilvien tyypit	14
5.3	Palvelumallit	15
6	Amazon Web Services	16
6.1	Perustapalveluiden kuvaukset	17
6.1.1	Laskentapalveluita	18
6.1.2	Tallennuspalveluita	19
6.1.3	Tietokantapalveluita	19
6.1.4	Tietoverkkopalveluita	19
7	Raportti	20
7.1	VPC ja EC2	20
7.2	Jenkins	22
8	Yhteenveto ja pohdinta	30
	Lähteet	31

## Lyhenteet

CI	Continuous Integration
CD	Continuous Delivery
VCS	Version Control System
OS	Operating system
VM	Virtual Machine
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
IAM	Identity and Access Management
AMI	Amazon Machine Image
URL	Uniform Resource Identifier
AWS	Amazon Web Services

## 1 Johdanto

Insinööriyössä tutkitaan omassa datakeskuksessa ajettavan jatkuvan integraation ympäristön siirtämistä julkiseen pilvipalveluun rakentamalla testiympäristö julkiseen pilvipalveluun. Pidättäydytään siis vain siirtämästä olemassa olevaa infrastruktuuria ajettavaksi julkisessa pilvessä ajettavilla virtuaalikoneilla eli instansseilla. Samalla pyritään etsimään ja kohtaamaan haastavia tilanteita rakennusvaiheessa sekä käyttönotossa, joita saattaa esiintyä sekä yritetään identifioida ongelmakohdat, joista on mahdollisesti hyötyä tulevaisuudessa, kun aloitetaan testiympäristön sijaan rakentamaan tuotannon asettamien vaatimusten perusteella täysimittaista jatkuvan integraation ympäristöä, jonka rakentamisessa voidaan mahdollisesti käyttää hyödyksi tässä työssä opittua tai tämän työn aikana esiintyneitä ongelmakohtia ja niiden mahdollisia ratkaisuita.

Tavoitteena on rakentaa pienimuotoinen jatkuvan integraation testiympäristö julkiseen pilvipalveluun, jotta voidaan tutkia miten tämä on mahdollista toteuttaa ja millaisen vaivannäön sekä miten laajalti tämä vaatii uuden omaksumista. Palveluntarjoajaksi on valittu markkinajohtaja Amazon Web Services. Jatkuvan integraation toteutuksen ohjelmistoksi on valittu avoimen lähdekoodin ohjelmisto Jenkins.

Ympäristön rakentamisella tavoitellaan tilaa, jossa julkiseen pilvipalveluun on pystytty rakentamaan ympäristö, jossa jatkuvaa integraatiota voidaan suorittaa siten, että uuden tiedostoversion saapuessa tiedostovarastoon, käynnistyy uuden ohjelmakoodin tarkastus automaattisesti aivan kuten paikallisessa datakeskuksessa.

Insinööriyössä uuden ohjelmakoodin tarkastus pyritään suorittamaan siten, että tarkastuspyynnön saapuessa jatkuvan integraation ohjelmistolle, luodaan saapuneen tehtävän suoritusta varten instanssi, joka sille annetun tehtävän suorittamisen jälkeen, vaihtoehtoisesti pysyy määritellyn ajan käytettävissä mahdollista tulevaa tarkastustehtävää varten tai vaihtoehtoisesti tuhoutuu välittömästi suorituksen jälkeen, jolloin instanssista ei jouduta maksamaan käytön mukaan enempää, kuin on välttämätöntä tehtävän suorittamista varten.

## 2 Versionhallinta

### 2.1 Toimintaperiaate

Yleisesti ottaen versionhallinnaksi kutsutaan järjestelmää, joka sekä hallinnoi, että jäljittää tallennetun sisällön, kuten tiedostojen, eri versioita eli muokkauksia. [1, s. 1.]

Mielestäni voidaan olettaa, että versionhallintaa moni on jo tietämättään tullut käyttäneeksi jonkinasteisessa muodossa elämänsä aikana. Esimerkiksi kokki näkee jossain mielenkiintoisen reseptin ja kokkaa ohjeiden mukaan illallisen itselleen. Illallinen on hyvää, mutta kaipaisi hieman korianteria ja sipuli on liian voimakkaan makuinen, joten kokki haluaa poistaa sipulin reseptistä. Kokki kirjoittaa reseptiin muutoksen, jossa muistuttaa lisäämään korianteria seuraavalla kerralla ja poistaa sipulin kokonaan. Versionhallinta näkee tämän vain lisäyksenä alkuperäiseen ja tallentaa korianterin reseptiin kokin määrittelemään kohtaan sekä poistaa sipulin kokin toiveiden mukaisesti reseptistä. Versionhallinnassa reseptiin tehtävä muutos on siis vain korianterin lisäys ja sipulin poisto eikä koko reseptiä kirjoiteta näin ollen kokonaisuudessaan uudelleen.

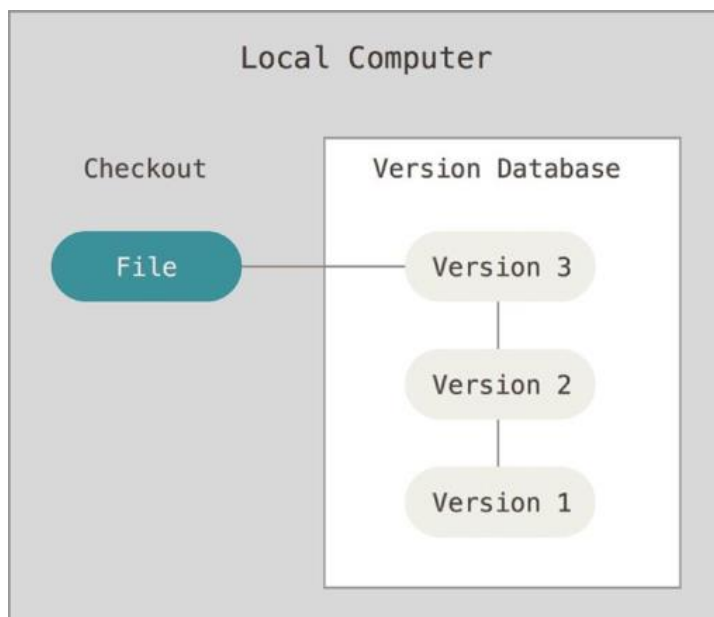
Versionhallinta tallentaa vain tehdyt muutokset eikä siten tallenna muokattua tiedostoa aina uudelleen, joka johtaisi uuden tiedoston luomiseen per muutos, vaan esittää muutoksen yhdessä tiedostossa. [2.]

Versionhallinnan avulla voidaan pitää kirjaa tiedostoon tai tiedostoihin tehdyistä muokkauksista. Niin halutessaan, voidaan aina palata tiettyinä hetkenä tehtyyn muokkaukseen ja jatkaa työskentelyä siitä kohdasta. [3, s. 1.]

Versionhallinnalla siis saavutetaan toiminnallisuus pitämällä yllä tiedostovarastoa (repository), joka sisältää uusimman mahdollisen version tiedostoista tai hakemistoista sekä koko muutoshistorian miten nykyiseen tilaan on päädytty. Lisätietona yleensä on mukana vähintäänkin kyseisen muutoksen suorittaja ja mahdollisesti muutoksen suorittajan kirjoittamat kommentit tekemästään muutoksesta. Näiden tietojen ansiosta seuraavaksi tiedostojen kanssa työskentelevän henkilön ei tarvitse käyttää kohtuuttomasti aikaa selvittääkseen kuka on muokannut viimeksi tiedostoa. Jos edellinen muokkaaja on myös kommentoinut mitä on muokattu ja miksi, helpottaa se entisestään seuraavan muokkajan työn aloitusta, koska kommenttien vuoksi on mahdollisesti tiedossa, mitä on jo aiemmin tehty. [4, s. 8.]

## 2.2 Versionhallintajärjestelmät

Versionhallintajärjestelmät voidaan jakaa karkeasti kolmeen eri kategoriaan. Yleisin on paikallinen versionhallinta. Paikallisella versionhallinnalla tarkoitetaan sitä, että käytettävät tiedostot, joita halutaan muokata, on paikallisesti saatavilla. Tiedostot voivat sijaita tietokoneen tai palvelimen kovalevyllä tai jollain muulla tallennusmedialla, joka on saatavilla fyysisesti. Muokattavat tiedostot ovat siis aina saatavilla, kunhan vain tallennettu media pysyy toimintakuntoisena eli sieltä voi lukea ja sinne voi tallentaa (kuva 1). [3, s. 1.]



Kuva 1. Paikallinen versionhallintajärjestelmä. [2, s. 1.]

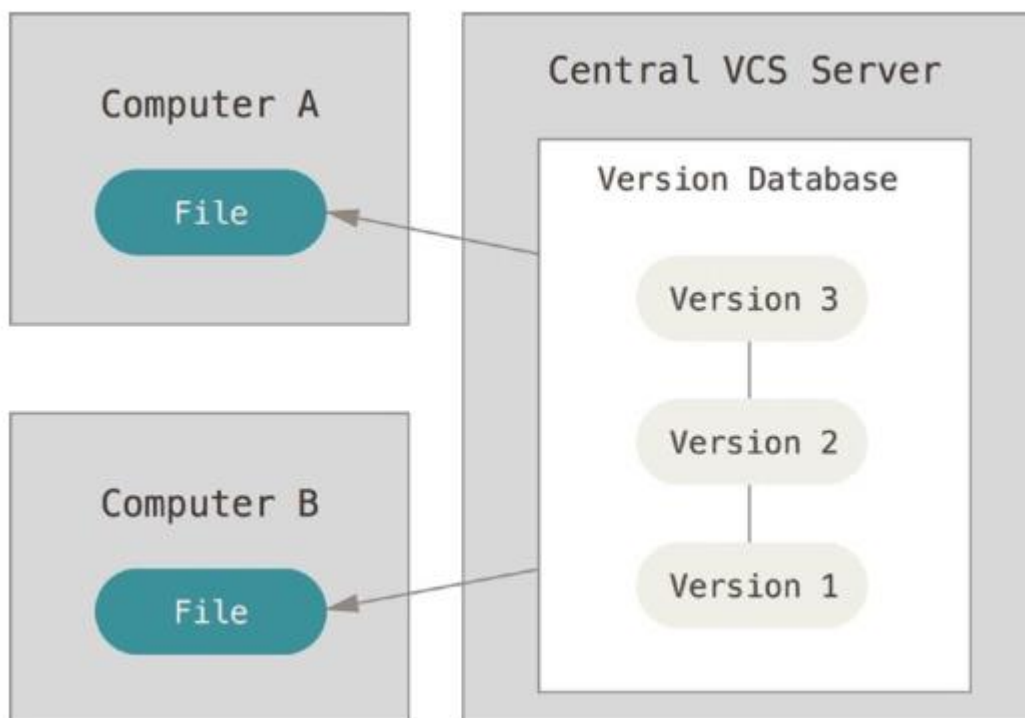
Omien kokemusten mukaan paikallisessa versionhallinnassa on ongelmakohtana tallennusmedioiden kestävyys ja mahdollisten varmuuskopioiden luominen sekä varmuuskopioiden luomistiheys. Jos varmuuskopiointia ei tehdä riittävän usein, on laiterikon tapahutuessa mahdollista, että saatavilla oleva tila tiedostosta poikkeaa hyvinkin paljon siitä tilasta, jossa viimeksi tiedoston kanssa työskenneltiin ja täten menetetään jokin määrä jo tehtyä työtä, koska varmuuskopiointia ei suoriteta tarvittavan usein. Positiivisena puolelana on nostettava esiin haluttujen tiedostojen ainainen saatavuus ja saatavuuden nopeus. Koska tiedostot ovat paikallisesti tallennettu ja ajan tasalla, on helppo jatkaa työskentelyä koska vain ja missä vain, kunhan tallennusmedia on mukana eikä näin ollen olla riippuvaisia palvelimista, joiden kanssa kommunikointi vaatisi lähiverkko- tai internet-yhteyttä. Huomattavaa on, että yhteistyö muiden kanssa on lähes mahdotonta, koska



tiedostot ovat saatavilla vain paikallisesti, joten paikallinen versionhallinta soveltuu lähinnä henkilökohtaisia tarpeita täyttämään.

### 2.2.1 Keskitetty versionhallintajärjestelmä

Keskitetyllä versionhallinnalla tarkoitetaan palvelinta, jossa sijaitsee tiedostovarasto, johon keskitetään tiedostot, joita käytetään tai tarvitaan kyseisessä työtehtävässä kuten ohjelman osan tai kokonaisen ohjelman luomisessa. Työntekijä ensin lataa keskitetyltä palvelimelta kopion halutusta tiedostosta ja alkaa muokkaamaan sitä esimerkiksi korjatakseen mahdollisia virheitä olemassa olevassa koodissa. Kun työntekijä on tehnyt halutut korjaukset, siirtää hän tiedostoon tekemänsä muokkaukset takaisin keskitetylle palvelimelle, josta muut tiimin tai projektin jäsenet näkevät missä tilannekuvassa tiedosto nyt on ja voivat ladata nykyisen eli uusimman version itselleen, johon edellinen työntekijä on luonut tarvittavan toiminnallisuuden tai korjannut mahdollisen ohjelmointivirheen. Keskitetty versionhallinta näin ollen mahdollistaa tiimin tai projektin työntekijöiden välisen yhteistyön. [3, s. 3.; 4, s. 2.] Kuva 2 havainnollistaa keskitetyn tallennusjärjestelmän toimintaa.



Kuva 2. Keskitetty versionhallintajärjestelmä. [2, s. 3.]

Mielestäni keskitetyn versionhallinnan suurimpiin etuihin kuuluu paikalliseen versionhallintajärjestelmään verrattuna juuri mainittu työntekijöiden mahdollisuus yhteistyöhön sekä projektin avoimeen näkyvyyteen, jota voidaan seurata keskitetyltä palvelimelta miten projekti etenee tai missä kohdin näyttäisi ongelmia esiintyvän ja mahdollisesti lisätä työvoimaa ratkaisemaan kyseisen osa-alueen ongelmaa.

Toki keskitettyä palvelinta voidaan hallita juuri niin kuin parhaaksi nähdään eli mahdollista avoimuutta on juuri niin paljon tarjolla, kuin sitä halutaan tarjota. Voidaan määrittellä, etteivät projektin työntekijät näe kuin oman projektihaaransa tiedostot tai voidaan halutessa tarjota nähtäväksi koko projektin tiedostovarasto muutoshistorioineen. On myös mahdollista asettaa vain lukuoikeudet koko projektin tiedostovarastoon ja kirjoitusoikeudet omaan projektin haaraan, jolloin mahdollisia vahinkoja, kuten väärään tiedostovarastoon tallentamista ei tapahdu. Keskitetyn versionhallinnan huonoihin puoliin kuuluu sen haavoittuvuus. [3, s. 3.]

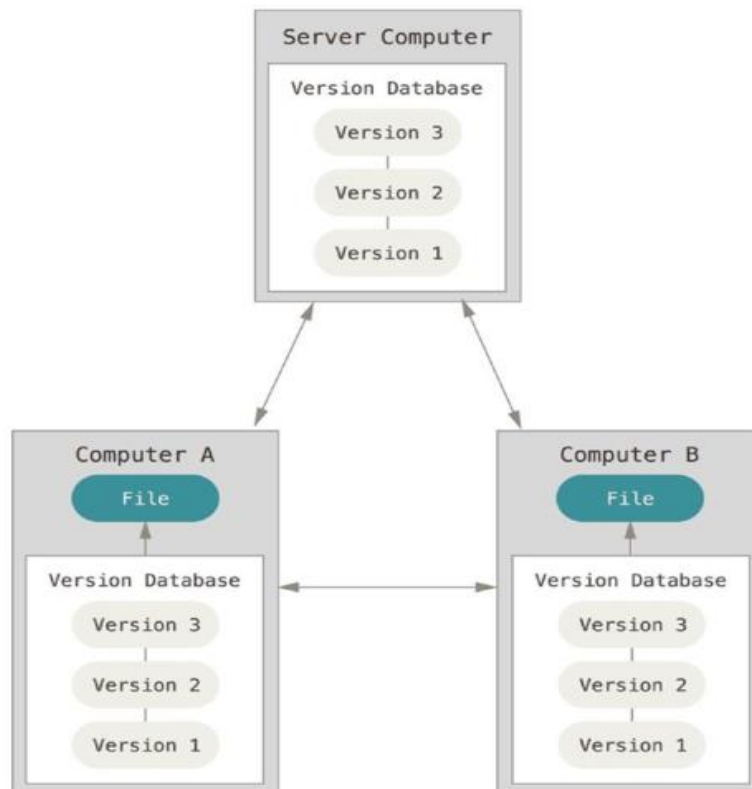
Jos kuvitteellisessa tilanteessa yhteys palvelimeen katkeaa useammaksi päiväksi, eivät työntekijät voi ladata palvelimelta itselleen tiedostoja joita työstää ja ongelma on myöskin toiseen suuntaan eli valmiiksi saatuja työtehtäviä ei voi lähettää palvelimelle.

Toinen merkittävä ongelma on mahdollinen tallennusmedian rikkoutuminen tai korrumpoituminen. Jos varmuuskopiot keskitetyn palvelimen tallennusmediasta eivät ole ajan tasalla, menetetään mahdollisesti useiden ihmisten tuntien ellei jopa päivien työ ja pahimmassa tapauksessa, jossa varmuuskopioita ei ole, on tehty työ menetetty lähes täysin. Työntekijöillä on mahdollisesti vielä työasemillaan heidän sen hetkinen tilansa projektista, jota kautta jotain saattaa vielä olla pelastettavissa. [3, s. 3.]

### 2.2.2 Hajautettu versionhallintajärjestelmä

Hajautetulla versionhallintajärjestelmällä tarkoitetaan järjestelmää, jossa tiedostovarasto ladataan kokonaisuudessaan palvelimelta työntekijän työasemalle. Kun työntekijä on tehnyt halutut muutokset tai lisäykset, voi hän lähettää oman tilannekuvansa palvelimelle, jonne lähetetään vain työntekijän luomien muutosten tilannekuva eikä koko tiedostovarastoa, joka kuormittaisi turhaan verkkoliikennettä ja huonon verkkoyhteyden päässä työskentelevä työntekijä ei mahdollisesti saisi edes muutoksiaan lähetettyä palvelimelle. [3, s. 4.]

Kuten huomataan, hajautetussa versionhallinnassa vältetään keskitetyn versionhallinnan ongelmakohdat tehokkaasti, koska verkkoyhteyttä palvelimelle tarvitaan vain, kun ladataan tiedostovarasto ja kun lähetetään tehdyt muutokset palvelimelle takaisin, joten jos verkkoyhteys katkeaa, voidaan jatkaa työtä toisen tiedoston kanssa. Ensimmäisen tiedostovaraston latauksen jälkeen siis ladataan palvelimelta vain tiedostovaraston saapuneet muutokset, eikä koko tiedostovarasto. Erittäin positiivisena asiana mainittakoon tiedoston tilannekuvan latauksen sijaan ladattava tiedostovarasto kokonaisuutenaan. Tämä tarkoittaa siis sitä, että jos palvelimen tallennusmedia rikkoutuu ja varmuuskopioita ei ole saatavilla, voidaan työntekijän työasemalle ladattu tiedostovarasto lähettää takaisin palvelimelle, jolloin työtä ei juurikaan katoa ja parhaimmassa tapauksessa koko tiedostovarasto voidaan palauttaa juuri siihen tilaan, kuin se tallennusmedian rikkoutumista edeltävässä tilannekuvassa oli. [3, s. 4.; 5, s. 3.] Kuvan 3 avulla voidaan havaita, että työntekijät voivat työskennellä yhteistyössä myös ilman yhteyttä palvelimeen.



Kuva 3. (2, s. 4).

### 3 Jatkuva integraatio

Ennen jatkuvan integraation kehittämistä ja käyttöönottoa oli ohjelmien tuottaminen ja etenkin julkaisuvaihetta edeltävä yhteensovittaminen, jossa ohjelmaan koodia tuottaneet ohjelmoijat sovittavat ohjelmoimiaan komponentteja ja ominaisuuksia yhteen kootakseen kokonaisen ohjelman kaikkine ominaisuuksineen, verrattaen haastavaa. Lisäksi projektin aikataulutusta tuo mukaan myös oman haasteensa, koska ohjelmistoa tuottava yritys luonnollisesti haluaa valmiin tuotteen asiakkaalleen mahdollisimman nopeasti. [6.]

Kun ohjelmoijat ovat jokainen tahoillaan saaneet työnsä valmiiksi, alkaa tuotetun koodin yhteen kokoaminen toimivaksi kokonaisuudeksi. Tähän kuuluu valtava määrä aikaa ja vaivaa, koska yhteensovittamisessa lähes varmasti ilmenee yhteensopivuusongelmia, joita joudutaan luonnollisesti korjaamaan. Ongelmana on muun muassa se, että ohjelmoijat ovat mahdollisesti työstäneet koodia, joka rikkoo kokonaisuuden toimivuuden huomattavan kauan sitten ja siten ei voida olettaa, että ohjelmoija voisi muistaa miksi on kirjoittanut ohjelmaan tuottamansa koodin juuri niin. Koodin yhteensopivuusongelmien ja niistä aiheutuvien korjausten vuoksi voidaankin olettaa, että toimitusaikataulu kärsii ja tuotteen toimitus asiakkaalle siis viivästyy sekä kustannukset kasvavat lisätyön vuoksi. [6.]

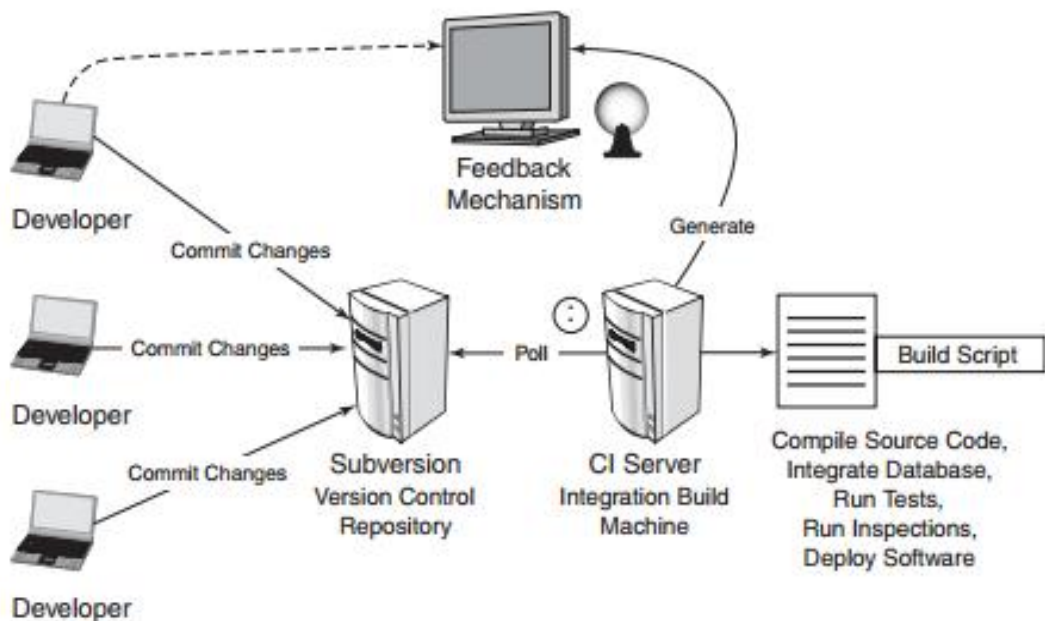
Historian valossa ei jatkuvaa integraatiota kehitetty ajettavaksi lukuisia kertoja saman päivän aikana, vaan jalostui tähän hieman myöhemmin. Idean nimeäjänä ja konseptin kehittäjänä pidetään Grady Boochia, joka esitti jatkuvan integraation idean teoriassaan 1991. [7.]

#### 3.1 Toimintaperiaate

Jatkuvan integraation tarkoitus on siis ratkaista ongelma, joka tulee esiin julkaisua edeltävässä ohjelmistokomponenttien integraatio- eli kokoamisvaiheessa vastaan. Jos integraatio on jatkuvaa, voidaan mahdolliset koodivirheet korjata lähes välittömästi kun ohjelmoijalla on vielä työhönsä tekemät muutoksensa tuoreessa muistissa ja siihen palaaminen on helpompaa sekä komponenttien yhteensopivuusongelma voidaan poistaa lähes kokonaan, kun kirjoitettua koodia testataan jatkuvasti yhteensopivuusongelmien löytämiseksi. [6.] Etenkin kun projekti etenee tai mahdollisesti lisätään uusia työntekijöitä projektiin, projekti monimutkaistuu, jolloin on tärkeää, että jatkuvaa integraatiota käytetään projektin alusta alkaen ja usein. [8, s. 21.]

Tyypillisen jatkuvan integraation prosessi menee jotakuinkin seuraavasti:

1. Kun ohjelmoija saa jonkin kohdan työstämästään osasta valmiiksi, hän kommitoi (commit) tuottamansa koodin koodivarastoon. Toisaalla CI-palvelin suorittaa kyselyitä tiedostovarastoon löytääkseen uusia muutoksia.
2. Hetken kuluttua kommitoinnista, huomaa CI-palvelin tiedostovaraston muutoksen ja noutaa uusimman version koodista tiedostovarastosta itselleen ja aloittaa rakennuskriptin ajon, joka testaa koodin yhteensopivuutta.
3. Ajettuaan koodin testauksen, suorittaa CI-palvelin ajon jälkeisen palautteen lähettämisen esimerkiksi sähköpostilla aiemmin määritellyille henkilöille, jossa informoi, että testi on ajettu ja listaa ajon tulokset, kuten oliko ajo onnistunut vai ei.
4. Tämän jälkeen CI-palvelin palaa kohtaan yksi tarkkailemaan muutoksia tiedostovarastossa. [8, s. 5.] Duvall esittää kuvassa 4 prosessin kulun.



Kuva 4. Jatkuvan integraation prosessin kulku. [8, s. 5.]

Pohjimmiltaan jatkuvalla integraatiolla pyritään siis minimoimaan riskejä tarjoamalla välitöntä palautetta koodin integraatiossa mahdollisesti esiintyvissä ongelmissa, jotta niihin voidaan paneutua välittömästi, joka siten vaikuttaa myönteisesti kirjoitetun koodin laatuun ja mahdollisten virheiden välittömään korjaamiseen. (6).

## 3.2 Jenkins

Jenkins on ohjelmointikieli Javalla toteutettu jatkuvan integraation ohjelma. Jenkins on myöskin avointa lähdekoodia (open source) sekä sillä on suuri ja aktiivinen yhteisö takanaan, joka selittää osittain miksi se on niin suosittu. Aluksi Jenkins ei ollut Jenkins, vaan Hudson, jonka kehityksen Kohsuke Kawaguchi vuonna 2004 aloitti harrasteprojektinaan. Vuonna 2011 Hudsonin kehittäjäyhteisö päätti vaihtaa nimen Jenkinsiksi ja migratoida Hudsonin koodin GitHub-tiedostovarastoon ja jatkaa kehitystyötään siellä. [6.]

## 4 Virtualisointi

Jotta voidaan ymmärtää mitä pilvipalveluilla tarkoitetaan, on sitä ennen syytä oppia ymmärtämään mitä virtualisointi käsitteenä tarkoittaa. Yksinkertaisimmillaan esimerkin aiheesta muotoilisin seuraavasti. Haja-asutusalueella asuvia, oppivelvollisia lapsia joudutaan päivittäin kuljettamaan perheen autolla kouluun. Näin ollen yksi oppilas kulkee yhdessä autossa koulumatkansa. Kun tämä toistuu jokaisen oppivelvollisen lapsen perheen kohdalla, kulkee huomattava määrä autoja päivittäin viemässä ja noutamassa lapsia koulumatkoillaan. Jossain vaiheessa eräs vanhemmista miettii auton polttoainekuluja maksaessaan, että miksi tuhlataan näin järjettömästi rahaa ja aikaa, kun kaikki lapset matkustavat koulumatkansa yksin vanhempiensa kyydissä ja keksii perustaa kyyditystä varten koulukyytiryhmän, jossa lapsia kuljetetaan samassa autossa useampia ja mieluiten niin, että saadaan auton istuinkapasiteetti mahdollisimman hyvin käytettyä. Tämä taas johtaa siihen, että yhdellä autolla voidaan suorittaa sama tehtävä kuin aiemmin ja saadaan samalla auton istuinresurssit selkeästi fiksummin valjastettua käyttöön eikä juurikaan hukkapaiikkoja jää, jolloin auto on tehokkaammassa käytössä, polttoaineen kulutuksen osuus per lapsi laskee kyytiläisten määrän mukaan ja huolto- sekä korjauskustannukset halpenevat, kun summa voidaan jakaa lasten vanhempien kesken. Huomionarvoista tällä järjestelyllä on se, että yksi auto pystyy suorittamaan usean tehtävän eli lapsen kuljetuksen samanaikaisesti, kun on ymmärretty käyttää auton istuinresursseja järkevämmiin hyödyksi ja voidaan mahdollisesti jopa luopua useammasta autosta, koska niitä ei yksinkertaisesti enää tarvita tätä tehtävää varten.

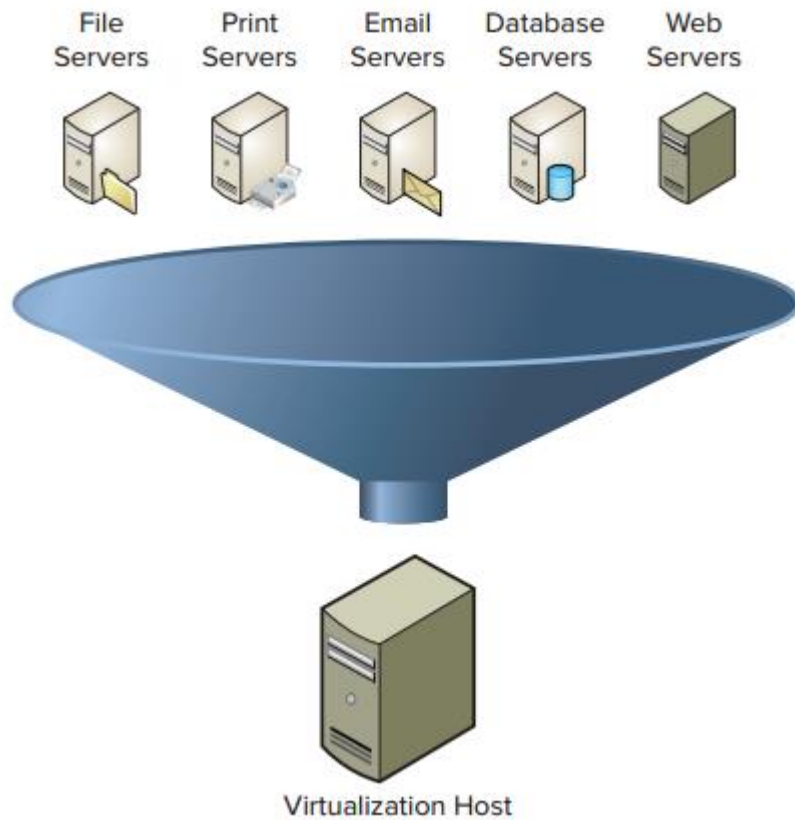
Tietotekniikassa, jossa perinteisesti on suoritettu tiettyjen ohjelmien ajo siten, että jokaista tehtävää varten, kuten esimerkiksi sähköpostipalvelua, on sille dedikoitu oma palvelin, joka tehtävä on pelkästään pitää huoli siitä, että esimerkiksi sähköposteja voidaan ottaa vastaan ja lähettää eteenpäin. Tämä yleensä kuormittaa palvelinta kohtuullisesti

tai jopa hyvin vähän, jolloin ollaan tilanteessa, jossa palvelin hoitaa tehtävänsä moitteettomasti, mutta palvelimen kapasiteetista jää paljon täysin käyttämättä, koska kapasiteettia suorittaa vaativampiakin tehtäviä olisi tarjolla. Kun vain yhtä tehtävää suorittavia palvelimia alkaa kertyä, ollaan tilanteessa, jossa on suuri määrä kapasiteettia tarjolla, kuten laskentatehoa prosessorien toimesta, mutta sitä käytetään vain vähän, koska muita palveluita ei joko haluta tai uskalleta asentaa ajettavaksi samalle palvelimelle mahdollisen vikatilanteen takia, jolloin vikatilanne aiheuttaa toimintakatkoja useammalle tarjotulle palvelulle. [9, s. 4-5.]

Virtualisoinnilla päästään hukkakapasiteetista ja useamman palvelun ongelmatilanteesta yhdellä palvelimella mainiosti eroon. Virtualisoinnilla tarkoitetaan sitä, että hypervisor-ohjelma luo kerroksen laitteen fyysisten resurssien päälle tai toisena vaihtoehtona luo kerroksen laitteen käyttöjärjestelmän päälle. Hypervisorin päälle voidaan asentaa virtuaalikoneita, jotka ovat ohjelmallisesti toteutettuja tietokoneita tietokoneen sisällä. [10, s. 21-24.]

Virtuaalikone luulee olevansa siis aivan aito tietokone, jolla on tietty määrä fyysisiä resursseja tarjolla, joita käyttää sillä ajettaviin tehtäviin. Kun aiemmin mainituille palvelimille asennetaan ensin virtualisointikerros, voidaan virtualisointikerroksen päälle asentaa virtuaalikoneita, joissa voidaan ajaa eri palveluita, kuten yhdessä sähköpostiohjelmistoa ja toisessa esimerkiksi tietokantaa. Näin ollen saadaan yhden palvelimen resurssit paremmin käyttöön ja samalla palvelimella voidaan ajaa useampia palveluita, jotka eivät ole samalla tavalla haavoittuvia, kuin perinteisesti yhdellä palvelimella ajettavia useampia palveluita.

Esimerkkini avulla voidaan ajatella, että koulukyytiryhmänä toimii hypervisor, joka mahdollistaa olemassa olevien resurssien paremman käytön, ottamalla useampia lapsia kyytiin kerralla samaan ajoneuvoon, tai ajamalla samalla palvelimella useampaa virtuaalikonetta, jotta jokaista tehtävää varten ei tarvita omaa palvelinta. Virtuaalikoneista mainittakoon vielä positiivisena puolena, että ennen virtualisaatiota jouduttiin jokaista käyttöjärjestelmää kohden pitämään yllä sille dedikoitua palvelinta, jossa kyseisen käyttöjärjestelmän yhteensopivaa ohjelmistoa tai palvelua suoritettiin, mutta virtuaalikoneissa voidaan ajaa mielivaltaisesti haluttua käyttöjärjestelmää eikä se ole sidonnainen palvelimen käyttöjärjestelmästä, jossa virtuaalikoneita ajetaan. Täten voidaan samalla palvelimella ajaa esimerkiksi yhdessä virtuaalikoneessa Windowsilla suoritettavaa palvelua tai ohjelmaa sekä toisessa virtuaalikoneessa Linuxilla suoritettavaa tietokantaa (kuva 5).



Kuva 5. [10, s. 10.]

#### 4.1 Hypervisor

Ilman hypervisorikerrosta virtuaalikoneet taistelisivat kilpaa suoraan fyysisten resurssien käytöstä, jolloin seuraamukset tuskin olisivat kovinkaan hyvät. Hypervisor isännöi siis olemassa olevien, fyysisten resurssien ja virtuaalikoneiden välissä välittäen virtuaalikoneiden esittämät resurssienkäyttöpyynnöt fyysisille resursseille päästämättä virtuaalikoneita itse komentamaan fyysisiä resursseja. [10, s. 19.]

Hypervisoreita on kahta tyyppiä. On tyyppin 1 hypervisor, joka on suoraan fyysisten resurssien päälle asennettava ohjelmisto ja tyyppin 2 hypervisor, joka asennetaan käyttöjärjestelmän, kuten Windows tai Linux, päälle. [10, s. 21-24.]

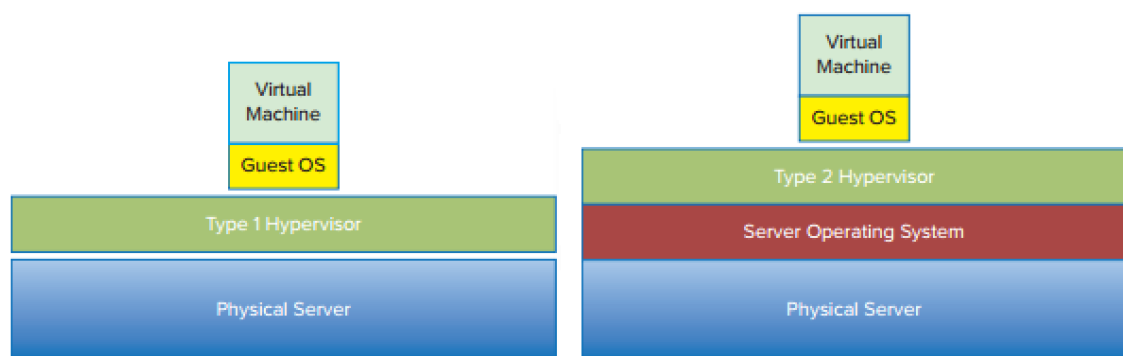
Tyyppin 1 hypervisor sijaitsee siis suoraan fyysisten resurssien ja virtuaalikoneiden välissä, jolloin erillistä käyttöjärjestelmää ei tarvita. Koska tyyppin 1 hypervisor kommunikoi suoraan fyysisten resurssien kanssa, johtaa tämä siihen, että virtuaalikoneita voidaan



luoda tällaiselle palvelimelle useampia, koska hukkaprosesseja välissä olevan OS:n (operating system) kanssa ei ole. [10, s. 21-22.]

Tyyppin 2 hypervisor sijaitsee taas OS:n päällä eli sen alla on käyttöjärjestelmä. Tämä tarkoittaa sitä, että kun VM (virtual machine) lähettää esimerkiksi pyynnön saada laskenta-aikaa prosessorilta, saa hypervisor siitä tiedon ja välittää sen alla olevalle käyttöjärjestelmälle, joka välittää sen taas eteenpäin prosessorille, josta palataan käänteisessä järjestyksessä takaisin. Tämä lisää siis hukkatyötä (overhead). [10, s.23-24.]

Tyyppin 1 ja tyyppin 2 hypervisorit esitellään kuvassa 6.



Kuva 6. Eri hypervisor-tyypit [10, s.22-23.]

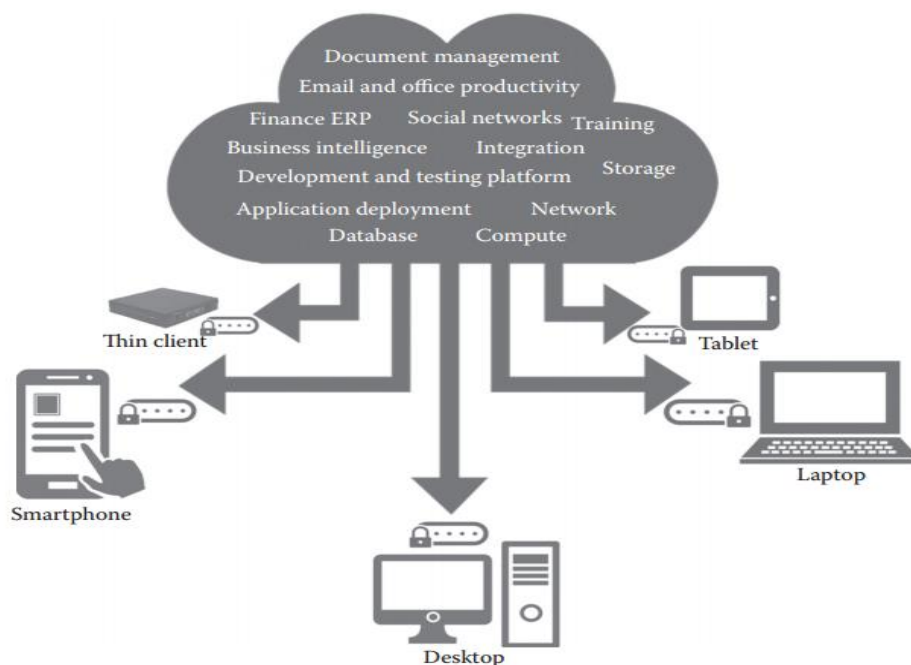
## 5 Pilvipalvelut

Pilvipalveluilla (cloud computing) tarkoitetaan palvelua, joka on aina saatavilla, verkko-yhteyden päässä, jossa on jaettu resurssivaranto ja josta voi pyynnöstä ottaa käyttöönsä tarvitsemansa resurssit, joita voidaan tarpeen kasvaessa skaalata suuremmiksi tai tarpeettomina vaivatta hävittää. [11, s. 47-48.] Pilvipalvelu voidaan siis määrittää resurssi-varannoksi, johon ollaan yhteydessä internetin välityksellä ja on aina saatavilla.

### 5.1 Pilvipalvelun keskeiset piirteet

Pilvipalvelun keskeisiin piirteisiin kuuluu itsepalvelu tarpeen mukaan. Kun asiakas tarvitsee esimerkiksi lisää kapasiteettia palvelimelleen, voi hän itse määrittää tarpeensa mukaan paljonko kapasiteettia haluaa lisätä. Toinen keskeinen piirre on saavutettavuus. Pilvipalveluiden resurssit ovat saavutettavissa internet-yhteyden välityksellä

laitealustasta riippumatta esimerkiksi tabletilla tai matkapuhelimella (kuva 7). Jaettu varanto on myös yksi keskeisistä piirteistä, jolla tarkoitetaan saatavilla olevien resurssien dynaamista saatavuutta, joka näyttäytyy asiakkaalle lähes loputtomana tietoteknisenä varantona, josta asiakas voi määrittää tarpeensa mukaan resurssit tarvitsemalleen palvelulle. Asiakaskohtaisia resursseja ei siis ole, vaan resurssit ovat saatavilla kaikille. Joustava provisiointi on myös osa keskeisiä piirteitä, jolla tarkoitetaan sitä, että asiakas voi pilvipalveluissa skaalata haluamaansa palvelua suuremmaksi tai osoittaa sille lisää suorittajia, myös automaattisesti. Joustavalla provisioinnilla asiakas saa siis suuremman vallan resurssiensa käytöstä. Mitattavuudella tarkoitetaan käytettyjen palveluiden käytön mittausta, jonka mukaan asiakas voi seurata käyttämiensä resurssien käytön määrää ja sen mukaan suunnitella käyttöönsä sekä pilvipalveluntarjoaja voi seurata resurssien käyttöä, jonka perusteella laskuttaa asiakasta. [11; 12, s.14-15.]



Kuva 7. [12, s. 11.]

## 5.2 Pilvien tyypit

Pilvestä puhuttaessa tulee ottaa huomioon, että pilvipalveluita on useita eri tyyppisiä ja ne jaetaan yleensä neljään erilaiseen pilvimalliin.

1. Yksityinen pilvi. Yksityistä pilveä voi käyttää vain sille erikseen määritelty taho. Tämä taho, kuten organisaatio, voi omistaa yksityisen pilven ja mahdollisesti pitää sitä omassa datakeskuksessaan tai julkisessa pilvipalvelussa. [12, s. 15.]

2. Julkinen pilvi. Julkinen pilvi on avoin kaikille, joten jokainen voi julkisen pilven pilvipalveluntarjoajalta ostaa tarvitsemaansa palvelua itsellensä käyttöön. [12, s. 15.]

3. Yhteisöpilvi. Yhteisöpilvessä voi olla käyttäjinä useampia organisaatioita tai tahoja, jotka mahdollisesti tekevät yhteistä projektia yhteisöpilvessä. Yhteisöpilvi voi sijaita joko julkisessa pilvessä tai yhteisön omilla palvelimilla. [12, s. 15.]

4. Hybridipilvi. Hybridipilvi koostuu kahdesta tai useammasta yllämainituista pilvimalleista. Osaa palveluista halutaan suorittaa yksityisessä pilvessä ja osaa julkisessa. [12, s. 16.] Esimerkkinä voisin mainita organisaation kotisivut, jotka sijaitsevat julkisessa pilvessä ja asiakastietokanta, joka sijaitsee organisaation omassa, yksityisessä pilvessä.

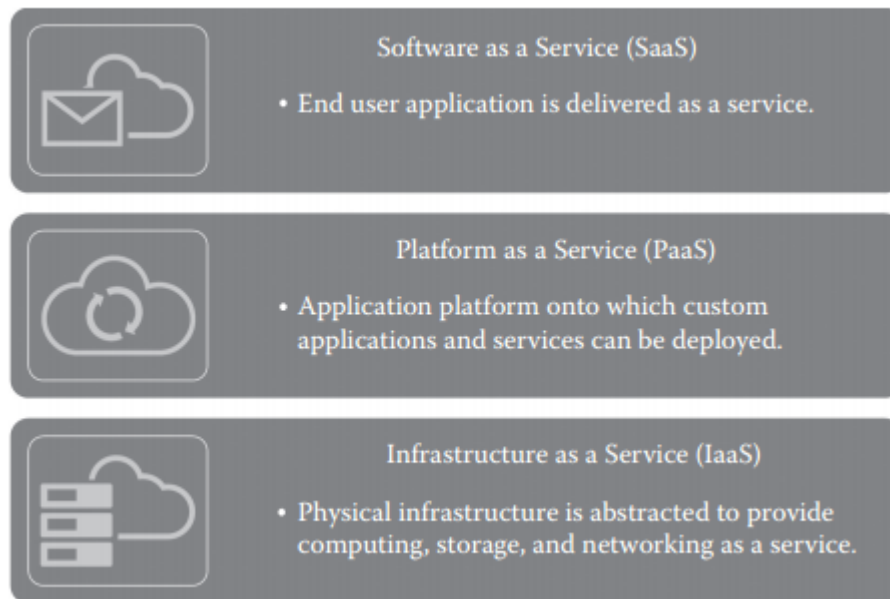
### 5.3 Palvelumallit

Palvelumalleja pilvessä on kolmea erilaista. SaaS (Software as a Service), PaaS (Platform as a Service) ja IaaS (Infrastructure as a Service) (kuva7).

SaaS-palvelumallissa asiakas ostaa palveluna pilvipalvelun, jossa ajetaan asiakkaan tarvitsemaa ohjelmistoa, kuten esimerkiksi kirjanpito-ohjelmistoa, johon asiakas voi ottaa yhteyden haluamallaan laitteella, kuten tietokone tai tabletti. Asiakas voi ottaa yhteyden ohjelmistoon esimerkiksi laitteensa selaimella tai ohjelmiston vaatiman asiakasohjelman avulla. Asiakas siis maksaa siitä, ettei hänen tarvitse itse konfiguroida ohjelmistoa tai sen tarvitsemia resursseja, vaan pelkästään ohjelmiston käytöstä. [12, s. 16-17.]

PaaS-palvelumallissa asiakas ostaa alustapalvelun itselleen, johon palveluntarjoaja tarjoaa alustan alla olevan infrastruktuurin sekä esimerkiksi palvelimella sijaitsevat kehitystyökalut, johon kehittäjä voi nyt sijoittaa kirjoittamaansa koodia ja ottaa sen käyttöön palvelimella asiakkaan ajamassa ohjelmassa. Asiakas ei siis kontrolloi alustamallissa pilviinfrastruktuuria koska ostaa tämän palveluna, vaan palvelimen sisällä ajettavaa asiakkaan ohjelmaa tai mahdollista kehitysympäristöä. [12, s. 17.]

IaaS-mallissa koko infrastruktuuri on asiakkaan konfiguroitava itse tarpeensa mukaan, josta pilvipalveluntarjoaja laskuttaa asiakasta käytön mukaan. Asiakas siis itse määrittelee millaisen virtuaalikoneen haluaa ottaa käyttöön, asentaa tarvitsemansa käyttöjärjestelmän ja ohjelmistot itse sekä määrittää tarvittavat verkkoyhteydet. Voidaan siis katsoa, että pilvipalveluntarjoaja tarjoaa vain resursseja joista asiakas valitsee tarvitsemansa, joista itse rakentaa koko infrastruktuurin tarpeidensa mukaisesti. [12, s. 17.]



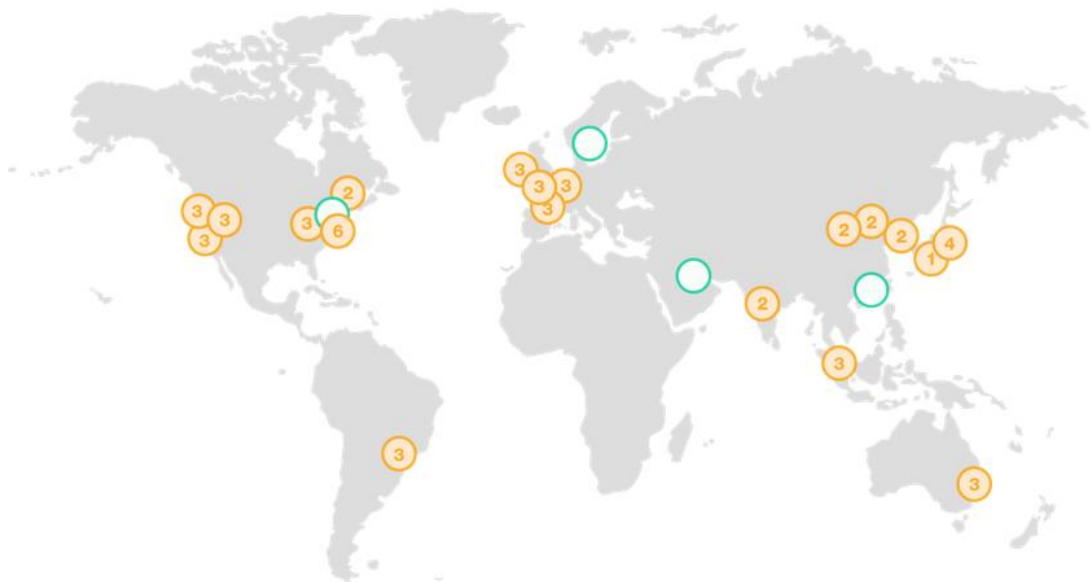
Kuva 8. Palvelumallit. [12. s, 16.]

## 6 Amazon Web Services

Amazon Web Services on pilvipalveluntarjoaja, joka usein lyhennetään yksinkertaisemmin AWS. Amazon Web Services on verkkokaupankäynnistä ja erityisesti verkkokirjakaupastaan tunnetun Amazonin tytäryhtiö.

Amazon Web Services julkistettiin virallisesti vuonna 2006 ja tarjosi vain pelkän tallennuspalvelun, Simple Storage Servicen, joka nykyisin tunnetaan nimellä S3. Myöhemmin samana vuonna tarjottiin uusina palveluina Simple Queue Service (SQS) ja Elastic Compute Cloud (EC2), joka tarjosi laskentakapasiteettia tarvittaessa ilman sitoumuksia käyttöaikojen suhteen. [13, s. 10-11.]

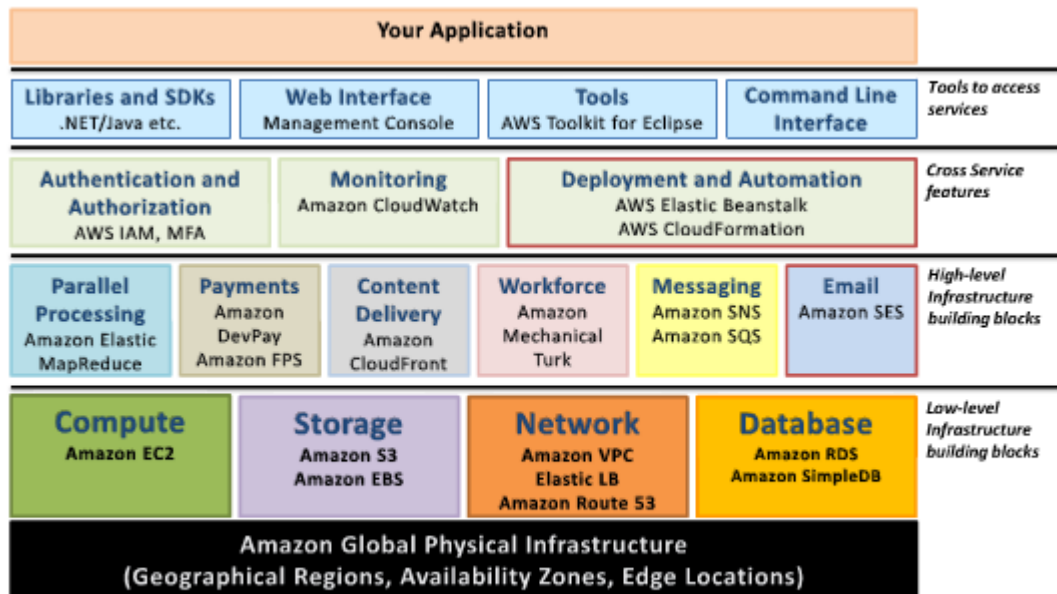
Amazon Web Servicesillä on alueita (region) lähes jokaisella mantereella. Aluetta valitessa tulee miettiä missä nyt tarvittavia palveluita käytetään eniten, jonka mukaan valitaan alue, koska alueen maantieteellinen sijainti vaikuttaa palvelun saatavuuden nopeuteen. Alueilla sijaitsee yksi tai useampi saatavuusalue (availability zone), joissa voi olla yksi tai useampi datakeskus. Yhdellä alueella sijaitsevat saatavuusalueet ovat yhdistetty toisiinsa erittäin nopeilla yhteyksillä. Kannattaa myös palvelua suunnitellessa jaella palvelu, mikäli mahdollista, useammalle saatavuusalueelle, jotta palvelu pysyy saatavilla jos jostain syystä yksi saatavuusalue vioittuu ja verkkoyhteyttä ei voida sinne muodostaa. [11, s. 7-8.] Alueita, kuten kuvassa 9 esitetään keltaisina ympyröinä, on AWS:llä lukuisia kapaleita eri maanosissa, joista löytyy useita saatavuusalueita, jotka esitetään keltaisen ympyrän sisällä olevalla luvulla. Kuvassa vihreällä on myös mukana tulevia, ei vielä käyttöönotettuja alueita.



Kuva 9. Amazon Web Servicesin alueet, saatavuusalueet sekä tulevat alueet. [21.]

## 6.1 Perustapalveluiden kuvaukset

Perustana Amazon Web Servicesin palveluissa on mainittava peruspilarit, joiden päälle rakentuu monia palveluita, laskenta (compute), tallennus (storage), tietokanta (database) ja verkkoyhteydet (networking), joiden päälle voidaan kasata tarvittavia palveluita (kuva 10). [11, s. 54.]



Kuva 10.

### 6.1.1 Laskentapalveluita

Laskennasta esiin on nostettava Elastic compute cloud (EC2), palvelu, joka tarjoaa laskentatehoa tarvittaessa, jota voi joustavasti tarpeen mukaan kasvattaa tai vähentää. EC2 on siis virtuaalikone, jonka asiakas voi pystyttää ja tarvittaessa pystyttää juuri niin monta kuin tarvitsee ja kun tarve poistuu, voi asiakas poistaa koneet käytöstä ja laskutus muodostuu pelkästään käytön mukaan. AWS kutsuu virtuaalikoneita instansseiksi. [11, s. 54-55.; 11, s. 110.]

Toinen tärkeä palvelu on virtuaalinen, yksityinen pilvi (VPC). VPC:n avulla voidaan rakentaa täysin eristettyjä sekä täysin muokattavissa olevia yksityisiä pilviä Amazon Web Servicesin sisällä. VPC:n avulla voidaan tuoda lisää turvallisuutta yhteyksien hallinnan muodossa jo olemassa olevien EC2 instanssien lisäksi. Esimerkiksi VPC:n avulla voidaan määrittää mitkä instanssit saavat olla yhteydessä internetiin ja mitkä eivät. VPC:n sisälle voidaan luoda omia verkkoja ja aliverkkona tarpeen mukaan sekä luoda käyttöoikeuslistoja (access control list), joiden avulla voidaan täsmällisesti määritellä mitkä yhteydet ovat sallittuja mihinkin ja mitkä ovat kiellettyjä. [11, s. 55, 188.]

### 6.1.2 Tallennuspalveluita

Tallennuksessa ensimmäisenä on AWS:n ensimmäinen palvelu, Simple Storage Service eli S3. S3 on objektitallennusjärjestelmä, jota voidaan hallita esimerkiksi selaimen kautta tai REST rajapinnan avulla. S3:een voidaan siis tallentaa esimerkiksi lomakuvia, jotka ovat aina saatavilla internet-yhteyden avulla. S3:n säiliöihin (bucket) voidaan myös antaa oikeuksia, kuten lataus ja tallennusoikeuksia, jolloin määritellyt käyttäjät voivat esimerkiksi ladata kuvia toisen käyttäjän säiliöstä. [14, s. 11.; 11, s. 343.]

Elastic block storage (EBS) tarjoaa loogisen levyblokin tallennusta varten ja jonka voi liittää ja irroittaa EC2 instanssiin. Kun blokin irroittaa instanssista, ei tallennettu data katoa loogiselta levyltä. [11, s. 55.; 14.]

Glacier tallennuspalvelu on taas S3:n kaltainen tallennuspalvelu, joka on tarkoitettu pitkäaikaisempaa datan varastointia varten, kuin S3 esimerkiksi varmuuskopiointiin. [11, s. 55.]

### 6.1.3 Tietokantapalveluita

Amazon Web Servicesin relaatiotietokantapalveluna toimii RDS (Relational Database Service), joka tarjoaa skaalutuvaa ja korkean suorituskyvyn relaatiotietokantapalvelua esimerkiksi MySQL järjestelmille. AWS:n tarjoaman tietokantapalvelu tarjoaa myös lisäpalveluita, kuten automatisoitua varmuuskopiointia ja useamman saatavuusalueen kesken replikoitavaa palvelua. [11, s. 55, 307.]

DynamoDB tarjoaa puolestaan NoSQL tietokantapalvelua

Redshift on taas suunniteltu suuren datan käsittelyä varten, kuten datalouhinnalle. [11, s. 55.]

### 6.1.4 Verkkopalveluita

Eräs palveluista on elastinen kuormantasaaja ELB (Elastic Load Balancer), joka nimensä mukaisesti on suunniteltu jakamaan kuormaa tasaisesti EC2 instansseille. Voidaan siis määritellä miten saapuva liikenne jaotellaan instansseille. Toisin sanoen verkkoliikenne kulkee ensin ELB:hen, joka jakaa liikenteen EC2 instansseille. [11, s. 56, 272.]

## 7 Raportti

### 7.1 VPC ja EC2

Aivan ensimmäiseksi pitää luoda tili Amazon Web Servicesiin, jota ei tässä työssä käydä läpi. Kun tili on luotu, voidaan kirjautua palveluun sisään. Kirjautumisen jälkeen voidaan aloittaa rakennusvaihe. Aluksi pohjalle luodaan virtuaalinen, privaattipilvi eli VPC. Aloitetaan siis valitsemalla Servicesin alta VPC. VPC:n luonti ei ole kovin monimutkaista. Apuna voidaan käyttää AWS:n omaa VPC Wizardia valitsemalla Start VPC Wizard. Wizard on tyypillisesti luotu helpottamaan asioiden määrittelyä helpoilla ja hyvin määritellyillä askelilla, joissa käyttäjältä vaaditaan lähinnä yksinkertaisiin kysymyksiin vastauksia. Wizardin avulla voidaan siis saada monimutkaiset määrittelyt yksinkertaistettua muutamilla kysymyksillä, joihin käyttäjä vastaa ja tämän perusteella jokin palvelu luodaan tai mahdollisesti jokin ohjelma luodaan. [15.]

Kun VPC Wizard on aloitettu, pitää määrittää minkälaisia, mahdollisia aliverkkoja tarvitaan ja sen mukaan valitaan vaihtoehtoista sopiva. Tässä tapauksessa ei tarvita kovin monimutkaista verkkoa, joten valitaan yhden julkisen aliverkon VPC. Verkon kooksi valitaan yksi yleisimmin käytetyistä aliverkoista eli aliverkon peitteellä 24 luotava verkko, joka sisältää tässä tapauksessa 251 vapaata IP-osoitetta. Tästä osasta voidaan vielä lohkoa pienempi verkko, jos VPC:hen halutaan luoda useampia, pienempiä aliverkkoja myöhemmin. Määritellään myös VPC:n nimi sekä saatavuusalue, johon verkko luodaan. Muutama muukin määrittely VPC:lle voidaan tehdä, mutta koska niitä ei tarvita tässä tapauksessa, niihin ei myöskään tarkemmin paneuduta. Tämän jälkeen voidaan luoda verkko valitsemalla Create VPC. Mikäli kaikki menee suunnitelmien mukaan, saadaan vastaus, jossa todetaan VPN:n luonnin onnistuneen ja voidaan jatkaa painamalla OK.

VPC:n luonnin jälkeen voidaan siirtyä luomaan EC2-instanssia eli virtuaalikonetta. Tämä on myöskin kovin yksinkertaista. Valitaan vain Services ja palveluiden alta EC2. Päästään näkymään, jossa on yhteenveto käytössä olevista resursseista, kuten montako instanssia on tällä hetkellä ajossa tai esimerkiksi montako tilannekuvaa on virtuaalikonesta otettu. Valitaan Launch Instance, josta päästään valitsemaan levykuvaa, joita on lukuisia eri käyttöjärjestelmistä kuten eri Linux-versioita tai esimerkiksi Windows Serverit. Saatavilla on myös eri käyttötarkoituksia varten luotuja levykuvia, kuten deep learning käyttötarkoitusta varten luotuja levykuvia, joissa on jo valmiiksi asennettuna tarpeellisia ohjelmistoja tätä käyttötarkoitusta varten. Nyt valitaan kuitenkin tätä



käyttötarkoitusta varten AWS:n tekemä Amazon Linux 64-bittinen versio, joka integroituu AWS:n työkalujen kanssa hienosti yhteen ja jossa on esimerkiksi valmiiksi asennettuna työkaluja, jotka ovat yhteensopivia AWS:n rajapinnan (API) kanssa. [16.] Jos mieleistä levykuvaa ei löydy tai halutaan se rakentaa itse, on se myös mahdollista.

Seuraavassa askeleessa määritellään instanssin resurssien tarpeet eli millainen virtuaalikoneline rakennetaan. Koska kyseessä on testiympäristö, jossa ei suurta kuormaa tulla ajamaan, vaan testataan ylipäättään toiminnallisuutta tulevaisuutta varten, valitaan t2.micro, joka on varustettu yhdellä virtuaalisella prosessorilla sekä yhdellä gigatavulla muistia. Saatavilla on lähes kaikkien kuviteltavaan käyttötarkoitukseen soveltuvia instansseja, jotka sisältävät muun muassa yhdestä virtuaalisesta prosessorista 128:aan virtuaaliseen prosessoriin tai puolesta gigatavusta lähes neljään tuhanteen gigatavuun muistia.

Siirrytään määrittämään instanssin konfiguraatiota. Määritellään montako instanssia tarvitaan, sekä valitaan juuri aiemmin luotu VPC ja sen yhteydessä luotu aliverkko, johon instanssi tullaan liittämään. Voidaan myös määrittää annetaanko uudelle instanssille heti julkinen IP-osoite tai halutaanko useampi verkkoliitäntä kytkeä virtuaalikoneseen. Näitä voidaan muokata myöhemmin, joten tässä vaiheessa ei esiinny tarvetta näitä tai muita mahdollisia asetuksia muokata.

Nyt voidaan joko siirtyä tarkastelemaan yhteenvetoa tulevasta instanssista tai jatkaa määrittelemään tulevaa tallennustilaa, jossa käytetään loogista levyblokkia (EBS), jonka koko voidaan määrittää tarpeen mukaan ja esimerkiksi tuhotaanko myös levyblokki, kun instanssi tuhotaan vai säästetäänkö levyblokki mahdollisia tulevia tarpeita varten. Voidaan myös määritellä merkintöjä (tags) instanssiin, jotka helpottavat identifioimaan esimerkiksi instanssin käyttötarkoitusta tai instanssia käyttävän tiimin. Seuraavaksi määritellään käyttöoikeusryhmä (security group), jolla voidaan erotella millaista liikennettä ohjataan instanssiin esimerkiksi porttien perusteella ja mistä IP-osoitteista tähän porttiin voidaan olla yhteydessä eli eräänlainen palomuuuri. [17.] Tämän jälkeen voidaan katsoa yhteenvedossa, että kaikki on halutulla tavalla ja aloittaa instanssin rakennusprosessi valitsemalla Launch.

Tämän jälkeen siirrytään avainparin luontiin. Avainparin avulla päästään kirjautumaan luotuun instanssiin. Huomionarvoista on, että avainparia ei voi ladata uudelleen, joten se kannattaa säilöä tämän vuoksi erittäin tarkasti, koska jos yksityinen avain, joka omalle tietokoneelle ladataan hukkuu, ei instanssiin voida kirjautua enää sisälle jos ei jo

aiemmin ole kirjautunut instanssiin ja esimerkiksi luonut toista käyttäjätiliä. Avainparin luonnin tai käytettävän avainparin valinnan jälkeen voidaan instanssin luonti aloittaa. Instanssin luonnissa vierähtää pieni tovi, joten voidaan mennä katsomaan onko instanssi jo valmis valitsemalla Services ja sen alta EC2. Kun instanssin tila on ajossa (running) sekä tarkastukset ovat menneet läpi, on instanssi valmiina käyttöön.

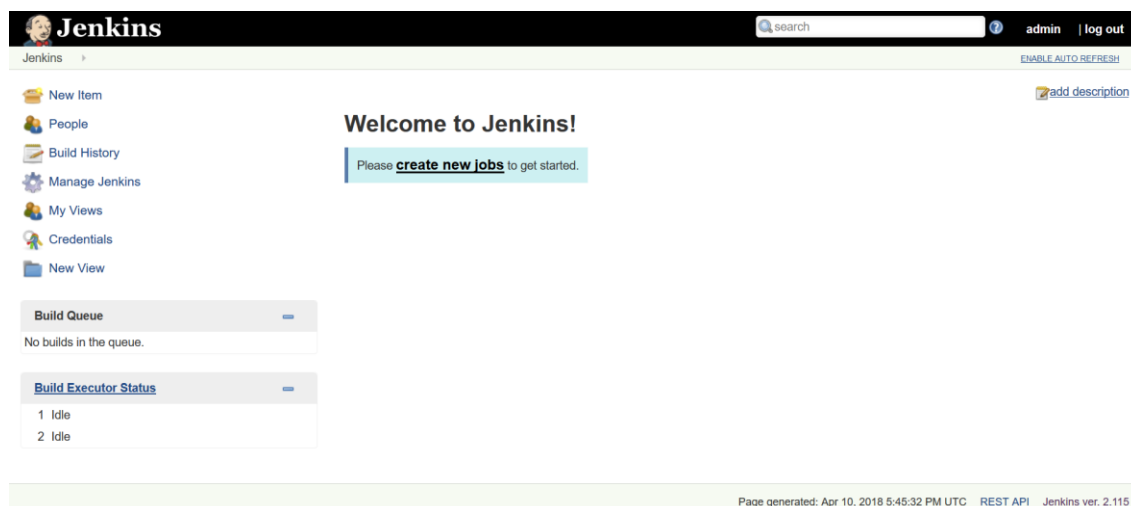
Jotta instanssiin voidaan muodostaa yhteys internetin välityksellä, tarvitsee instanssi julkisen IP-osoitteen. Valitaan klikkaamalla esimerkiksi instanssin nimeä hiiren oikealla napilla, jolloin saadaan avattua valikko, josta valitaan networkingin alta Manage IP Addresses. Valitaan Allocate an Elastic IP. Nyt luotu elastinen IP-osoite pitää liittää vielä instanssiin, jotta yhteys voidaan muodostaa. Valitaan Elastic IPs vasemman reunan valikosta, etsitään juuri luotu IP-osoite, avataan valikko klikkaamalla hiiren oikealla napilla juuri luotua IP-osoitetta ja valitaan Associate address. Seuraavaksi valitaan luotu instanssi, johon IP-osoite liitetään sekä yksityinen IP-osoite, jonka instanssi sai luotaessa ja varmistetaan liittäminen painamalla Associate. Nyt yhteyden muodostus instanssiin onnistuu käyttämällä ssh-yhteyttä. Sisään kirjaudutaan tarjoamalla yksityistä avainta, joka valittiin käytettäväksi instanssin määrittelyvaiheessa, sekä käyttämällä käyttäjänimeä ec2-user.

## 7.2 Jenkins

Kun alusta Jenkinsille on asennettu, voidaan siirtyä asentamaan jatkuvan integraation ohjelmisto Jenkinsiä. Jenkinsin asennukseen löytyy selkeät, muutaman komennon ohjeet Jenkinsin dokumentaatiosta, joten en käy tätä prosessia tarkemmin lävitse.[18.] Ensimmäiseksi rakennetaan isäntäkone. Asennuksen jälkeen palvelu käynnistetään ja jotta voidaan ottaa yhteys selaimella instanssiin, joudutaan määrittelemään käyttöoikeusryhmään salliva sääntö saapuvalla liikenteelle porttiin 8080, jota Jenkins käyttää. Tämä on konfiguroitavissa myös muuhunkin porttiin jos niin haluaa. Security group valitaan vasemmalta valikosta ja avataan oikean käyttöoikeusryhmän valikko hiiren oikealla napilla ja valitaan valikosta Edit inbound rules. Luodaan uusi sääntö Add rule:sta painamalla ja valitaan Custom TCP rule sekä määritellään portti ja sallittu IP-osoite tai IP-osoiteavaruus josta yhteydet sallitaan ja tallennetaan muutos Save napilla.

Nyt voidaan avata selain ja osoitekenttään syötetään instanssin julkinen IP-osoite sekä määritellään ottamaan yhteys oikeaan porttiin lisäämällä osoitteen perään kaksoispiste sekä käytettävä portti. Ensimmäistä kertaa käynnistettäessä Jenkins pyytää ylläpitäjän

salasanaa ja antaa ohjeet mistä tämä löytyy. Annetaan pyydetty salasana ja palvelu avautuu käyttäjälle, jossa ensimmäisenä on valittavana liitännäisten (plugins) asennus joko suositeltujen liitännäisten tai valitsemalla itse listasta liitännäiset, jotka asennetaan. Tämän jälkeen luodaan ylläpitäjän tili, jolloin palvelu avautuu käyttövalmiina eteen (kuva 11).



Kuva 11. Jenkinsin aloitusnäky.

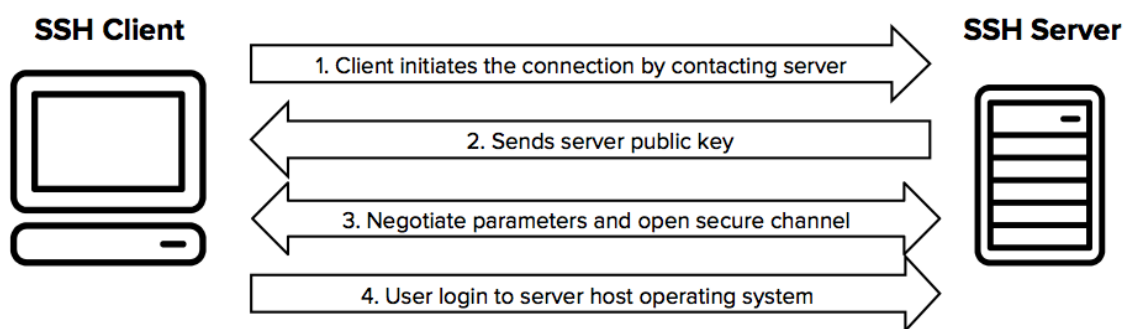
Omasta kokemuksesta voin suositella lisäämään heti uuden käyttäjän, jottei ylläpitäjän tiliä käytetä turhaan sekä mahdollisen ongelmatilanteen sattuessa, on mahdollisuuksien mukaan toinen käyttäjätili vielä vaihtoehtona korjaamaan ongelmatilannetta. Tämä tapahtuu valitsemalla Manage Jenkins, Manage Users, Create User ja antamalla pyydyt tiedot.

Seuraava askel on asentaa tarvittavat liitännäiset joita tullaan käyttämään eli asennetaan Amazon EC2-liitännäinen sekä GitHub-liitännäinen. Valitaan taas Manage Jenkins, jonka alta Manage Plugins. Avautuvasta näkymästä valitaan Available-välilehti ja haku-kenttään kirjoitetaan ensimmäinen liitännäinen ja valitaan Install without Restart. Kun tarvittavat liitännäiset ovat ladattu ja asennettu, voidaan siirtyä rakentamaan agenttia juuri rakennetulle isännälle.

Luodaan aiemman mukaan uusi EC2 instanssi, mutta nyt luotavan instanssin kohdalla voidaan valita käyttöön jo aiemmin luotu ja määritelty turvallisuusryhmä sekä jo käytössä oleva avainpari, joten näiden luontia ei tarvitse tehdä uudelleen. Kun instanssi on käyttövalmis, pitää määrittellä turvallisuusryhmään uudet säännöt isännän ja agentin välille,

jotta yhteys voidaan muodostaa. Tämä voidaan tehdä olemassa olevaan turvallisuusryhmään tai luoda tätä varten uusi. Turvallisuusryhmiä voidaan liittää useita samaan instanssiin, joten liikenne voidaan seuloa tarkasti halutun kaltaiseksi. Esimerkiksi ei ole järkeä käyttää samoja sääntöjä agentin ja isännän välillä, kun isännällä on, koska isäntä on julkiseen internetiin yhteydessä, jota taas agentin ei tarvitse muodostaa. Luodaan siis uusi ryhmä aiemman ohjeistuksen avulla, jossa sallitaan ssh-liikenne isännän ja agentin välillä.

Kun sallivat muutokset on saatu tehtyä, ja instanssille, johon yhdistetään, julkinen avain on sijoitettu asiaankuuluvalla tavalla, voidaan kirjautuminen suorittaa. Julkisen avaimen sijoituksen AWS tekee oletuskäyttäjän kohdalla aina, joten voidaan kirjautua oletuskäyttäjällä sisään uuteen instanssiin niin halutessa.



Kuva 12. SSH-avainperusteinen kirjautuminen. [19.]

Jos halutaan testaamalla todeta toiminnallisuus, voidaan agentti tuoda käyttövalmiuteen lisäämällä isännän konfiguraatioon uusi agentti, joka tapahtuu Manage Jenkinsin kautta. Täältä valitaan Manage Nodes ja New Node. Valitaan minkä tyyppinen agentti luodaan ja määritellään nimi. Seuraavaksi aukeavassa ikkunassa tässä tapauksessa määritetään vain agentilla sijaitseva juurihakemisto Jenkinsille, agentin käynnistystapa, johon valitaan ssh-yhteyden kautta sekä syötetään agentin IP-osoite ja mitä pääsytietoja (credentials) käytetään kirjautumiseen. Nämä pitää toki ensin luoda Jenkinsiin, jotta niitä voidaan käyttää. Tähän voidaan syöttää jo olemassa oleva käyttäjä eli oletuskäyttäjää, jonka avainpari on jo olemassa ja täten niitä ei tarvitse luoda uudelleen. Kun tiedot on syötetty ja tallennettu, käynnistää isäntä yhteyden agenttiin ja ilmoittaa pääsivulla tämän olevan käytettävissä (online).

Luodaan AWS käyttäjätili Jenkinsiä varten. Valitaan AWS:n valikosta palvelu IAM (Identity and Access Management), josta valitaan Users sekä Add user. Kun nimi on annettu

ja valittu Programmatic access, määritellään tulevalle käyttäjälle uusi ryhmä (group). Uudelle ryhmälle puolestaan tarvitsee määritellä käytäntöjä (policy), joita ryhmän jäsenet saavat suorittaa. Kun tarvittavat policyt on asetettu, voidaan luoda ryhmä.

Kun ryhmä on luotu ja käyttäjä on valmis, voidaan asettaa juuri luodut pääsytiedot Jenkinsiin, jotta Jenkins voi luoda instansseja AWS:ään tulevia ajoja varten. Valitaan Manage Jenkins, Configure System, valitaan Add a new cloud ja Amazon EC2. Koska uusia pääsytietoja ei ole vielä luotu Jenkinsiin, luodaan ne samalla valitsemalla Add ja Jenkins. Nyt voidaan syöttää käyttäjätiedot ja salasanat sekä mahdollinen kuvaus, jotta pystytään erottelemaan helpommin mikä käyttäjätili on kyseessä tai mitä tarkoitusta varten on tämä käyttäjätili luotu.

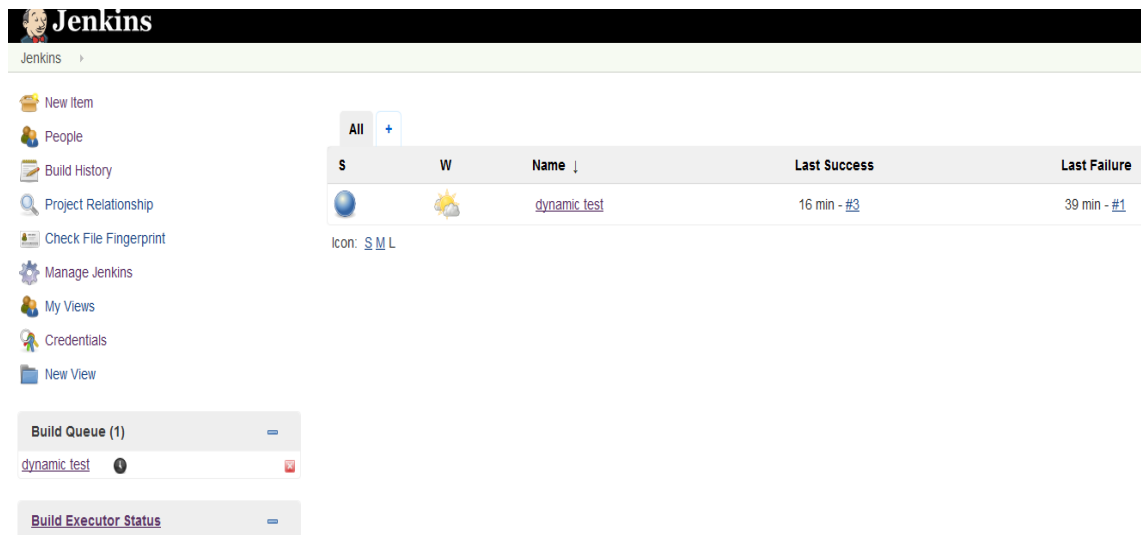
Kun pääsytiedot on syötetty, voidaan jatkaa valitsemalla mille alueelle halutaan tulevat instanssit sijoitettavan. Samalla syötetään myös yksityinen avain sille varattuun kenttään, jotta pystytään kirjautumaan tuleviin instansseihin sisään tulevia ajoja varten. Seuraavaksi määritellään mitä Amazonin virtuaalikoneen kuvaa (AMI) halutaan käyttää, jonka pohjalta automaattisesti rakennetaan tuleva agentti suorittamaan annettua ajoa.

On myös määritettävä mitä turvallisuusjoukkoa käytetään, jotta saadaan tuleva instanssi sidottua oikeaan aliverkkoon, jossa Jenkinsin isäntäkone on tai pitää luoda sääntöjä, jotka mahdollistavat yhteyden isännän ja agentin välillä. Määritellään myös mitä aliverkkoa käytetään sekä voidaan määrittää myös leimalla (label), että leimaa vastaavat ajot ajetaan vain tietyssä instanssissa. EC2 laajenuksessa on oletuksena, että joutilaana olevat instanssit tuhoetaan 30 minuutin joutenolon jälkeen. Tähän kannattaa myös kiinnittää huomiota. Jos ajetaan paljon pieniä ajoja, ei ole kovin järkevää tuhota instanssia minuutin joutenolon jälkeen ja hetken päästä pystyttää uutta, vaan mieluummin pidetään jo olemassa oleva hetken valmiudessa, jos lisää tehtäviä mahdollisesti saapuu lyhyen ajan sisällä.

Kun määrittely on saatu valmiiksi, voidaan rakentaa ensimmäinen työ Jenkinsiin. Valitaan New Item, annetaan sille nimi sekä valitaan Freestyle project. Voidaan asettaa kuvaus työstä niin halutessaan, jonka jälkeen valitaan GitHub project ja jonka kenttään syötetään tiedostovaraston osoite. Määritetään myös ehto, että tämä työ voidaan ajaa vain tietyn leiman sisältävässä instanssissa. Voidaan siirtyä määrittämään Source Code Managementiin versionhallintajärjestelmä, jota halutaan käyttää, joten valitaan Git. Avautuviin kenttiin annetaan tiedostovaraston URL ja valitaan pääsytiedot tiedostovarastoon. Jos näitä ei ole vielä luotu, voidaan ne luoda tässä kohden Add-valikon avulla.

Valitaan Build Triggers valikosta GitHub hook trigger for GITScm polling, jonka avulla siis saamme käynnistettyä automaattisen ajon, kun tiedostovarastoon tulee päivitys. Tallennetaan määrittymiset valitsemalla Save ja voidaan testata toiminnallisuutta kommitoimalla jotain tiedostovarastoon.

Kun kommitointi on suoritettu, saapuu Jenkinsin työjonoon uusi työ, kuten kuvasta 18 nähdään.



The screenshot shows the Jenkins dashboard. On the left is a navigation menu with options like 'New Item', 'People', 'Build History', etc. The main area displays a table of build jobs. A table with columns 'S', 'W', 'Name', 'Last Success', and 'Last Failure' shows one job named 'dynamic.test' with a success icon and a duration of 16 min. Below the table is a 'Build Queue (1)' section containing the job 'dynamic.test' with a status icon.

S	W	Name ↓	Last Success	Last Failure
		<a href="#">dynamic.test</a>	16 min - <a href="#">#3</a>	39 min - <a href="#">#1</a>

Icon: [S](#) [M](#) [L](#)

Build Queue (1)

[dynamic.test](#)

[Build Executor Status](#)

Kuva 13. Ruutukaappaus kommitoinnin jälkeen Jenkinsin työjonoon saapuneesta tehtävästä.

Hetken päästä ilmestyy agentti, joka käynnistetään suorittamaan tehtävää.

The screenshot shows the Jenkins dashboard. On the left is a navigation menu with items like 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'My Views', 'Credentials', and 'New View'. The main area displays a table of build jobs. A table with columns 'S', 'W', and 'Name' shows a job named 'dynamic test'. Below the table, there is a 'Build Queue (1)' section showing 'dynamic test' in progress. A 'Build Executor Status' section contains a warning: 'This slave has only 2 minutes idle time before termination (i-0a08cc09497920e5e) (offline)'. The interface includes a search bar at the top right and a table with columns 'S', 'W', and 'Name'.

S	W	Name ↓
		<a href="#">dynamic test</a>

Icon: [S](#) [M](#) [L](#)

**Build Queue (1)**

[dynamic test](#)

**Build Executor Status**

This slave has only 2 minutes idle time before termination (i-0a08cc09497920e5e) (offline)

Kuva 14. Ruutukaappaus agentista, jota käynnistetään suorittamaan tehtävää.

Kun instanssi on saatu rakennettua, muodostaa Jenkins SSH-yhteyden agenttiin, suorittaa annetun ajonsa, kuten kuvasta 15 huomataan, jonka jälkeen agentti on valmis ja odottaa tuhoutumistaan.

```
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision fe6f4449f05e5daa7b6d0a98ff17daed9462de57 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f fe6f4449f05e5daa7b6d0a98ff17daed9462de57
Commit message: "This works. I'm happy."
> git rev-list --no-walk be7c071318fe9de2bc34454a53185e7449d0a4b0 # timeout=10
[dynamic test] $ /bin/sh -xe /tmp/jenkins7931383548500585889.sh
+ echo 'this is running on a dynamic slave, which is nice'
this is running on a dynamic slave, which is nice
Finished: SUCCESS
```

Kuva 15. Ruutukaappaus osasta Jenkinsin konsolinäkymää ajon jälkeen.



Kuva 16. Agenti on suorittanut ajon ja jää jouten odottamaan tuhoamistaan.

Kun jouten oloon aiemmin määritelty aika täytyy, tuhotaan agenti, jota voidaan seurata AWS:n konsolista (kuva 17).

	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
-gitbox	i-028ae814505423c01	t2.micro	eu-west-1a	● running	✓ 2/2 checks pas...
-jenkins-master-testbox	i-0f13b756c5ed3d8f1	t2.micro	eu-west-1a	● running	✓ 2/2 checks pas...
	i-0a08cc09497920e5e	t2.micro	eu-west-1a	● terminated	

Kuva 17. Jouten ollut agenti on tuhottu määritetyn joutenoloajan jälkeen.

## 8 Yhteenveto ja pohdinta

Insinööriyössä selvitettiin jatkuvan integraation ympäristön rakentamista julkiseen pilvipalveluun. Lisäksi tavoitteeksi asetettiin, että ympäristö olisi automatisoitu sekä työssä pyrittiin myös kiinnittämään pilvipalveluiden kustannusmalliin ja tätä silmällä pitäen automatisoida työtä suorittavien agenttien staattista pystyttämistä sekä dynaamisten agenttien joutenolon minimointi. Työllä pyrittiin selvittämään onko kuvatus kaltainen ympäristö mahdollista toteuttaa ja siinä onnistuttiin.

Ensin tutustuttiin Amazon Web Services pilvipalvelun palveluihin, jotta opitaan ymmärtämään mitä on mahdollista tehdä, mitä tarvitsee tehdä ja mitä määrittämiä joudutaan tekemään, jotta ympäristöllä on tarvittavat oikeudet ja määrittämiä, joiden pohjalta ympäristö voidaan AWS:ään rakentaa. Jenkinsin kohdalla tutustuttiin eri liitännäisiin, jotta voidaan miettiä onko tarpeellisia liitännäisiä saatavilla, identifioitiin tarpeet ja niihin sopivat liitännäiset, joiden avulla pystyttiin rakentamaan halutut toiminnallisuudet. Versionhallinnan tiedostovarastona käytettiin GitHubia, jonne luotiin tili kommitointia varten ja josta noudetaan agenteille työtehtävät.

Työssä havaittiin, että AWS:n toiminnallisuus on hyvällä tasolla, palvelu on mielestäni käyttäjäystävällinen sekä tarpeelliset palvelut löytyvät helposti. Joskin palveluiden valtava määrä hieman arvelutti aluksi ennen kuin tarvittavat palvelut pystyttiin identifioimaan ja tutustumaan tarkemmin. Jenkinsin kanssa toimittaessa huomattiin, että palvelu oli ajoittain hidas ja syytä oli vaikea identifioida, joten tämä saattaa johtua verkkoyhteydestä tai virtuaalikoneesta, jossa Jenkins palvelua ajettiin. Myöskin Jenkinsin liitännäisten määrä teki vaikutuksen, joka omalta osaltaan sai vakuuttumaan, että sillä on takanaan aktiivinen yhteisö, joka kehittää palvelua ja liitännäisiä jatkuvasti sekä korjaa mahdollisia toiminnallisuuden ongelmia aktiivisesti.

Tehtävän saapuessa Jenkinsiin, on AMI:n pohjalta käyttöön otettavan instanssin käynnistys hieman hidasta, jossa kestää muutaman minuutin, joten sen perusteella aivan kaikkia tehtäviä varten ei kannata aina tuhota olemassa olevaa agenttia suorituksen jälkeen, vaan määrittellä pidempi elinaika mahdollisia tulevia työtehtäviä varten.

Jatkoa ajatellen kiinnostaisi testata miten Jenkinsin agentit suoriutuisivat konteissa eli containereissa ja miten nopeasti kontissa ajettava agentti olisi valmiina suorittamaan annettua tehtävää. Etenkin kun verrataan työssä käytettyyn AMI:sta rakennettavaan

agenttiin. Myöskin AWS:n CodeCommit-palvelua tulisi harkita korvaamaan GitHubia tiedostovarastona.

Työtä en tällaisenaan käyttäisi vielä yritysten tuotantoympäristöissä, vaan siinä tapauksessa on tarpeellista identifioida käyttäjiä tarkemmin ja tutkia heidän pääsyoikeuksia ympäristöön tai sen tarpeellisuutta ylipäättään. Myöskin AWS:n puolella ajoittain suurta päänvaivaa aiheutti turvallisuusjoukon käyttö, joka käytännössä jakelee oikeuksia liikenteelle, kuten mistä saa tulla, minne mennä ja millä protokollalla. Tämän vuoksi sen kanssa työskentely on todella tarkkaa ja on otettava huomioon pikkutarkkoja yksityiskoh-  
tia, jotta liikenne kulkee toivotusti ettei päädytä tilanteeseen, jossa etsitään toiminnalli-  
suuden ongelmakohtaa turhaan aivan väärästä paikasta.

Kiinnostusta herätti AWS:n palvelu CloudFormation, jossa pilveä hallitaan koodilla. Voi-  
daan rakentaa halutun kaltainen instanssi tai kokonainen pilvi pääsyoikeuksineen, verk-  
koineen ja instansseineen kirjoittamalla ne koodina malliin ja ajamalla malli CloudFor-  
mationissa, joka luo määritellyt resurssit ja toiminnallisuudet.

Työssä uuden omaksumista on lähtötasosta riippuen joko paljon tai kohtuullisesti, mutta  
etenkin AWS on pyrkinyt tuomaan esiin palvelunsa selkeästi ja materiaalia löytyy paljon  
ja monipuolisesti, jotta uuden omaksuminen helpottuisi. Jenkins on hieman mutkik-  
kaampi lähestyttävä, mutta eri foorumeilta löytyy paljon tietoa mahdollisista ongelmati-  
lanteista. GitHub taasen on helppokäyttöinen, mutta jos tiedostovarastot, käyttötarkoitukset  
ja niiden kanssa toimiminen on täysin vierasta, on vaikea ymmärtää mikä ylipäättään Git-  
Hub on, joten vähintäänkin perusteet on syytä omaksua aiheesta.

## Lähteet

- 1 Loeliger, Jon ja McCullough, Matthew. 2012. Version Control with Git, Second edition. USA: O'Reilly media.
- 2 Outlaw, Robert. Luettavissa: [<https://www.visualstudio.com/learn/what-is-version-control/>]
- 3 Chacon, Scott ja Straub, Ben. 2014. Pro Git, Second edition. USA: Apress.
- 4 Seppänen, Vili. 2015. Open source version control software. Oulu: Oulun yliopisto. Luettavissa: [<http://jultika.oulu.fi/files/nbnfioulu-201503311195.pdf>]
- 5 Escobar Garcia, Jose Antonio. 2011. Software Development and Collaboration: Version Control Systems and Other Approaches. Berliini: Technische Universität Berlin. Luettavissa: [[https://www.snet.tu-berlin.de/fileadmin/fg220/courses/SS11/snet-project/version-control-systems\\_escobar.pdf](https://www.snet.tu-berlin.de/fileadmin/fg220/courses/SS11/snet-project/version-control-systems_escobar.pdf)]
- 6 Smart, John Ferguson. 2011. Jenkins: The Definite Guide. USA: O'Reilly media.
- 7 [[https://en.wikipedia.org/wiki/Continuous\\_integration](https://en.wikipedia.org/wiki/Continuous_integration)] Päivitetty: 30.3.2018, Luettu: 25.3.2018.
- 8 Duvall, Paul M., Matyas, Steve ja Glover, Andrew. 2007. Continuous integration: Improving Software Quality and Reducing Risk. USA: Addison-Wesley Professional.
- 9 Ruest, Danielle ja Ruest, Nelson. 2009. Virtualization: A Beginner's Guide. USA: McGraw-Hill.
- 10 Portnoy, Matthew. 2012. Virtualization Essentials. USA: John Wiley & Sons. Inc.
- 11 Wadia, Yohan. 2016. AWS Administration – The Definitive Guide. UK: Packt Publishing Ltd.
- 12 Chandrasekhar, K. 2015. Essentials of Cloud Computing. USA: CRC Press.
- 13 Golden, Bernard. 2013. Amazon Web Services for Dummies. USA: John Wiley & Sons, Inc.
- 14 Overview of Amazon Web Services. [<https://d0.awsstatic.com/whitepapers/aws-overview.pdf>] luettu: 5.4.2018.
- 15 [[https://en.wikipedia.org/wiki/Wizard\\_\(software\)](https://en.wikipedia.org/wiki/Wizard_(software))] päivitetty viimeksi: 17.3.2018, luettu 10.4.2018

- 16 [<https://aws.amazon.com/amazon-linux-ami/>] luettu: 10.4.2018.
- 17 [<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html>] luettu 10.4.2018
- 18 [<https://jenkins.io/doc/book/installing/>] luettu: 20.3.2018
- 19 [<https://www.ssh.com/ssh/>] muokattu 6.3.2018, luettu 8.4.2018.
- 20 [<https://aws.amazon.com/about-aws/global-infrastructure/>] luettu 3.4.2018
- 21 [<https://rctom.hbs.org/submission/amazon-web-services-seeing-through-the-clouds/>] luettu 3.4.2018