Tampereen yliopisto

Sofianna Mäenpää

KONTTIEN KÄYTTÖ VIRTUALISOINNISSA

Docker- ja LXC-tekniikat

Kandidaatintyö
Tekniikan ja luonnontieteiden tiedekunta
Tarkastajat: Mikko Salmenperä
Syyskuu 2022

TIIVISTELMÄ

Sofianna Mäenpää: Konttien käyttö virtualisoinnissa
Kandidaatintyö
Tampereen yliopisto
Automaatiotekniikka, Teknisten tieteiden koulutus
Syyskuu 2022

Konttitekniikka on uudehko tekniikka, joka haastaa virtuaalikoneita osana virtualisaatiota. Konttien tarkoituksena on eristää esimerkiksi käyttöjärjestelmä tai sovellus omaksi yksikökseen. Tällöin konttia on helppo liikuttaa, käyttöönottaa ja muokata eri laitteilla.

Tässä kandidaatintyössä tarkastellaan kahta eri konttitekniikkaa Dockeria ja LXC:tä. LXC:tä pidetään ensimmäisenä konttitekniikkana, ja se julkaistiin vuonna 2008 [1]. Dockerista on muodostunut suosituin konttitekniikka sen julkaisun jälkeen. Dockerin pääkäyttökohde on sovellusten pakkaaminen kontteihin. LXC-kontteihin pakataan pääasiallisesti käyttöliittymiä. Konttitekniikoiden lisäksi tutkitaan kahta eri konttienhallintaohjelmaa DockerSwarmia ja Kubernetestä.

Koska kontit tarjoavat vaihtoehtoa virtuaalikoneille, työssä tutkitaan myös virtuaalikoneiden ja virtuaalikonttien eroja. Virtuaalikonttien ja -koneiden erot ilmenevät niiden rakenteessa. Työssä todetaan myös, että virtuaalikoneet ja -kontit voivat toimia rinnakkain yhteistyössä.

Konttitekniikkalla on monia eri hyödyntämiskohteita. Kontit ovat virtuaalikoneita kevyempi ratkaisu, joten rajoitettujen resurssien kanssa toimiessa konttien käyttö voi olla virtuaalikonetta järkevämpää. Kontit voivat olla myös osa CI/CD-työnkulkua, jonka avulla voidaan nopeuttaa ja sujuvoittaa ohjelmistojen kehitystä ja toimitusta.

Avainsanat: konttietekniikka, Docker, LXC, Kubernetes, Docker Swarm, virtuaalikone

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

binääritiedosto	Tiedosto, joka on tietokoneen luettavissa
CI/CD	Continuous Integration and Continuous Delivery
CPU	Central Processing Unit, prosessori, joka suorittaa tietokoneohjelman käskyjä
de facto-standardi	Yleisesti käytetty standardi
OS	Operating System, käyttöjärjestelmä
REST API	Representational State Transfer Application programming interface, ohjelmointirajapinta, joka hyödyntää HTTP:tä pyyntöjen käsittelyssä
VM	Virtual Machine, virtuaalikone

1. JOHDANTO

Tämä kandidaatintyö on kirjallisuustutkielma konttien käytöstä virtualisoinnissa. Kontit ovat eristettyjä yksiköitä, jotka sisältävät kaiken informaation, joka tarvitaan tietyn prosessin suorittamiseen. Näin tiettyä prosessia voidaan hyödyntää useassa eri paikassa, ilman ongelmia yhteensopivuuden kanssa. Tämä helpottaa esimerkiksi ohjelmistokehitystä, kun ohjelmisto pystytään siirtämään kehitysympäristöstä testiympäristöön ja edelleen tuotantoympäristöön ilman yhteensopivuusongelmia. Tietoa prosessista, joka voidaan pakata konttiin, on esimerkiksi lähdekoodi ja tarvittavat kirjastot. Kontit ovat siis osa virtualisaatiota, joka on yksi automaatiomaailman trendeistä.

Kontteja hyödynnetään jatkuvasti yhä enemmän, joten niiden hallinta on tärkeää virtualisoinnin parissa työskenteleville. Kontit tarjoavat vaihtoehtoa virtuaalikoneille, joiden implementointi on raskasta ja vie paljon resursseja. Kontit ovat huomattavasti virtuaalikoneita kevyempiä rakenteeltaan ja vievät vähemmän suoritustehoa. Virtuaalikoneet ja kontit toimivat kuitenkin myös toistensa rinnalla, kuten tässä työssä myöhemmin todetaan.

Seuraavassa luvussa esitellään konttien rakennetta ja käyttökohteita yleisellä tasolla, vertaillaan kontteja ja virtuaalikoneita sekä tutkitaan näiden toimintaa yhdessä. Tämän jälkeen luvussa 3 tutkitaan eri konttiratkaisuja Dockeria ja LXC:tä tarkemmin. Luvussa esitellään näiden konttien rakenteita ja käyttökohteita. Luvussa 4 tutkitaan konttien hallintaan käytettäviä ohjelmistoja Docker Swarmia ja Kubernetesta. Viimeinen luku eli luku 5 on työn yhteenveto.

2. KONTTITEKNIIKKA YLEISESTI

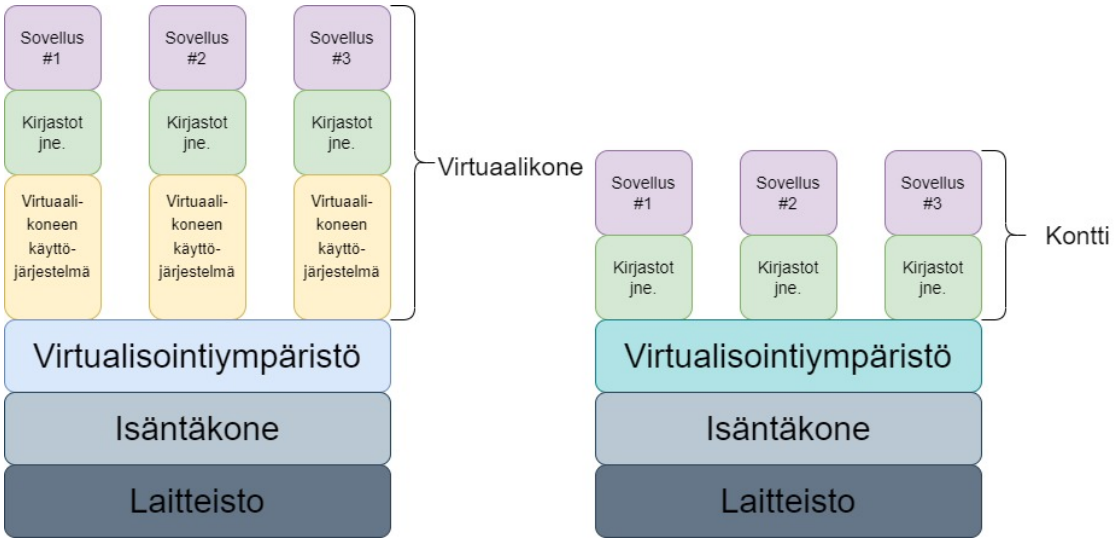
Jotta voidaan tutustua eri konttiteknologioiden (engl. container technology) eroihin, on ensin ymmärrettävä mitä kontit ovat ja miten niitä käytetään. Kontti on eristetty ympäristö, joka pitää sisällään yksittäisen sovelluksen, eli kaikki jonkin tietyn prosessin suoritamiseen tarvittavat tiedot [2]. Tällaisia tietoja ovat esimerkiksi prosessin suorittamiseen tarvittavat kirjastot ja sovelluksen lähdekoodi. Kontin kaikki tieto löytyy kuvatiedostosta, jonka perusteella kontti voidaan rakentaa ja käynnistää uudelleen missä tahansa ympäristössä. Konttiin pakattu sovellus toimii kaikissa ympäristöissä samalla tavalla. Tällöin ohjelmistojen käyttö, kehittäminen, ylläpito ja käyttöönotto helpottuvat [3]. Kontit mahdollistavat myös modulaarisuutta, eli niiden avulla ohjelmisto voidaan jakaa pienempiin osiin. Tällaisia osia voi olla esimerkiksi tietokanta ja sovelluksen lähdekoodi, jotka voidaan pakata omiin kontteihinsa. Kontteja voidaan myös helposti lisätä tai poistaa, jolloin ohjelmiston tai palvelimen skaalaaminen on helppoa.

2.1 Konttien ja virtuaalikoneiden erot

Kontteja pidetään usein vaihtoehtona virtuaalikoneille (engl. Virtual Machine, VM). Kontit ja virtuaalikoneet ovat käyttötarkoitukseltaan samankaltaisia: molemmat pyrkivät eristämään sovelluksen ja sen suorittamiseen tarvittavan datan. Niiden eroavaisuudet tulevat ilmi niiden arkkitehtuurissa [4]. Virtuaalikoneissa virtualisaatiota tapahtuu laitteiston tasolla, kun taas konteissa virtualisaatio tapahtuu ohjelmistossa. Kontit ovat virtuaalikoneita kevyempi ratkaisu. Virtuaalikoneiden implementointi on yleensä huomattavasti raskaampaa, ja vie enemmän aikaa ja resursseja. A. Vartiaisen artikkelissa [2] mainitaan esimerkkinä, että uuden kontin käynnistäminen tapahtuu millisekunteissa, kun taas virtuaalikoneessa käynnistysaika on useita minuutteja. Tilaa virtuaalikoneet vievät yleensä gigabittejä, kun taas kontit vain kymmeniä - satoja megabittejä [5, Esipuhe]. Tiettyä tallennustilaa pystytään hyödyntämään useampaan sovellukseen, kun käytetään kontteja, eikä virtuaalikoneita.

Tavallisten virtuaalikoneiden ja konttien rakenteellinen ero on nähtävissä kuvassa 2.1. Virtuaalikone on luotu käyttöjärjestelmä (engl. guest OS) toisen isäntäkoneen käyttöjärjestelmän (engl. host OS) päällä. Virtuaalikoneen pohjana toimii virtualisointiympäristö (engl. hypervisor), joka toimii isäntäkoneen päällä. Laitteisto (engl. hardware) tarjoaa virtuaali-

koneelle resursseja, kuten muistia. Nämä resurssit jaetaan kaikkien käytössä olevien virtuaalikoneiden kesken. [4] Itse virtuaalikone muodostuu käyttöjärjestelmästä, sovelluksen tarvitsemista tiedoista kuten kirjastoista ja binääritiedostoista, sekä itse sovelluksesta.



Kuva 2.1. *"Virtuaalikoneen ja kontin rakenteet"*

Kontit toimivat myös laitteiston ja isäntäkoneen päällä kuten virtuaalikoneetkin. Virtualisointiympäristönä konteilla on kuitenkin kontin moottori (engl. container engine) 2.1, joka hallitsee kontteja. Konteilla ei ole luotua käyttöjärjestelmää, vaan kontit muodostuvat vain itse sovelluksesta, sekä sovelluksen tarvitsemista tiedoista, kuten kirjastoista ja binääritiedostoista. Käytännössä siis virtuaalikoneet virtualisoivat laitteistoa niin, että voidaan ajaa useaa käyttöjärjestelmän ilmentymää, ja kontit virtualisoivat käyttöjärjestelmiä niin, että voidaan ajaa useaa työtaakkaa yhdellä käyttöjärjestelmällä [6].

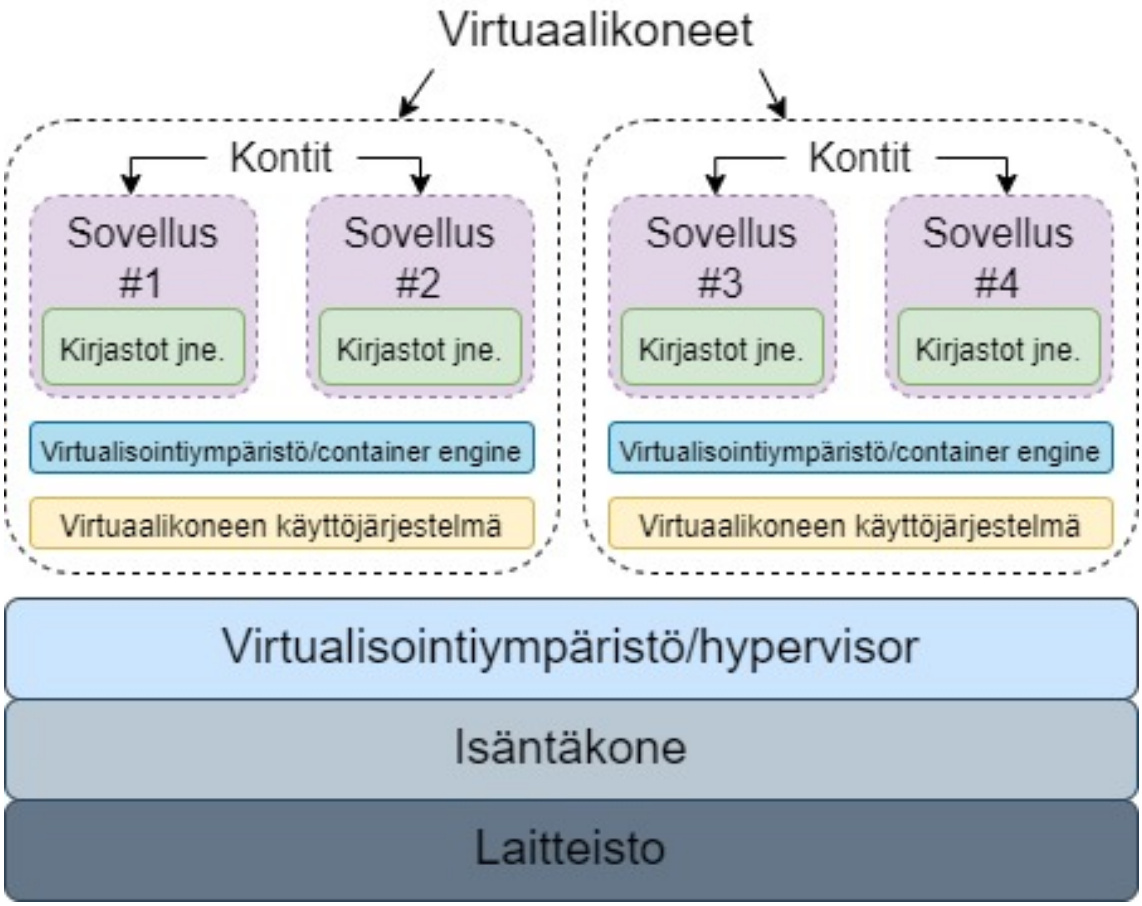
Virtuaalikoneiden ja konttien käyttökohteet eroavat toisitaan osittain, mutta kuten seuraavassa alaluvusa todetaan, ne toimivat myös hyvin yhdessä. Virtuaalikoneiden käyttökohteena voi olla esimerkiksi useiden sovellusten ajo useilla servereillä, kun taas konttien avulla on helpompi ajaa useaa kopiota samasta sovelluksesta [4]. Molemmille virtualisointityökaluille on lukuisia käyttökohteita, joista konttien käyttökohteita esitellään lisää luvussa kolme.

2.2 Kontit ja virtuaalikoneet yhdessä

Vaikka kontit kasvattavatkin suosiotaan, virtuaalikoneet eivät ole poistumassa käytöstä, sillä virtuaalikoneet ja kontit toimivat myös yhdessä. Tällöin voidaan hyödyntää molempien parhaita puolia, ja luoda systeemejä, jotka olisivat pelkästään virtuaalikoneilla tai konteilla mahdotonta luoda. Systeemien muuttaminen virtuaalikoneista konteiksi on hidasta ja vaikeaa, joten on järkevämpää kehittää systeemeitä sellaisiksi, että virtuaalikoneet ja kontit toimivat yhdessä. Tällöin virtualisaatiota tapahtuu niin laitteiston kuin ohjel-

miston tasolla. Tämä mahdollistaa joustavia systeemeitä, joissa prosessien vaatimuksia voidaan toteuttaa laajasti.

Kuvassa 2.2 on nähtävissä virtuaalikoneiden ja konttien rinnakkainen malli.



Kuva 2.2. *"Konttien ja virtuaalikoneiden muodostaman systeemin rakenne"*

Kuten kuvasta 2.2 nähdään, kontit toimivat virtuaalikoneiden sisällä. Pelkkiä kontteja sisältävässä systeemissä konttien virtualisointiympäristö (eli konttien moottori) toimii isäntäkoneen päällä. Jos kontteja ja virtuaalikoneita käytetään rinnakkain, konttien moottori sijoittuu virtuaalikoneen sisään virtuaalikoneen käyttöjärjestelmän päälle. Pohjan rakenne on sama kuin normaalillakin virtuaalikoneella: laitteisto, isäntäkone ja virtualisointiympäristö eli hypervisor.

Konttien turvallisuus ei ole yleensä yhtä hyvä kuin virtuaalikoneen, sillä konttien välinen eristys on huomattavasti ohuempi, kuin virtuaalikoneiden. Tästäkin syystä konttien ja virtuaalikoneiden rinnakkain käyttäminen voi olla järkevää, sillä enemmän suojausta vaativat prosessit voidaan suojata paremmin sijoittamalla ne virtuaalikoneisiin.

3. ERILAISET KONTTITEKNIIKAT

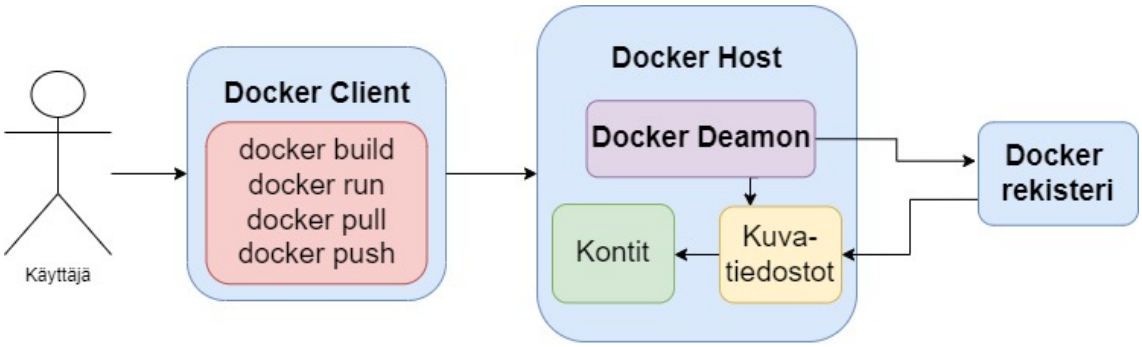
Konttitekniikoita on kehitetty useampia erilaisia eri tarkoituksiin. Tässä luvussa esitellään nykypäivänä kaikista suosituin konttitekniikka Docker sekä ennen Dockeria suosittu LXC. Luvussa tutkitaan näiden konttitekniikoiden rakenteita, eroja sekä niiden käyttötarkoituksia.

3.1 Docker

Docker-konttitekniikka on avoimen lähdekoodin (engl. open source) projekti, joka julkaistiin vuonna 2013 [7]. Docker on saavuttanut konttiteknologioista suurimman suosion, ja sitä pidetään konttiteknologian de facto -standardina. Docker pohjautuu LXC-kontteihin [4], joita käsitellään alaluvussa 3.2. Dockeria hyödynnetään sovellusten kehityksessä, siirtämisessä ja ajamisessa eristetyssä ympäristössä [8]. Dockerin suosion, sen suhteellisen helppouden sekä sen käyttäjäystävällisyyden takia Docker on hyvä lähtökohta virtuaalikonttien aloittelijalle.

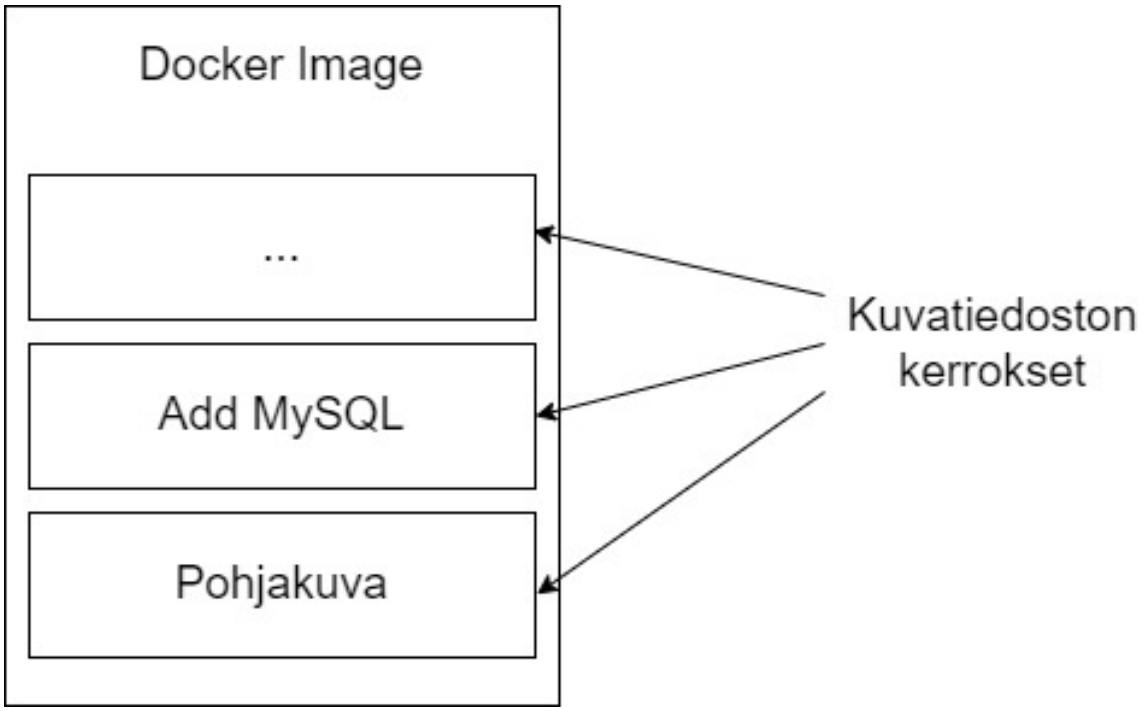
3.1.1 Dockerin rakenne

Kuvassa 3.1 esitellään Dockerin rakenne ja eri osien väliset kommunikaatiot. Dockerkontin rakentaminen aloitetaan Docker-tiedostolla (engl. DockerFile) eli tekstitiedostolla. Tämä tekstitiedosto sisältää tiedon siitä, kuinka Docker-kuvatiedosto (engl. Docker Image) rakennetaan. Käyttäjä kirjoittaa komentoja Docker-asiakasohjelmaan (eng. Docker Client) (kuvassa 3.1 ensimmäinen laatikko vasemmalla) eli esimerkiksi komentorivile. Docker-asiakasohjelma kommunikoi Docker Daemonin (kuvassa 3.1 violetti laatikko) kanssa käyttäen REST API -rajapintaa, ja Docker Daemon toteuttaa käyttäjän komennon. Docker Daemon pyörii aina isäntäkoneella, Docker-asiakasohjelma puolestaan joko isäntäkoneella tai muulla laitteella. Docker-asiakasohjelmaan syötettäviä komentoja voivat olla esimerkiksi docker build, joka rakentaa Docker-tiedostosta Docker-kuvatiedoston [9], tai docker push, joka työntää kuvatiedoston tai repositorion rekisteriin (kuvassa 3.1 oikealla) [10]. Docker Daemonin Docker-tiedoston pohjalta rakentama kuvatiedosto on luettava tiedosto, joka määrittelee, miltä kontti näyttää ja mitä prosesseja ajetaan, kun kontti käynnistetään.



Kuva 3.1. *Dockerin rakenne ja kommunikointi*

Docker Daemonin luomaan kuvatiedostoon muodostuu uusi kerros (engl. layer) jokaista Docker-tiedoston ohjetta varten. Jokainen kerros edustaa osuutta kuvatiedoston tiedostojärjestelmästä, joko lisää alla olevan kerroksen tai korvaa sen kokonaan. Tässä hyödynnetään Union File System -palvelua. Union File System mahdollistaa tiedostojen ja hakemistojen (engl. directory) kerrostamisen yhdeksi tiedostojärjestelmäksi. Kun tiettyä kerrosta halutaan muuttaa, Union File System luo kopion tiedostosta muuttamatta alkuperäistä tiedostoa. Näin saavutetaan Docker-kuvatiedoston ominaisuus, joka mahdollistaa vain luettavuuden. [4] Docker-kuvan kerrosten muodostuminen on nähtävissä kuvassa 3.2. Kuvatiedoston pohjana on pohjakuva, jonka päälle muodostuvat kerrokset Docker-tiedoston mukaan. Kerroksena voi olla esimerkiksi MySQL-tietokanta, kuten kuvassa 3.2.



Kuva 3.2. *Docker-kuvatiedoston kerrosrakenne*

Kerroksellisen rakenteen takia uutta konttia luotaessa tai ajettaessa kaikkia tiedostoja ei tarvitse kopioida. Tämä nopeuttaa prosessia ja pienentää myös kustannuksia. Kerroksellista kuvatiedostoa on myös helppo muuttaa, sillä Docker muuttaa vain sitä kerrosta,

johon muutoksia tehtiin, eikä koko konttia. [4]

Docker hyödyntää eristyksessään nimiavaruus-, kontrolliryhmä- ja Union File System -ominaisuuksia. Nimiavaruudet muodostavat abstraktion globaalille järjestelmäresurssille, jolloin määritetyn nimiavaruuden prosessit näkyvät erillisinä esiintyminään kyseisestä globaalista resurssista. Nimiavaruus siis tarjoaa eristyksen kontin ja isäntäjärjestelmän välillä. [1]

Nimiavaruuksia, joita Docker hyödyntää, on useita, kuten esimerkiksi NET, PID ja USER. NET-nimiavaruus mahdollistaa kontille oman näkymän järjestelmän verkkoon eli esimerkiksi IP-osoitteisiin ja verkkolaitteisiin. PID:n (Process ID) avulla kontti saa oman näkymän prosesseihin, joiden kanssa se voi vuorovaikuttaa. USER-nimiavaruus luo eron uid (user ID)- ja gid (group ID) -alueille verrattuna isäntäsysteemiin, joka eristää käyttäjät toisistaan. [4]

Googlen vuonna 2007 julkaisema cgroups (control groups, kontrolliryhmät) -projekti mahdollistaa resurssien parempaa hallintaa systeemissä [1]. Kontrolliryhmien avulla systeemin resursseja, kuten muistia, CPUta ja verkonkäyttöä voidaan esimerkiksi jakaa eri konttien kesken eri määrissä. Docker-kontit käyttävät vain tarvitsemansa resurssit, ja cgroups varmistaa, ettei yksi kontti vie liikaa resursseja ja kaada ohjelmistoa. [4]

3.1.2 Dockerin käyttökohteet

Docker mahdollistaa kontin hallinnan koko sen elinkaaren ajan. Docker-konttien avulla sovellusta ja siihen liittyviä komponentteja voidaan kehittää, sovellusta voidaan jakaa ja testata, sekä lopulta siirtää sovellus tuotantoympäristöön. Tuotantoympäristöön siirtämisessä ei tule yhteensopivuusongelmia Docker-kontteja käytettäessä, sillä Docker-kontit toimivat yhtä hyvin lokaalilla datakeskuksella kuin pilvipalvelussa sekä näiden yhdistelmässä. [8]

Docker-kontit toimivat hyvin CI/CD-työnkulussa, eli jatkuva integrointi, ja jatkuva toimitusmallissa. Esimerkki kyseisestä työnkulusta on seuraava: kehittäjät koodaavat sovellusta paikallisesti, ja jakavat sitä keskenään Docker-kontteja hyödyntämällä. Sovellus voidaan työntää testiympäristöön käyttäen Dockeria, jolloin sovellusta voidaan testata manuaalisesti tai automaattisesti. Kun sovelluksesta löydetään ohjelmointivirheitä, kehittäjä voi korjata niitä paikallisesti kehitysympäristössä ja lähettää sitten uudelleen kontin testiympäristöön. Kun testaus on saatu valmiiksi, kontti voidaan toimittaa asiakkaalle työntämällä viimeisin versio Docker-kuvatiedostosta tuotantoympäristöön. [8]

3.1.3 Docker-ohjelmisto ja DockerHub

Docker-ohjelmiston voi ladata omalle tietokoneelleen Dockerin omilta internetsivuilta [11]. Docker on saatavilla joko maksullisina Pro-, Team- tai Business-versioina tai ilmaisena Personal-versiona. Tämä mahdollistaa Docker-konttien käytön laajalla käyttäjäkunnalla. Docker kehitettiin alun perin Linux-ympäristöön, mutta nykyään Docker-kontteja voidaan ajaa myös Windowsilla ja Macilla [12].

DockerHub-pilvipalvelusta [13] käyttäjä voi hakea ilmaisia valmiita Docker-kuvatiedostoja, jotka sisältävät esikonfiguroidun ympäristön, johon käyttäjä voi luoda lisää kerroksia. DockerHubista saatavista kuvatiedostosta löytyvät esimerkiksi ohjelmointikieli ja siihen tarvittavat tiedostot ja rakenteet (engl. framework) [3]. Kehittäjä voi myös jakaa omia Docker-kuvatiedostojaan DockerHubissa, joko yksityisenä tai julkisena.

3.2 LXC

Ennen Dockerin suosiota LXC (Linux Containers) oli ensimmäinen konttitekniikka, joka sai laajempaa suosiota [14]. LXC-kontit julkaistiin vuonna 2008 [1]. LXC-konttien ensisijaisena käyttökohteena on käyttöjärjestelmien kontitus, mutta LXC-kontteihin voidaan pakata myös yksittäisiä sovelluksia. LXC-ohjelmisto on ilmaiseksi saatavilla Linux-jakelusta.

3.2.1 LXC-konttien rakenne

Koska Docker-kontit pohjautuvat LXC-kontteihin, on niissä paljon samankaltaisuuksia. Myös LXC-kontit käyttävät Linux-ytimen cgroups- (control groups, kontrolliryhmät) ja nimiavaruus-ominaisuuksia saadakseen luotua eristetyn tai pakatun alueen isäntäkoneessa [1]. Docker-luvussa 3.1 esitellyt nimiavaruudet ovat siis myös LXC-konttien käytössä. LXC-kontit hyödyntävät chroot-komentoa, eli ne voidaan muodostaa nin, että ne näkevät juurihakemistonaan valitun tiedostojärjestelmän alihakemiston [15]. LXC muodostuu neljästä eri komponentista: liblxc-kirjastosta, useista eri ohjelmointikielten yhteensopivuuksista API:n kanssa, konttien hallintaan käytettävistä standardityökaluista, sekä jakelukonttimalleista [16].

3.2.2 LXC-konttien käyttökohteet

LXC-kontit ovat pääasiallisesti käyttöjärjestelmäkontteja, joiden avulla voidaan ajaa useampaa Linux-käyttöjärjestelmää samanaikaisesti yhdellä isäntäkoneella. Mutta kuten aiemmin mainittiin, on yksittäisten sovellusten pakkaaminen LXC-konttiin myös mahdollista. Myös LXC-konttien pohjakuvia löytyy internetistä valmiiksi ladattavissa. Esimerkiksi LXC:n oma GitHub-repositorio sisältää muutaman mallin LXC-konteista [17].

Esimerkkinä LXC-konttien käyttökohteesta voidaan ottaa testipäivitysten ja -muutosten

teko. LXC-kontista voidaan helposti ja nopeasti luoda kopio, johon testipäivityksiä ja -muutoksia voidaan tehdä kopioon. Kun kopioon tehdyt muutokset ovat valmiita, kopio voidaan helposti ottaa käyttöön systeemissä, ja alkuperäinen viallinen kontti poistaa. [12]

3.3 LXC- ja Docker-konttien erot

Kuten aiemmin on esitelty, Docker- ja LXC-konttien käyttökohteet eroavat hiukan toisistaan. Niissä on kuitenkin myös pieniä rakenteellisia eroja.

Docker muodostui LXC-konttien pohjalta, ja alkuun Docker käyttikin samaa eristystapaa isäntäsysteemistä kuin LXC. Versio 1.0:n jälkeen Docker on kuitenkin kehittänyt oman version eristyksestä, libcontainerin. Libcontainer mahdollistaa säiliön elinkaaren hallitsemisen kontin luomisen jälkeen lisätoimintoja käyttämällä [18]. Docker-moottori mahdollistaa abrstraktion laitteistokohtaisille asetuksille, kuten esimerkiksi verkolle, tallennustilalle ja niin edelleen. Tämä tekee Docker-konteista helpommin siirrettäviä kun LXC-kontit, sillä ne eivät ole niin riippuvaisia allaan toimivasta fyysisestä koneesta. [12]

Molemmat konttityypit ovat huomattavasti nopeampia käynnistää ja suorittaa, kuin virtuaalikoneet. Docker-kuvatiedosto on mahdollista muodostaa distroless-tyyppinen kontti. Distroless tarkoittaa, että kontti sisältää vain täysin pakolliset eli sovelluksen ja sen ajamiseen tarvittavat riippuvuudet [19]. LXC-kuvatiedostot pyrkivät yleensä kopioimaan Linux jakeluita, kuten Ubuntua ja Alpinea. Suurella mittakaavalla distroless Docker-kontit voivat olla nopeampia toimittaa. [12]

LXC:n turvallisuutta pidetään parempana, kuin Dockerin. LXC:hen on lisätty useita ominaisuuksia, jotka parantavat sen turvallisuutta, kuten AppArmor-profiilin, joka varmistaa ettei host käytä vahingossa oikeuksiaan väärin. Docker konttien turvallisuusongelmana on, että Docker toimii pääkäyttäjänä, joka voi lisätä riskejä altistua haittaohjelmille. Docker Daemon toimii siis pääkäyttäjänä myös isäntäkoneella. [12]

Docker on enemmän skaalattavissa kuin LXC, sillä Docker-kuvatiedostot ovat kevyempiä kuin LXC-kuvatiedostot. Dockerilla sovellus voidaan jakaa pienempiin kuvatiedostoihin, jolloin skaalaaminen helpottuu. Vaikka LXC-kuvatiedostot ovatkin Docker-kuvia raskaampia, ovat LXC-kuvat kuitenkin virtuaalikoneita kevyempi ratkaisu. LXC-kontteja voidaan kuitenkin käyttää kevyenä virtuaalikoneena ilman hypervisoria, joten sekin on skaalattava vaihtoehto. [12]

4. KONTTIEN HALLINTA

Kontit toimivat toistensa rinnalla ja suurissa ohjelmistoissa kontteja voi olla lukuisia. Eri kontteja täytyy hallita samanaikaisesti, joten tarvitaan konttien hallintaan suunniteltuja ohjelmistoja. Tällaisia ohjelmistoja ovat esimerkiksi Docker Swarm ja Kubernetes.

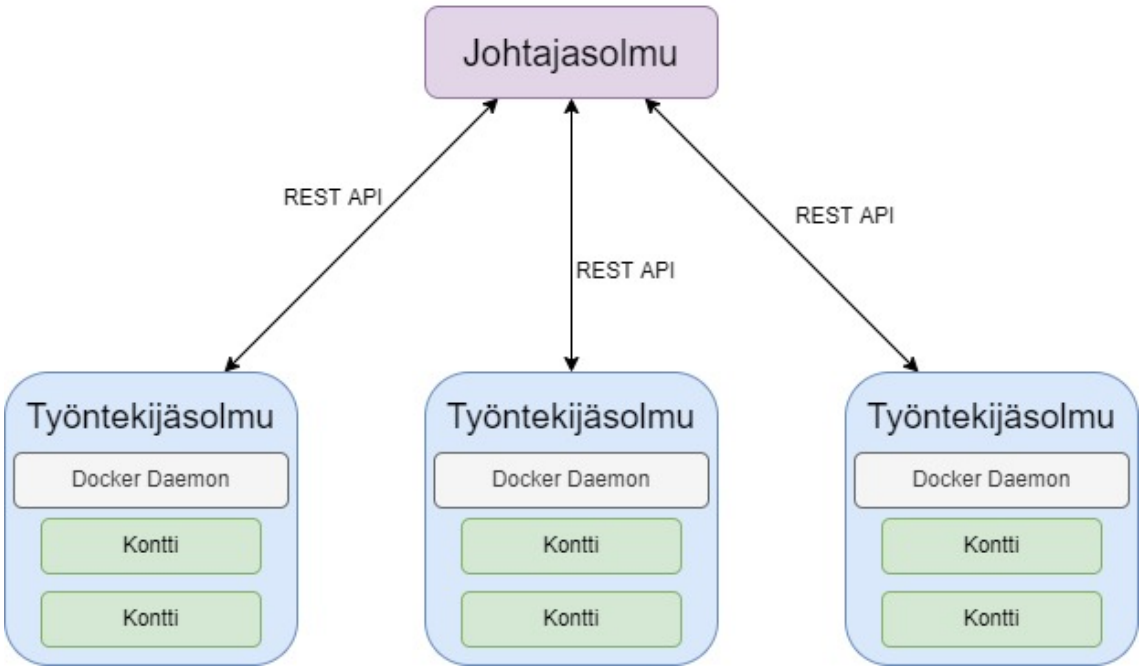
Konttien klusterointi on tärkeä osa konttitekniikkaa, sillä se muodostaa konteista yhdessä toimivan kokonaisuuden, joka huolehtii esimerkiksi konttien kaatumisen korjaamisen. Koska kontit ovat toisistaan eristettyjä, niiden muutokset eivät vaikuta muiden konttien toimintaan. Kontteja voidaan esimerkiksi lisätä, poistaa tai muokata systeemin sisäisesti, ilman että se vaikuttaa muihin kontteihin. Näiden muutoksien tekemisessä konttienhallintaohjelmat ovat hyvänä apuna.

4.1 Docker Swarm

Docker Swarm on Docker-ohjelmiston mukana tuleva konttienhallintatyökalu. Kuten Docker-kontitkin, Docker Swarm on suhteellisen helppokäyttöinen ja aloittelijaystävällinen. Tässä luvussa käydään läpi Docker Swarmin rakennetta, sekä sen käyttökohteita.

Docker Swarmin avulla voidaan luoda ja hallita useampaa eri Docker-konttia samanaikaisesti ja automaattisesti. Docker Swarmia käytetään Docker Clientin, eli esimerkiksi komentorivin kautta, aivan kuten yksittäisiä Docker-konttejakin. Docker Swarm on tehokas, mutta se ei sisällä niin laajaa valikoimaa komentoja, kuin osa muista konttienhallintaohjelmista.

Kuvassa 4.1 on esitelty Docker Swarmin rakenne. Docker Swarm muodostuu solmuista (node), joista yksi on johtajasolmu, ja loput työntekijäsolmuja. Johtajasolmuja voi olla useampikin, mutta silloin yksi niistä on ensisijainen johtajasolmu. Johtajasolmu, kuvassa 4.1 ylin laatikko, kommunikoi työntekijäsolmujen kanssa käyttäen REST API-rajapintaa. Työntekijäsolmu muodostuu Docker Daemonista, sekä yhdestä tai useammasta Docker-kontista. Johtajasolmu vastaanottaa tietoa työntekijäsolmujen statuksesta, kuten siitä onko solmu käynnissä vai kaatunut. Johtajasolmu lähettää tehtäviä työntekijäsolmuille, jotka joko hyväksyvät tai hylkäävät tehtävän. Jos isäntäsolmu huomaan jonkin konteista kaatuneen, se korjaa tilanteen esimerkiksi käynnistämällä uuden vastaavan kontin.



Kuva 4.1. *Docker Swarmin rakenne*

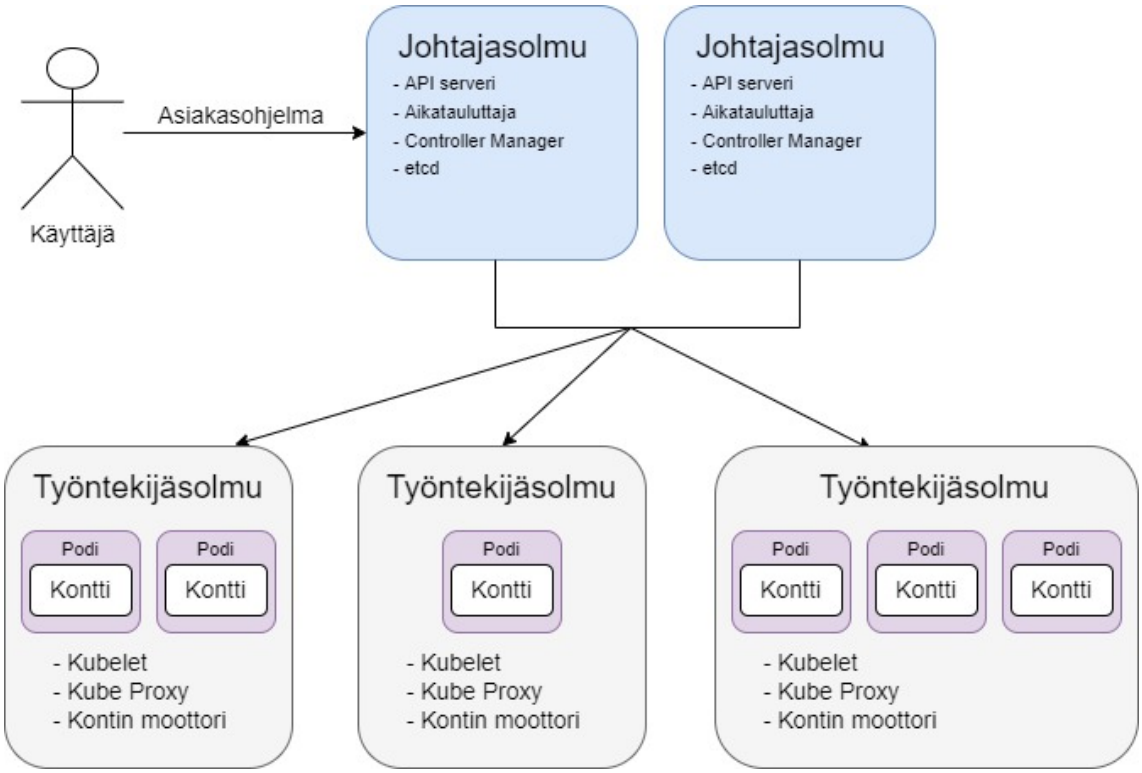
Swarm varmistaa, että jokaisella solmulla on riittävät resurssit käytössään. Swarm määrittää taustalla toimivia solmuja, ja määrää niille kontteja niin, että konttien työkuormat pyörivät kaikkein optimaalisimmalla isännällä. Näin saavutetaan tasapainoinen järjestelmä, jossa työtaakat jakautuvat tasaisesti. [20] Tämä helpottaa kehittäjän työtaakkaa, kun Docker Swarm hoitaa automaation avulla järjestelmän tasapainoa.

Docker Swarmissa on useita hyviä puolia. Se helpottaa sovelluksien kehitystyötä, sillä se mahdollistaa hajautetun pääsyn kontteihin. Tällöin useamman kehittäjän on helppo hallita samoja kontteja ja tehdä niihin muutoksia. Kehitystyössä auttaa myös se, että Swarmin avulla voidaan palauttaa aikaisempia turvallisia ympäristöjä, jos jokin menee vikaan. Docker Swarmissa on korkea turvataso. Kommunikaatio solmujen välillä on turvallista, vaikkei käyttäjä tekisikään muutoksia Docker Swarmin asetuksiin. Kontteja voidaan lisätä ja poistaa Swarmin kautta, jolloin infrastruktuuria voidaan skaalata helposti. [21]

4.2 Kubernetes

Kubernetes (joskus myös K8s) on avoimen lähdekoodin alusta, jota käytetään Linux-konttien hallintaan. Google julkaisi Kubernetes-projektin vuonna 2014. [22] Kubernetteksellä kontteja voidaan hallita niin yksityisissä, julkisissa, kuin myös hybridi-muotoisissa pilviympäristöissä. Kubernetesin avulla kehittäjä voi automaattisesti käyttöönottaa, skaalata, ylläpitää, ajoittaa ja operoida useita kontteja. [23] Kuten Docker Swarm, myös Kubernetes voi automaattisesti uudelleenkäynnistää kaatuneita kontteja, seurata konttien statusta ja esimerkiksi poistaa kontteja, jotka eivät vastaa.

Kubernetesin rakenne on nähtävissä kuvassa 4.2, ja se eroaa DockerSwarmin rakenteesta jonkin verran.



Kuva 4.2. *Kubernetesin rakenne*

Kubernetes klusteri muodostuu työntekijäsolmuista eli jokaisessa klusterissa on vähintään yksi työntekijäsolmu. [24] Työntekijäsolmu sisältää podeja, jotka sisältävät kontteja. Jokaisessa solmussa on Kubelet, Kube Proxy ja kontin moottori (kuten Docker Engine) [25]. Kubelet-agentti varmistaa, että kaikki kontit ajetaan podeissa ja hyödyntää PodSpecsejä varmistaakseen, että kontit ovat toiminnassa. Kube Proxy on verkonvälityspalvelin, joka pitää huolta solmujen verkon kautta käytävästä kommunikaatiosta. Kommunikaatiota voi tapahtua joko klusterin sisäisesti tai sen ulkopuolelta. Kontin moottori pitää huolen konttien ajamisesta. [24]

Työntekijäsolmut toimivat yhden tai useamman johtajasolmun alla. Käyttäjä kommunikoi johtajajasolmun kanssa asiakasohjelman (client) kautta. Johtajasolmu sisältää API serverin, aikatauluttajan, ohjauksenhallinnan ja etcd:n. Käyttäjä lähettää johtajasolmulle pyynnön, joka käsitellään API serverissä. API serveri validoi pyynnön ja lähettää sen eteenpäin. Aikatauluttaja tarkkailee podeja, joilla ei ole vielä tiettyä työntekijäsolmua, ja sijoittaa ne sopiviin solmuihin. Sopivan solmun löytämiseksi aikatauluttaja tarkastelee esimerkiksi yksittäisiä ja kollektiivisia resursseja, rajoitteita, affiniteettiä, datan sijaintia, työkuormia ja määräaikoja. Ohjauksenhallinta eli kube-controller-manager ajaa ohjainprosesseja. Ohjaimia on useita erilaisia, esimerkkinä solmuohjain. Solmuohjain tarkkailee solmujen statusta ja reagoi jos solmuja kaatuu.

Kuberneteksessä on myös muutamia ominaisuuksia, joita voidaan pitää heikkouksina. Kubernetes ei ole sovelluksen rakentamiseen tarkoitettu työkalu, joten se ei ole osa CI/CD-työnkulkua. Sovelluksen kehityksessä Kubrenetes tulee siis mukaan vasta myöhemmässä vaiheessa. Kubernetes ei myöskään tarjoa välitysohjelmistoa, eikä sovelluspalveluita. Kubernetesin avulla voidaan kyllä ajaa palveluita, jotka ovat konttien sisällä, mutta ne eivät ole natiiveja Kubernetekselle. [26]

5. YHTEENVETO

Kuten tässä työssä on todettu, konteilla on useita eri käyttökohteita. Kontteja voidaan käyttää monipuolisesti niin sovellusten, kuin käyttöjärjestelmien eristämiseen. Kontit ovat muodostuneet vaihtoehdoksi virtuaalikoneille, sillä virtuaalikoneet ja virtuaalikontit pyrkivät molemmat eristämään yksiköitä. Mutta kuten tässä työssä aiemmin todettiin, virtuaalikoneet ja virtuaalikontit toimivat myös rinnakkain, jolloin voidaan luoda järjestelmiä, joita pelkästään konteilla tai virtuaalikoneilla ei olisi mahdollista luoda.

Docker- ja LXC-konttien eristys todettiin perustuvan Linux-ytimen ominaisuuksiin, kuten nimiavaruuksiin ja kontrolliryhmiin. Docker-konttitekniikka on muodostunut LXC-konttien pohjalta, joten rakenteelliset samankaltaisuudet ovat peräisin siitä. Vaikka Docker hyödyntääkin Linux-ytimen ominaisuuksia, todettiin että Docker-kontit toimivat nykyisin myös Windowsilla ja Macilla. Tämä onkin varmasti yksi syy Dockerin suosioon. Dockerin suosiota tukee myös sen suhteellinen helppokäyttöisyys, ja käyttökohteiden monipuolisuus.

DockerSwarm- ja Kubernetes-konttienhallintatyökaluissa todettiin olevan rakenteellisia eroja. Kubernetes käsittelee kontteja podeissa, kun taas DockerSwarm suoraa työntekijäsolmuissa. Konttienhallintatyökalut automatiosoivat konttien hallintaa seuraamalla konttien statusta, ja esimerkiksi käynnistämällä uuden kontin automaattisesti kaatuneen kontin tilalle.

Tässä työssä tarkoituksena oli antaa yleiskuva konttitekniikasta, sekä syventyä Docker- ja LXC-kontteihin. Tarkoituksena oli, että käyttäjä saa hyvän yleiskuvan konttien ja virtuaalikoneiden rakenteista, sekä niiden käyttökohteista. Myös konttienhallintatyökalut DockerSwarm ja Kubernetes pyrittiin selittämään lukijalle niin rakenteen, kuin käyttötarkoituksen tasolla.

LÄHTEET

[1] S., S. K. *Practical LXC and LXD. Linux Containers for Virtualization and Orchestration*. Apress, elokuu 2017.

[2] Vartiainen, A. *Konttiteknologia on merkittävimpiä tekniikan murroksia juuri nyt*. 10. maaliskuuta 2015. URL: https://blog.digia.com/tech/2015/03/10/konttiteknologia-on-merkittavimpia-tekniikan-murroksia-juuri-nyt (viitattu 19. 11. 2021).

[3] *What is Docker in 5 minutes*. Aug. 19, 2018. URL: https://www.youtube.com/watch?v=_dfLOzuIg2o.

[4] *A Beginner-Friendly Introduction to Containers, VMs and Docker*. Mar. 4, 2016. URL: https://www.freecodecamp.org/news/a-beginner-friendly-introduction-to-containers-vms-and-docker-79a9e3e119b/ (visited on 11/30/2021).

[5] Matthias, K. ja Kane, S. P. *Docker: Up & Running. Shipping reliable containers in production*. O'Reilly Media, 3. heinäkuuta 2015.

[6] Synytsky, R. *How Containers and VMs Can Coexist*. Apr. 23, 2021. URL: https://containerjournal.com/editorial-calendar/kubernetes-in-the-enterprise/how-containers-and-vms-can-coexist/ (visited on 08/16/2022).

[7] *Docker-internetsivu, why Docker*. Mar. 4, 2016. URL: https://www.docker.com/why-docker.

[8] *Docker-nettisivun yleiskatsaus*. URL: https://docs.docker.com/get-started/overview/ (visited on 08/08/2022).

[9] *docker build*. URL: https://docs.docker.com/engine/reference/commandline/build/ (visited on 08/04/2022).

[10] *docker push*. URL: https://docs.docker.com/engine/reference/commandline/push/ (visited on 08/04/2022).

[11] *Docker-verkkosivu*. URL: https://www.docker.com (visited on 08/07/2022).

[12] *LXC vs Docker: Which Container Platform Is Right for You?* Feb. 18, 2022. URL: https://earthly.dev/blog/lxc-vs-docker/ (visited on 08/07/2022).

[13] *DockerHub-internetsivu*. Mar. 4, 2016. URL: https://hub.docker.com/.

[14] Kovács, Á. Comparison of different Linux containers. *2017 40th International Conference on Telecommunications and Signal Processing (TSP)*. 2017, s. 47–51. URL: https://ieeexplore-ieee-org.libproxy.tuni.fi/document/8075934.

[15] *Chroot*. 20. marraskuuta 2020. URL: https://www.linux.fi/wiki/Chroot (viitattu 23. 08. 2022).

[16] *What's LXC?* URL: https://linuxcontainers.org/lxc/introduction/ (visited on 08/12/2022).

[17] *LXC GitHub-repositorio*. URL: https://github.com/lxc/lxc (visited on 08/23/2022).

[18] *libcontainer GitHub-repositorio*. URL: https://github.com/opencontainers/runc/tree/main/libcontainer (visited on 08/23/2022).

[19] *"Distroless" Container Images GitHub-repositorio*. URL: https://github.com/GoogleContainerTools/distroless (visited on 08/23/2022).

[20] Casey, K. *What is Docker Swarm?* URL: https://www.techtarget.com/searchitoperations/definition/Docker-Swarm (visited on 08/12/2022).

[21] Simplilearn. *What is Docker Swarm: Modes, Example and Working*. Mar. 2, 2022. URL: https://www.simplilearn.com/tutorials/docker-tutorial/docker-swarm (visited on 08/12/2022).

[22] *Kubernetes-verkkosivujen yleiskatsaus*. URL: https://kubernetes.io/docs/concepts/overview/ (visited on 08/25/2022).

[23] Earls, A. R. *TechTarget-verkkosivun Kubernetes-artikkeli*. URL: https://www.techtarget.com/searchitoperations/definition/Google-Kubernetes (visited on 08/25/2022).

[24] *Kubernetes-verkkosivujen Kubernetes Components*. URL: https://kubernetes.io/docs/concepts/overview/components/ (visited on 08/31/2022).

[25] *Kubernetes Architecture explained | Kubernetes Tutorial 15*. Jan. 23, 2020. URL: https://www.youtube.com/watch?v=umXEmn3cMWY&list=PLy7NrYWoggjziYQIDorlXjTv index=3 (visited on 08/31/2022).

[26] *What is Kubernetes? Kubernetes Explained in 5 Minutes*. Dec. 16, 2019. URL: https://www.youtube.com/watch?v=fO3vUMc_WbI (visited on 08/25/2022).