TAMPEREEN TEKNILLINEN YLIOPISTO

JUUSO NUIKKA

DOCKERIN KÄYTÖN HYÖDYT OHJELMISTOKEHITYKSESSÄ

Kandidaatintyö

Tarkastaja: Maria Törhönen
Tarkastaja ja aihe hyväksytty:
8.12.2018

TIIVISTELMÄ

NUIKKA, JUUSO: Dockerin käytön hyödyt ohjelmistokehityksessä
Tampereen teknillinen yliopisto
Kandidaatintyö, 18 sivua
Joulukuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: Maria Törhönen
Avainsanat: Virtualisointi, Virtuaalikone, Konttiteknologia, Docker, Jatkuva toimitus

Virtualisaation tarkoituksena on luoda jostakin asiasta virtuaalinen versio. Tietotekniikassa monet asiat voidaan virtualisoida, kuten käyttöjärjestelmä, muisti, tallennustila, tietoverkko ja dataan liittyvät erilaiset abstraktiotasot. Aiemmin ohjelmistokehityksessä suuressa suosiossa on ollut käyttää virtuaalisointiin virtuaalikoneita. Ne kuitenkin huonon skaalautuvuudensa vuoksi eivät ole kovin tehokkaita moniin tilanteisiin. Tämän vuoksi on ollut tarve kehittää uudenlaisia tekniikoita ja virtuaalikoneiden tilalle onkin tullut konttiteknologia.

Tässä työssä tulen tutkimaan kirjallisuuskatsauksen muodossa Dockeria, joka on tällä hetkellä suosituin konttiteknologiaa käyttävä virtualisointityökalu. Työn tarkoitus on selvittää, minkälaisia etuja ohjelmoija ja ohjelmistokehitystä tekevä yritys saavat Dockerin käytöstä.

Ensiksi työssä kerrotaan taustaa virtualisaatiosta, sekä tutustutaan virtuaalikoneeseen ja konttiteknologiaan lähemmin. Sen jälkeen tutustutaan tarkemmin Dockerin teknisiin yksityiskohtiin ja selvennetään, miten nämä auttavat ohjelmoijaa. Lisäksi tutustutaan jatkuvan toimituksen malliin, jonka tehokkaan käytön Docker mahdollistaa.

Tutkimus osoittaa, että Dockerin käytöllä on paljon erilaisia etuja ohjelmoijan ja ohjelmistokehitystä tekevän yrityksen näkökulmasta. Dockerin käyttö lisää nopeutta, varmuutta, ennustettavuutta ja ylläpidettävyyttä ohjelmistokehityksessä.

SISÄLLYS

1 JOHDANTO....1
2 VIRTUALISOINTI....2
2.1 Virtualisoinnin kehitys....2
2.2 Virtualisointi ja abstraktio....2
2.3 Virtuaalikone....3
2.4 Konttiteknologia....4
3 DOCKER-SOVELLUS....6
3.1 Docker-moottori....6
3.2 Docker-kuva....7
3.3 Docker-kontti....8
3.4 Dockerin tehokkuus....11
3.5 Dockerin turvallisuus....12
3.6 Jatkuva toimitus....14
4 YHTEENVETO....16
LÄHTEET....17

LYHENTEET JA MERKINNÄT

Apache	HTTP-palvelinohjelma.
ARP-väärennös	Eng. ARP spoofing.
CPU	Eng. Central Processing Unit, tietokoneen suoritin.
Client	Asiakas.
Datavolume	Paikka, johon tallennetaan dataa.
Emacs	Tekstieditori.
GitHub	Versionhallintatyökalu.
Harkinnanvarainen pääsynvalvonta	Eng. Discretionary access control, lyhenne DAC.
Hypervisori	Eng. Hypervisor, suorittaa virtuaalikoneen toimintoja.
Isäntäkäyttöjärjestelmä	Eng. Host OS, käyttöjärjestelmä, joka ajaa vieraskäyttöjärjestelmää.
Java	Java-ohjelmointikieli.
Kernel	Linux-pohjaisen käyttöjärjestelmän ydin.
Kontrolliryhmä	Eng. Control Group.
Kontti	Eng. Container, konttiteknologian ominaisuus, jolla sovellusta ajetaan.
Konttimoottori	Eng. Container Engine, hypervisoria vastaava ohjelma konttiteknologiassa.
Kubernetes	Googlen suunnittelema orkesterointityökalu konteille.
Liittotiedostojärjestelmä	Eng. Union File System.
Linux	Linux-pohjainen käyttöjärjestelmä.
Mac	macOS-pohjainen käyttöjärjestelmä.
MAC-tulva	Eng. MAC flooding.
Makefile	Ohjelmointikehitykseen tehty automaatiotyökalu.
Muutoksenlisäys	Toiminta lisättäessä muutos versionhallintaan.
Nimiavaruus	Eng. Namespace.
Pakollinen pääsynvalvonta	Eng. Mandatory access control, lyhenne MAC.
RAM	Eng. Random Access Memory, tietokoneen keskusmuisti.
Red Hat	Avoimen lähdekoodin sovelluksia yrityksille tuottava yritys.
REST API	HTTP-protokollaan perustuva ohjelmointirajapinta.
Serveri	Palvelin.
Ubuntu	Linuxiin pohjautuva käyttöjärjestelmä.
Vieraskäyttöjärjestelmä	Eng. Guest OS, käyttöjärjestelmä, jota ajetaan isäntäkäyttöjärjestelmä sisällä.
Versiotunniste	Eng. Version Identifier.
Windows	Windows-pohjainen käyttöjärjestelmä.

1 JOHDANTO

Virtualisointiteknologiat ovat kasvattaneet suosiotaan viimevuosina täyttääkseen tarpeet ohjelmiston laadun, jatkuvuuden ja ennustettavuuden takaamiseksi. Virtualisaatio ei ole teknologiana uusi, mutta erityisesti pilviteknologioiden suosion kasvu on tuonut virtualisoinnin osaksi joka päiväistä työntekoa ohjelmistoalan yrityksissä.

Docker on avoimen lähdekoodin sovellus, joka mahdollistaa nopean virtualisaation monipuolisilla konteillaan. Kontti on paketti, johon paketoidaan ohjelman koodit ja riippuvuudet, jotta ohjelmaa voidaan ajaa eristetysti, nopeasti ja luotettavasti monissa ympäristöissä. Kontti sisältävää kuvan, johon sisällytetään ohjelman ajamiseen tarvittavat komponenit, kuten itse koodin, ajojärjestelmän, systeemityökalut, kirjastot ja asetukset. Näitä kuvia sisältäviä kontteja ajetaan Docker-moottorin avulla, joka varmistaa konttien toimivan oikein. Konttiteknologiat ja Docker ovat nykyään perustyökaluja ohjelmistokehityksessä ja niiden ymmärtäminen pitäisi kuulua jokaisen ohjelmoijan taitoihin.

Tämä työ analysoi Docker-sovellusta, koska se on tällä hetkellä suosituin konttiteknologiaan perustuva ja ominaisuuksiltaan monipuolisin työkalu. Tutkimus pyrkii analysoimaan kirjallisuuskatsauksen muodossa, miten Docker helpottaa ohjelmointityötä sen eri toimivuuksien kautta.

Tämän työn alussa, luvussa 2, esitellään virtualisointiin liittyviä aiheita. Ensiksi selvitetään, mitä virtualisointi yleisesti on ja miten se on kehittynyt tultaessa nykyaikaan. Tämän jälkeen kerrotaan, mitä eroa on virtualisoinnilla ja abstraktiolla. Tämän jälkeen kerrotaan oleelliset tekniset yksityiskohdat liittyen virtuaalikoneisiin ja konttiteknologiaan.

Luvussa 3 tutustaan itse Dockeriin ja pyritään avaamaan tärkeimmät tekniset yksityiskohdat sen tärkeimmistä ominaisuuksista ja kertomaan, millä tavalla ne helpottavat ohjelmistokehitystä. Näitä ovat Dockerin moottori, kuvat, kontit, turvallisuus, ja tehokkuus. Sen jälkeen selvennetään Dockerin yhteyttä jatkuvan toimituksen malliin ja kerrotaan, miten se lisää ohjelmistokehityksen tehokkuutta. Tämän jälkeen tulee yhteenvetoluku.

2 VIRTUALISOINTI

2.1 Virtualisoinnin kehitys

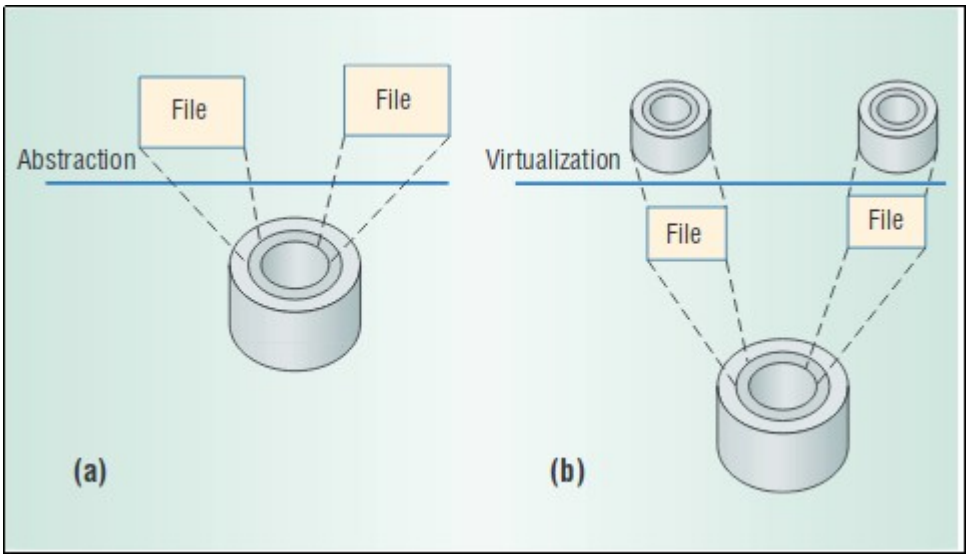
Tietotekniikassa on monenlaista virtualisaatiota. Varhaisimmat virtualisaatiotekniikat ovat peräisin jo 1960-luvulta, sen jälkeen virtualisaation käsite on laajentunut. Ennen nykyaikaisia virtualisointitekniikoita virtualisoitiin emulaattoriohjelmilla, jotka muodostavat virtuaalisen suorittimen. Tätä käytettiin ohjelman suoritukseen samoin, kuin fyysistäkin suoritinta.

Ennen pilvilaskennan kehitystä, stantarditapa ohjelmiston ympäristönä on ollut virtuaalikone. Virtuaalikone ei kuitenkaan huonon skaalautuvuutensa vuoksi sovellu tehokkaasti tähän tarkoitukseen. Näin ollen alalla on ollut tarve kehittää tehokkaampi ja kevyempi teknologia näiden ongelmien ratkaisemiseksi [1].

Nykypäivänä pilvilaskennan muuttaessa ohjelmistotuotantoa on suuri tarve helpoille ja älykkäille ohjelmiston hallintatyökaluille. Tämä tarkoittaa, että ohjelmistolle annetaan resursseja käyttäjän tarpeen mukaan, ilman pullonkaulojen syntymistä. Tähän tarkoitukseen on kehitetty konttiteknologia. Konttiteknologia ratkaisee monet vaikeudet, jotka ohjelmistojen tuottamisessa ovat aiemmin olleet ongelmana.

2.2 Virtualisointi ja abstraktio

Tietokoneiden jatkuvan kehityksen mahdollistaa hyvin määritellyt rajapinnat, jotka erottelevat eri abstraktiotasot toisistaan riippumattomiksi. Ylempi abstraktiotaso yksinkertaistaa alemman tason ratkaisut oman rajapintansa taakse. Tietotekniikassa virtualisaatiolla tarkoitetaan toimintaa, jossa luodaan jostain asiasta virtuaalinen versio. Virtualisaatio ei luo abstraktiota, mutta sitä käytetään työkaluna sen luomiseen. Esimerkiksi virtuaalisen levytilan luominen ei luo abstraktiota, vaan edelleen sama rajapinta on olemassa niin virtuaaliselle, kuin fyysisellekin [2].



Kuva 1: Abstraktio ja virtualisaatio levytallennuksessa [2].

Virtualisaatio voi nykyään liittyä laitteistoon, ohjelmistoon, muistiin, tallenustilaan, dataan tai tietoverkkoihin. Ohjelmistotasolla virtualisoinnin tavoitteena on, että ohjelmiston ajaminen eri laiterajapinnoilla ei muuta ohjelman toimintaa, vaan samoilla sisäänmenoarvoilla saadaan täysin samat ulostuloarvot [3].

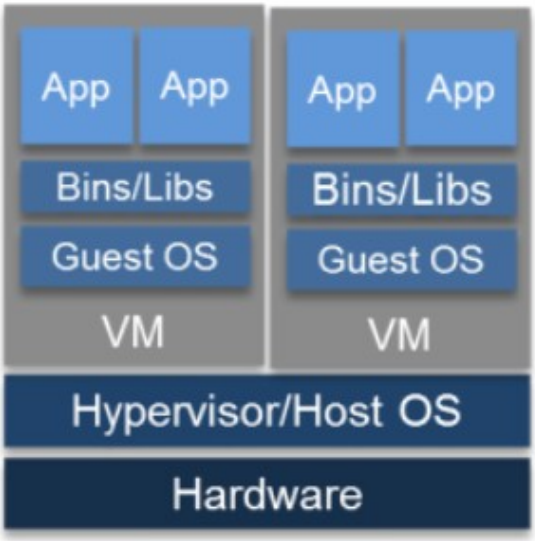
Erilaisia ohjelmistokehityksessä käytettäviä virtuaalisointiteknologioita ovat virtuaalikoneet sekä konttiteknologiat. Molemmat teknologiat ovat nykyään aktiivisessa käytössä ohjelmistokehityksessä, mutta niitä käytetään yleensä eri tarkoituksiin. Näistä konttiteknologia tuottaa tehokkaamman ja kevyemmän tavan ohjelmistojen ajamiseen, ylläpitoon ja kehittämiseen [4].

2.3 Virtuaalikone

Modernit tietokoneet ovat monimutkaisia järjestelmiä, jotka pitävät sisällään monia läheisiä vuorovaikutuksessa olevia komponentteja. Tällöin virtualisaatio toimii yhdistävänä teknologiana eri kerrosten välillä. Virtualisaation lisääminen sovellus- ja laitekerroksen väliin voi mahdollistaa muuten epäsopivien järjestelmien yhteentoimivuuden [2].

Virtuaalikone on käyttöjärjestelmä, joka on luotu virtuaalisesti toisen käyttöjärjestelmän päälle. Tämä luotu virtuaalinen käyttöjärjestelmä, sekä fyysinen käyttöjärjestelmä jakavat laitekerroksen resurssit. Suosituin virtuaalikoneen luomiseen käytetty sovellus on Oraclen VirtualBox, jolla pystyy luomaan lähes kaikista PC-pohjaisista käyttöjärjestelmistä virtuaalikoneen. Virtuaalikoneella pystyy tekemään lähes kaiken saman, kuin fyysiselläkin tietokoneella. Virtuaalikonetta ohjataan

hypervisorin avulla. Hypervisori taas pyörii fyysisen tietokoneen ja käyttöjärjestelmän päällä, kuten kuvasta 2 näkee.



Kuva 2: Virtuaalikoneen kerrokset isäntäkäyttöjärjestelmän päällä [5].

Hypervisori luo sovelluskerrokseen abstraktion, joka jäljittelee kokonaista käyttöjärjestelmää ja tätä kutsutaan virtuaalikoneeksi [6]. Hypervisoria kutsutaan myös virtuaalikonemonitoriksi, sillä se huolehtii virtuaalikoneen ajamisesta ja virtuaalikonetta voidaan hallita tämän avulla.

Virtuaalikonetta kutsutaan myös nimellä vieraskäyttöjärjestelmä ja sitä suorittavaa käyttöjärjestelmää isäntäkäyttöjärjestelmäksi. Jokainen virtuaalikone ei pelkästään aja kopiota käyttöjärjestelmästä, vaan myös virtuaalista kopiota kaikesta mitä vaaditaan käyttöjärjestelmä suorittamiseen. Tämä kuluttaa paljon RAM- ja CPU-resursseja.

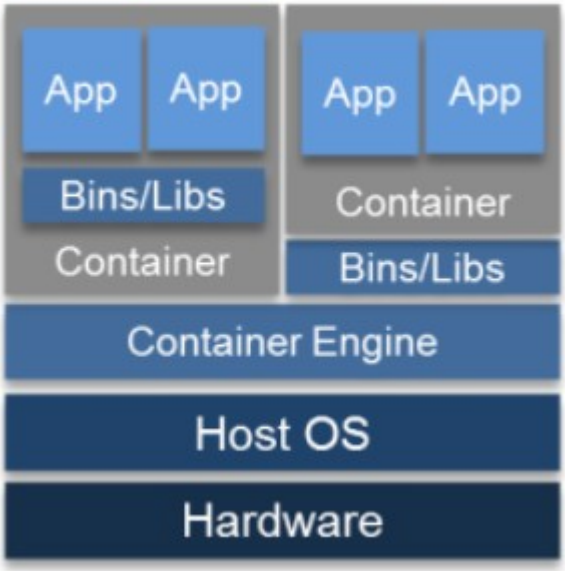
Eräs kuuluisimmista virtuaalikoneista on Java Virtual Machine. Se mahdollistaa käyttöjärjestelmässä Java-pohjaisten ohjelmien ajamisen. Sillä myös pystyy ajamaan kaikkia ohjelmointikieliä, jotka kääntyvät Java bittikoodiksi. Tämänlaisia ovat esimerkiksi Clojure, Scala ja Groovy.

2.4 Konttiteknologia

Konttiteknologia on viimevuosina nostanut suosiotaan pilvilaskennan suosion myötä. Konttiteknologia ei ole kovin uusi keksintö, mutta konttien luomisen ja kehittämisen vaikeuden myötä ne ovat vasta nyt nousseet varteenotettavaksi kilpailijaksi virtuaalikoneille. Sana kontti tulee laivojen tyypillisestä tavasta kuljettaa tavaroita, eli konteissa. Tämä kuvaa sitä, että tavara on eristetty muista standardikokoisiin kappaleisiin.

Päinvastoin kuin virtuaalikoneen tapauksessa, konteissa vain käyttöjärjestelmä on virtualisoitu. Tämä tarkoittaa, että virtuaalikoneiden suorittamaa laitetason virtualisaatiota ei tehdä. Kontit sijaitsevat fyysisen isäntäkäyttöjärjejestelmän päällä, joka on tyypillisesti Linux tai Windows. Tällöin isäntäkäyttöjärjestelmä jakaa oman ytimensä myös kontille. Näin saadaan vähennettyä paljon työmäärää isäntäkoneelta.

Tämän ominaisuuden avulla kontit ovat virtuaalikoneeseen verrattuna pienikokoisia ja käynnistyvät nopeasti. Konttien käynnistämiseen menee tyypillisesti sekunteja, kun taas virtuaalikoneen käynnistykseen menee jopa minuuteja.



Kuva 3: Konttiteknologian kerrokset isäntäkäyttöjärjestelmän päällä [5].

Kuvasta 3 nähdään tyypillinen kerrosrakenne käytettäessä konttiteknologiaa. Isäntäkoneen päällä ajetaan konttimoottoria, joka on vastaava hypervisorille virtuaalikoneen tapauksessa. Konttimoottori hoitaa kaiken liittyen konttien käynnistämiseen, ajamiseen, lopettamiseen ja muuhun hallintaan. Kuvassa 3 samat sovellukset ajetaan tällä kertaa kontin sisällä, kuvassa 2 ne ajettiin virtuaalikoneen sisällä. Kirjastojen ja binääritiedostojen ei välttämättä tarvitse olla kontin sisällä, kuten virtuaalikoneen tapauksessa.

Normaalisti isäntäkäyttöjärjestelmässä suoritettavalla ohjelmalla on oikeudet kaikkiin tietokoneeseen liitettyihin resursseihin, kuten yhdistettyihin laitteisiin ja tiedostoihin. Kontin sisällä ajettava ohjelma on oikeutettu vain kontin sisäisiin resursseihin, eli jotka käyttäjä on erikseen asettanut kontin nähtäväksi.

3 DOCKER-SOVELLUS

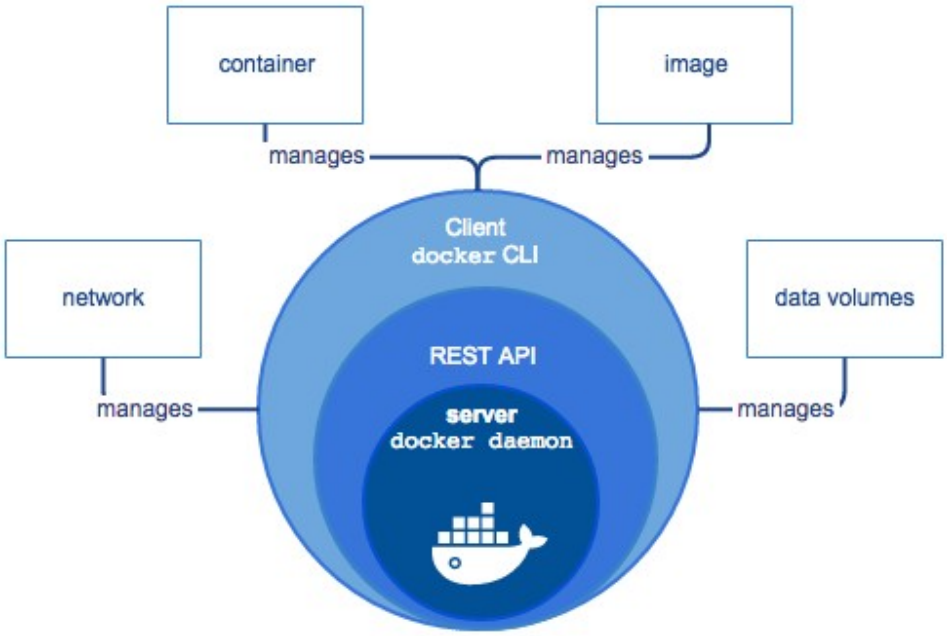
Dockerin tarina alkoi, kun sen keksijä Solomon Hykes piti puheen Dockerista Python-kehittäjien konferenssissa Kaliforniassa vuonna 2013. Kontit olivat aiemmin teknologisesti vaikeasti luotavia ja ylläpidettäviä, mutta Dockerin kehitystyön myötä ne ovat nykyään ratkaisu monenlaisiin ohjelmistokehityksen ongelmiin.

Nykyään Dockeria kehittää ja ylläpitää yritys nimeltään Docker Inc. Sovelluksen ensimmäinen versio julkaistiin vuonna 2013, ja tällä hetkellä viimeisin vakaa versio siitä on 18.09. Dockerin voi ladata tietokoneelle joko sovelluksen internetsivulta tai Linuxin paketinhallintaohjelmalla. Se on saatavilla Linuxille, Windowsille ja Macille. Siitä on saatavilla kaksi versiota, maksullinen Enterprise Edition ja ilmainen Community Edition.

Docker on Go-kielellä kirjoitettu avoimeen lähdekoodiin perustuva sovellus, jota voidaan käyttää ohjelmistojen rakentamiseen, ajamiseen ja jakamiseen. Näiden ominaisuuksien suorittamisen hoitaa Docker-moottori, joka toimii alustana sovellusten ajamiseen. Docker-moottorin pakkaamia ja sen suorittamia ohjelmia on tarkoitus ajaa konteissa. Dockerin kontit ovat kevyitä ja ominaisuusrikkaita kontteja, joiden suorittamisesta ja rakentamisesta Docker-moottori huolehtii. Kontit rakennetaan perustuen kuviin. Kuviin sisällytetään kaikki ohjelman toimintaan liittyvät asiat, kuten kirjastot, riippuvuudet ja itse ohjelmakoodi. Kontit ajavat ohjelmaa aina samalla tavalla jokaisessa tietokoneessa riippumatta esimerkiksi käyttöjärjestelmästä tai tietokoneen laitteistosta, jos suorittavaan tietokoneeseen on myös asennettu Docker.

3.1 Docker-moottori

Docker-moottori toimii sovelluskerroksessa isäntäkäyttöjärjestelmän päällä ja käyttää isäntäkäyttöjärjestelmän resursseja toimintaansa. Kuva 4 havainnollistaa moottoriin sisältyvät komponentit ja ominaisuudet.



Kuva 4: Docker-moottorin komponentit [7].

Docker-mottori on asiakas-palvelinsovellus, jonka pääkomponentteja ovat client, server ja REST API. Client ja server kommunikoivat keskenään REST APIn avulla. Clientiksi kutsutaan komentorivinäkymää, jonka kautta käyttäjä hallitsee esimerkiksi Dockerissa käytettäviä kuvia ja kontteja. Client välittää käyttäjän antaman käskyn serverille ja tarkemmin sanottuna Docker daemonille. Daemon on prosessi, joka huolehtii käyttäjän kuvien, konttien, datavolumien ja tietoliikenteen hallitsemisesta ja ylläpidosta Dockerin sisällä. [7]

3.2 Docker-kuva

Docker-kuva on tiedosto, joka sisältää tarvittavat vaatimukset Docker-kontin ajamiseen. Docker-konteilla on vain kuvassa määritellyt resurssit. Käyttäjän pitää muuttaa resursseja manuaalisesti, mikäli kontin toimintaan halutaan muutosta. Luomalla monia saman kuvan sisältäviä kontteja saadaan tasattua kuormaa ja näin skaalattua ohjelman toimintaa.

Käytännössä on kaksi tapaa luoda Docker-kuvia. Ensimmäinen tapa on rakentaa oma kuva pohjakuvan päälle. Tyypillisesti pohjakuva on käyttöjärjestelmäkuva, esimerkiksi Ubuntu. Tämän pohjakuvan päälle voi lisätä omia resursseja tarpeen mukaan. Näin toimittaessa omasta kuvasta saadaan luotua myöhemmin kontti, joka

sisältää omien koodien lisäksi myös kokonaisen käyttöjärjestelmän toiminnallisuuden. Myös on mahdollista luoda pohjakuva kokonaan itse. Tarvittavat ohjelmat voidaan lisätä suoraan tyhjään pohjakuvaan, mutta tällöin on pakollista luoda kokonaan uusi kuva. Uuden kuvan luomista kutsutaan muutoksenlisäykseksi. [8]

Toinen tapa uuden kuvan luomiseksi on määritellä kuva Docker-tiedostossa. Se on tekstipohjainen tiedosto ja aina nimeltään ”Dockerfile”. Docker-tiedostossa määritellään lista käskyjä, jotka ajetaan käyttäjän rakentaessa kuvansa. Docker-tiedostolla on samankaltaisuuksia Makefilen kanssa.

Tyypillinen Docker-kuvien tallennuspaikka on Docker Hub. Sen käyttö on verrattavissa ohjelmointikoodin säilytykseen versionhallinnan isännöintipalvelussa, esimerkiksi GitHubissa. Docker Hubissa kuvia voi säilyttää joko julkisina tai piilotettuina. Julkisia kuvia voi hakea itselleen kuka tahansa ja käyttää niitä vapaasti. Docker Hub on erinomainen alusta kuvien jakamiseen kehittäjien kesken. Docker Storesta käyttäjät voivat myös ostaa maksullisia kuvia ja hakea niitä itselleen. Kuvien kanssa toimiminen on osittain automatisoitu Dockeria käytettäessä. Esimerkiksi mikäli käyttäjällä ei ole tarvittavaa kuvaa paikallisesti, Docker osaa hakea sen Docker Hubista nimen ja version perusteella.

Kuvien käyttö kontin luomiseen lisää ohjelmointityön tehokkuutta ja varmuutta, sillä kuva voidaan määritellä tarkasti erilliseen Docker-tiedostoon. Lisäksi kuvat ovat kooltaan tyypillisesti pieniä ja niiden siirtäminen on yksinkertaista.

3.3 Docker-kontti

Docker käytti aiemmin Linux Containers-projektin LXD-kontteja konttien luomiseen. Tämä kuitenkin muuttui vuonna 2015 version 0.9 yhteydessä, jonka jälkeen Docker on käyttänyt libcontainer-kontteja. Silloin Docker ja muut konttiteknologian johtavat yritykset julkistivat Open Container Initiative-projektin. Tämän projektin tarkoituksena on luoda avoimet stantardit konttien rakenteelle ja ajamiselle [9]. Kuitenkin monet LXD-konttien tekniset toteutukset ovat libcontainer-konttien luomisen perustana tänäkin päivänä. [4]

Docker-kontit ovat monin tavoin parempia verrattaessa niitä muihin konttiteknologioihin ja virtualisointitekniikoihin. Pääasiallisina etuina ovat nopeus, siirrettävyys ja skaalautuvuus [8].

Nopeus on yksi suurimmista eduista Dockerin käytössä. Kontin rakentamiseen käytettävä aika on lyhyt, koska kontit ovat hyvin pieniä. Tällöin myös ohjelmointi, testaus ja käyttöönotto on nopeaa. Tyypillisesti konttien käyttämä kuva työnnetään rakentamisen jälkeen testaamisvaiheeseen, ja testien läpäisyn jälkeen tuotantoympäristöön [8].

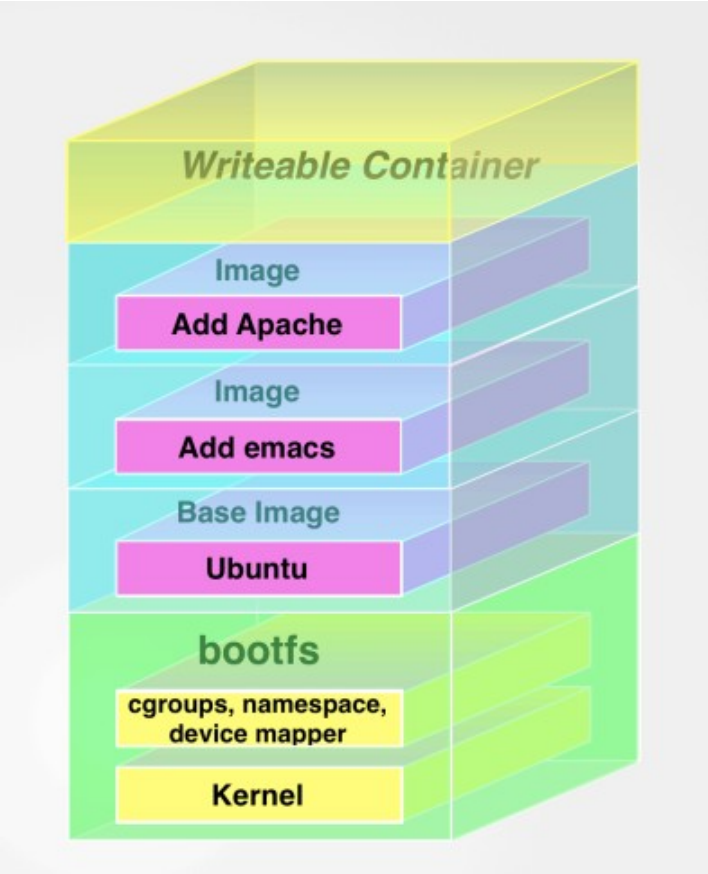
Siirrettävyys on myös yksi suurimmista eduista. Kontin sisällä rakennetut sovellukset voidaan siirtää yksittäisenä kappaleena ilman, että niiden toiminta muuttuu esimerkiksi eri kehittäjien välillä [8].

Skaalautuvuus näkyy siten, että Docker-kontteja voidaan ajaa fyysisillä servereillä, dataservereillä, sekä pilviympäristössä. Kontit voidaan helposti liikuttaa pilviympäristöstä paikalliselle fyysiselle koneelle, sekä takaisin nopeassa tahdissa. Tämä mahdollistaa esimerkiksi kehittäjän suorittamat nopeat virheidenkorjaukset [8].

Docker käyttää erityisesti kolmea Linuxin ominaisuutta konttiensa luomiseen turvallisesti ja tehokkaasti. Konttien luomiseen käytetään nimiavaruuksia, kontrolliryhmiä ja liittotiedostojärjestelmää [4] [10].

Kontrolliryhmät mahdollistavat tehokkaan resurssien hallinnan, kontrolliryhmien avulla käyttäjä voi hallinnoida konteissa oleville prosesseille näkyviä resursseja [10]. Tämänlaisia resursseja ovat tyypillisesti CPU:n, muistin, verkkokaistan, levymuistin ja prioriteetteihin liittyvät rajoitukset. Kontrolliryhmien avulla voidaan esimerkiksi antaa tietylle konttiryhmälle enemmän oikeuksia CPU:n käyttöön. Nimiavaruudet kokoavat käyttöjärjestelmän resurssit erillisiin instansseihin. Näiden instanssien käyttö luo konttien sisällä ajaville prosesseille illuusion, että resurssit ovat heidän itsensä omistuksessa. [4]

Liittotiedostojärjestelmää käytetään silloin, kun käyttäjä tekee muutoksia kuvaan. Ennemmin kuin kirjoittaen muutokset suoraan kontin käyttämään kuvaan, lisätään käyttäjän tekemät muutokset ylimääräisenä kerroksena kuvan päälle. Esimerkiksi käyttäjän asentaessa Ubuntu-pohjakuvaan emacsin ja Apachen, Docker-kuvan tilanne näyttää kuvan 5 kaltaiselta. Kuvassa päällimmäisenä on kontti, johon käyttäjä teki muutoksia.



Kuva 5: Ubuntu-pohjakuvan päälle lisätyt kerrokset [11].

Kehittäjän lisäämät emacsin ja Apachen sisältämät datakerrokset lisätään pohjakuvan päälle ja liittotiedostojärjestelmä tekee niistä yhtenäisen kokonaisuuden. Liittotiedostojärjestelmän käyttäminen tekee lisäksi kuvien jakamisesta tehokkaampaa, koska tällöin vain muuttunut kerros pitää päivittää [10].

Docker-konttien käytön haittapuoliin lukeutuu täydellisen virtualisaation puuttuminen, sillä Docker on riippuvainen isäntäkäyttöjärjestelmän ytimestä. Kontit eivät kykene kaikilla alueilla myöskään samanlaisiin nopeuksiin kuin puhtaasti fyysisessä käyttöjärjestelmässä ilman virtuaalisaatiota [12]. Lisäksi tällä hetkellä Docker ei tue 32-bittisiä käyttöjärjestelmiä. Tyypillisesti konttien klusterointiin on myös käytettävä lähes aina jotain ulkopuolista työkalua, tälläinen on esimerkiksi Googlen kehittämä Kubernetes. Myöskin konttiteknologian ekosysteemit ovat osittain jakautuneet eri yritysten välillä. Vaikka Docker perustuu avoimeen lähdekoodiin, niin se ei kuitenkaan toimi kaikkien yritysten kanssa täysin ongelmitta. Esimerkiksi Red Hatin konttien orkesterointityökalu toimii vain Kuberneteksen kanssa. [13]

3.4 Dockerin tehokkuus

Monet tämän hetken suurimmat pilvipalveluiden toimittajat, kuten Amazon EC2 käyttävät konttiteknologiaa palveluidensa tuottamiseen [1]. Konttiteknologia mahdollistaa skaalautuvuuden ja tehokkuuden, sekä ’pay-as-you-go’-laskutuksen käyttäjältä. Konttiteknologian ja virtuaalikoneen tehokkuutta vertaillaan taulukossa 1.

Taulukko 1: Konttiteknologian ja virtuaalikoneen tehokkuuden vertailu [12].

Parameter	Containers	Virtual Machines
Guest OS	Containers an OS and kernel. The image of the kernel is loaded into the physical memory.	All VMs run on virtual hardware, and a kernel is loaded into its own memory region.
Communication	Standard IPC mechanism such as signals, pipes, sockets, etc.	Through Ethernet devices
Security	Mandatory access control can be leveraged.	Depends on the implementation of a hypervisor
Performance	Containers provide near native performance	Virtual Machines suffer from a small overhead.
Isolation	Sub-directories can be transparently mounted and shared.	Sharing libraries, files between guests and between hosts is not possible.
Startup time	Containers can be booted up in a few seconds	Virtual Machines take a few minutes to boot up.
Storage	Containers take less amount of storage.	Much more storage usage is required as the whole OS kernel and its associated programs must be installed and run.

Dockeria verrattaessa fyysiseen käyttöjärjestelmään käy ilmi, että CPU:n ja muistin käytössä ei ollut juuri minkäänlaista eroa nopeudessa. Kuitenkin syöttö-, lähtö- ja käyttöjärjestelmäoperaatioissa Docker suoriutui huonommin [12].

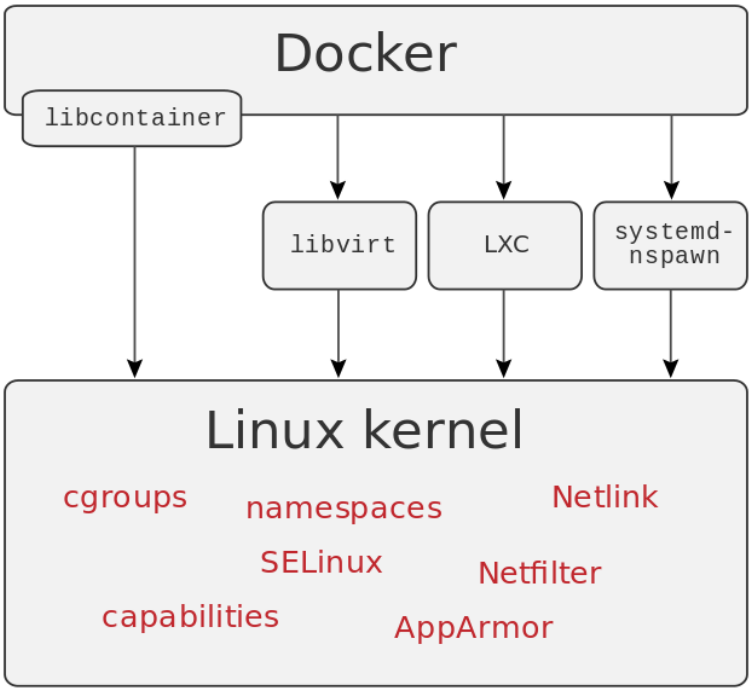
Docker-kontteja ja virtuaalikoneen tehokkuutta vertailtaessa fyysisessä käyttöjärjestelmässä päästiin tulokseen, että virtuaalikone käyttää 3.6-4.6 kertaa enemmän muistia, kuin Docker-kontit suoritettaessa samoja tehtäviä [14]. Lisäksi kontit suorittivat tehtävät nopeammin. Tutkimuksessa mitattiin CPU:n ja muistin käytön määrää, niiden tyhjäkäynnin aikaa ja verkkopalvelimen tehokkuutta [14].

Docker voittaa virtuaalikoneen käytön tehokkuudessa usealla alueella, kuten taulukosta 1 käy ilmi. Docker suorituu hyvin erityisesti suurilla datamäärillä. Konttiteknologia ja Docker suoriutuvat näistä tehtävistä paremmin, kuin perinteiset virtualisointitekniikat [12].

3.5 Dockerin turvallisuus

Ohjelmistokehityksessä on otettava myös ohjelmiston turvallisuus huomioon sovellusta kehitettäessä. Tämä tarkoittaa esimerkiksi tiedon luottamuksellisuuden ja eheyden takaamista. Tavoitteena on myös estää kaikenlaisten tietomurtojen ja muiden haitallisten hyökkäysten mahdollisuus. Yleisesti ajatellaan hypervisoripohjaisten virtualisointitekniikoiden olevan turvallisempia, kuin konttiteknologiaan pohjautuvien. Tämä johtuu siitä, että hypervisori on kokonaan eristyksissä ytimestä ja hypervisoripohjaisia tekniikoita on testattu jo vuosien ajan. Konttiteknologia on suhteellisen uusi ja kontit jakavat osan ytimestä isäntäkäyttöjärjestelmän kanssa. Ytimen jakamista pidetään epävarmuustekijänä [4].

Dockerin turvallisuuteen liittyy pääasiassa neljä eri osa-aluetta: kontin nimiavaruudet ja kontrolliryhmät, daemon, kontin konfiguraatio, sekä ytimen turvallisuus [15]. Tärkeimmät konttien turvallisuuteen liittyvät ominaisuudet näkyvät kuvassa 6.



Kuva 6: Dockerin tärkeimmät turvallisuuteen liittyvät ominaisuudet [16].

Käyttäjän luodessa uuden kontin, Docker luo sille myös omat nimiavaruudet ja kontrolliryhmät. Nimiavaruudet mahdollistavat kontin eristämisen muista prosesseista. Lisäksi jokainen kontti saa oman tietoverkkopinonsa, jonka avulla kontit voidaan asettaa keskustelemaan keskenään, mutta automaattisesti ne eivät saa oikeuksia päästä toisen kontin tietoliikennerajapintoihin. Tyypillinen tapa on asettaa portit, joiden kautta kontti voi keskustella toisen kontin kanssa. Ytimen nimiavaruus-ominaisuutta on käytetty jo vuodesta 2008 lähtien Linux-ytimessä ja useissa eri sovelluksissa, joten sen voidaan olettaa olevan luotettava tapa toteuttaa konttien eristäminen [15]. Nykyään Docker käyttää viittä eri nimiavaruutta, jotka vastaavat jokainen kontin eristämisestä eri osa-alueilla. Pid vastaa prosessien eristämisestä, net vastaa tietoverkon rajapinnoista, ipc vastaa prosessien välisestä kommunikaatiosta, mnt vastaa tiedostojärjestelmien liitoskohdista ja uts vastaa ytimen, sekä versiotunnisteiden eristämisestä [10].

Kontrolliryhmiä käytetään yksittäisen kontin tai konttiryhmän resurssien hallinnoimiseksi. Ne on otettu ominaisuutena käyttöön Linux-ytimeen vuonna 2006. Niitä käytetään, jotta yksittäinen kontti ei voi kuluttaa kaikkia käyttöjärjestelmän resursseja loppuun ja näin aiheuttaa koko järjestelmään pullonkaulaa. Hyökkääjä yrittää saada ohjelmaa tämänlaiseen tilaan esimerkiksi palvelunestohyökkäyksessä. Lisäksi ne ovat tärkeässä roolissa pilvisovelluksissa, joissa pyritään maksimoimaan ohjelmiston ylläoloaika. [15]

Docker daemonin on saatava toimiakseen pääkäyttäjän oikeudet, joka saattaa altistaa ohjelmaa monenlaisille haavoittuvuuksille [15]. Kaikki kontit ajetaan ilman pääkäyttäjän oikeuksia, eivätkä ne saa millään tavalla vaarantaa daemonin toimintaa. Kaikkien konttien ollessa eristyksissä toisistaan rajoitetuilla oikeuksilla, turvallisuus parantuu myös yhden kontin vaarantuessa. Tällöin daemon ja muut kontit ovat edelleen turvassa, jos vaarantuminen tapahtuu yhden kontin osalta.

Käyttäjä voi lisätä ytimen turvallisuutta esimerkiksi käyttämällä SELinuxin ja AppArmorin tuomia ominaisuuksia. Linuxin tavallisen harkinnanvaraisen pääsynvalvonnan, eli DACin tarkoitus on asettaa oikeuksia erilaisille järjestelmäobjekteille ja näin hallita niiden oikeuksia. SELinux tuo tähän lisäksi pakollisen pääsynvalvonnan, eli MACin. MAC on tarkoitus suorittaa DACin jälkeen. MAC perustuu otsikoiden käyttöön, siinä jokaisella prosessilla, tiedostolla ja järjestelmäobjektilla on oma otsikkonsa. MACin avulla varmistutaan siitä, että hyökkääjällä on oikeudet vain kontin sisäisiin resursseihin. [4]

AppArmorin suojausmekanismit perustuvat myös MACiin, mutta se lähestyy ongelmaa sovelluskohtaisesti. Se toteuttaa jokaiselle ohjelmalle ladattavan turvallisuusprofiilin, jossa on määrätty ohjelman oikeudet. Dockerin kanssa AppArmoria käytetään siten, että jokaiselle kontille määritetään oma turvallisuusprofiili. Mikäli sitä ei määritellä, daemon käyttää konttiin vakioprofiilia, joka suojaa kaikista tärkeimmät tiedostot järjestelmässä. [4]

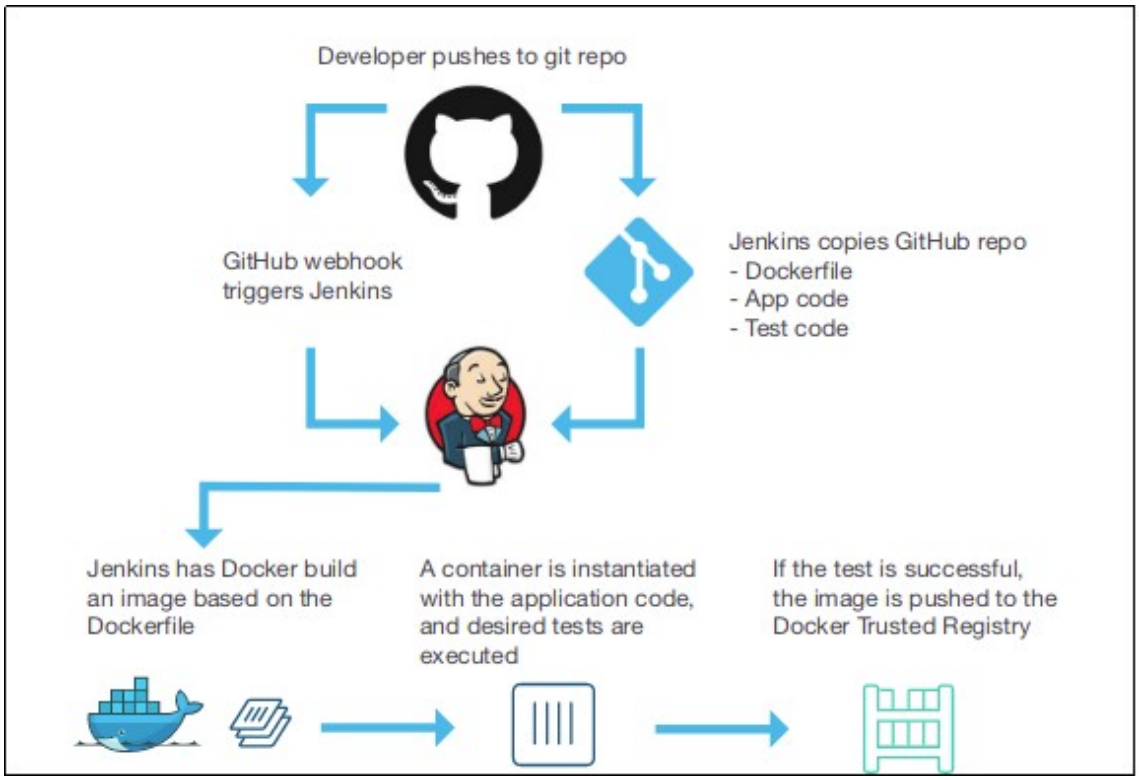
Ainoana turvallisuutta haittaavaa ominaisuus Dockerissa löydettiin sen käyttämässä oletusverkkomallissa. Sen käyttämä virtuaalinen tietoliikennesilta antaa mahdollisuuden ARP-väärennöksille ja MAC-tulville. Tämä saadaan kuitenkin estettyä lisäämällä sen estävä suodatin tai vaihtamalla yhteys käyttämään virtuaaliverkkoa. [4]

Edellämainituiden ominaisuuksien oikeanlainen käyttö Dockeria käytettäessä tuo riittävän turvallisuuden ohjelmistokehitykseen. Kontit eristävät toisensa hyvin käyttäen erilaisia turvallisuusmekanismeja. Tärkeää on ymmärtää, että riippumatta turvallisuusominaisuuksien oikeanlaisesta käytöstä, silti erilaisia tietovuotoja tapahtuu.

3.6 Jatkuva toimitus

Jatkuva toimitus tarkoittaa ohjelmistokehityksen mallia, jossa jokainen vaihe on automatisoitu siten, että tuotettavaa sovellusta voi päivittää nopealla tahdilla asiakkaalle. Näin sovelluksen kehittäjä pystyy parantamaan luotettavuutta ohjelmointityössä, vähentämään riskiä tehdä asioita väärin ja mahdollistamaan nopea asiakkaan palautteen huomioon ottamisen. Dockerilla on tärkeä osuus tämän toiminnan mahdollistamisessa.

Perinteinen ohjelman julkaisukierros tarkoittaa tuotettavan ohjelmiston uuden version julkaisua esimerkiksi aina tietyn ajan jälkeen. Tämä aika on voinut olla esimerkiksi laskettu kuukausissa, jonka jälkeen on julkaistu aina seuraava versio. Tähän julkaisukierrokseen sisällytetään kehitystyö, testaaminen ja tarvittavat infrastruktuurimuutokset julkaisuympäristöön. Ongelmia tämänlaisessa julkaisutavassa on monia. Ensinnäkin koska julkaisuja tehdään verrattaen harvoin, on oletettavissa ohjelmiston muuttuvan paljon. Tämä tarkoittaa automaattisesti sitä, että on suoritettava monia manuaalisia tehtäviä esimerkiksi testaamisen ja infran suhteen. Nämä vievät automaattisesti paljon aikaa ja maksavat rahaa. Tehtäessä suuria muutoksia on myös suuri todennäköisyys, että päivitettävät asiat eivät välttämättä täysin toimikaan yhdessä aiemman tuotteen kanssa. Tämä voi tarkoittaa esimerkiksi sitä, että joudutaan käyttämään paljon aikaa erilaisten riippuvuuksien selvittämiseen. Lisäksi itse ohjelmasta voi löytyä virheitä ja niiden korjaaminen on monimutkaista suuren muutosmäärän vuoksi. Näiden ongelmien vuoksi on kehitetty jatkuvan toimituksen malli, jossa kehittäjät pystyvät suorittamaan nopeasti kehitystä, testausta ja päivitystä mahdollisimman pienellä vaivalla suoraan asiakkaalle. Kuva 7 havainnollistaa kokonaisuutta jatkuvassa toimituksessa käytettävästä putkesta.



Kuva 7: Jatkuvan toimituksen esimerkkiputki [17].

Jatkuvan toimituksen tavoittteena on automatisoida kaikki mahdollinen ohjelman julkaisuun liittyvä toiminta. Docker tarjoaa monet tämän toiminnan mahdollistamiseen liittyvät tekniset ratkaisut. Aiemmin paljon työtä on vaatinut erilaisten riippuvuuksien halliintaan liittyvät ongelmat. Docker-tiedostoa käytettäessä kontin luomiseen, kaikki kuvaan liitetyt riippuvuudet, kuten kirjastot ja muut sovellukset ovat aina samat. Tämä tarkoittaa, että ohjelman ajoympäristö on aina sama, eikä ongelmia pääse syntymään.

Käytettäessä automaatiopalvelinta, kuten Jenkinsiä, voidaan helposti lisätä suorittavia kontteja tarpeen mukaan. Ennen konttiteknologiaa oltiin sidottuina vain tiettyyn määrään virtuaalikoneita. Tämä parantaa tehokkuutta vähentämällä aikaa, jonka palvelin on tyhjäkäynnillä ja mahdollistamalla ohjelman nopeat uudelleennkäynnistykset. Käytettäessä kuormituksentasaajana esimerkiksi Docker Swarmia pidetään huoli siitä, että ohjelman kuormitus pysyy tasaisena, eikä pullonkauloja pääse syntymään. Dockeria käyttämällä hyväksi ohjelman julkaisukierroksessa saadaan käytettyä tarjolla olevia resursseja tehokkaammin pitäen samalla turvallisuus ja käytettävyys mielessä [12].

4 YHTEENVETO

Docker on tämän hetken suosituin konttiteknologiaa käyttävä virtualisointityökalu. Dockerin pääkompontentteja ovat moottori, kuva ja kontti. Moottori pitää huolen kuvien hallinnasta, sekä konttien hallinnasta ja ajamisesta. Kuva pitää sisällään kaiken sovelluksen ajamiseen liittyvät asiat, kuten ympäristön, kirjastot ja itse sovelluskoodin. Näin varmistutaan kehittäjän ajaessa kuvaa kontissa, että koodiin tai ympäristöön ei tule muutoksia eri kehittäjien välillä. Kontti on standartoitu, kevyt ja turvallinen kokonaisuus. Kontin tarkoituksena on suorittaa sille annetussa kuvassa määritelty ohjelma. Kontin käytön suurimpina etuina ovat nopeus, siirrettävyys ja skaalautuvuus. Dockerin turvallisuudessa ei ole aukkoja, jotka altistaisivat sen todennäkösille hyökkäyksille ja sitä voidaan käyttää huoletta tuotantokoodissa.

Tässä tutkimuksessa on esitetty monia Dockerin käytöstä saatavia hyötyjä. Taulukko 1 antaa tarkemman kuvauksen vertailtaessa Dockeria virtuaalikoneisiin resurssienkäytön ja tehokkuuden näkökulmasta. Tiivistyksenä voidaan sanoa, että Docker suorittaa monet osa-alueet paremmin kuin virtuaalikone. Dockerin käyttö lisää näin nopeutta, varmuutta, ennustettavuutta ja ylläpidettävyyttä ohjelmistokehityksessä. Tästä hyvänä todisteena voidaan pitää jatkuvan toimituksen mallia, jonka Dockerin käyttö mahdollistaa.

LÄHTEET

[1] N. Dragoni, M. Mazzara, Joining Jolie to Docker, A. Giaretta, 2017, p. 1, https://arxiv.org/pdf/1709.05635.pdf

[2] J.E. Smith, R. Navi, The Architecture of Virtual Machines, 2015, pp. 1–2, https://ieeexplore-ieee-org.libproxy.tut.fi/stamp/stamp.jsp?tp=&arnumber=1430629&tag=1

[3] M. Pearce, S. Zeadally, R Hunt, Virtualization: Issues, Security Threats, and Solutions, 2013, p. 2, https://profsandhu.com/cs6393_s14/csur_virt_2013.pdf

[4] T. Buy, 2014, Analysis of Docker Security, pp. 1–7, https://arxiv.org/pdf/1501.02967.pdf

[5] C. Pahl, B. Lee, 2015, Containers and Clusters for Edge Cloud Architectures – a Technology Review, p. 3, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.726.832&rep=rep1&type=pdf

[6] S. T. King, G. W. Dunlap, P. M Chen, Operating System Support for Virtual Machines, 2003, p. 1, http://web.eecs.umich.edu/virtual/papers/king03.pdf

[7] Docker Engine, Docker Inc, viitattu 1.12.2018, https://docs.docker.com/engine/docker-overview/#docker-engine

[8] B. B. Rad, G. J. Bhatti, M. Ahmadi, 2017, An Introduction to Docker and Analysis of its Performance, pp. 2–4, http://paper.ijcsns.org/07_book/201703/20170327.pdf

[9] OCI-project, The Linux Foundation, viitattu 1.12.2018 https://www.opencontainers.org/about

[10] The underlying technology, Docker Inc, viitattu 1.12.2018, https://docs.docker.com/engine/docker-overview/#the-underlying-technology

[11] The Docker EcoSystem, viitattu 1.12.2018 https://washraf.gitbooks.io/the-docker-ecosystem/content/Chapter%201/Section%203/union_file_system.html

[12] T. Vase, Integrating Docker to A Continuous Delivery Pipeline – A Pragmatic Approach, 2016, pp. 22–50, https://jyx.jyu.fi/bitstream/handle/123456789/52756/URN%3ANBN%3Afi%3Ajyu-201701181181.pdf?sequence=1&isAllowed=y

[13] Hybrid cloud, enterprise Kubernetes, viitattu 1.12.2018, https://www.openshift.com/learn/topics/kubernetes/

[14] M. Chae, H. Lee, K. Lee, A performance comparison of linux containers and virtualmachines using Docker and KVM, 2016, p. 8, https://link-springer-com.libproxy.tut.fi/content/pdf/10.1007%2Fs10586-017-1511-2.pdf

[15] Docker security, Docker inc, viitattu 1.12.2018, https://docs.docker.com/engine/security/security/

[16] M. Dwars, W. Van Geest, R. Nijessen and R. Wieman, Docker: Build, Ship, and Run Any App, Anywhere, viitattu 1.12.2018, https://delftswa.github.io/chapters/docker/

[17] Docker Inc, Building a Continuous Integration Pipeline with Docker, 2015, p. 4, https://www.docker.com/sites/default/files/UseCase/RA_CI%20with%20Docker_08.25.2015.pdf