Pienyrityksen siirtyminen Linux-pohjaiseen konttiympäristöön

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tieto- ja Viestintätekniikan Insinööri

Kevät 2022

Nikolas Nilivaara

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja Viestintätekniikka Tiivistelmä
Tekijä Nikolas Nilivaara Vuosi 2022
Työn nimi Pienyrityksen siirtyminen Linux-pohjaiseen konttiympäristöön.
Ohjaaja Teemu Järvenpää

Opinnäytetyössä pyrittiin selvittämään pienyrityksen näkökulmasta, miten yrityksen tuotanto voitaisiin siirtää Linux-pohjaiseen konttiympäristöön. Tarkoituksena oli vertailla ja pohtia, millainen ratkaisu olisi optimaalinen pienyrityksen tarpeisiin. Opinnäytetyössä ohjeistetaan myös se, miten pienyritys voisi siirtää tuotantonsa tietynlaiseen konttiympäristöön.

Opinnäytetyössä esitellään erityisesti virtualisoinnin historiaa ja teoriaa. Näiden jälkeen paneudutaan konttiteknologioiden historiaan ja teoriaan. Niiden lisäksi esitellään myös erilaisia konttiteknologioita, sekä orkestrointiohjelmia.

Työssä ohjeistetaan myös, miten pienyritys voisi teoriassa siirtää tuotantoansa Docker-pohjaiseen konttiympäristöön. Tämä toteutetaan käytännössä siten, että Ubuntu-Linuxilla ylläpidetyt verkkosivut laitetaan Docker-konttiin. Tämän toiminnallisen esimerkin kaikki vaiheet dokumentoidaan.

Edellä mainitun lisäksi työssä ohjeistetaan se, kuinka verkkosivuista luotu Docker Image julkaistaan Docker Hub -sivustolla, ja pohditaan, että miksi näin kannattaa toimia. Työssä on dokumentoitu myös GitHub -ohjelmavaraston käyttöönottaminen osana tehokasta Docker-pohjaista konttiympäristöä.

Opinnäytetyön yhteenvedossa pohditaan mitkä ovat toiminnalliseen esimerkkiin valittujen teknologioiden heikkoudet ja vahvuudet. Siinä esitellään myös, mitä olisi voitu tehdä toisin esimerkiksi tilanteessa, jossa yrityksen tuotanto, tai yritys itsessään olisi laajempi. Pohditaan etenkin Kubernetesin mahdollista käyttöönottoa, mutta todetaan, että pienyritys ei välttämättä kaipaa konttiympäristökseen mitään Docker-pohjaista asetelmaa monimutkaisempaa.

Avainsanat Konttiteknologia, Docker, Virtualisointi
Sivut 26 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information- and Communications technology
Author Nikolas Nilivaara
Subject Migration of a small business to a Linux-based container environment.
Supervisors Teemu Järvenpää

Abstract
Year 2022

The aim of the thesis was to investigate from the perspective of a small business how to transfer business production to a Linux-based container environment. The purpose was to compare and contemplate what kind of solution would be optimal for the needs of a small business. The thesis also instructs how a small business could transfer their production to a specific kind of container environment.

Especially the history and theory of virtualization are showcased in the thesis. After these it delves into the history and theory of container technologies. In addition, different kinds of container technologies and container orchestration systems are presented.

The thesis examines how a small company could in theory transfer their production into a Docker-based container environment. This basically means that a website that is maintained on Ubuntu-Linux will be containerized. All the steps of this functional example are documented.

In addition to the above, the thesis instructs how a Docker Image, created of the website, is published on the Docker Hub -website, and contemplates why this is worth doing. The introduction of GitHub as a part of an efficient Docker-based container environment is presented in the thesis too.

The summary of the thesis considers what are the strengths and weaknesses of the technologies selected for the functional example. It also presents what could have been done differently in a situation where the company's production or the company itself had been larger for example. Especially the introduction of Kubernetes is considered, but it is concluded that a small business does not necessarily require a container environment more complicated than a Docker-based setup.

Keywords Container technology, Docker, Virtualization
Pages 26 pages

Sisällysluettelo

1 Johdanto

Palvelimet ovat olleet oleellinen osa modernia yritystä jo vuosia. Esimerkiksi asiat, kuten yrityksen verkkosivut, maksupalvelimet, tietokannat, ja sähköpostipalvelimet, on jo vuosien ajan ylläpidetty jonkinlaisilla paikallisilla palvelimilla. Tämä kuitenkin vaatii yritystä allokoimaan huomattavan määrän resursseja sekä palvelimien perustamiseen että niiden ylläpitämiseen. Teknologian kehittyessä palvelinten ylläpitämistä onkin pyritty tehostamaan, jotta etenkin pienemmät yritykset kykenisivät keskittämään resurssinsa tehokkaammin. Näinpä yrityskäytössä on ruvettu käyttämään virtualisoituja ympäristöjä.

Tässä opinnäytetyössä pyritään osoittamaan, miksi pienyrityksen olisi hyvä siirtyä käyttämään konttiratkaisuja palveluidensa ylläpitämisessä perinteisten ratkaisuiden sijasta. Konttiratkaisuiden ymmärtäminen vaatii kuitenkin osittain myös virtualisoinnin ja virtuaalikoneiden perusteiden sisäistämistä. Siksi opinnäytetyössä tullaan esittelemään virtualisointia ja sen historiaa, josta aihe johdatellaan konttiteknologioihin sekä niiden historiaan. Edellä mainittujen lisäksi tullaan teoriaosuudessa esittelemään ja vertailemaan erilaisia konttiteknologioita sekä pohtimaan, mitkä niistä soveltuisivat tehokkaasti pienyrityksen vaihteleviin tarpeisiin. Tarkoitus on pyrkiä tuomaan esille erityisesti se, miksi virtualisoituja ympäristöjä sekä konttiympäristöjä ylipäätään on olemassa.

Toiminnallisessa osassa pyritään järkeistämään ja ohjeistamaan konttiteknologioihin siirtymistä. Tarkoituksena on luoda yksinkertainen ohjeistus siitä, miten palvelut voidaan ottaa käyttöön, ja kuinka ylläpitää niitä järkevästi konttiympäristössä.

Osana toiminnallista esimerkkiä tullaan dokumentoimaan myös ohjeistus siitä, kuinka nettisivut voidaan siirtää Docker-pohjaiseen konttiympäristöön. Työssä esitellään myös, kuinka sivustoja Docker Hub ja GitHub kannattaa hyödyntää tehokkaan konttiympäristön rakentamisen ja ylläpitämisen apuna.

Opinnäytetyöllä on tarkoitus antaa vastaukset seuraaviin kysymyksiin:

- Miten pienyritys hyötyy konttiteknologioiden käyttöönotosta?
- Miten konttiteknologiat otetaan hyötykäyttöön pienyrityksessä?
- Minkälaiset konttiratkaisut sopivat pienyrityksen tarpeisiin?

2 Palvelittoman ympäristön teoria

2.1 Virtualisointi

2.1.1 Virtualisoinnin historia

Virtualisoinnin teknologiana voidaan todeta syntyneen 1960-luvulla. Hypervisor-ohjelmistokerrokset kehitettiin alun perin mahdollistamaan se, että usea henkilö kykenisi käyttämään tietokoneita, joita käytettiin eräajossa. Virtualisointi ei kuitenkaan yleistynyt käytössä, kuin vasta 1990-luvulla, ja 2000-luvun alussa. Tällöin elettiin tilanteessa, jossa monilla yrityksillä oli fyysisiä palvelimia, joissa oli paljon yhteensopivuusongelmia. Yrityksillä saattoi olla useiden eri valmistajien laitteita, mikä saattoi johtaa tilanteeseen, jossa tiettyjä sovelluksia ei voitu suorittaa, kuin vain tiettyjen valmistajien laitteilla. Usein palvelimet kykenivätkin suorittamaan vain yhtä niille räätälöityä tehtävää. (Red Hat, 2018a; Red Hat, 2018b)

Virtualisoinnin suosio kasvoi, sillä se muodostui hyväksi ratkaisuksi fyysisten palvelimien aiheuttamien ongelmakohtien korjaamiseen. Virtualisoidut ympäristöt mahdollistivat sen, että yritykset kykenivät ajamaan sovelluksiaan ja palvelujaan useissa eri versioissa ja käyttöjärjestelmissä, yksittäisellä tietokoneella. Tämä johti myös siihen, että palvelimet alkoivat hyötysuhteeltaan olemaan tehokkaampia ja ylläpitokustannuksiltaan halvempia. (Red Hat, 2018b)

Vuonna 2009 IDC raportoi, että virtuaaliset palvelimet ovat jo yleisemmässä käytössä, kuin fyysiset. Moderneissa yrityksissä fyysiset palvelimet on jo pitkälti hylätty. Tämä johtuu yksinkertaisesti siitä, että virtuaalisien palvelimien hallinnoiminen on tehty helpoksi ja tehokkaaksi, ja kustannukset fyysisten palvelinten pyörittämiseen verrattuna ovat paljon pienemmät. Virtuaaliympäristöt kykenevät myös skaalautumaan ja joustamaan käytännössä niin paljon, kuin tarve vaatii. Yritysten ei tarvitse enää investoida esimerkiksi uuteen fyysiseen laitteistoon teknologian vanhetessa, vaan riittää, että rahaa käytetään virtuaalisen ympäristön päivittämiseen. (Portnoy, 2016, s. 12)

Nykypäivänä myös hyvin yleiset pilvipalvelut perustuvat pitkälti virtualisoinnin tarjoamiin mahdollisuuksiin. Käytännössä pilvipalvelut ovat virtualisoituja ympäristöjä, jotka tarjoavat virtuaalisia alustoja, sovelluksia ja resursseja erilaisiin tarpeisiin. Oleellista on kuitenkin se, että siinä missä virtualisointi mahdollistaa useiden erilaisten käyttöjärjestelmien ja palveluiden suorittamisen yhden fyysisen laitteiston päällä, pilvipalvelut mahdollistavat yhden käyttöjärjestelmän tai palvelun suorittamisen useissa erillään olevissa fyysisissä verkoissa. (Oludele ym., 2014; ks. myös Red Hat, 2018c)

2.1.2 Virtualisoinnin teoria

Virtualisointi tietotekniikassa tarkoittaa usein jonkin fyysisen komponentin tai laitteen virtuaalista vastaparia tai abstraktiota, jonka avulla käytössä olevia resursseja voidaan hyödyntää laajemmin. Esimerkiksi paikallisverkon sisälle voidaan rakentaa virtuaalisia paikallisverkkoja, jotka tekevät verkon hallinnoimisesta ja resurssien käytöstä tehokkaampaa ja joustavampaa. (Portnoy, 2016, s. 2)

Tietokoneiden virtualisoiminen mahdollistaa useiden simuloitujen ympäristöjen ylläpitämisen, yhden fyysisen laitteiston päällä. Hypervisor-sovellus mahdollistaa tämän luomalla yhteyden tietokoneen raudan kanssa, ja mahdollistaen tietokoneen jakamisen useisiin eristettyihin virtuaaliympäristöihin, joita kutsutaan virtuaalikoneiksi (Kuva 1, s. 6). Hypervisor on nimittäin vastuussa siitä, että nämä virtuaalikoneet kykenevät hyödyntämään fyysisen tietokoneen tarjoamia resursseja, ja siitä, että niille jaetaan niitä tarpeen mukaan. Virtuaalikoneet nimittäin käsittelevät tietokoneen tarjoamaa muistia tai suoritintehoa varantoina, joista ne tarpeen vaatiessa ottavat itselleen lisää resursseja, jos tämä on sallittu. Kaikki virtuaalikoneiden suorittaminen tapahtuu silti siis fyysisellä tasolla. (Red Hat, 2020)

Tietotekniikan kehittyessä, on saavutettavuus muodostunut kriittiseksi tekijäksi. Virtuaalikoneiden suurimpia etuja onkin juuri se, että oikein käsiteltyinä, ne ovat aina saatavilla. Siinä missä fyysiset laitteet ovat aina olleet suojattomia ulkopuolisia voimia, kuten esimerkiksi luonnonkatastrofeja vastaan, voidaan virtuaalikone kokonaisuudessaan siirtää fyysiseltä isännältä toiselle ilman, että siitä aiheutuu keskeytystä palvelulle. Virtualisoitu ympäristö mahdollistaa myös sen, että palvelimen palauttaminen varmuuskopiosta voidaan suorittaa hyvin lyhyessä ajassa. Tämän lisäksi asioiden kuten suoritintehon, ja muistin

lisääminen virtuaalikoneelle, voidaan suorittaa ilman, että palvelinta tarvitsee käynnistää uudestaan. (Portnoy, 2016, ss. 13–14)

2.2 Konttiratkaisut

2.2.1 Konttiratkaisujen historia

Aikana ennen virtualisoituja ympäristöjä, sovelluskehittäminen yrityksissä oli hidasta ja työlästä. Monesti ohjelmakehitys toimi niin, että ensin yksittäinen kehittäjä tuotti sovelluksen tai ohjelman parhaansa mukaan. Tämä sitten jaettiin seuraavassa portaassa oleville operatiivisille insinööreille, joiden tehtäväksi jäi ohjelman asentaminen ja suorittaminen tuotantopalvelimilla. Käytännössä tämä saattoikin toimia ihan hyvin niin kauan, kun kehittäjiä oli vähäinen määrä ja he työskentelivät samojen ohjelmien parissa. (Schenker, 2018, s. 10)

Ongelmia alkoi ilmenemään, kun yrityksessä toimi useita eri kehittäjiä, tai kehittäjätiimejä, jotka työskentelivät erilaisten ohjelmien parissa, jotka tulisi kuitenkin lopulta asentaa ja suorittaa samoilla fyysisillä palvelimilla. Useasti jokaisella sovelluksella on omat riippuvuutensa erilaisista kirjastoista ja ohjelmistokehyksistä. Joskus on jopa sellainen tilanne, että kahdella erillisellä sovelluksella on riippuvuutena saman ohjelmistokehyksen eri versiot, jotka saattavat olla yhteensopivia keskenään, tai sitten eivät ole. (Schenker, 2018, s. 10)

Ensimmäinen ratkaisu tähän oli rakentaa virtuaalisoituja ympäristöjä ja virtuaalikoneita, joiden avulla yhden fyysisen laitteiston päällä voitaisiin ajaa kaikkia sovelluksia omissa virtuaalikoneissansa. Yksittäinen virtuaalikone yksittäistä sovellusta kohden on kuitenkin suhteellisen raskasta, ja paljon resursseja kuluu hukkaan, kun kokonaista käyttöjärjestelmää ja kaikkia sen tarvitsemia resursseja joudutaan virtualisoimaan vain yhtä sovellusta varten. Näinpä tarpeelliseksi tuli kehittää jotain, joka olisi virtuaalikoneita kevyempi, mutta mahdollistaisi tuotannon ja ylläpidettyjen sovellusten siirtelyn ja päivittämisen helposti. (Schenker, 2018, s. 10—11)

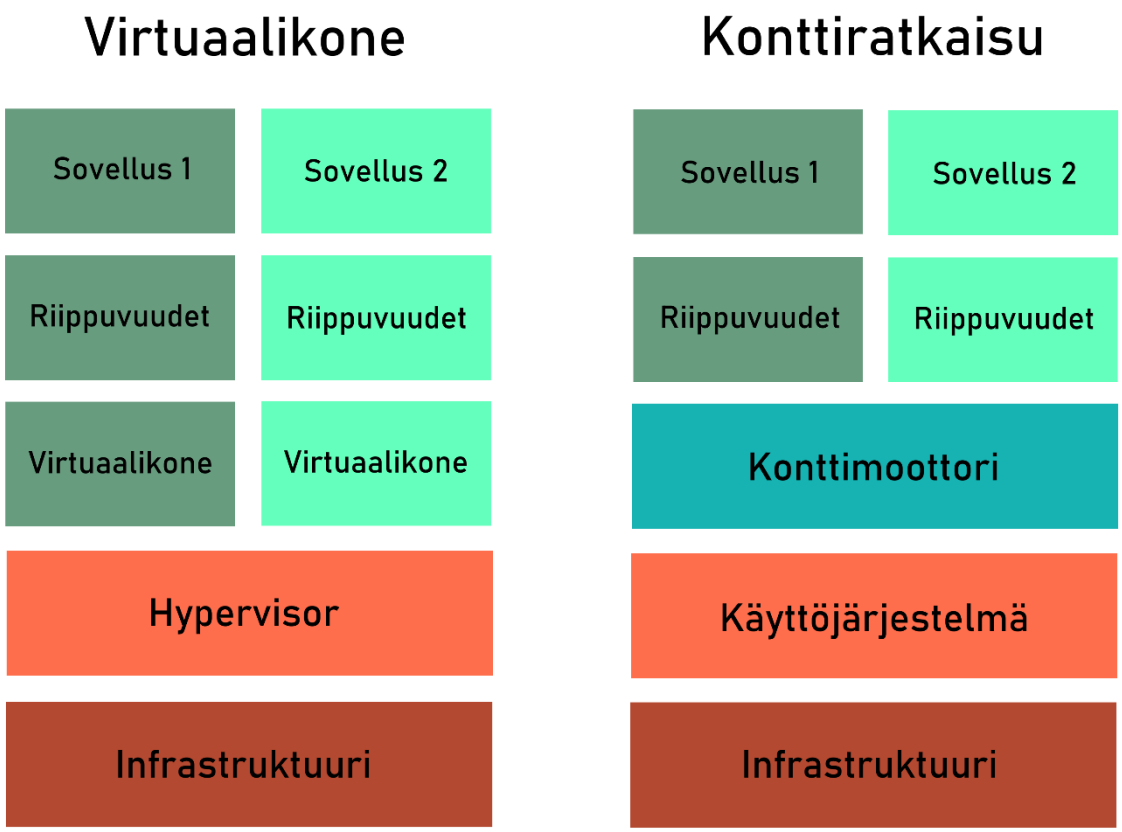
Näin kehitettiin konttiteknologiat, jotka mahdollistaisivat sen, että yhdellä isäntäkäyttöjärjestelmällä voidaan samanaikaisesti ylläpitää useita sovelluksia. Tämän lisäksi nämä syntyneet kontit ovat kevyitä, ja sisältävät vain ja ainoastaan kaiken sen tarpeellisen, mitä kontissa oleva sovellus tarvitsee. (Schenker, 2018, s. 10—11)

2.2.2 Konttiratkaisujen teoria

Konttiratkaisut ja -teknologiat ovat huomattavan resurssiystävällinen tapa toteuttaa virtualisoitu ympäristö virtuaalikoneisiin verrattuna. Siinä missä virtuaalikoneet hyödyntävät ja tarvitsevat Hypervisor-sovellusta saadakseen virtualisoituja resursseja käytettäväkseen, konttiratkaisuissa tätä ei tarvita. Konttiratkaisuissa toimintaperiaate on se, että yksittäisessä kontissa virtualisoidaan käyttöjärjestelmä, ja se sisältää vain sovelluksen ja kaikki sen riippuvuudet (Kuva 1, s. 6). Konttimoottorin avulla se toimii omana eristettynä prosessinaan isäntänä toimivan käyttöjärjestelmän päällä. (Morabito, 2017; ks. myös IBM Cloud Education, 2021b)

Kaikki ympäristössä ylläpidetyt kontit jakavat yhdessä käyttöjärjestelmän ytimen eli kernelin. Tämä jaettu kerneli yhdessä sen kanssa, ettei tarvetta virtualisoiduille resursseille tai ajureille ole, tekee mahdolliseksi tiiviimmän ja levykooltaan pienemmän ympäristön saavuttamisen. (Morabito, 2017)

Kuva 1. Virtuaalikoneiden ja konttiratkaisuiden eroavaisuudet visuaalisesti esitettynä.



2.3 Erilaisten konttiratkaisujen esittely

2.3.1 Docker

Docker on avoimeen lähdekoodiin perustuva kevyt konttialusta. Se mahdollistaa sovellusten ajamisen ja kehittämisen kevyissä konttiympäristöissä. Yhdessä Docker Hub -varaston kanssa, se tekee esimerkiksi yrityksen tuotantoympäristön ylläpitämisestä ja päivittämisestä yksinkertaista ja turvallista. Se perustuu ideaan, jossa kaikki sovelluksen tarvitsemat riippuvuudet ja koodi ovat kontin sisällä, ilman mitään tarpeetonta. (IBM Cloud Education, 2021a; Docker, n.d.)

Docker on myös rakennettu siten, että etenkin päivitysten testaaminen on nopeaa, ja vanhoihin versioihin on helppo palata ongelmatilanteissa. Dockerin versionhallinta mahdollistaa myös sen, että useita versioita samasta sovelluksestakin voidaan ylläpitää

samanaikaisesti. Tarvittaessa on sovelluksia mahdollista päivittää myös siten, että osa sovelluksesta on toiminnassa, samalla kun osaa siitä päivitetään. (IBM Cloud Education, 2021a; Docker, n.d.)

2.3.2 Podman

Podman on avoimeen lähdekoodiin perustuva ja Red Hatin kehittämä, konttien ja imagejen etsimiseen, käyttämiseen, rakentamiseen, ja jakamiseen perustuva konttimoottori. Podman mahdollistaa sen, että kontteja voi käyttää ilman root-oikeuksia, mikä tekee siitä esimerkiksi Dockeria tietoturvallisemman. Podman toimii hyvin samankaltaisesti kuin Docker, ja sitä onkin mahdollista käyttää, vaikka Dockerin rinnalla. Podman ei kuitenkaan tue esimerkiksi Docker Swarmia, ja toisin kuin Docker, vaatii imagejen rakentamiseen erillisen työkalun nimeltään Buildah. (Gamela & Figueiredo, 2021; ks. myös Podman, n.d.)

Podmanissa erikoista on se, että perinteisen konttijaon ohella, Podman kykenee luomaan Pod-pohjaisia rakenteita, joissa kontit kykenevät tekemään yhteistyötä toistensa kanssa. Käytännössä siis niin, että useat erilliset kontit kykenevät esimerkiksi jakamaan resursseja tai tasaamaan työtaakkaa yhden podin sisällä. Yksi kontti voisi vastata esimerkiksi sovelluksen visuaalisesta puolesta, toinen tietokannasta, ja niin edelleen. Podman onkin hyvin samanlainen, kuin Kubernetes, ja nämä sovellukset toimivatkin hyvin yhdessä. Podman toimii myös konttiratkaisuvakiona Red Hat, ja CentOS Linux-jakeluissa. (Gamela & Figueiredo, 2021; ks. myös Podman, n.d.)

2.3.3 CRI-O

CRI-O on Red Hatin kehittämä kevyt vaihtoehto Dockerille. Se on räätälöity toimimaan sovelluspalveluna nimenomaan Kubernetekselle. Se mahdollistaa sen, että Kubernetes voi hyödyntää mitä tahansa OCI-standardin mukaista konttiratkaisua podien käyttämiseen. Käytännössä CRI-O hoitaa kommunikaation Kuberneteksen ja konttien välillä. (CRI-O, n.d.; Brockmeier, 2017)

Sitä ei kuitenkaan voida juuri käyttää konttien hallintaan ja imagejen luontiin, vaan näitä varten on käytettävä muita työkaluja. CRI-O sisältää kuitenkin komentoliittymän, mutta sitäkään ei voida juuri käyttää kuin CRI-O:n toiminnan testaamiseen. (Brockmeier, 2017)

2.4 Orkestrointiohjelmia

2.4.1 Docker Swarm

Docker Swarm, on Dockerin kehittämä avoimen lähdekoodin orkestrointityökalu. Sen päällimmäisenä tehtävänä on muuttaa joukko Docker-hosteja yhdeksi virtuaaliseksi hostiksi, mikä mahdollistaa sen, että niitä kaikkia kyetään hallitsemaan yhdestä pisteestä. Hyvä syy ottaa Docker Swarm käyttöön on esimerkiksi se, että se on alusta lähtien luotu toimimaan Dockerin kanssa. Se on helppo ottaa käyttöön tuotannossa, ja sen asentamiseen ei tarvita edes mitään erillisiä työkaluja, mikäli Docker on jo asennettuna palvelimelle. (Soppelsa & Kaewkasi, 2016, ss. 8—11)

Käytännössä Swarm kuuluu orkestrointityökaluista kevyimpiin. Näinpä se onkin optimaalinen valinta sellaisissa yrityksissä, joissa tuotantoa on palvelimilla niin paljon, että sitä varten kannattaa tällainen järjestelmä ottaa käyttöön. Jos yrityksen konttiympäristö on kuitenkin laajempi, kannattaa yrityksen harkita mahdollisesti esimerkiksi Kubernetestä. Se nimittäin sisältää enemmän toiminnallisuuksia, ja sen suuria vahvuuksia on automatisointi. (Soppelsa & Kaewkasi, 2016, ss. 13—14; Soppelsa & Kaewkasi, 2016, s. 17)

2.4.2 Kubernetes

Kubernetes on alun perin Googlen vuonna 2014 kehittämä avoimen lähdekoodin orkestrointityökalu konttiympäristöjen hallintaan. Se tarjoaa luotettavan ja erittäin skaalautuvan ympäristön, jossa voidaan helposti ylläpitää vaikka isonkin yrityksen tuotantoa. Nykypäivänä Kubernetes onkin vakiintunut ohjelmointirajapintana, jota hyödynnetään monissa erilaisissa pilviympäristöissä. (Burns ym., 2018, s. 1)

Kuberneteksellä on useita ominaisuuksia, jotka tekevät siitä houkuttelevan vaihtoehdon kaikenkokoisille yrityksille. Esimerkiksi sen luotettavuus tulee hyvin esille siinä, että se pyrkii

korjaamaan itse itseään. Mikäli jokin menee pieleen tai hajoaa, se pyrkii suojaamaan ympäristöä, ja tekee kaikkensa palauttaakseen ympäristön siihen tilaan, joka siihen on konfiguroitu. Tämä tarkoittaa sitä, että mikäli jotain tuotannolle kriittistä hajoaa, ei sitä tarvitse käsivoimin olla korjaamassa. Näin säästetään aikaa ja kuluja järjestelmän ylläpitämisessä, ja niitä voidaan allokoida muualle. (Burns ym., 2018, s. 5)

3 Toiminnallisen esimerkin esittely

3.1 Konttiratkaisun esittely

Toiminnallisen esimerkki yritystuotannon siirtämisestä konttiympäristöön tullaan toteuttamaan käyttämällä Ubuntu 20.04 LTS -Linuxia alustana. Se on kevyt ja helppokäyttöinen palvelinympäristö, johon esimerkiksi Docker-alustan voi valita asennettavaksi jo itse palvelimen asennusvaiheessa. Se on myös hyvin yleisessä käytössä, mikä tarkoittaa, että dokumentointi sen suhteen on parempaa, ja kattavampaa.

Asia, joka siirretään konttiympäristöön, on tässä tapauksessa yksinkertainen verkkosivu, joka simuloi yrityksen verkkosivuja. Käytännössä kyseessä voisi olla vaikkapa tietokanta, ja kaikki mitä sen ylläpitäminen tarvitsee. Verkkosivut ovat kuitenkin kevyempi vaihtoehto näin esimerkin kannalta. Itse verkkosivu tulee olemaan ylläpidettynä Nginx-palvelimella.

Esimerkissä käytettäväksi konttiympäristöksi valittiin Docker. Tämä siksi, koska se on nopea ja helppo pystyttää, ja sopii täydellisesti pienyrityksen tarpeisiin. Jos tuotanto ei ole liian laaja, niin sitä on käytännössä mahdollista ylläpitää vaikka pelkällä Dockerilla.

3.2 Konttivarasto konttiratkaisulle

Docker hub on Dockerin ylläpitämä pilvipohjainen versio avoimen lähdekoodin Docker Registry konttivarastosta. Docker Imaget voidaan varastoida sinne, tai ladata sieltä käytännössä hyvin samankaltaisesti, kuin miten GitHubissa käsiteltäisiin koodia. Sivusto mahdollistaa versionhallinnan, eli käytännössä useita eri versioita samasta imagesta voidaan kehittää samanaikaisesti ja esimerkiksi vanhoihin versioihin voidaan palata. Merkittävää on myös se, että käyttäjät kykenevät sivustolla esimerkiksi skannaamaan haavoittuvuuksia

Imageissa. Edellä mainittujen lisäksi sivustolla pystyy halutessaan myös ostamaan ja myymään omia imageja.

Käytännössä Docker Hubissa maksumuurin takana ovat mahdollisuus ylläpitää loputonta määrä yksityisiä ohjelmavarastoja, sekä mahdollisuus kehittää useita versioita samasta imagesta samanaikaisesti. Näiden lisäksi maksumuurin takana ovat tietoturvatyökalut, kuten käyttäjäoikeuksien hallinta tiimin sisällä, sekä koodin skannaaminen haavoittuvuuksien varalta. Myös mahdollisuus GitHubin käyttämiseen Docker Hubin kanssa, on maksumuurin takana.

Docker Hub ei ole kuitenkaan ainoa konttivarasto, vaan vaihtoehtoisiakin konttivarastoja löytyy. Esimerkiksi Amazon tarjoaa konttivarastoksi palvelua Amazon Elastic Container Registry, tai tuttavallisemmin Amazon ECR. Toiminnallisuuksiltaan se on samanlainen, kuin Docker Hub, ja maksumuuri ei ole rajoittava tekijä ominaisuuksien suhteen. Käytännössä palvelu muuttuu maksulliseksi, kun dataa ruvetaan siirtämään useita satoja gigoja kuukausittain. Tällöin palvelu rupeaa veloittamaan kiinteää hintaa siirretyn datan määrän perusteella.

Amazon ECR:stä tekee yrityskäytössä houkuttelevan se, että se on tehty toimimaan yhdessä palveluiden Amazon Elastic Container Service, eli Amazon ECS, sekä Amazon Elastic Kubernetes Service, eli Amazon EKS, kanssa. Käytännössä palvelua Amazon ECS käytetään konttien ylläpitämiseen, ja Amazon EKS toimii Kubernetes klustereiden ylläpitäjänä ja luojana.

Kokonaisuudessaan tämä ympäristö on siis toimiva, mutta koska Amazon ECR on räätälöity toimimaan juuri näiden kanssa, ei se ole houkutteleva vaihtoehto tämän toiminnallisen esimerkin konttivarastoksi. Kuitenkin suuremmassa yrityksessä, jossa halutaan myös Kubernetes käyttöön, se voi olla juuri sopiva.

Toinen merkittävä moderni tarjoaja konttivarastoille on Microsoftin Azure Container Registry. Se tarjoaa pitkälti samat palvelut, kuin mitä Amazon ECR, mutta datamääristä maksamisen sijasta, palvelun hinta määräytyy sen mukaan, millaista ympäristöä siellä ylläpidetään. Esimerkiksi tilaltaan suuremman varaston ylläpitäminen maksaa enemmän, ja hinta muodostuu päivä kerrallaan.

Mainittakoon vielä Googlen tarjoama Google Container Registry. Se tarjoaa pitkälti kaikki samat palvelut, kuin mitä Azuren tai Amazonin vastaavat, ja toimii Googlen pilvipalvelussa. Palvelun hinta määräytyy pitkälti sen mukaan, kuinka paljon dataa siirretään, ja millaisen varastoratkaisu valitaan.

Loppuratkaisuna todettakoon, että vaikka kaikki edellä mainitut konttivarastot on räätälöity toimimaan yhdessä Dockerin kanssa, niin toiminnallista esimerkkiä varten päädyttiin käyttämään Docker Hubia. Pienyrityksen tarpeita varten se on nopeasti käyttöönotettava, ja yksinkertainen käyttää.

Lopuksi työssä esitellään se, miten GitHub otetaan käyttöön osana Docker-pohjaista ympäristöä. GitHub on verkkosivusto, joka tarjoaa graafisen käyttöliittymän versionhallintaan ja koodin ylläpitämiseen erinäisille ohjelmistokehitysprojekteille. Se on nykypäivänä niin yleisessä käytössä sekä yritys-, että siviilipuolella, että sen voidaan todeta olevan hyvä ratkaisu myös Docker Hubin rinnalle. Käytännössä sitä voidaan hyödyntää sekä imagejen koodin varastoimiseen että niiden kehittämiseen. Siellä varastoidusta koodista voidaan esimerkiksi rakentaa vaikkapa image suoraa Docker Hubiin, jos näin halutaan.

3.3 Lähtötilanne

Lähtötilanteena projektille toimivat siis verkkosivut, joita ylläpidetään yhdellä fyysisellä palvelimella. Käytännössä tämä yksittäinen tietokone on kokonaisuudessaan dedikoitu vain tätä yhtä merkitystä varten.

Käytössä on puhdas Ubuntu 20.04 LTS -Linux, jolle on asennettu vain ja ainoastaan Nginx-palvelin, ja sen riippuvuudet. Sen lisäksi kansiosta ”/var/www/html”, löytyy ”testisivu.html”, jota Nginx käynnistyttyään alkaa isännöimään. Tähän päästään sitten kiinni selaimella, menemällä osoitteeseen ”http://192.168.1.121/testisivu.html”, jossa ”192.168.1.121” on palvelimen osoite paikallisverkossa.

3.4 Termien selittämistä

Docker Image on yksittäinen tiedosto, johon on sisällytetty sovellus, sekä kaikki sille tarpeelliset riippuvuudet ja ajurit. Se sisältää periaatteessa myös virtualisoidun käyttöjärjestelmän. Toiminnallisen esimerkin konttiratkaisun tapauksessa se sisältää kaiken, mitä web-palvelimen ylläpitäminen vaatii. Kun työssäni puhun imageista, niin tarkoitan juuri tällaisia tiedostoja. Tälle termille ei ole suomenkielistä käännöstä.

Docker Engine on konttimoottori, joka suorittaa Dockeria ja sen kontteja palvelimellamme. Käytännössä juuri sillä luodaan ja ylläpidetään kontteja, kuten tässä toiminnallisessa esimerkissä luotava web-palvelin.

4 Toiminnallisen esimerkin toteuttaminen

4.1 Docker

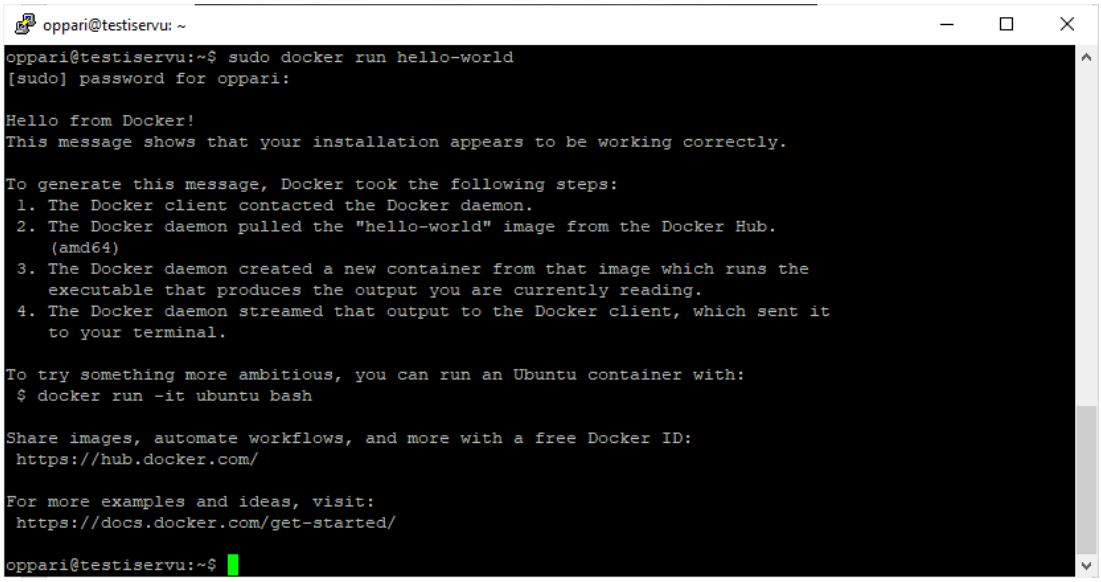
Toiminnallisen esimerkin työstämisen aloittamiseksi tarvitaan paketti Docker Engine. Koska esimerkiksi Docker on muodostunut erittäin suosituksi työkaluksi nykypäivänä, tarjoaa Ubuntu-palvelin mahdollisuuden sen asentamiseen jo itse palvelimen asennusvaiheessa. Jos sitä ei kuitenkaan asenneta tässä vaiheessa, tai käytössä on jokin toinen Apt-paketinhallintajärjestelmää käyttävä Linux, voidaan se, ja kaikki sen tarvitsemat riippuvuudet asentaa komennolla:

- sudo apt-get install docker-ce docker-ce-cli containerd.io

Kun Docker-Engine saadaan asennettua, tulee varmistaa sen toiminnallisuus. Se voidaan todeta tulosteesta (kuva 2, s. 13), joka saadaan komennolla:

- sudo docker run hello-world

Kuva 2. Docker-enginen toiminnallisuuden toteaminen.

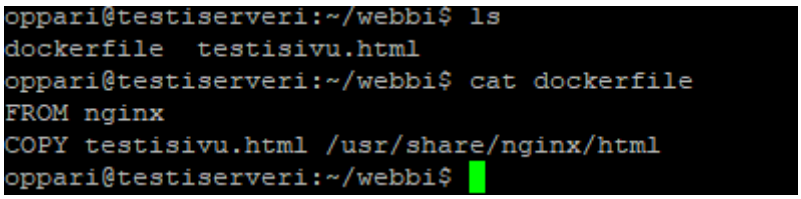


Seuraavaksi tarkoituksena on saada kuvitellun yrityksen verkkosivut laitettua konttiin, ja ylläpidettyä niitä sitä kautta. Ensiaskel tätä varten on ladata Nginx-palvelimen Docker Image, Docker Hub - varastosta komennolla:

- sudo docker pull nginx

Tätä hyväksikäyttäen voidaan luoda Docker File eli konfiguraatiotiedosto, jolla kyetään luomaan uusi Docker Image. Tässä tapauksessa kyseessä on siis yksinkertainen tekstitiedosto, jossa ”FROM” komennolla rakennetaan image Nginx-imagen pohjalta, ja ”COPY”-kohdassa kopioidaan nykyisestä kansiosta nettisivut kansioon ”/usr/share/nginx/html” (Kuva 3). Tätä varten on tiedosto ”testisivu.html” siirretty tähän kansioon kohteesta ”/var/www/html”.

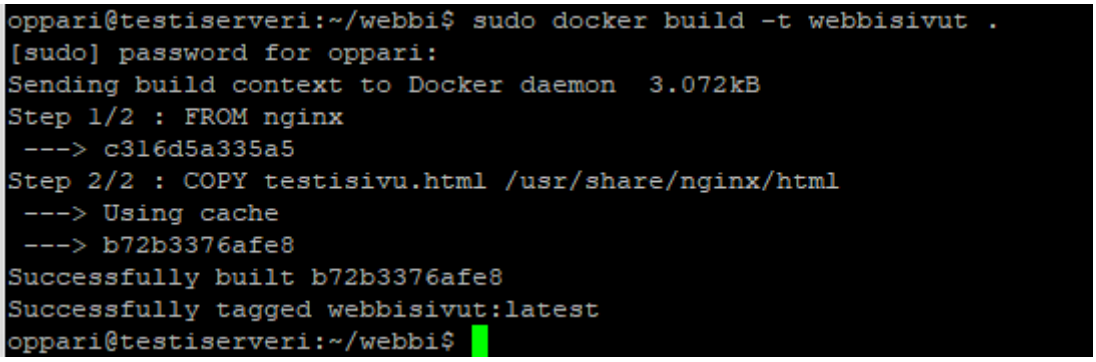
Kuva 3. Kansion ja Docker Filen sisältö



Tuotetun Docker Filen avulla, voidaan nyt luoda uusi image nimeltään ”webbisivut” (Kuva 4, s. 14). Tämä tapahtuu komennolla:

- sudo docker build -t webbisivut .

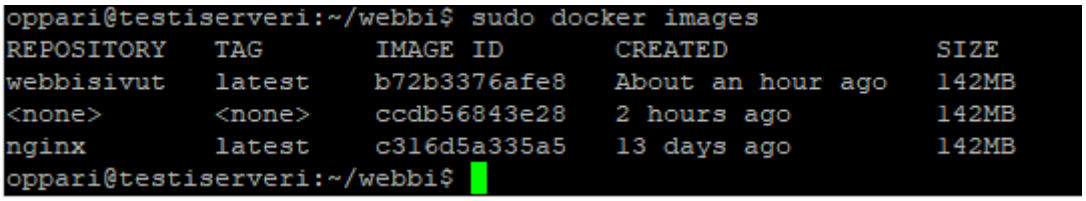
Kuva 4. Docker-imagen luonti.



Nyt löytyy image, joka sisältää kaikki oleelliset ohjelmistot ja käyttöjärjestelmän, ja kykenee näin itsenäisesti ylläpitämään verkkosivuja. Samalla voidaan todeta sekä luodun imagen että Nginx-imagen molempien löytyvän (Kuva 5) komennolla:

- sudo docker images

Kuva 5. Docker-imaget listattuna



Nyt täytyy vain luoda kontti, hyväksikäyttäen tätä luotua imagea. Teoriassahan yhtä Docker-imagea voidaan käyttää, vaikka kuinka monen kontin luomiseen tarvittaessa. Tämä kuitenkin tapahtuu esimerkiksi komennolla:

- sudo docker run --name webbipalvelin -d -p 8080:80 webbisivut

Komennossa “sudo docker run” käynnistää ja luo uuden kontin, jolle ” --name” parametrillä annetaan nimeksi ”webbipalvelin”. Parametri taas ”-d” mahdollistaa kontin ajamisen taustalla.

Docker kontilla on omasta näkökulmastaan käytännössä kaikki samat verkko-osoitteet, kuin esimerkiksi tietokoneillakin. Mikäli konttien verkkoajureihin ei kosketa, kykenevät samalla isännällä olevat kontit vakiona keskustelemaan keskenään IP-osoitteiden avulla, bridge-tyylisessä verkossa.

Kontti ei kuitenkaan näy samalla isännällä olevien konttien muodostaman verkon ulkopuolelle vakiona, vaan tarvitaan parametri ”-p”, joka ohjeistaa kontin käyttämään tiettyjä portteja. Tässä tapauksessa 8080:80 tarkoittaa, että NGINX käyttää porttia 80, eli perinteistä http-protokollan porttia liikennöimiseen. Itse kontti taas käyttää porttia 8080 näkyäkseen kontin ulkopuolelle. Huomattavaa on, että tuo portti 8080 voisi olla mikä tahansa muukin portti. Käytännössä tämä siis tarkoittaa, että päästäkseen käsiksi verkkosivuihin paikallisverkkomme sisällä, on mentävä osoitteeseen ”localhost:8080/testisivut.html” (Kuva 6). Tässä tapauksessa ”localhost” viittaa IP-osoitteeseen ”192.168.1.122”, eli palvelimemme IP-osoitteeseen paikallisverkossa. Komennossa viimeisenä oleva ”webbisivut” ilmoittaa, että mitä imagea käytetään tämän kontin luontiin. Julkiverkosta ei konttiin tai verkkosivuihin päästä käsiksi.

Kuva 6. Nettisivusto ylläpidettynä Docker-kontista.

Joskus voi olla tarpeellista käynnistää kontti uudelleen, kuten vaikka tilanteessa, jossa se on mennyt jumiin syystä tai toisesta. Kontti voidaankin käynnistää uudelleen komennolla:

- sudo docker restart webbipalvelin

Käytännössä kontin sisällön päivittäminen vaatii, että luodaan kokonaan uusi image, jonka avulla käynnistetään ja luodaan uusi kontti. Tämä kuitenkin vaatii joko eri portin käyttämistä, tai vanhan ”webbipalvelin” kontin pysäyttämistä ja poistamista komennoilla:

- sudo docker stop webbipalvelin
- sudo docker rm webbipalvelin

4.2 Docker Hub

Seuraava askel tuotannon tehokkaassa muuttamisessa konttipohjaiseksi, on säilöä nämä Docker-kuvat jossakin konttivarastossa. Tätä varten on olemassa esimerkiksi Docker Hub -sivusto, josta aikaisemmin ladattiin Nginx-image.

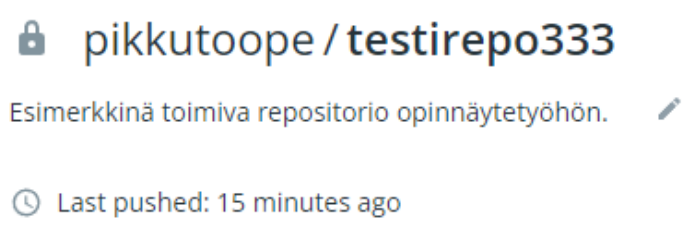
Ensimmäiseksi tulee luoda käyttäjä sivustolle https://hub.docker.com/. Tämä mahdollistaa sen, että sinne voidaan julkaista useita julkisia varastoja, tai yksi yksityinen, kuten kuvassa 7. Mikäli yrityksen tarkoituksena on julkaista enemmänkin yksityisiä ohjelmavarastoja sivustolle, tarjoaa Docker Hub mahdollisuudet siihen lisämaksusta.

Kun käyttäjä on luotu, tulee sinne luoda esimerkiksi yksityinen ohjelmavarasto, johon voidaan ryhtyä työntämään tuotantoa. Ennen kuin tämä on tehtävissä, tulee kirjautua Docker Hub -käyttäjälle myös palvelimelta komennolla:

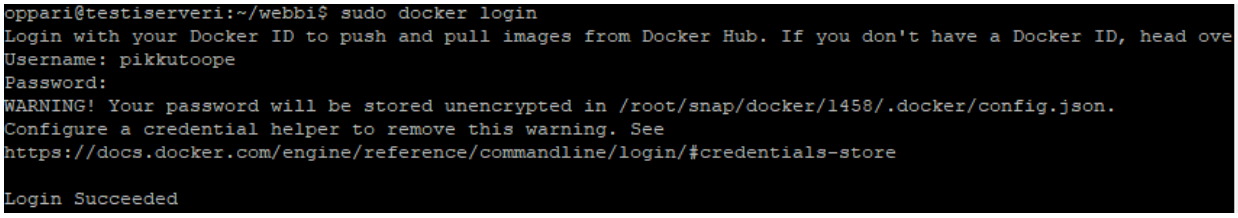
- Docker login

Tämän jälkeen ohjelma kysyy käyttäjänimen ja salasanan. Kun nämä on syötetty oikein, ilmoittaa Docker kirjautumisen onnistumisesta (Kuva 8, s. 17).

Kuva 7. Luotu yksityinen Docker Hub-ohjelmavarasto.



Kuva 8. Docker Hub -kirjautuminen komentoriviltä.



Nyt voidaan luoda uusi image käyttämällä samaa Docker Fileä, kuin aikaisemminkin. Ensiksi tulee kuitenkin nimetä image uuden ohjelmavaraston nimen mukaisesti, jotta sen voi Docker Hubiin työntää. Tässä tapauksessa se tapahtuu komennolla

- sudo docker build -t pikkutoope/testirepo333 .

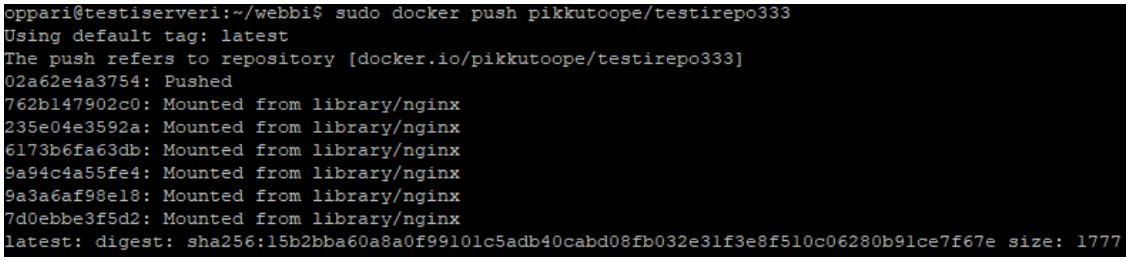
Tämä uusi image voidaan nyt työntää Docker Hubiin (Kuva 9). Se tapahtuu komennolla:

- sudo docker push pikkutoope/testirepo333

Ja tarvittaessa se voidaan sieltä kopioida. Tämä tapahtuu komennolla:

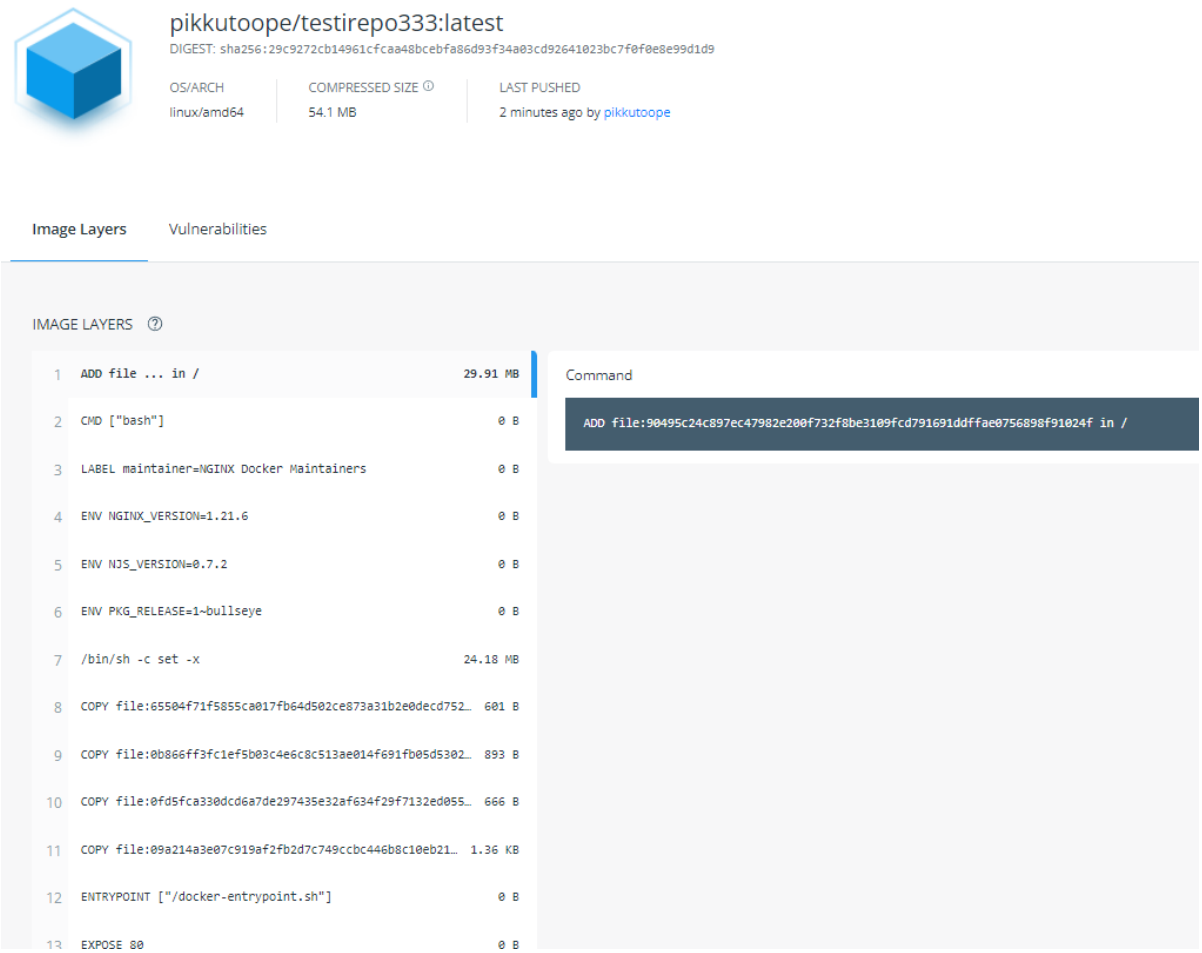
- sudo docker pull pikkutoope/testirepo333

Kuva 9. Uuden Docker-imagen työntäminen Docker Hub ohjelmavarastoon.



Nyt uusi image on työnnettyä ohjelmavarastoon, ja se on sieltä kaikkien oikeuksien omaavien henkilöiden kopioitavissa ja editoitavissa. He kykenevät myös työntämään uusia versioita imagesta, mikäli he ovat tehneet siihen muutoksia. Itse imagea tarkasteltaessa voidaan nähdä kaikki mitä se pitää sisällään (Kuva 10, s. 18). Käytännössä näkyvissä on kaikki ne rakennuspalikat, josta kyseinen image koostuu.

Kuva 10. Docker Hubiin työnnetyn imagen sisältö.

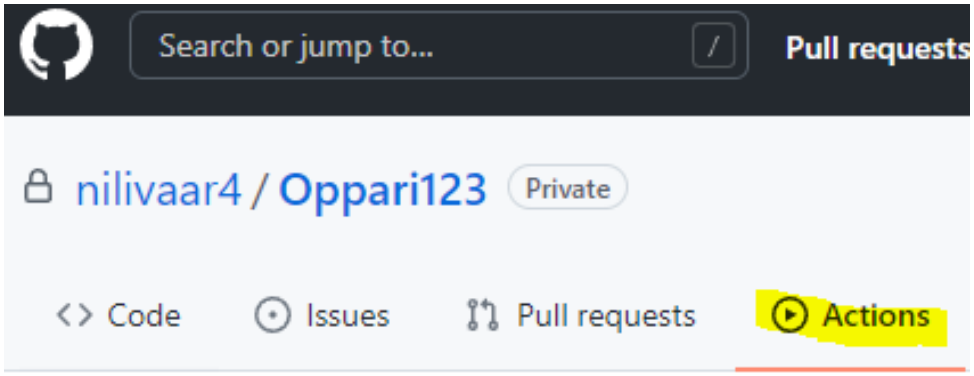


4.3 GitHub

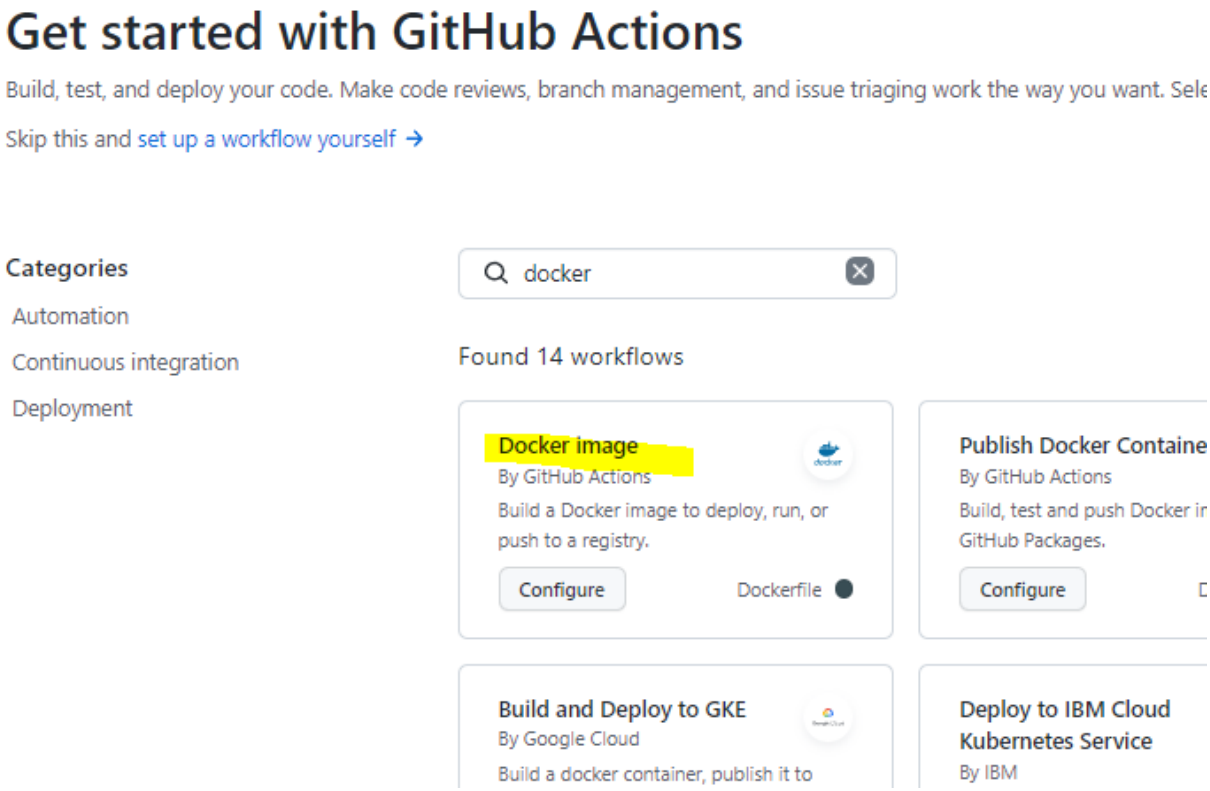
Myös GitHub -sivuston käyttö Dockerin kanssa on myös tarpeellista tuoda esiin. Etenkin sellaisessa yrityksessä tai organisaatiossa, jossa sovelluksia kehitetään GitHubia hyväksikäyttäen, kannattaa Docker Hub ja GitHub käyttäjät linkittää toisiinsa. Tämä mahdollistaa sen, että esimerkiksi sovelluksien lähdekoodia varastoidaan ja kehitetään GitHubissa, mutta niistä voidaan suoraan rakentaa imageja, jotka varastoidaan Docker Hubin puolelle.

Yksinkertaisuudessaan tämä tapahtuu GitHubissa siten, että ensin tulee luoda ohjelmavarasto, ja valita (Kuva 11, s. 19) paneeli ”Actions”. Sieltä tulee sitten etsiä Docker Image (Kuva 12. s. 19).

Kuva 11. Actions -välilehden valinta

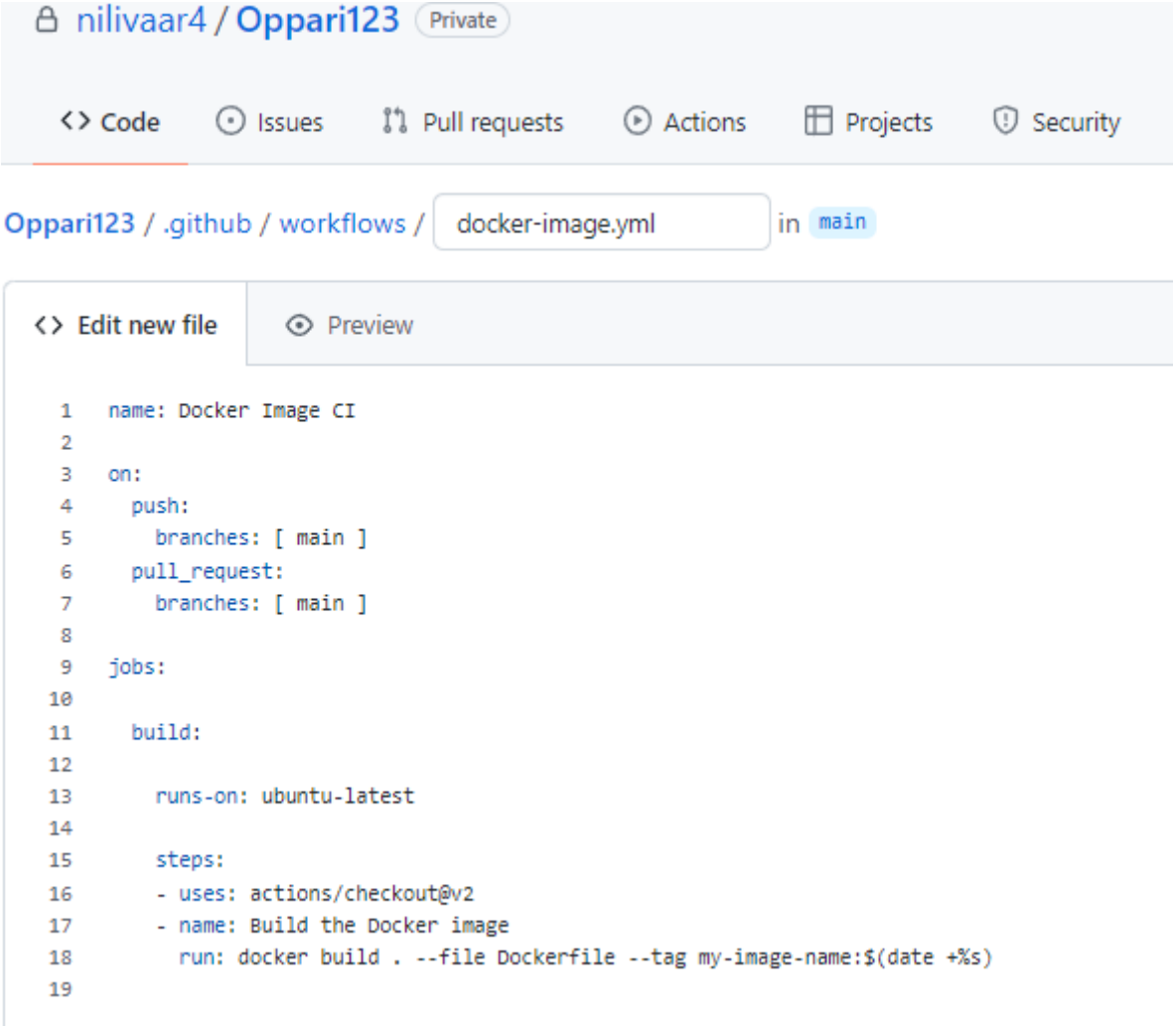


Kuva 12. Docker Imagen etsiminen GitHubista.



Valitsemalla ”Configure” päästään tilaan, jossa voidaan käytännössä rakentaa Docker Image käsin, jos näin halutaan (Kuva 13. s. 20). Tässä voidaan myös tarvittaessa konfiguroida tiedosto siten, että kun se julkaistaan valmiina, niin tämä sama image julkaistaan automaattisesti myös Docker Hubin puolella.

Kuva 13. Esimerkki Docker Imagen luomisesta GitHubissa.

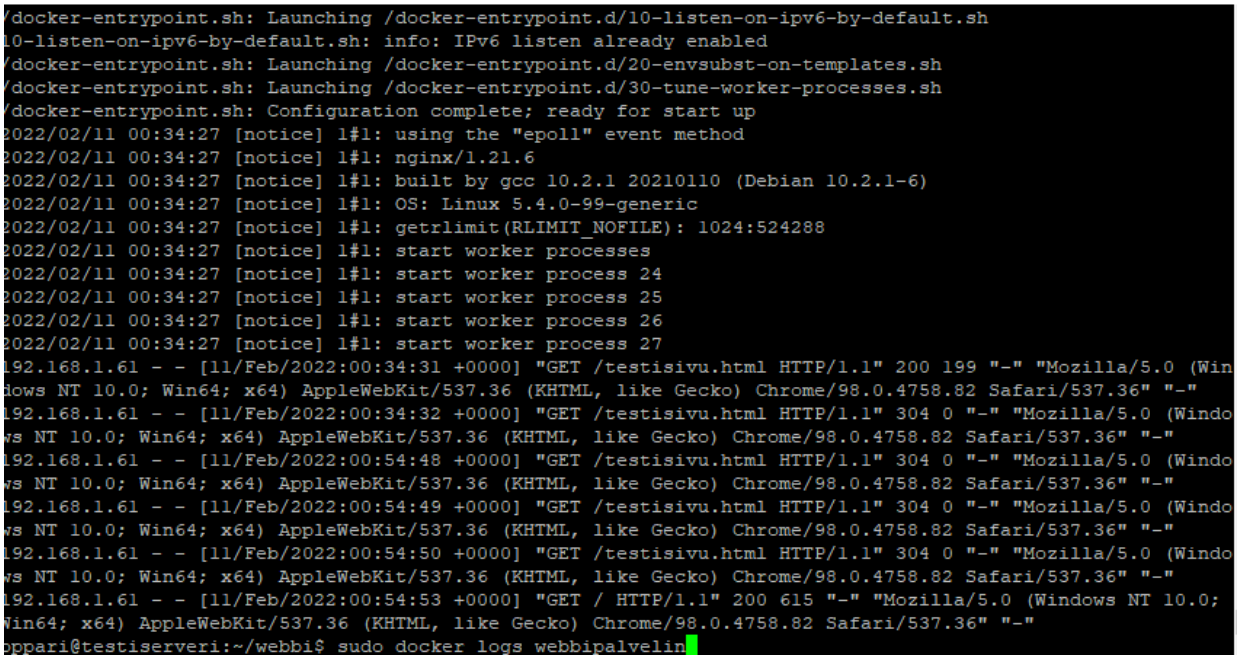


4.4 Muita tuotannon kannalta oleellisia komentoja

Etenkin tuotantoympäristössä on tärkeää kyetä seuraamaan, että missä tilassa kontit ovat, tai mikä on mahdollisesti aiheuttanut ongelmia. Etenkin sovelluksien lokien tarkastelu (Kuva 14, s. 21) on tärkeää ja tämä voidaankin suorittaa komennolla:

- sudo docker logs webbipalvelin

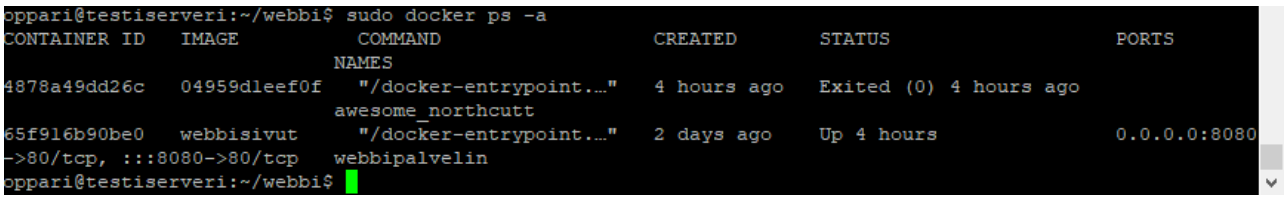
Kuva 14. Pätkä Dockerin sovelluksen lokitietoja.



Lokien tarkastelun lisäksi, on hyvä kyetä tarkastamaan kaikkien käytössä olevien konttien tila nopeasti (Kuva 15). Tämä voidaan tehdä esimerkiksi komennolla:

- sudo docker ps -a

Kuva 15. Listaus käytössä olevista konteista.



Docker-kontteja on myös mahdollista ylläpitää privileged-tilassa, jossa ne saavat oikeudet koko isäntänä toimimaan käyttöjärjestelmään. Käytännössä tämä tarkoittaa, että kontit saavat oikeudet käyttää sellaisia resursseja, joihin tavallisilla konteilla ei ole oikeuksia. Yksi yleinen tapaus, jossa näin toimitaan, on tilanne, missä Docker halutaan asentaa ja ylläpitää toisen Docker-kontin sisällä. Voimme tarkistaa, onko konttimme privileged-tilassa komennolla:

- docker inspect --format='{{.HostConfig.Privileged}}' “kontin nimi”

Tässä tapauksessa kontti ei ole kyseisessä tilassa. Se voidaan todeta syötteen (Kuva 16) sanasta “false”. Jos se olisi “true” olisi kontti privileged-tilassa. Kontti kuitenkin saadaan privileged tilaan, luomalla kontti “docker run” komennolla käyttämällä parametriä “privileged” (Kuva 17).

Kuva 16. Tarkistaminen, onko kontti privileged-tilassa.

```
oppari@testiserveri:~/webbi$ docker inspect --format='{{.HostConfig.Privileged}}' webbipalvelin
false
oppari@testiserveri:~/webbi$
```

Kuva 17. Kontin luominen privileged-tilassa.

```
oppari@testiserveri:~/webbi$ sudo docker run --privileged --name webbipalvelin -d -p 8080:80 webbisivu
t
5a87a76ae36cb6715f649c7e706c4d34d28404b596cf00abb34eb3a8dda32de2
oppari@testiserveri:~/webbi$ docker inspect --format='{{.HostConfig.Privileged}}' webbipalvelin
true
```

Konttien ylläpitäminen privileged-tilassa ei kuitenkaan ole suositeltavaa, mikäli se ei ole välttämätöntä. Se nimittäin tekee järjestelmän haavoittuvaiseksi mahdollisille hyökkäyksille. Jos konttiin pääsee kiinni jokin ulkopuolinen taho, olisi heillä nyt täydet oikeudet koko järjestelmään ja sen resursseihin.

Joskus voidaan kokea tarpeelliseksi tehdä kopio jo olemassa olevasta kontista, vaikkapa siksi, että kontti voitaisiin siirtää uudelle isännälle. Tämä käytännössä tapahtuu siten, että jo olemassa olevasta kontista tehdään uusi image (Kuva 18). Sitä voidaan sitten siirrellä vapaasti tai vaikka ylläpitää samanaikaisesti vanhan kontin kanssa, mutta eri portissa. Käytännössä tämä kontin kopiointi tapahtuu komennolla:

- sudo docker commit <kopioitavan kontin nimi> <uuden imagen nimi>

Kuva 18. Uuden imagen luominen kontista.

```
oppari@testiserveri:~/webbi$ sudo docker commit webbipalvelin kopioitupalvelin
sha256:d1acd2603c747a19b69666a9f07c496520d7528839abc716f942a2c86a0ce0c5
oppari@testiserveri:~/webbi$ sudo docker images
REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
kopioitupalvelin            latest    d1acd2603c74   18 seconds ago   142MB
<none>
webbisivut                  latest    04959d1eef0f   6 weeks ago      142MB
```

5 Johtopäätökset ja pohdinta

On vaikea todeta, että jokin konttiratkaisu olisi ylivertaisesti parempi kuin muut. Konttiratkaisut yleisesti ovat toiminnaltaan hyvin samankaltaisia, mutta sisältävät joitain pieniä eroavaisuuksia. Tehtyjen havaintojen perusteella voidaan todeta, että pienyrityksessä, jossa on tarkoitus siirtyä konttiympäristöön, kannattaa ehkä harkita juuri Docker-pohjaista ratkaisua ensimmäiseksi. Se on helppo ja nopea ottaa käyttöön hyvällä ohjeistuksella, ja on edellä mainittujen lisäksi myös todella kevyt.

Mitä imagejen varastoimiseen konttivarastoissa tulee, on Docker Hub todennäköisesti varmin vaihtoehto siihen ainakin alustavasti. Tätä työtä varten pohdittiin myös erityisesti Google Cloudin tarjoamaa Container Registryä, mutta sen dokumentaatioon tutustumisen jälkeen, tultiin siihen tulokseen, ettei sitä välttämättä kannata ottaa käyttöön pientä ympäristöä varten. Docker Hub tarjoaa huomattavasti helpommin lähestyttävämmän varaston, ja tekee Dockerista ylipäätään houkuttelevamman vaihtoehdon pienyrityksen tarpeisiin. Todettakoon kuitenkin myös se, että mikäli yrityksellä on jo Amazonin, Microsoftin, tai Googlen pilvipalveluympäristö tuotantokäytössä, on niiden käyttöönotto huomattavasti luonnollisempi ratkaisu.

Docker Hubin rinnalla myös GitHub on erittäin hyödyllinen työkalu. Tämä korostuu toki enemmän sellaisissa yrityksissä, jotka itse kehittävät sovelluksiansa jo valmiiksi esimerkiksi juuri GitHubin puolella. Sitä voidaankin käyttää, vaikka imagejen luomiseen ja päivittämiseen.

Jos yritys päätyy Docker-pohjaiseen ympäristöön, ja tarvitsee sen lisäksi myös orkestrointityökalun, on siihenkin helppo ratkaisu. Dockerin kanssa luonnostaan yhteensopiva Docker Swarm, on optimaalinen pienyrityksen tarpeisiin. Se on kevyt, helposti ymmärrettävä ja nopeasti käyttöönotettava. Sen asentaminen ei myöskään vaadi mitään ylimääräistä itse Dockerin lisäksi.

Periaatteessa myös Podman on hyvä vaihtoehto konttiratkaisuksi, mutta sopii todennäköisesti paremmin hieman suurempien yritysten käyttöön. Sen käyttämisestä saa myös enemmän irti, mikäli sitä käytettään yhdessä Kuberneteksen kanssa. Se ei myöskään toimi yhdessä Docker Swarmin kanssa.

Erityisesti sellaisessa yrityksessä, joka haluaa pitää tuotantoympäristön yksinkertaisena, on Docker houkuttelevampi vaihtoehto. Sen lisäksi, että se toimii luonnostaan Swarmin kanssa, se ei vaadi erillistä työkalua imagejen rakentamiseen kuten Podman.

Laajemman tuotantoympäristön tarpeisiin olisi todennäköisesti järkevintä valita Kubernetes. Se on vakiinnuttanut paikkansa useiden modernien yritysten tuotantoympäristöjen orkestrointityökaluna hyvistä syistä. Sen skaalautuvuus ja automaattinen ympäristön monitorointi ja ylläpito, ovat hyvin oleellisia ominaisuuksia nykypäivänä. Näiden lisäksi se on erittäin hyvin dokumentoitua, ja monet pilvipalveluntarjoajat ja käyttöjärjestelmät tukevat sitä. Esimerkiksi juuri Microsoftin, Googlen ja Amazonin kaikkien tarjoamat konttiympäristöt on luotu sellaisiksi, että Kuberneteksen käyttäminen niiden kanssa sujuisi helposti. Sen käyttöönotto kuitenkin tilanteessa, jossa edellä mainittujen yritysten palveluja ei hyödynnetä, ei ole läheskään yhtä yksinkertaista, kuin Docker Swarmin. Näinpä suosittelisin sen käyttämistä vain tilanteessa, jossa konttiympäristöstä halutaan helposti laajennettava ja automatisoitava. Vaihtoehtoisesti se sopisi tilanteeseen, jossa yrityksellä on jo käytössään, joko Microsoftin, Googlen ja Amazonin pilvipalvelut.

Lopuksi todettakoon, että konttiteknologiat ovat jatkuvassa kehityksessä, ja uusia teknologioita ilmestyy markkinoille jatkuvasti. Ne ovat kuitenkin vielä sen verran tuore teknologia, että on vaikea ennustaa, mitkä ratkaisut ovat valta-asemassa vuosien päästä. Todennäköistä on kuitenkin se, että Docker tulee olemaan merkittävimpien teknologioiden joukossa yhdessä Kuberneteksen kanssa vielä vuosien ajan. Molemmat näistä ovat erittäin hyvin tuettuja kaikkialla, ja tämä ei oletettavasti tule muuttumaan pian.

Lähteet

Brockmeier, J. (2017). *Introducing CRI-O 1.0.* https://www.redhat.com/en/blog/introducing-cri-o-10

Burns, B., Beda, J. & Hightower, K. (2018) *Kubernetes: Up and Running.* O'Reilly Media.

CRI-O. (n.d.) *What is CRI-O?* Haettu 10.3.2022 https://cri-o.io/

Docker, Inc. (n.d.). *Docker Oveview*. Haettu 11.02.2022. https://docs.docker.com/get-started/overview/

Gamela, A. & Figueiredo, R. (2021). *Podman vs Docker: What are the differences?* https://www.imaginarycloud.com/blog/podman-vs-docker/

IBM Cloud Education. (2021a). *Docker*. https://www.ibm.com/ae-en/cloud/learn/docker

IBM Cloud Education. (2021b). *Containers*. https://www.ibm.com/cloud/learn/containers

Morabito, R. (2017). Virtualization on Internet of things Edge Devices With Container Technologies: *A Performance Evaluation.* IEEE Access, (5), 8835–8840. https://doi.org/10.1109/ACCESS.2017.2704444

Oludele, A., Ogu, E,A., 'Shade, K. & Chinecherem, U. (2014). *On the Evolution of Virtualization and Cloud Computing: A Review.* Journal of Computer Sciences and Applications, *2*(3), 40–43. https://www.doi.org/10.12691/jcsa-2-3-1

Podman. (n.d.). *What is Podman? Simply put: alias docker=podman.* https://podman.io/whatis.html

Portnoy, M. (2016) *Virtualization Essentials.* John Wiley & Sons, Incorporated.

Red Hat. (2018a). *Understanding virtualization*. https://www.redhat.com/en/topics/virtualization

Red Hat. (2018b). *What is virtualization*?
https://www.redhat.com/en/topics/virtualization/what-is-virtualization

Red Hat. (2018c). *What's the difference between cloud and virtualization*?
https://www.redhat.com/en/topics/cloud-computing/cloud-vs-virtualization

Red Hat. (2020). *What is a hypervisor*?
https://www.redhat.com/en/topics/virtualization/what-is-a-hypervisor

Schenker, G. N. (2018). *Learn Docker – Fundamentals of Docker 18.x*. Packt Publishing Limited.

Soppelsa, F. & Kaewkasi, C. (2016). *Native Docker Clustering with Swarm*. Packt Publishing Limited.