SOVELLUSKONTTIEN HALLINNOINTI KUBERNETES-ALUSTALLA

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Visamäki, tietojenkäsittely

kevät, 2019

Joni Hannuksela

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutusohjelma
Visamäki

Tekijä Joni Hannuksela **Vuosi** 2019

Työn nimi Sovelluskonttien hallinnointi Kubernetes-alustalla

Työn ohjaaja/t Erkki Laine

TIIVISTELMÄ

Opinnäytetyössä perehdyttiin sovelluskontteihin Docker-ohjelmiston kautta sekä niiden hallinnointiin Kubernetes-alustalla. Teoriaosuudessa käsiteltiin myös tietokonevirtualisointi konseptina, joka auttaa sovelluskonttien ymmärtämisessä. Työn tavoitteena oli toteuttaa Kubernetes-ympäristö Docker-sovelluskonteille parhaiden käytäntöjen mukaan.

Työ pyrkii myös selvittämään Dockerin sekä Kuberneteksen toimintaperiaatteet ja kartoittamaan, mitä hyötyjä tekniikat tuovat palveluiden toteutuksessa sekä kuinka haastava prosessi ympäristön asentaminen ja käyttöönotto on ensikertalaiselle, jolla on hieman Linux-osaamista. Työn käytännön osuus painottui suurimmaksi osaksi palvelun sijasta itse tekniikoiden sisäistämiseen, tarjoten monikäyttöisen ja yleispätevän ratkaisun eri käyttötarkoituksiin. Käyttöjärjestelmänä toimi Linux-ytimeen pohjautuva Ubuntu Server-jakelu.

Käytännön toteutuksessa Kubernetes toimi yksittäisen koneen paikallisessa ratkaisussa. Lopputuloksena luotiin toimiva LAMP-ympäristö Dockeria ja Kubernetestä käyttäen. LAMP-ympäristössä sovelluskontteja hyödyntävä WWW-palvelinohjelmisto Apache yhdistettiin eri instanssissa olevaan MySQL-tietokantapalveluun Kuberneteksen avulla. Tämän lisäksi ympäristön PHP-ohjelmointikieli sallii dynaamisen web-sisällön näyttämisen. Työ toimii oppaana sovelluskonteista kiinnostuneille henkilöille sekä organisaatioille.

Avainsanat sovelluskontti, Kubernetes, Docker, virtualisointi

Sivut 44 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Visamäki

Author Joni Hannuksela **Year** 2019

Subject Managing application containers with Kubernetes

Supervisors Erkki Laine

ABSTRACT

The purpose of this thesis was to get familiar with application containers through the Docker software and to manage them via the Kubernetes platform. The theoretical part of this thesis also described computer virtualization as a concept, helping to understand application containers better. The goal was to implement a Kubernetes environment for Docker containers using best practices.

The thesis also aims to examine both Docker's and Kubernetes' operating principles, map out the benefits of these technologies when it comes to implementing services and find out how challenging the process of installing and deploying the environment is for a first timer with some Linux experience. The practical part of the thesis focused mostly on the technologies behind the environment instead of a specific service. By doing so, this thesis offers a versatile solution for multiple use cases. Operating system used on this solution was a Linux-based Ubuntu Server distribution.

In the practical implementation (part of the thesis) Kubernetes was hosted locally on a single virtual computer. The result was a working LAMP stack utilizing Docker and Kubernetes. In the LAMP environment, a web server software Apache connected to a MySQL database service located in a different instance by using Kubernetes. In addition, the PHP programming language included with the environment allows the display of dynamic web content. This thesis works as a guide for people and organizations interested in application containers.

Keywords application container, Kubernetes, Docker, virtualization

Pages 44 pages

SISÄLLYS

1 JOHDANTO

Tietokoneiden virtualisoinnin hyödyntäminen yrityksissä on ollut arkipäivää jo 2000-luvun alusta lähtien. Kuitenkin internetyhteyksien jatkuva kehittyminen, älylaitteiden räjähtänyt suosio sekä SaaS-tuotteiden (Software as a Service) yleistyminen on tuonut uusia ongelmia palveluiden skaalautuvuudessa sekä käyttöönotossa, joihin perinteiset virtualisointitekniikat eivät ole riittäviä.

Tämän vuoksi viimeisen muutaman vuoden aikana useat yritykset ovat perehtyneet vaihtoehtoisiin tekniikoihin ratkaistakseen edellä mainittuja ongelmia. Yksi näistä vaihtoehdoista on käyttöjärjestelmätason virtualisointi eli sovelluskontit. Sovelluskontit ovat eristettyjä ohjelmistopaketteja, jotka sisältävät ainoastaan palvelulle olennaiset osat toimiakseen.

Sovelluskonteilla voidaan eliminoida tarve kokonaisille virtuaalikoneille, jotka sisältävät palvelun lisäksi usein kokonaisen käyttöjärjestelmän, käyttäen huomattavasti enemmän resursseja. Kontit sallivat myös ohjelmistojen pyörittämisen usealla eri käyttöjärjestelmällä ilman yhteensopivuusongelmia.

Näiden sovelluskonttien hallinta isommassa skaalassa voi olla haastavaa, ja tämän vuoksi siihen on luotu useita työkaluja sekä alustoja. Nämä työkalut auttavat esimerkiksi sovelluskonttien automatisoidussa hallinnassa, konfiguroinnissa sekä käyttöönotossa. Yksi suosittu vaihtoehto on alun perin Googlen kehittämä Kubernetes, jota käytetään tässä opinnäytetyössä.

Opinnäytetyössä perehdytään sovelluskontteihin Docker-ohjelmiston kautta sekä hallinnoidaan näitä kontteja käyttäen Kubernetes-alustaa. Työn tavoitteena on toteuttaa Kubernetes-ympäristö Docker-sovelluskonteille parhaiden käytäntöjen mukaan. Tässä toteutuksessa Kubernetes tulee toimimaan yksittäisen koneen paikallisessa ratkaisussa, joka on hyvä vaihtoehto Kuberneteksen alkeiden opetteluun. Usean koneen klusterointia käsitellään teoriaosiossa silti kattavasti sen tärkeyden vuoksi.

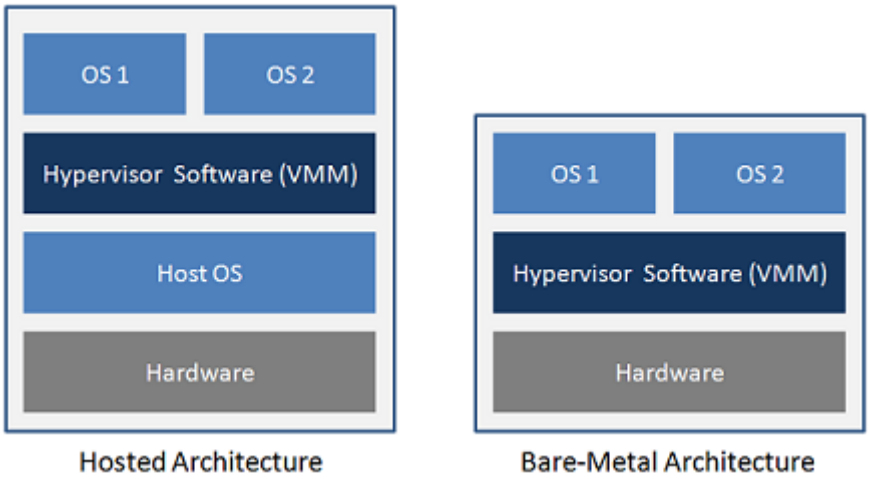
Työ pyrkii selvittämään Dockerin sekä Kuberneteksen toimintaperiaatteet, kartoittamaan mitä hyötyjä tekniikat tuovat palveluiden toteutuksessa sekä kuinka haastava prosessi ympäristön asentaminen ja käyttöönotto on ensikertalaiselle, jolla on hieman Linux-osaamista.

Opinnäytetyön käytännön osuus painottuu suurimmaksi osaksi palvelun sijasta itse tekniikoiden sisäistämiseen, tarjoten monikäyttöisen ja yleispätevän ratkaisun eri käyttötarkoituksiin. Käyttöjärjestelmänä toimii Linux-ytimeen pohjautuva Ubuntu Server-jakelu, joten Linuxin terminaalin sekä Debian-pohjaisen Linux-jakelun perusosaamisesta on hyötyä käytännön osuutta lukiessa.

2 VIRTUALISOINTI

Lyhyesti, virtualisointi terminä tarkoittaa palvelun tai resurssin erottamista sitä tarjoavasta fyysisestä tasosta. Yleisesti tietokonevirtualisoinnissa tämä tarkoittaa virtualisointitason lisäämistä tietokoneen käyttöjärjestelmän sekä fyysisen tietokonelaitteiston väliin. Tämä virtualisointitaso sallii useamman käyttöjärjestelmän yhtäaikaisen ajamisen samalla tietokoneella hyödyntäen virtuaalikoneita. (VMWare, 2007, s. 2)

Virtuaalikoneet ovat keskeisessä osassa virtualisointia. Ne ovat säiliöitä perinteiselle käyttöjärjestelmälle ja sovelluksille, jotka toimivat hypervisorin päällä fyysisellä palvelimella (Portnoy, 2016, s. 54). Hypervisor on ohjelma, joka asennetaan isäntäkoneelle; tästä tietokoneesta ja sen ohjaamista resursseista tulee silloin isompi kokonaisuus resursseja, joita voidaan jakaa uusien virtuaalikoneiden kanssa (Rathod & Townsend, 2014, s. 11).



Kuva 1. Tyypin 1. ja 2. hypervisor-ohjelmien vertailu. (National Instruments n.d.).

Saatavilla on erilaisia hypervisor-ohjelmia, joista yleinen on 1. tyypin hypervisor, josta käytetään myös termiä ”bare metal”. Nämä hypervisorit toimivat isäntäkäyttöjärjestelmänä fyysisellä palvelimella ja ohjaavat suoraan tietokoneen komponentteja sekä hallitsevat virtuaalikoneita. Hypervisor jakaa resurssit useammalle virtuaalikoneelle, jotka toimivat itsenäisesti mutta jakavat verkkoresursseja. (Rathod & Townsend, 2014, s. 11)

Toisena yleisenä ratkaisuna on 2. tyypin hypervisor eli ”hosted hypervisor”, joka toimii jonkin jo asennetun käyttöjärjestelmän päällä ohjelmana, kuten Windows Server tai Linux. Hypervisor kommunikoi isäntäkäyttöjärjestelmän kanssa, jakaen resursseja virtuaalikoneille tarpeen mukaan. (Rathod & Townsend, 2014, s. 12)

2.1 Täysvirtualisointi sekä muut virtualisointitavat

Täysvirtualisoinnissa x86-prosessoriarkkitehtuuriin pohjautuva käyttöjärjestelmä (esimerkiksi Windows) virtualisoidaan käyttämällä binäärikäännöksiä sekä suoria komentotekniikoita prosessorille. Tämä lähestymistapa kääntää käyttöjärjestelmän ytimen koodin korvatakseen virtualisointikelvottomat käskyt uudella sarjalla komentoja, joilla saadaan haluttu tulos virtuaalilaitteistolle. Samalla käyttäjätason koodi suoritetaan suoraan prosessorilla, jotta saavutetaan korkea suorituskyky virtualisoinnissa. (VMWare, 2007, s. 4)

Tämä yhdistelmä binäärikäännöksiä sekä suoria prosessorikomentoja muodostavat täysvirtualisoinnin, tehden isäntäkoneella toimivasta virtuaalikoneesta täysin irrallisen fyysiseen laitteistoon. Virtuaalikone ei ole tietoinen, että sitä virtualisoidaan eikä vaadi minkäänlaisia muutoksia. Hypervisor kääntää kaikki käyttöjärjestelmäkäskyt lennosta, ja tallentaa ne välimuistiin tulevaisuuden käyttöä varten. Täysvirtualisointi tarjoaa parhaan eristyksen ja turvallisuuden virtuaalikoneille ja selkeyttää migraatioita sekä siirrettävyyttä. (VMWare, 2007, s. 4)

Paravirtualisoinnilla eli käyttöjärjestelmän osittaisella virtualisoinnilla pyritään parantamaan hypervisorin ja virtuaalikoneen kommunikointia keskenään. Paravirtualisoinnissa kääntämisen sijaan muokataan käyttöjärjestelmän ydintä, korvaten virtualisointikelvottomat käskyt hypercall-kutsuilla. Hypercall-kutsut kommunikoivat suoraan virtualisointitason hypervisorin kanssa ja tarjoavat rajapinnan kriittisille käyttöjärjestelmän ydinoperaatioille kuten muistinhallinnalle sekä ajan pitämiselle. (VMWare, 2007, s. 5)

Paravirtualisoinnin etuna on useissa käyttötarkoituksissa parantunut suorituskyky, yhteensopivuusongelmien ehdoilla. Paravirtualisointimalli tarjoaa suorituskykyetuja, kun virtuaalikone tai sovellus on tietoinen sen virtuaaliympäristöstä ja sitä on mukautettu hyödyntämään tätä. Yksi mahdollinen haittapuoli tässä lähestymistavassa on migraation vaikeus verrattuna esimerkiksi täysvirtualisointiin. (VMWare, 2006, s. 8)

Edellä mainittujen perinteisten virtualisointitapojen lisäksi muita tekniikoita ovat esimerkiksi laitteistoavustettu virtualisointi. Laitevalmistajat ovat tukeneet kasvavaa virtualisointimarkkinaa kehittämällä uusia ominaisuuksia sekä yksinkertaistamalla virtualisointitekniikoita. Ensimmäisen sukupolven parannuksiin kuuluu prosessoripuolella Intelin VT-x sekä AMD:n AMD-V -tekniikat, jotka tukevat virtualisointia uusilla suoritustiloilla. (VMWare, 2007, s. 6)

Toisena isona esimerkkinä on sovellusvirtualisointi, jolla voidaan virtualisoida yksittäisiä ohjelmia tai ohjelmistokokonaisuuksia. Sovellusvirtualisointia hyödyntävä palvelu ei ole riippuvainen esimerkiksi tietystä käyttöjärjestelmästä. Tällä menetelmällä voidaan huomattavasti nopeuttaa ohjelmistojen käyttöönottoa organisaatiossa verrattuna kokonaisiin virtuaalikoneisiin tai manuaalisiin asennuksiin verrattuna. Sovellusvirtualisoinnilla voidaan myös välttää yhteensopivuusongelmia ja ohjelmien välisiä ristiriitoja. Tuoteratkaisuja tälle menetelmälle ovat esimerkiksi Microsoftin App-V ja VMWaren ThinApp. (Portnoy, 2016, s. 38)

2.2 **Virtualisoinnin edut**

Virtualisoinnista hyötyvät erityisesti jatkuvasti kasvavat tietoliikenteeseen sekä pilvipalveluihin kytköksissä olevat yritykset. Nykypäivänä melkein kaikki datakeskukset ovat jollain tavalla virtualisoituja; tämä tekee niistä kilpailukykyisempiä ja helpottaa pilvipohjaisten palveluiden hallinnointia. (Rathod & Townsend, 2014, s. 8)

Yksi merkittävistä virtualisoinnin eduista on alhaisemmat kustannukset; ajaakseen useita palvelimia yrityksen täytyy maksaa huomattavasti sähköstä sekä poistaa lämpöä. Tämän lisäksi kulut palvelimien asentamisessa, ylläpidossa sekä päivittämisessä voivat riistäytyä nopeasti käsistä. Virtualisointia hyödyntävät yritykset ovat vastuussa pienemmästä määrästä fyysisiä palvelimia. (Rathod & Townsend, 2014, s. 8)

Toisena suurena virtualisoinnin hyötynä on nopeampi käyttöönotto verrattuna fyysisiin palvelimiin. Resurssien määrittäminen perinteisessä datakeskuksessa voi viedä merkittävästi aikaa, sekä palvelinkapasiteetin uudelleenjakaminen saattaa koitua haastavaksi. Sen sijaan virtualisoidussa ympäristössä resurssien uudelleenjakaminen on usein vain muutaman klikkauksen päässä. (Rathod & Townsend, 2014, s. 9)

Käyttöönoton ja resurssien siirron hitauden lisäksi perinteiset palvelimet käyttävät saatavilla olevaa kapasiteettia varsin epätehokkaasti; Syyskuussa 2013 tehdyn tutkimuksen mukaan fyysiset palvelimet hyödyntävät usein vain alle kymmenen prosenttia saatavilla olevasta laskentatehosta. Tallennustilan hyödyntäminen on korkeampaa, mutta silti alle viisikymmentä prosenttia useissa tapauksissa. Virtualisoinnilla voidaan välttää tällaisia tilanteita jakamalla laitteistoa, ohjelmistoja sekä infrastruktuuria. (Rathod & Townsend, 2014, s. 10)

Virtualisointi yksinkertaistaa myös varmuuskopioiden ottoa sekä tilannevedosten (snapshot) luontia; esimerkiksi virtuaalikoneiden, tietokantojen sekä erilaisten palveluiden asetustiedostojen varmuuskopiointi on joustavampaa. Virtuaaliympäristö voidaan varmuuskopioida aina järjestelmätasolta ohjelmatasolle niin useasti kun on tarvetta. (Rathod & Townsend, 2014, s. 9)

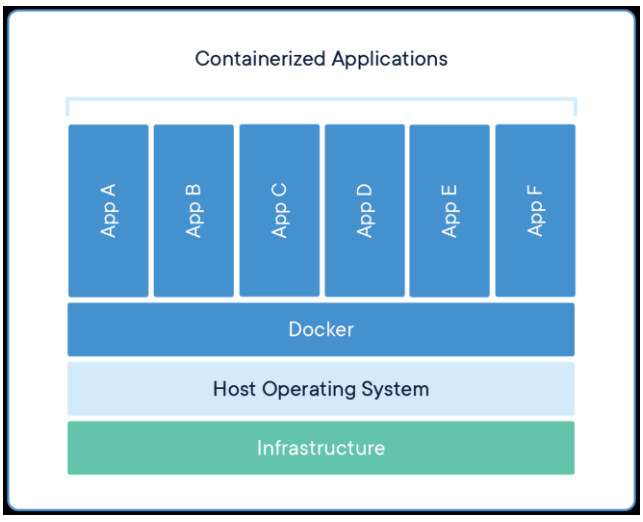
3 KÄYTTÖJÄRJESTELMÄTASON VIRTUALISOINTI – SOVELLUSKONTIT

Tässä opinnäytetyössä keskeisimmässä osassa virtualisointitekniikoista ovat sovelluskontit – eli käyttöjärjestelmätason virtualisointia hyödyntävät ratkaisut. Vaikka sovelluskontit eivät varsinaisesti ole perinteistä virtualisointia, auttavat aiemmin käsitellyt tekniikat ymmärtämään sovelluskonttien periaatteita. Työssä käsiteltävä Docker on tällä hetkellä ylivoimaisesti suosituin ohjelma sovelluskonttiratkaisuista (Carter, 2018).

Sovelluskontti on ohjelmisto, joka pakkaa halutun sovelluksen koodin sekä kaikki siihen tarvittavat osat, jotta sovellusta voidaan ajaa nopeasti ja luotettavasti eri käyttöjärjestelmäympäristöissä (Docker Inc., 2018). Yleisesti sovelluskontit pohjautuvat virtuaaliseen eristämiseen ja käyttävät isäntäkoneen omaa käyttöjärjestelmäydintä ilman tarvetta erillisille virtuaalikoneille. Sovelluskontit sallivat sovelluksen koodin eristämisen alla olevasta infrastruktuurista. (Rouse, 2018).

Sovelluskonttien levykuvat (container image) sisältävät tiedot sovelluksesta, joka suoritetaan niin kutsutun konttimoottorin (container engine) avulla. Kontitetut sovellukset voivat koostua useasta levykuvasta. Esimerkiksi kolmetasoinen sovellus voi sisältää etupään web-palvelimen, sovelluspalvelimen ja tietokantakontin, joista jokainen toimii itsenäisesti. (Rouse, 2018).

Alla olevassa kuvassa 2. esitetään visuaalisessa muodossa sovelluskontteja hyödyntävä ympäristö. Alimpana on infrastruktuuri, joka voisi olla esimerkiksi fyysinen palvelinkone. Toisella tasolla on isäntäkäyttöjärjestelmä; tämä voi olla kokonainen käyttöjärjestelmä tai hypervisor-ohjelma. Käyttöjärjestelmätason päällä sijaitsee konttimoottori, jolla suoritetaan halutut sovelluskontit. Viimeisenä tasona ovat itse sovelluskontit, jotka on eristetty muusta ympäristöstä konttimoottorin avulla.



Kuva 2. Visualisointi sovelluskontin sekä kontteja suorittavan Docker-ohjelman sijainnista ympäristössä. (Docker Inc. n.d.).

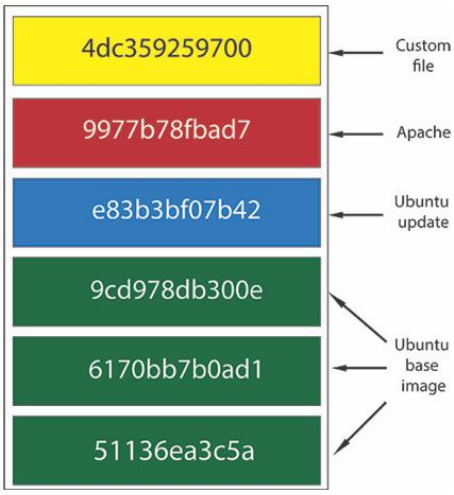
Sovelluskontti sisältää kaikki tarvittavat osat sovelluksen suorittamiseen, kuten tiedostot, ympäristömuuttujat, riippuvuudet sekä ohjelmistokirjastot. Isäntäkäyttöjärjestelmällä voidaan myös rajoittaa sovelluskontin pääsyä fyysisiin resursseihin kuten suorittimeen, tallennustilaan sekä keskusmuistiin. Tällä tavoin yksittäinen sovelluskontti ei vie kaikkia resursseja isäntäkoneelta. (Rouse, 2018).

3.1 Sovelluskontin levykuva

Sovelluskontti käynnistetään suorittamalla levykuva. Levykuva sisältää kaikki tarvittavat osat sovelluskontin toimimiseen (Docker Inc. 2019). Levykuvat (container image) ovat staattisia versioita sovelluksesta tai palvelusta, jotka eroavat riippuen tekniikasta. Docker-levykuvat on tehty useasta tasosta, jotka alkavat niin kutsutusta pohjakuvasta (base image). Pohjakuva sisältää kaikki riippuvuudet, joita tarvitaan koodin suorittamiseen sovelluskontissa. (Rouse, 2018).

Jokaisessa levykuvassa on luettava sekä kirjoitettava taso staattisten, eimuuttuvien tasojen päällä. Koska jokaisella sovelluskontilla on oma konttitaso, alla olevat konttitasot voidaan tallentaa ja käyttää uudelleen useassa sovelluskontissa. Konttimoottori suorittaa nämä levykuvat, luoden toimivan sovelluskontin. (Rouse, 2018).

Alla olevassa kuvassa 3. on esimerkki Linux-pohjaisesta, Ubuntu-jakelulla luodusta Docker-levykuvasta. Tämä levykuva koostuu yhteensä kuudesta tasosta; alimmalla tasolla pohjakuvana toimii Ubuntun omat levykuvat, jotka määrittävät yleisesti rakentamiseen käytetyn käyttöjärjestelmän.



Kuva 3. Esimerkki Docker-levykuvan eri tasoista. (InfoWorld 2016).

Seuraavalla tasolla ovat mahdolliset Ubuntun levykuvaan tulevat päivitykset, jonka jälkeen lisätään Apache-webpalvelin. Viimeisenä tasona on käyttäjän omat muokkaukset, joissa voidaan tehdä lisäyksiä lopulliseen levykuvaan. Tämä voisi esimerkiksi olla Apachea hyödyntävän palvelun lisääminen, sallien kokonaisen palvelun kontittamisen.

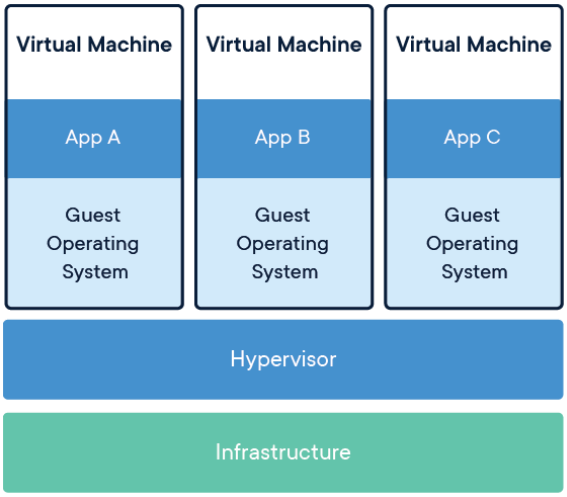
Näissä prosesseissa usein organisaatiot käyttävät apuna sovelluskonttien järjestelijäohjelmia sekä muita hallinnointialustoja, kuten tässä työssä myöhemmin käsiteltävää Kubernetes-alustaa. Sovelluskontit ovat helposti siirrettäviä, koska jokainen levykuva sisältää kaiken tarvittavan sovelluksen suorittamiseen (Rouse, 2018).

Esimerkkinä voisi toimia verkkopelipalvelinsovellus, jonka levykuva on luotu Linux-pohjaista käyttöjärjestelmää hyödyntäen. Tämä levykuva voidaan suorittaa konttimoottorin avulla Windows-pohjaisella käyttöjärjestelmällä ilman muutoksia itse sovellukseen.

3.2 Sovelluskontit verrattuna virtuaalikoneisiin

Sovelluskonteilla ja virtuaalikoneilla on samankaltaisia resurssien eristämis- ja allokointiominaisuuksia, mutta toimivat eri tavalla koska sovelluskontit virtualisoivat käyttöjärjestelmän laitteiston sijaan. (Docker Inc. n.d.).

Sovelluskontit vievät virtuaalikoneisiin verrattuna huomattavasti vähemmän tallennustilaa, voivat käsitellä useampia sovelluksia yhtäaikaisesti ja vaativat suurempien palveluiden toteuttamiseen vähemmän käyttöjärjestelmäasennuksia (Docker Inc. n.d.). Tämä mahdollistaa useamman palvelun ajamisen samalla fyysisellä palvelimella, laskien kustannuksia (Microsoft, 2018). Alla olevassa kuvassa 4. ilmenee yleinen ongelma perinteisessä virtualisoinnissa.



Kuva 4. Perinteisen virtualisointiympäristön ongelmana ovat hukatut resurssit; usein yksittäiset virtuaalikoneet suorittavat vain yhtä palvelua. (Docker Inc. n.d.).

Virtuaalikoneita voidaan käyttää myös yhdessä sovelluskonttien kanssa; näin voidaan yhdistää molempien tekniikoiden hyvät puolet, tarjoten joustavuutta ja käyttöönoton nopeuttamista. (Docker Inc. n.d.).

3.3 **Sovelluskonttien hyödyt ja haitat**

Koska sovelluskontit jakavat saman käyttöjärjestelmäytimen isäntäjärjestelmän kanssa, kontit voivat olla tehokkaampia kuin virtuaalikoneet, jotka vaativat erillisen käyttöjärjestelmäasennuksen. Kuten aiemmin mainittu, sovelluskontteja voidaan myös liikuttaa järjestelmien välillä, joissa on sama käyttöjärjestelmäydin. Tämän sovelluksen koodin kapseloinnin vuoksi ei ole tarvetta käsitellä konttia suorittavan käyttöjärjestelmän ympäristömuuttujia tai kirjastoriippuvuuksia. (Rouse, 2018).

Tällä ratkaistaan yleinen ”toimii minun koneellani” -ongelma, jonka kehittäjät usein kohtaavat siirtäessään koodiaan uusiin ympäristöihin; esimerkkinä koodin tuominen tuotantoympäristöön, jossa saattaa olla erilaisia yhteensopivuus- sekä ohjelmistoriippuvuusongelmia. (Docker Inc. 2017).

Sovelluskontit tuovat etuja resurssienkäytössä; keskusmuistin, suoritinkäytön sekä tallennuskapasiteetin tehokkaampi hyödyntäminen on yksi avainominaisuuksista verrattuna perinteisiin virtualisointiratkaisuihin. Koska sovelluskontit eivät tarvitse virtuaalikoneiden käyttämiä lisäresursseja, kuten erillisiä käyttöjärjestelmäasennuksia, on mahdollista tukea useampia sovelluskontteja samassa ympäristössä (Docker Inc. 2017).

Mahdollinen haittapuoli sovelluskonteissa on niiden eristyksen puute isäntäkäyttöjärjestelmästä, tämän vuoksi tietoturvauhilla on helpompi pääsy koko järjestelmään verrattuna hypervisor-pohjaiseen virtualisointiin. Yksi lähestymistapa tämän riskin minimointiin on suorittaa sovelluskontteja virtuaalikoneessa. Tämä tapa varmistaa tietoturvamurron sattuessa konttitasolla, että hyökkääjällä on pääsy ainoastaan virtuaalikoneen käyttöjärjestelmään. (Rouse, 2018).

Toinen haittapuoli sovelluskontituksessa on käyttöjärjestelmän joustamattomuus. Tyypillisissä käyttöönotoissa jokaisen sovelluskontin tulee käyttää samaa käyttöjärjestelmäydintä sen pohjakuvassa, kun taas hypervisor-instansseilla on enemmän joustavuutta. Esimerkiksi Linux-pohjaisella isäntäjärjestelmällä luodulla levykuvalla ei voida suorittaa Windows-pohjaisia instansseja tai sovelluksia. Myös näkyvyyden monitorointi saattaa tuottaa ongelmia. Mahdollisesti satojen sovelluskonttien toimiessa palvelimella, voi olla vaikeaa seurata mitä tapahtuu jokaisessa sovelluskontissa. (Rouse, 2018).

Useat teknologiat konttitekniikoiden luojilta sekä muilta tahoilta, avoimen lähdekoodin ratkaisujen lisäksi, auttavat sovelluskonttien operointihaasteissa. Näihin ratkaisuihin kuuluu esimerkiksi turvallisuuden seurantajärjestelmiä, lokitietoja hyödykseen käyttäviä monitorointijärjestelmiä sekä järjestelijäohjelmia, jotka seuraavat alustan toimivuutta. (Rouse, 2018)

4 DOCKER

Docker on avoimen lähdekoodin projekti, joka pyrkii automatisoimaan sovellusten käyttöönottoa; tehden sovelluksista siirrettäviä, ympäristöriippumattomia sovelluskontteja, joita voidaan suorittaa pilvessä sekä paikallisilla palvelimilla. Docker on myös yritys, joka tukee ja kehittää tätä teknologiaa yhdessä eri pilvipalvelu-, Linux- sekä Windows-toimittajien kanssa. (Microsoft 2018).

Docker-levykuvat toimivat natiivisti Linuxilla sekä Windows-alustoilla, mutta Windows-levykuvia voi suorittaa ainoastaan Windows-isäntäjärjestelmissä. Linux-levykuvia voi suorittaa molemmissa ympäristöissä, Windowsilla käyttäen hyödyksi virtualisointia. (Microsoft 2018).

Sovellusten kontitus ei ole ideana kovinkaan tuore; jo 1970-luvulla Unix-käyttöjärjestelmä hyödynsi *chroot*-komentoa. Chroot-komennolla voidaan käynnistää ohjelma siten, että kyseinen ohjelma näkee jonkin alihakemiston juurihakemistonaan (Linux.fi n. d.).

Linux-kontit, johon Docker alun perin perustui, esiteltiin jo vuonna 2008. Docker julkaistiin avoimen lähdekoodin projektina dotCloud-alustayrityksen toimesta vuonna 2013. Dockerin markkinoinnin varapääjohtajan David Messinan mukaan ”Ennen Dockeria, sovelluksen tai palvelun siirrettävyys ei ollut koskaan varmaa.” Pian Dockerin julkaisemisen jälkeen, useat kehittäjät huomasivat tämän uuden lähestymistavan edut, ja kuinka se voisi ratkaista pitkään vaivanneita ongelmia palveluiden käyttöönotossa ja siirrettävyydessä. (Martin, 2015).

Elokuussa 2013, kuukausi Dockerille julkaistun interaktiivisen kurssin jälkeen sitä oli kokeillut Dockerin mukaan noin 10 000 kehittäjää. Vuoden päästä suuret yritykset kuten Amazon ja Red Hat olivat lisänneet kaupallisen tuen Dockerille, joskin Dockerin johtohenkilöt varoittivat Dockerin käytöstä tuotannossa tässä vaiheessa. Kesäkuussa 2014 version 1.0 julkaisuun mennessä Dockeria oltiin ladattu 2,75 miljoonaa kertaa. (Martin, 2015).

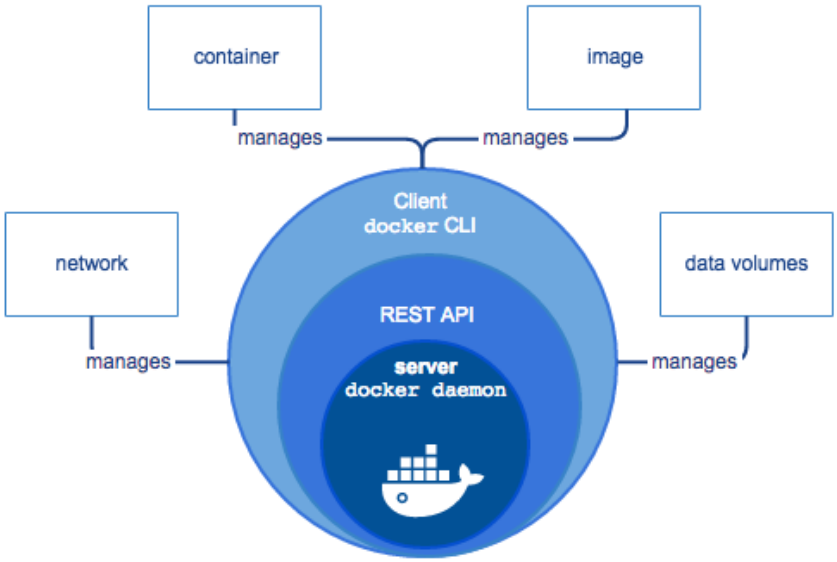
Dockerin suosio on jatkanut kasvamistaan, ja esimerkiksi vuoden 2018 raportissa konttihallintaratkaisuihin erikoistuvan Sysdigin mukaan, 83 prosenttia heidän asiakkaistansa käyttää sovelluskonttiratkaisunaan Dockeria (Carter, 2018). Pilvipalveluiden monitorointityökaluihin erikoistuvan Datadogin mukaan lähes neljännes heidän asiakkaistaan on hyödyntänyt Dockeria palveluissaan (Datadog, 2018). Valtaosa suurista yrityksistä hyödyntää jollain tasolla Dockeria; esimerkkeinä Twitter, Spotify ja Starbucks (Stackshare, 2018).

4.1 Docker Engine

Docker koostuu kahdesta osasta; Docker Enginestä – joka on sovelluskontituksen mahdollistava teknologia sekä Docker Hubista, joka on Docker Inc:n SaaS (Software as a Service) -palvelu konttien hallintaan ja jakamiseen muiden käyttäjien kanssa. Yleensä kun käyttäjät puhuvat Dockerista, tarkoitetaan sillä ainoastaan Docker Engineä (Docker Inc. 2019a). Docker Engine mahdollistaa konttien rakentamiseen, toimittamiseen ja suorittamiseen liittyvät tehtävät sekä työnkulun.

Docker Engine on asiakas-palvelinohjelmisto, joka koostuu kolmesta suuresta komponentista, jotka on visualisoitu alla olevassa kuvassa:

- *dockerd*-palvelinohjelmasta, joka toimii taustaprosessina käyttöjärjestelmässä ja on vastuussa Docker-objektien luonnista, joita ovat esimerkiksi levykuvat, sovelluskontit sekä verkot.
- REST-arkkitehtuurimalliin pohjautuvasta, eli HTTP-protokollaan perustuvasta sovellusrajapinnasta, joka määrittää käyttäjäliittymät joita ohjelmat voivat käyttää kommunikoidakseen taustaprosessin kanssa
- Komentorivikäyttöliittymästä, jota ohjataan *docker*-komennolla. Docker-asiakasohjelma on yleisin tapa useimmille Dockerin käyttäjille ohjata ja toimia Docker-ympäristössä (Docker Inc. 2019a).



Kuva 5. Docker Enginen rakenne visualisoituna, sisältäen sen kolme tärkeintä komponenttia. (Docker Inc. 2019a).

Komentorivikäyttöliittymä käyttää Dockerin REST-rajapintaa ohjatakseen ja toimiakseen *dockerd*-taustaprosessin kanssa skriptauksen sekä suorien komentojen kautta. Myös monet muut Docker-sovellukset käyttävät tätä rajapintaa ja komentorivikäyttöliittymää. (Docker Inc. 2019a).

4.2 Dockerin asennus ja sovelluskontin ajaminen

Dockerista on saatavilla kaksi eri versiota – Docker Community Edition (Docker CE) sekä Docker Enterprise. Docker CE on suositeltu versio kehittäjille sekä pienille tiimeille, jotka ovat aloittamassa sovelluskonttien kanssa työskentelyä. Docker Enterprise taas tarjoaa yrityksille pidempiaikaisen tuen sekä erilaisia tietoturvaominaisuuksia (Docker Inc. 2019f).

Tässä opinnäytetyössä käyttöjärjestelmäympäristönä toimii Linux-pohjainen Ubuntu Server, joten asennusvaiheet perustuvat tähän käyttöjärjestelmään. Docker Community Edition on saatavilla työpöytäversiona macOS:lle ja Windowsille, sekä virallisesti tuettuja palvelinkäyttöjärjestelmiä ovat CentOS, Debian, Fedora sekä Ubuntu (Docker Inc. 2019f).

Docker CE täyttää työn tavoitteet, eikä Enterprise-version ominaisuuksille ole tarvetta tässä toteutuksessa. Käytetyt komennot ovat varsin yleispäteviä kaikkiin Debian-pohjaisiin Linux-jakeluihin, joten asennusprosessia voi soveltaa halutessaan muihin Debian-pohjaisiin jakeluihin.

Dockerin voi asentaa usealla eri tavalla; useimmat käyttäjät hyödyntävät Dockerin tietovarastoa (repository) sen helppouden sekä päivitettävyyden vuoksi, joka on myös suositeltu tapa. Toinen vaihtoehto on asentaa sovelluspaketti manuaalisesti, mutta tässä tapauksessa myös sovelluspäivitykset joudutaan tekemään käsin. (Docker Inc. 2019f). Tässä opinnäytetyössä käytetään ensimmäistä eli tietovarastoa käyttävää asennustapaa.

Jotta Dockerin käyttämää HTTPS-tietovarastoa voidaan käyttää apt-pakетinhallinnassa, tulee suorittaa seuraavat komennot, asentaen useamman ohjelmapaketin:

```
sudo apt-get install \
apt-transport-https \
ca-certificates \
curl \
gnupg-agent \
software-properties-common
```

Seuraavaksi lisätään Dockerin virallisen tietovaraston sovelluspakettien tietoturvan lisäämisessä ja salauksessa hyödynnetty GPG-kryptausavain komennolla:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
```

Nyt Dockerin tietovarasto voidaan lisätä Ubuntun tietovarastolistaan alla olevalla komennolla. Tämä komento määrittää apt:n hakemaan uusimman vakaan version Dockerista.

sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"

Lopuksi päivitetään apt-paketinhallinnan listaus komennolla *sudo apt-get update* ja asennetaan viimeisin versio Docker CE:stä sekä containerd-taustaprosessista komennolla *sudo apt-get install docker-ce docker-ce-cli containerd.io.*

Sovelluskontti käynnistetään Dockerissa suorittamalla levykuva. Sovelluskontti on ajoaikainstanssi levykuvasta – se mitä levykuvasta tulee tietokoneen muistiin suorittaessa, kuten käyttäjäprosessi. Dockerin sovelluskontit toimivat natiivisti Linux-pohjaisilla käyttöjärjestelmillä ja jakavat isäntäjärjestelmän ytimen muiden sovelluskonttien kanssa. (Docker Inc. 2019b).

Sovelluskontit käynnistetään levykuvasta Dockerilla komennolla *sudo docker run.* Komento oletusarvoisesti etsii levykuvaa ensin paikalliselta koneelta, jonka jälkeen haluttua pakettia etsitään Docker Hub-konttirekisteristä. (Docker Inc. 2019b).

Toimivan Docker-asennuksen voi helposti todeta käyttämällä *hello-world* -levykuvaa. Kyseinen levykuva suoritetaan käyttämällä komentoa *sudo docker run hello-world*. Tämä komento lataa testilevykuvan Docker Hub-palvelusta ja suorittaa sen sovelluskontissa. Kun sovelluskontti käynnistyy, se tulostaa alla olevan kuvan 6. mukaisen viestin onnistuneesta asennuksesta ja sulkeutuu.

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:2557e3c07ed1e38f26e389462d03ed943586f744621577a99efb77324b0fe535
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Kuva 6. Onnistunut Docker-asennus Ubuntu-ympäristössä todettiin käyttäen *hello-world* -levykuvaa. Docker haki *hello-world* -levykuvan käyttäen Docker Hub -konttirekisteriä.

Tietokoneelle ladatut levykuvat voidaan listata komennolla *sudo docker image ls.* Komento listaa esimerkiksi äskettäin ajetun *hello-world* -levykuvan. Komennolla *sudo docker container ls --all* voidaan listata kaikki aiemmin suoritetut sovelluskontit, myös aiemmin sulkeutunut *hello-world* alla olevassa kuvassa 7.

```
CONTAINER ID        IMAGE               COMMAND
1bc67caa6ca0        hello-world         "/hello"
```

Kuva 7. *container ls* -komennolla voidaan listata parhaillaan käynnissä olevia sovelluskontteja, sekä *--all* -parametrillä myös jo suljettuja kontteja, kuten aiemmin suoritettu *hello-world*.

4.3 **Rekisteri ja Docker Hub**

Dockerin rekisteriominaisuus on laajasti skaalautuva palvelinsovellus, jolla voidaan tallettaa sekä jakaa Docker-levykuvia. Kuten muut Dockerin osa-alueet, myös rekisteri on avointa lähdekoodia. Erillisen rekisterin luominen on suositeltavaa, jos käyttäjä tahtoo tarkan hallittavuuden levykuvien säilömisestä omilla palvelimillaan, omistaa täysin jaettujen levykuvien toteutukseen käytetyn alustan tai integroida levykuvien talletuksen ja jakamisen organisaation kehitystyönkulkuun. (Docker Inc. 2019c).

Toisena vaihtoehtona käyttäjille on Docker Hub, joka on Docker Inc:n toteuttama konttirekisteri. Docker Hub on maailman suurin tietovarasto (repository) sovelluskonttien levykuville. Se sisältää vapaasti ladattavia levykuvia useisiin eri käyttötarkoituksiin aina suuryrityksiltä alan harrastajiin. Myös yksityisten tietovarastojen luominen on mahdollista tilausmuotoisen maksun kautta. Normaali käyttäjätunnus voi luoda ilmaiseksi Docker Hubiin yhden yksityisen tietovaraston. (Docker Inc. 2019d).

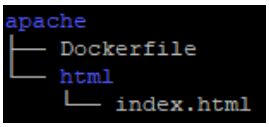
Docker Hubin etuna on sen yksinkertaisuus; käyttäjä välttyy ylläpidollisilta velvoitteilta saaden valmiin, ilmaisen sekä laajavalikoimaisen konttirekisterin. Docker Hub tarjoaa käyttäjille helppokäyttöisyyttä yksityisyyden ja oman infrastruktuurin integroinnin ollessa oman rekisterin hyötypuolia. Kuitenkin Docker Hubin laajuus sekä lisäominaisuudet ovat useimmille käyttäjille hyvä vaihtoehto.

4.4 **Dockerfile**

Eri konttirekistereistä sekä muista Internet-lähteistä saatavat valmiit levykuvat riittävät useille käyttäjille toteuttamaan halutun palvelun tai toteutusympäristön. Kuitenkin on paljon tilanteita, joissa kustomoitu levykuva voi tulla tarpeeseen. Esimerkiksi jos organisaatiolla on verkkopalvelu, jota tahdotaan skaalata useamman palvelimen ratkaisuille, itse luotu levykuva sallii huomattavasti nopeamman käyttöönoton.

Uuden levykuvan muodostaminen aloitetaan Dockerfile-tiedoston luomisella. Dockerfile määrittää, mitä sovelluskontin sisällä oleva ympäristö tulee sisältämään. Dockerfile-tiedostoa käytetään lopullisen levykuvan rakentamiseen, jonka avulla sovelluskontti voidaan suorittaa. Tässä ympäristössä pääsy resursseihin kuten verkkoadaptereihin ja kiintolevyihin on virtualisoitu sekä eristetty muusta järjestelmästä. Tämän vuoksi joudutaan myös kartoittamaan käytettävät portit, joilla on pääsy eristetyn ympäristön ulkopuolelle. (Docker Inc. 2019b).

Esimerkkinä opinnäytetyössä yksinkertaisesta Dockerfile-tiedostoa hyödyntävästä levykuvasta toimii Apache-webpalvelin, jolle kopioidaan yksi HTML-tiedosto. Aloitetaan luomalla kansio sovelluskontin tiedostoille, alikansio ”html” ja siirtämällä sinne jokin HTML-tiedosto. Tämän lisäksi luodaan alustavasti tyhjä Dockerfile-tiedosto. Kansiorakenne näyttää alla olevan kuvan 8. mukaiselta. Tämän jälkeen voidaan lisätä Dockerfile-tiedostoon tarvittavat muuttujat levykuvan muodostamiseksi.



Kuva 8. Kansiorakenne Apache-webpalvelimen levykuvan luomiseen

Kaikki Docker-levykuvat perustuvat aina johonkin pohjakuvaan, jonka päälle oma levykuva rakennetaan. Tämä ilmaistaan Dockerfile-tiedoston alussa ympäristömuuttujalla FROM. Apache-webpalvelimen sekä useimpien Ubuntu-käyttöjärjestelmällä toimivien ohjelmien tapauksessa pohjakuvana toimii Ubuntu. Myös pohjakuvat haetaan oletusarvoisesti Docker Hub-konttirekisteristä.

```
FROM ubuntu:latest

RUN apt update
RUN apt -y upgrade

RUN DEBIAN_FRONTEND=noninteractive
RUN apt -y install apache2

COPY html /var/www/html

EXPOSE 80
CMD /usr/sbin/apache2ctl -D FOREGROUND
```

Kuva 9. Dockerfile-tiedostoon lisätyt ympäristömuuttujat Apache-webpalvelimen levykuvan luomista varten.

Yllä olevassa kuvassa 9 nähdään esimerkkilevykuvassa käytetyt ympäristömuuttujat. RUN-muuttujalla suoritetaan komentoja levykuvan rakennusvaiheessa. COPY-muuttujalla voidaan kopioida tiedostoja isäntäjärjestelmästä sovelluskontin eristettyyn ympäristöön. EXPOSE-muuttuja määrittää, mikä portti halutaan avata sovelluskontista ulkomaailmaan. CMD-muuttujalla voidaan määrittää komento, jonka sovelluskontti suorittaa käynnistyessään.

Esimerkkilevykuvassa aluksi päivitettiin apt-paketinhallinnan ohjelmalistaukset Ubuntulle, jonka jälkeen asennetaan mahdolliset päivitykset käyttöjärjestelmälle komennolla *apt -y upgrade*. Y-parametrilla vastataan päivitysvaiheessa tuleviin kysymyksiin automaattisesti hyväksyvästi, ohittaen käyttäjältä pyydetyt syötteet.

Seuraavaksi suoritettu komento DEBIAN_FRONTEND kertoo Dockerille rakennusvaiheessa, ettei käyttäjältä pyydetä syötettä. Näin voidaan automatisoida esimerkiksi pakettien asennusvaiheet sekä muita konfigurointeja. Tämän jälkeen asennettiin Apache-webpalvelin komennolla *apt -y install apache2*. Aiemmin luotu HTML-kansio saadaan kopioitua sovelluskontin eristettyyn ympäristöön komennolla COPY html /var/www/html; joista ensimmäinen on kansion nimi ja jälkimmäinen Apachen oletuspolku HTML-sisällölle.

EXPOSE-muuttujalla määritettiin, mikä portti tahdotaan avata ulkomaailmalle Dockerin eristetystä ympäristöstä. Valittiin HTTP-liikenteen vakioportti 80. Tämä muuttuja ei vielä yksinään avaa porttia ympäristön ulkopuolelle, vaan se asetetaan sovelluskonttia suorittaessa. Lopuksi CMD-muuttujalla voidaan suorittaa komento sovelluskontin ajovaiheessa, esimerkkikuvan tapauksessa suoritetaan Apache-webpalvelin komennolla */usr/sbin/apache2ctl -D FOREGROUND*, joka käynnistää Apachen taustalla.

4.5 Levykuvan rakentaminen ja esimerkkikontin suorittaminen

Esimerkkikuvassa tarvittiin vain muutamia yleisimpiä ympäristömuuttujia, joita Dockerfile-tiedoston rakentamiseen on tarjolla. Levykuvan laajuuden kasvaessa erilaisia ominaisuuksia on laajasti saatavilla. Nyt Dockerfile-tiedosto on valmis, ja levykuva voidaan rakentaa komennolla *sudo docker build*. Komento vaatii argumenttina Dockerfile-tiedoston polun.

Valmiin levykuvan nimeäminen helpottaa sen hallinnoimista, joten on suositeltavaa antaa levykuvalle tunniste käyttäen *-t* -parametriä. Tälle parametrille annetaan ensiksi haluttu nimi, kuten *apache* ja sen jälkeen kaksoispisteellä eroteltu tunniste, esimerkiksi *v1*. Alla olevassa kuvassa 10. komento suoritettiin samassa polussa Dockerfile-tiedoston kanssa, joten polku on yksittäinen piste:

```
joni@ubuntu:~/apache$ sudo docker build . -t apache:v1
Sending build context to Docker daemon  16.38kB
Step 1/9 : FROM ubuntu:latest
latest: Pulling from library/ubuntu
6cf436f81810: Pull complete
987088a85b96: Pull complete
b4624b3efe06: Pull complete
d42beb8ded59: Pull complete
```

Kuva 10. Suoritettiin levykuvan rakennuskomento, joka hakee aluksi tarvittavat pohjakuvat Docker Hubista.

Komento aloittaa monivaiheisen rakentamisprosessin, jossa haetaan aluksi tarvittavat pohjakuvat Docker Hubista ja suoritetaan Dockerfile-tiedostossa määritetyt komennot. Lopuksi Docker ilmoittaa onnistuneesta levykuvan rakentamisesta sekä tunnisteen asettamisesta. Komennolla *sudo docker images* voidaan tarkistaa koneella olevat levykuvat. Alla olevassa kuvassa 11. näkyy juuri luotu apache-levykuva v1-tunnisteella; lisäksi ubuntu-levykuva latautui koneelle, sen ollessa pohjakuvana.

```
joni@ubuntu:~/apache$ sudo docker images
REPOSITORY          TAG                 IMAGE ID
SIZE
apache              v1                  a449178db989
210MB
ubuntu              latest              47b19964fb50
88.1MB
hello-world         latest              fce289e99eb9
1.84kB
```

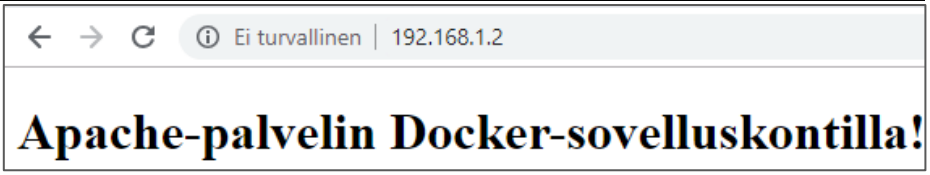
Kuva 11. Docker images -komennolla listatut levykuvat, joista yksi on äskettäin luotu apache:v1.

Itse luodun levykuvan suorittaminen sovelluskontiksi tapahtuu samaan tapaan *docker run* -komennolla. Jotta sovelluskontti ei komennon suorittamisen jälkeen sulkeudu ja pysyy taustalla käynnissä, tulee käyttää komennon yhteydessä *-d* eli detached-parametriä. Tämän lisäksi tulee määrittää sovelluskontin käyttämä portti Dockerin sisäverkosta sekä isäntäjärjestelmästä *-p* parametrillä. Dockerfile-tiedostossa määritettiin Dockerin sisäverkon portti, joka oli 80. Esimerkissä asetetaan myös ulkoverkon portiksi HTTP-protokollan oletusportti eli 80.

Lopuksi määritetään levykuvan nimi, joka halutaan suorittaa, jotta sovelluskontti voidaan luoda. Esimerkin tapauksessa nimi on apache:v1. Lopullinen komento on *sudo docker run -d -p 80:80 apache:v1.* Komennolla *sudo docker ps* voidaan tarkistaa tällä hetkellä käynnissä olevat sovelluskontit sillä koneella.

Alla olevassa kuvassa 12. nähdään käynnissä olevan Apache-webpalvelimen sovelluskontti ja selaimen näkymä yhdistäessä samassa lähiverkossa sijaitsevaan palvelimeen työasemalla.

```
joni@ubuntu:~/apache$ sudo docker run -d -p 80:80 apache:v1
1a37396c5bc42885176dc4306ddc493cc04c28b2e8795cd6bfbb2b5a722fcf42
joni@ubuntu:~/apache$ sudo docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
1a37396c5bc4        apache:v1           "/bin/sh -c '/usr/sb…"   8 seconds ago
joni@ubuntu:~/apache$
```



Kuva 12. Docker run -komennon jälkeen listattiin parhaillaan käynnissä olevat sovelluskontit. Yhdistäessä palvelimeen selaimella samassa lähiverkossa, voidaan todeta sovelluskontin toimivuus.

5 PALVELUT JA NIIDEN SKAALAUS DOCKERISSA

Myös konttisovellukset tarvitsevat skaalautuvuutta ja kuormantasausta. Hajautetussa sovelluksessa eri sovelluksen osia kutsutaan ”palveluiksi”. Esimerkiksi videontoistosivustolla on luultavasti palvelu sovellusdatan tallettamiseen tietokantaan, palvelu videon transkoodaamiseen taustalla käyttäjän lähettäessä tiedostoja, palvelu asiakkaan käyttöliittymälle ja niin edelleen. (Docker Inc. 2019b).

Palvelut ovat yksinkertaisesti sovelluskontteja tuotannossa. Palvelua suoritetaan ainoastaan yhdellä levykuvalla mutta kyseisessä levykuvassa määritetään, kuinka palvelu toimii – kuten mitä portteja käyttää ja kuinka monta kopiota sovelluskontista luodaan, jotta palvelulla on tarvittava kapasiteetti toimimiseen. Palvelun skaalaus muuttaa sovellusta suorittavien kontti-instanssien määrää, asettaen palvelulle lisää resursseja. (Docker Inc. 2019b).

5.1 Docker Compose

Docker Compose on työkalu useamman sovelluskontin Docker-palveluiden määrittämiseen sekä suorittamiseen. Compose-työkalua hyödyntävät palvelut konfiguroidaan käyttäen *docker-compose* YAML-tiedostoa. Kun tämä tiedosto on luotu, voidaan kaikki palvelut käynnistää yhdellä komennolla. (Docker Inc. 2019h). Alla olevassa kuvassa 13. nähdään esimerkki yksinkertaisesta skaalautuvasta palvelusta, jolle on määritetty resurssirajat.

```yaml
version: "3"
services:
  web:
    image: username/repo:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "4000:80"
    networks:
      - webnet
networks:
  webnet:
```

Kuva 13. Esimerkki valmiista docker-compose YAML-tiedostosta.

Tässä esimerkissä haetaan ensin haluttu levykuva Docker Hub-rekisteristä. Tästä levykuvasta käynnistetään viisi instanssia palveluksi nimellä ”web”, joista jokaiselle on annettu lupa käyttää maksimissaan kymmenen prosenttia prosessorin laskentatehoista sekä 50 megatavua keskusmuistia.

Condition -arvolla esimerkissä määritetään sovelluskonttien käynnistyvän automaattisesti uudestaan niiden kaatuessa tai sammuessa. (Docker Inc. 2019b)

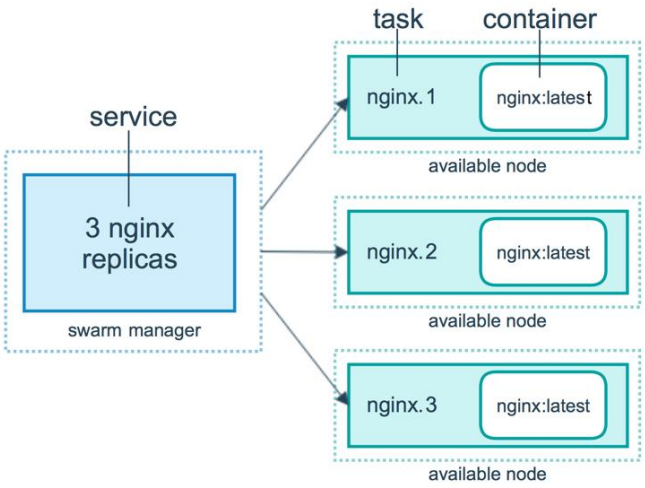
Seuraavaksi tiedostossa kartoitetaan isäntäjärjestelmän portti 4000 ohjautuman *web* -palvelun porttiin 80. Lisäksi käsketään *web* -palvelun sovelluskontteja jakamaan porttia 80 *webnet* -nimisen kuormantasaajaverkon läpi. Tätä varten tiedostossa määritetään lopuksi käytetty *webnet* -verkko vakioasetuksilla. (Docker Inc. 2019b)

Tässä työssä ei käsitellä Compose-työkalulla palveluiden luomista tai suorittamista. Docker Compose antaa silti hyvän esimerkin Kubernetes-alustan käyttöönoton määritystiedostoista, jotka mukailevat Composessa käytettyjä YAML-tiedostoja. Compose-työkalua käytetäänkin lähinnä pienissä testiympäristöissä palvelun suorittamisen helpottamiseksi, eikä niinkään suuriin tuotantoskaalauksiin. Silti on tilanteita, jolloin esimerkiksi Composen ja Kuberneteksen yhdistämisestä voidaan saada etuja, joten Compose ei ole yksinomaan testiympäristön työkalu. (Docker Inc. 2019h).

5.2 **Docker Swarm**

Docker Compose -tiedosto mahdollistaa sovelluskontteja hyödyntävän palvelun skaalauksen, mutta toimii yksinään parhaiten yhden tietokoneen paikallisissa ratkaisuissa. Tarvittaessa suuremman luokan skaalautuvuutta, saatetaan tarvita useampia fyysisiä tai virtuaalisia palvelimia, jolloin Compose-tiedoston tueksi hyödynnetään muita ratkaisuja. Dockerin tarjoama ratkaisu tähän on fyysisiä palvelimia tai virtuaalikoneita klusteriin lisäävä Docker Swarm.

Swarm eli ”parvi” on joukko koneita, jotka suorittavat Dockeria ja ovat liittyneet klusteriin. Tässä ympäristössä voidaan syöttää Docker-komentoja tavalliseen tapaan, mutta ne ajetaan klusterissa pääpalvelimella, jota kutsutaan ”swarm manageriksi”. Koneet parvessa voivat olla joko fyysisiä tai virtuaalisia. Parveen liittymisen jälkeen, yksittäisiä koneita kutsutaan nimellä ”node”. (Docker Inc. 2019b). Seuraavan sivun kuvassa 14. nähdään esimerkki yksinkertaisesta Docker Swarm ympäristöstä.



Kuva 14. Esimerkki kolmen noden Docker Swarm-ympäristöstä. Pääpalvelin pyytää saatavilla olevia nodeja antamaan resursseja kolmelle nginx-webpalvelimen sovelluskontille. (Docker Inc. 2019e).

Swarm managerit voivat käyttää useita eri strategioita suorittaakseen sovelluskontteja, kuten ns. tyhjin node -tekniikkaa – joka täyttää vähiten hyödynnetyt koneet sovelluskonteilla. Toisena esimerkkinä on globaali -tekniikka, joka takaa jokaiselle koneelle täsmälleen yhden instanssin määritetystä sovelluskontista. Näiden strategioiden käyttöä konfiguroidaan aiemman tapaan Compose-tiedostossa. Swarm managerit ovat ainoita koneita, jotka voivat suorittaa komentoja tai valtuuttaa muita koneita liittymään parveen ns. työkoneina (workers). Työkoneet ovat ainoastaan ympäristön kapasiteetin kasvattamista varten, eikä niillä ole valtuutta käskyttää muita ympäristön nodeja. (Docker Inc. 2019b).

Normaali Docker CE-asennus voidaan kytkeä swarm-tilaan, joka sallii parvien käytön kyseisessä ympäristössä. Swarm-tilan kytkeminen tekee kyseisestä koneesta välittömästi swarm managerin, johon voidaan liittää työkoneita tarpeiden mukaan. Tämän jälkeen Docker suorittaa kaikki komennot hallitussa parvessa, yksittäisen senhetkisen koneen sijaan. (Docker Inc. 2019b).

5.3 Docker Swarm verrattuna Kubernetekseen

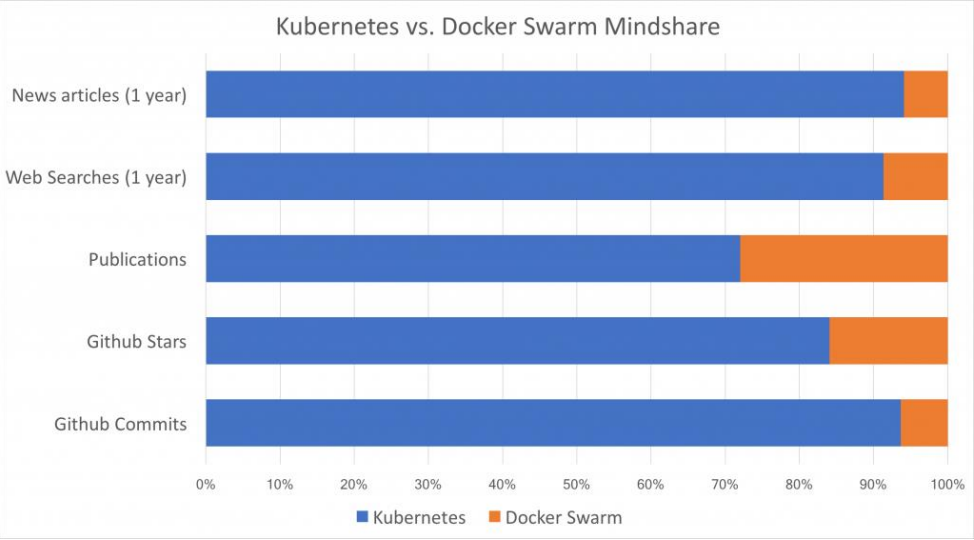
Kubernetes ja Docker Swarm ovat molemmat suosittuja vaihtoehtoja sovelluskonttien skaalaukseen sekä hallinnointiin. Niillä on paljon samankaltaisuuksia, joten en käsittele Dockerin Swarm-tilaa tässä opinnäytetyössä käytäntöön asti. Sen sijaan seuraavassa kappaleessa perehdyn syvemmin Kuberneteksen toimintaan ja sen käyttöönottoon kuvitteellisessa yritysympäristössä. Molemmat ratkaisut ovat nykyään avoimen lähdekoodin projekteja, joita kehittävät useat eri tahot aina suurista yrityksistä yksittäisiin henkilöihin.

Docker Swarm on Dockeriin sisäänrakennettu konttijärjestelijä, joten se on heti natiivisti saatavilla Docker-ympäristöön. Tämän vuoksi Swarm saattaa olla helpommin käyttöönotettava, sekä ihanteellisempi pienemmille työmäärille. Muita Swarmin hyötyjä ovat saman komentokehotteen käyttö itse Dockerin kanssa sekä oman rajapinnan tuomat edut. Swarmin huonona puolena on pääasiallisesti sen muokkaamattomuus ja vähäiset ominaisuudet. (Rosen, 2018).

Kubernetes oli alun perin Googlen kehittämä, yrityksen sisäisessä käytössä ollut alusta. Nykyään Kubernetes on ylivoimaisesti suosituin tapa sovelluskonttien hallinnointiin ja skaalaukseen, jonka vuoksi kehitys on aktiivista sekä tuen saaminen helppoa. Kuberneteksen vahvuutta suuremmissa ympäristöissä osoittaa sen käyttö suurissa yrityksissä, kuten Google ja IBM. Haittapuolena nopeasta kehityksestä ovat jatkuvat päivitykset, jotka voivat aiheuttaa yhteensopivuusongelmia. (Rosen, 2018).

Kubernetes on yleisesti tehokkaampi, muokattavampi ja joustavampi verrattuna Swarmiin; näin se soveltuu usein paremmin suuren skaalan ympäristöihin. Kubernetes onkin enemmän kokonainen alusta sovelluskonteille kuin yksinkertainen työkalu. Kuberneteksen oppimiskynnys on aluksi varsin suuri verrattuna Swarmiin sen monipuolisuuden vuoksi. Kuitenkin Kuberneteksen suorittaminen hallinnoidun lisäpalvelun kautta voi yksinkertaistaa näitä hallintavastuita, joka mahdollistaa muihin osa-alueisiin keskittymisen. Kubernetes on yleisesti myös liian raskas yksittäisten kehittäjien käyttöön, jos kyseessä on yksinkertaisen sovelluksen tai palvelun ajaminen. (Rosen, 2018).

Alla olevassa kuvassa 15. Platform9-pilvipalvelu vertaili Docker Swarmin ja Kuberneteksen suosiota erilaisten web-lähteiden sekä kehitystyön perusteella. Kuten huomataan, Kubernetes oli kesäkuun 2017 aikaan ylivoimaisesti suosituin vaihtoehto.



Kuva 15. Kuberneteksen ja Docker Swarmin suosion vertailu. (Wright 2017).

6 KUBERNETES

Kubernetes on käyttöönoton jälkeen siirrettävissä ja laajennettavissa oleva avoimen lähdekoodin alusta sovelluskontteja hyödyntävien palvelujen hallitsemiseen. Kuberneteksellä voidaan helpottaa palveluiden skaalauksen alustavaa konfigurointia sekä automatisointia huomattavasti; se tarjoaa myös laajan ja nopeasti kasvavan ekosysteemin sovelluskonteille. Suuren suosionsa vuoksi Kubernetesille on saatavilla paljon erilaisia palveluita, teknistä tukea sekä työkaluja. (The Kubernetes Authors 2019a).

Kuten edellisessä kappaleessa mainittiin, Kubernetes on alun perin Googlen sisäiseen käyttöön kehitetty työkalu. Google kuitenkin vuonna 2014 teki Kubernetes-alustasta avoimen lähdekoodin projektin. Kubernetes perustuu Googlen vuosien kokemukseen tuotannossa olevien palveluiden skaalauksesta. Tämän lisäksi avoimen lähdekoodin ansiosta yhteisön parhaita ehdotuksia ja käytäntöjä on tuotu projektiin. (The Kubernetes Authors 2019a).

Kubernetes sisältää useita eri ominaisuuksia, joten sitä voidaan pitää useana erilaisena kokonaisuutena – esimerkiksi sovelluskontti-, mikropalvelu- tai siirrettävänä pilvialustana. Kubernetes tarjoaa sovelluskonttikeskeisen hallintaympäristön. Docker Swarmin tapaan, se jakaa saatavilla olevat tietokoneresurssit automaattisesti ilman käyttäjän manuaalista työtä. Tämä antaa alustaa hyödyntäville palveluille tarvittua joustavuutta ja yksinkertaisuutta, joita nähdään usein esimerkiksi PaaS-tuotteissa. (Platform as a Service). (The Kubernetes Authors 2019a).

Vaikka Kubernetes tarjoaa paljon toiminnallisuuksia, on silti aina uusia tilanteita, joissa hyödyttäisiin uusista ominaisuuksista. Esimerkiksi sovelluskohtaista työnkulkua voidaan virtaviivaistaa nopeuttaen ohjelmistokehitystä. Tämän vuoksi Kubernetes suunniteltiin myös tarjoamaan alustaa komponenttien ja työkalujen ekosysteemin rakentamiselle. Näillä tavoilla voidaan helpottaa sovellusten käyttöönottoa, skaalausta sekä hallinnointia infrastruktuurissa (The Kubernetes Authors 2019a). Kubernetes tarjoaakin resurssien organisointiin useita eri työkaluja, joita käydään myöhemmin läpi tässä opinnäytetyössä.

Kubernetes ei kuitenkaan ole perinteinen, kaikenkattava PaaS (Platform as a Service) -järjestelmä. Koska Kubernetes toimii sovelluskonttitasolla laitteistotason sijaan, se tarjoaa joitain yleisesti PaaS-järjestelmälle miellettyjä ominaisuuksia, kuten skaalausta ja kuormantasausta. Kubernetes ei ole kuitenkaan monoliittinen ja nämä vakio-ominaisuudet ovat valinnaisia. Kubernetes siis tarjoaa rakennusosat kehittäjäalustan rakentamiseen, mutta säilyttää käyttäjävalinnan sekä joustavuuden näille tärkeissä kohdissa. (The Kubernetes Authors 2019a).

6.1 **Kubernetes-alustalla työskenteleminen**

Kubernetes sisältää useita samoja konsepteja kuin Dockerin Swarm, liittyen sovelluskonttien skaalaukseen. Kubernetes on huomattavasti laajempi kokonaisuus, tarjoten kokonaisen alustan ominaisuudet. Esimerkiksi Kubernetes-klusterit koostuvat samalla tavalla isäntänodesta, joka tunnettiin Docker Swarm -ympäristössä nimellä swarm manager. Myös työkoneiden rooli on hyvin pitkälti samantyyppinen kuin Docker Swarmissa.

Työskennellessä Kuberneteksellä käytetään hyödyksi sen oman rajapinnan objekteja (Kubernetes API object). Näillä kuvaillaan esimerkiksi sovelluskonttiklusterin tila, mitä palveluja tai työkuormia tahdotaan suorittaa sekä sovelluskonttien kopioiden määrä. Yleisesti näitä objekteja luodaan käyttämällä Kuberneteksen rajapinnan *kubectl* -komentokehotetyökalua. Objekteja määritetään Dockerin ja Swarmin tapaan YAML-tiedostoilla. On myös mahdollista käyttää rajapintaa suoraan klusterin käskyttämiseen, joskin aloittelijoille on suositeltavaa aloittaa edellä mainitulla työkalulla. (The Kubernetes Authors 2018a).

Kun objektille on asetettu haluttu toimintatila, Kuberneteksen ohjaustaso (Kubernetes Control Plane) varmistaa, että nykyisen klusterin tila vastaa luodun objektin haluttua tilaa. Tämän tehdäkseen Kubernetes suorittaa useita toimia automaattisesti – kuten sovelluskonttien uudelleenkäynnistyksiä, konttien määrän lisäystä tai muita skaalaustoimista. Kuberneteksen ohjaustaso koostuu muutamasta klusterissa toimivasta prosessista. (The Kubernetes Authors 2018a).

Näistä keskimmäisessä roolissa on Kubernetes Master -kokoelma, joka koostuu kolmesta eri prosessista, joita suoritetaan yksittäisellä isäntänodella. Nämä prosessit ovat *kube-apiserver*, *kube-controller-manager* sekä *kube-scheduler.* Nämä prosessit ovat vastuussa rajapinnan tuomisesta käyttäjälle, Kuberneteksen ohjauselementtien hallinnasta sekä objektien, esimerkiksi uusien sovelluskonttien sijoittamisesta klusterissa. (The Kubernetes Authors 2019b).

Tämän lisäksi jokainen ei-isäntänode, eli työnode klusterissa ajaa kahta prosessia. Prosessit ovat *kubelet*, joka kommunikoi isäntänoden eli Kubernetes Masterin kanssa sekä *kube-proxy*, joka toimii verkon välityspalvelimena tarjoten Kuberneteksen verkkopalvelut jokaiseen nodeen. (The Kubernetes Authors 2018a).

Kubernetes sisältää useita abstraktioita, jotka edustavat järjestelmän nykyistä tilaa; kuten käyttöönotettuja sovelluskontteja, niiden verkko- sekä levyresursseja ja muuta tietoa klusterin toiminnasta. Nämä abstraktiot esitetään Kubernetes-rajapinnassa objekteina. Yleisimpiä Kubernetes-objekteja ovat *pod, service* sekä *volume.* (The Kubernetes Authors 2018a). Seuraavissa alaluvuissa esitellään tarkemmin jokainen näistä objekteista.

6.2 **Pod-objektit eli sovelluskonttien kapselointi**

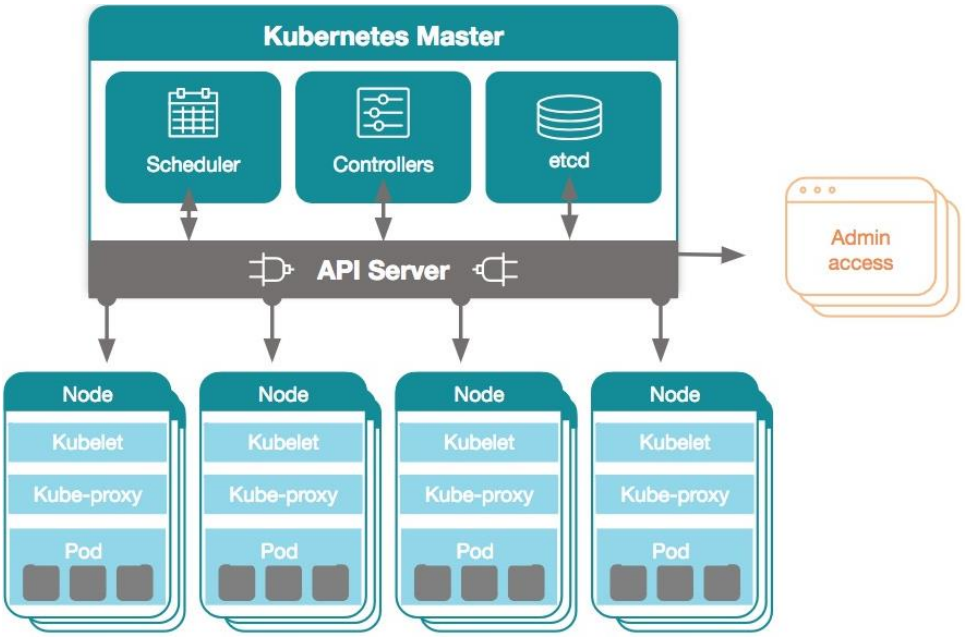
Pod-rajapintaobjekti on yleinen Kubernetes-alustan rakennusosa. Se on pienin ja yksinkertaisin yksikkö Kuberneteksen objektimallissa, joita voidaan luoda ja ottaa käyttöön ympäristössä. Pod-objekti esittää käynnissä olevaa prosessia Kubernetes-klusterissa. Pod-objekti kapseloi sovelluskontteja, talletusresursseja, uniikin verkon IP-osoitteen sekä asetuksia, jotka määrittävät kuinka sovelluskonttien kuuluu toimia. (The Kubernetes Authors 2019c).

Podit toimivat käyttöönottojen yksikköinä; eli pod-objekti on yksi instanssi sovelluksesta Kuberneteksessä. Tämä instanssi voi koostua yhdestä tai muutamasta sovelluskontista, jotka ovat kytköksissä toisiinsa ja jakavat keskenään resursseja. Docker on eniten käytetty sovelluskonttiteknologia Kubernetes-podien kanssa, mutta podit tukevat myös muita konttitekniikoita. Podeja käytetään Kuberneteksessä kahdella eri tavalla; yhden sovelluskontin pod-objekteilla, joka on kaikista yleisin Kuberneteksen käyttötapaus. Tässä tapauksessa Pod-objektia voidaan ajatella päällysteenä yhdelle sovelluskontille, jota Kubernetes hallitsee itse sovelluskontin sijaan. (The Kubernetes Authors 2019c).

Toisena podien hyödyntämistapana on usean sovelluskontin ratkaisut. Tässä tavassa yksittäinen pod-objekti sisältää useita sovelluskontteja, jotka toimivat jollain tavalla keskenään. Pod voi kapseloida sovelluksen, joka koostuu useammasta sovelluskontista, joiden täytyy jakaa resursseja. Nämä samassa paikassa sijaitsevat sovelluskontit voivat muodostaa yhtenäisen palvelukokonaisuuden. Pod-objekti päällystää nämä sovelluskontit ja talletusresurssit yhdeksi hallittavaksi kokonaisuudeksi. (The Kubernetes Authors 2019c).

Jokaisen podin on tarkoitus suorittaa yksittäistä instanssia annetusta sovelluksesta. Jos käyttäjä haluaa skaalata sovellustaan vaakasuorasti, eli ajaa useampaa instanssia, tulee käyttäjän hyödyntää useita podeja. Kuberneteksessä tätä kutsutaan yleisesti replikaatioksi (replication). Replikoituja podeja luodaan ja hallitaan yleisesti ryhmässä Controller -nimisen abstraktio-objektin avulla. (The Kubernetes Authors 2019c).

Controller-ohjainobjektilla voidaan automatisoida useamman podin luominen ja hallinnointi. Se on vastuussa podien replikoinnista ja julkistamisesta ympäristöön, tarjoten samalla automatisoituja korjausmenetelmiä koko klusterin laajuisesti ongelmatilanteiden sattuessa. Esimerkiksi jos yksittäinen node kaatuu, Controller-objekti saattaa automaattisesti korvata tällä nodella toimineen podin, siirtämällä siitä identtisen kappaleen eri nodelle. (The Kubernetes Authors 2019c).



Kuva 16. Esimerkkikuva neljän noden Kubernetes-klusterista. Pod-objektien sisällä sijaitsevat harmaat laatikot edustavat sovelluskontteja. (Gerrard, 2018).

Ylhäällä olevassa kuvassa 16. on ilmaistu nämä konseptit visuaalisessa muodossa. Esimerkissä on Kubernetes-klusteri, joka sisältää isäntänoden (Kubernetes Master) lisäksi neljä työnodea. Jokainen työnode sisältää edellä mainitut kubelet- sekä kube-proxy -objektit. Tämän lisäksi nodet sisältävät yksittäisen podin, joka kapseloi kolmea sovelluskonttia, joita ilmaistaan kuvassa harmailla laatikoilla. Admin access -laatikko voi tarkoittaa esimerkiksi työasemaa, jolla hyödynnetään Kuberneteksen rajapinnan *kubectl*-työkalua klusterin hallinnointiin komentorivikäyttöliittymällä.

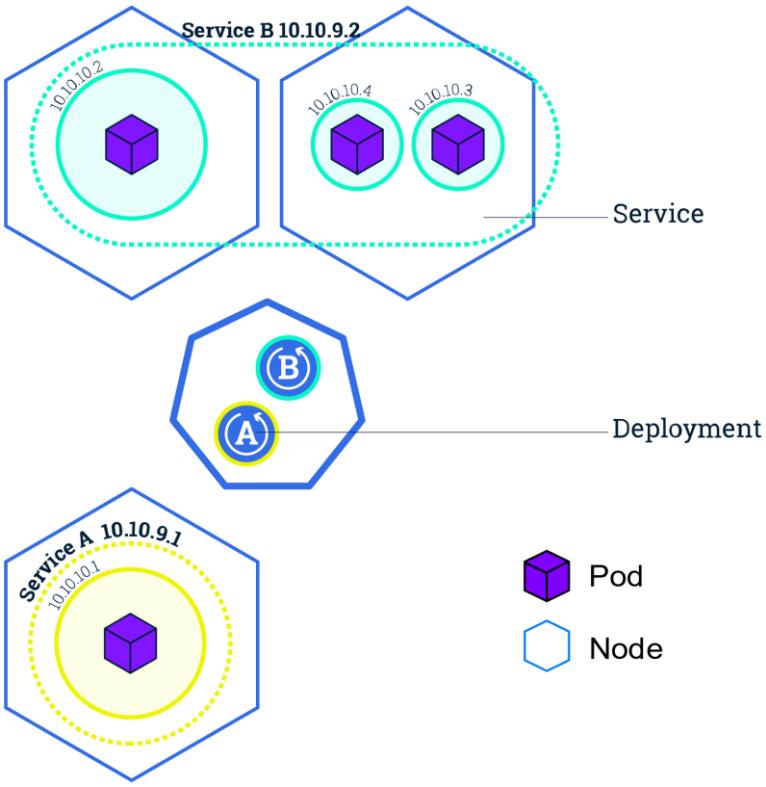
6.3 **Service-objektit yleisesti**

Kuberneteksen pod-objektit ovat vain väliaikaisia. Niitä voidaan luoda ja niiden sulkeutuessa, samaa pod-objektia ei käytetä uudestaan. Podien replikoinnista vastuussa oleva ReplicaSets-ohjain on erityisesti suunniteltu luomaan ja poistamaan podeja dynaamisesti. Vaikka jokainen pod saa oman IP-osoitteen, näihin osoitteisiin ei voida turvautua pidemmällä aikavälillä. (The Kubernetes Authors 2019d).

Tämä johtaa ongelmaan; jos esimerkiksi tietyt backend-podit tarjoavat toiminnallisuuksia muille frontend-podeille Kubernetes-klusterissa, kuinka nämä frontend-podit pitävät kirjaa tarvittavista backend-podeista? Tähän ratkaisuna Kuberneteksessä toimivat Services-objektit. Kubernetes Service-objekti on podien tapaan abstraktio. Sillä voidaan määrittää podeja loogisiin joukkoihin ja antaa sääntöjä sille, kuinka ne kommunikoivat keskenään. Palveluobjekti määritellään YAML-tiedostolla kuten muutkin Kubernetes-objektit. (The Kubernetes Authors 2019d).

Palvelut voidaan linkittää haluttuihin podeihin käyttämällä nimikkeitä (label) ja valitsimia (selector). Nämä ovat alkeellisia tapoja Kubernetes-objektien ryhmittelyyn, jotka sallivat loogisten operaatioiden tekemisen objekteissa. Nimikkeet ovat avain- tai arvopareja, jotka on liitetty objekteihin ja niitä voidaan käyttää esimerkiksi versioiden erotteluun kehitys- ja tuotantoympäristöistä. Nimikkeitä voidaan liittää objekteihin luontivaiheessa tai niitä voidaan muokata milloin vain objektin luomisen jälkeenkin. Nimikkeet ja valitsimet ovat hyvä tapa organisoida olemassa olevia palveluita suuremmissa Kubernetes-ympäristöissä. (The Kubernetes Authors 2018b).

Vaikka jokaisella podilla on uniikki IP-osoite, nämä IP-osoitteet eivät näy ulkomaailmaan klusterin ulkopuolelle ilman Service-objektia. Palveluobjektit sallivat sovelluksen vastaanottamaan verkkoliikennettä. Palveluita voidaan altistaa ulkoverkolle usealla eri tavalla, kuten sallien palvelun näkymisen ainoastaan klusterin sisällä tai avaamalla tietyn portin ulkoverkkoon. (The Kubernetes Authors 2019e).



Kuva 17. Kahden palvelun Kubernetes-ympäristö, jossa yhdistetään nodejen podeja keskenään. (The Kubernetes Authors 2019e).

Yllä olevassa kuvassa 17. on yksinkertainen esimerkki kahdesta palvelusta Kubernetes-ympäristössä. Molemmat palvelut sisältävät pod-objekteja, joilla on omat eristetyn verkon IP-osoitteet. Services-objekteja hyödyntäen voidaan eri nodeissakin olevia podeja linkata toisiinsa sekä avaa ne ulkomaailmaan uudella IP-osoitteella ja portilla.

Kuberneteksen Services -dokumentaatiossa löytyy hyvä esimerkki Service-objektin käytöstä, jossa hyödynnetään kolmea replikaatiota toteuttamaan palvelimella suoritettava kuvanprosessointipalvelu. Nämä replikaatiot ovat jokainen samankaltaisia; on yhdentekevää mitä näistä podeista asiakaspuolen laite käyttää. Vaikka varsinaiset podit, jotka muodostavat backendin voivat muuttua, frontend-asiakkaiden ei tarvitse olla tietoisia taustalla tapahtuvista muutoksista. Service-abstraktio mahdollistaa tämän tyyppisen erottelun. (The Kubernetes Authors 2019d).

6.4 **Volume — datan tallennus Kuberneteksessä**

Levyllä olevat tiedostot sovelluskontissa ovat lyhytikäisiä, joka esittää joitain ongelmia sovelluksissa niiden toimiessa sovelluskonteissa. Kun sovelluskontti kaatuu, kubelet-prosessi käynnistää kontin uudelleen, mutta tiedostot katoavat – sovelluskontti käynnistyy puhtaassa tilassa. Sovelluskonttien toimiessa yhdessä podien avulla, jakavat kontit usein olennaisia tiedostoja keskenään. Kuberneteksen Volume-abstraktio, eli looginen levy ratkaisee molemmat näistä ongelmista. (The Kubernetes Authors 2019f).

Myös Dockerilla on mahdollista tallettaa dataa sovelluskonteista, myös Dockerin sisältäessä volumen kaltaisen konseptin. Tämä on tosin hieman rajoittuneempi ja vähemmän hallittavissa oleva ratkaisu. Dockerissa volumet ovat yksinkertaisesti kansioita levyllä tai toisessa sovelluskontissa. Näiden loogisten levyjen elinaikaa ei voi hallita ja vasta hiljattain on syntynyt ratkaisuja paikallisen varmuuskopioinnin lisäksi. (The Kubernetes Authors 2019f).

Kuberneteksessä volume-objekti sen sijaan omaa täsmällisen elinajan, kuten sen kapseloiva pod-objektikin. Sen seurauksena volume säilyy sovelluskonttien sulkeutuessakin, näiden ollessa samassa pod-objektissa. Kaikki data voidaan tämän ansiosta säilöä sovelluskonttien uudelleenkäynnistysten välillä. Podin kuitenkin sulkeutuessa myös kyseinen volume katoaa. Kubernetes tukee laajasti erityyppisiä volume-objekteja, joista podit voivat käyttää useita niistä yhtäaikaisesti. (The Kubernetes Authors 2019f).

Lyhykäisyydessään volume-objekti on vain hakemisto, joka saattaa sisältää dataa. Podissa olevilla sovelluskonteilla on pääsy tähän kansioon ja riippuen volume-objektin tyypistä, sen toteutustapa vaihtelee. Pod-objektin YAML-määritystiedostossa määritetään, mitä loogisia levyjä halutaan käyttää sekä mihin nämä ns. kansiot kiinnitetään sovelluskontissa. (The Kubernetes Authors 2019f).

Sovelluskontissa toimiva prosessi näkee Docker-levykuvan rakentaman tiedostojärjestelmän. Kyseinen Docker-levykuva on tiedostojärjestelmän hierarkian juuressa ja kaikki volume-objektit kiinnitetään määriteltyihin polkuihin levykuvan sisällä. Volume-objekteja ei voida kiinnittää tai linkittää toisiinsa eli jokaiselle sovelluskontille podissa tulee erikseen määrittää, minne volume-objektit kiinnitetään. (The Kubernetes Authors 2019f).

6.5 Paikallisen Kubernetes-ympäristön käyttöönotto

Kubernetes on saatavilla useille eri alustoille, aina kannettavista tietokoneista suuren skaalan pilvipalvelinratkaisuihin. Kubernetes-klusterin käyttöönoton haastavuus vaihtelee aina muutamasta komennosta monimutkaisiin kustomoituihin klustereihin. Tämä riippuu käyttäjän tarpeista ja valitusta toteutusalustasta. Hyvä vaihtoehto Kuberneteksen opetteluun on paikallinen, Docker-pohjainen ratkaisu (The Kubernetes Authors 2019g).

Tässä opinnäytetyössä Kubernetes-klusteri otetaan käyttöön pienemmässä skaalassa, jotta perusasiat ovat helpommin ymmärrettävissä. Ratkaisuksi valikoitui Minikube, jolla voidaan luoda paikallinen yhdessä nodessa toimiva Kubernetes-klusteri. Minikube on suosittu ratkaisu esimerkiksi ohjelmistokehitys- sekä testauskäytössä. Asentamisprosessi on yksinkertaistettu eikä tarvetta pilvipalvelutunnuksille ole. (The Kubernetes Authors 2019g).

Paikallisen Kubernetes-ympäristön asennuksessa ja käyttöönotossa hyvänä apuna toimii Kuberneteksen virallisen dokumentaation Minikube -asennusohjeistus, joka löytyy työn kirjoitushetkellä osoitteesta https://kubernetes.io/docs/setup/minikube/. Sivustolta löytyy myös yleisimpiä komentoja Minikuben käyttöön liittyen sekä linkkejä muihin tiedonlähteisiin. Kuten Docker, myös Minikube asennetaan tässä opinnäytetyössä Linux-pohjaiselle Ubuntu Server -käyttöjärjestelmälle.

6.6 Minikuben asentaminen Linux-pohjaisella käyttöjärjestelmällä

Minikubella toteutettu Kubernetes-klusteri vaatii oletusarvoisesti laitteistoavustetun virtualisoinnin käyttöönottoa; tämä on myös suositeltua tiettyjen haavoittuvuuksien estämiseksi. Työssä käytetty Ubuntu-asennus toimii fyysisellä palvelimella, joten laitteistoavustettu virtualisointi tulee kytkeä käyttöön tietokoneen BIOS:in kautta. Intel-prosessoreilla tätä tekniikkaa kutsutaan nimellä VT-x ja AMD:n prosessoreilla AMD-v. Linuxissa voidaan tarkistaa ominaisuuden tila komennolla *egrep --color 'vmx|svm' /proc/cpuinfo*. Jos komento antaa tulosteen, virtualisointi on käytössä.

Minikube tarvitsee myös hypervisor-ohjelman toimiakseen. Tähän vaihtoehtoina ovat Linuxilla joko KVM (Kernel-based Virtual Machine) tai Oracle-yhtiön VirtualBox. Näistä VirtualBox on Minikuben oletusvalinta ja asennusohjelma sisältää sen tarvittavat ajurit mukanaan. KVM on Linuxin ytimen sisäänrakennettu tuki virtualisoinnille, mutta tämä ratkaisu vaatii ylimääräisten ajurilisäosakirjastojen asentamista käyttöjärjestelmälle, jotta Minikuben saa käyttöön. Tässä tapauksessa käytetään Minikuben oletusarvoista VirtualBox-ohjelmaa.

Ubuntu Serverille VirtualBox-ohjelma voidaan asentaa yksinkertaisesti käyttämällä *apt*-komentoa. Kuitenkin ensiksi täytyy sallia Ubuntun *multiverse*-ohjelmavaraston käyttäminen, joka sisältää laajan kokoelman tekijänoikeudellisesti ja laillisesti rajoittuneempia ohjelmia. Tätä ohjelmavarastoa ylläpitävät yhteisön jäsenet, eikä se ole Ubuntun virallisesti tukema. Tämä ei kuitenkaan tarkoita sitä, etteikö ohjelmavaraston ohjelmia olisi turvallista käyttää. *Multiverse*-ohjelmavaraston saa käyttöön komennolla *sudo add-apt-repository multiverse*, jonka jälkeen pakettilistaus voidaan päivittää komennolla *sudo apt-get update.*

Tämän jälkeen VirtualBoxin asennus onnistuu komennolla *sudo apt install virtualbox.* Jos tietokoneessa on käytössä Secure Boot -ominaisuus, asennusohjelmassa tulee muutamia lisävaiheita, jotka sallivat kolmannen osapuolen ajureiden käytön järjestelmässä. Työssä käytetyllä palvelimella Secure Boot on poistettu käytöstä prosessin yksinkertaistamisen vuoksi.

Hypervisor-ohjelman asennuksen jälkeen voidaan itse Minikuben asennusprosessi aloittaa. Minikuben ohjaamista varten asennetaan aluksi *kubectl*-komentokehotetyökalu; tämä onnistuu Ubuntulla helposti snap-paketinhallintaa käyttäen. Asentaminen tapahtuu komennolla *sudo snap install kubectl –classic. Kubectl*-työkalun toimivuuden sekä version voi tarkistaa komennolla *kubectl version*.

Minikuben käännetyn ohjelmakoodin saa helposti ladattua komentorivillä käyttämällä esimerkiksi *curl*-käyttöliittymää. Ohjelman lataamisen lisäksi Minikube tarvitsee suoritusoikeudet käyttöjärjestelmässä, jotka annetaan *chmod +x* -komennolla. Alla oleva komento suorittaa edellä mainitut asiat:

curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 \
&& chmod +x minikube

Kun ohjelma on ladattu, se voidaan kopioida */usr/local/bin* -kansioon, josta normaalit käyttäjätunnukset voivat ajaa Linuxille asennettuja ohjelmia. Tämän jälkeen voidaan poistaa latauskansion ohjelmakopio:

sudo cp minikube /usr/local/bin && rm minikube

Nyt Minikube voidaan käynnistää komennolla *minikube start*. Ensimmäisellä käynnistyskerralla VirtualBox luo virtuaalikoneen Minikube-klusterille, lataa Minikuben ISO-levykuvakkeen, konfiguroi Dockerin sekä alustaa Kuberneteksen käyttövalmiiksi. Kubernetes-klusteri saa tässä vaiheessa myös IP-osoitteen ja *kubectl*-työkalu osoitetaan toimimaan tuoreen Minikube-asennuksen kanssa. Onnistunut asennus voidaan todeta *Done! Thank you for using minikube!* -tulosteesta.

7 TEKNIIKOIDEN SOVELTAMINEN

Docker-sovelluskontteihin perehtymisen sekä Kubernetes-ympäristön asennusvaiheen jälkeen tämän kappaleen tarkoituksena on soveltaa teoriaosuudessa käsiteltyjä tekniikoita. Käytännön osion tavoitteena on rakentaa Docker-sovelluskontteja hyödyntävä Kubernetes-ratkaisu, jonka tehtävänä on toimia LAMP-ympäristön alustana. Osiossa käytetään hyödyksi esimerkkiratkaisua osoitteesta https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/

LAMP-lyhenne tulee sanoista:

- Linux, avoimen lähdekoodin käyttöjärjestelmä.
- Apache, joka on maailman käytetyin WWW-palvelin.
- MySQL on tietokanta, joka on suosittu Linux-käyttäjien keskuudessa.
- PHP on komentosarjakieli, jolla luodaan dynaamista verkkosisältöä.

Yhdessä nämä muodostavat LAMP-ympäristön, joka on kokoelma avoimen lähdekoodin ohjelmia, joilla voidaan suorittaa verkkopalveluita dynaamisesti. Suurin osa maailman verkkosivuista ja -palveluista käyttää tätä ohjelmaympäristöä (Easttom 2011, s. 317). Tämän ympäristön toteuttaminen toimii hyvänä ponnahduslautana monipuolisempien palveluiden toteuttamiseen Kubernetes-ympäristöissä.

Toteutusympäristö koostuu fyysisestä palvelimesta, jonka käyttöjärjestelmänä toimii Linux-pohjainen Ubuntu Server. Kubernetes-alusta käyttää aiemmassa kappaleessa luotua Minikube-virtuaalikonetta tämän fyysisen palvelimen päällä. Virtuaalikone sisältää myös oman virtuaalisen verkkoadapterin, joten yhtenä haasteena tässä ratkaisussa tulee olemaan valmiiden palveluiden avaaminen fyysiseen lähiverkkoon. Minikube-virtuaalikoneelle on määritetty kaksi virtuaaliydintä sekä kaksi gigatavua keskusmuistia.

Tässä kokonaisuudessa ainoastaan Apache-palvelu on saavutettavissa fyysisestä ulkoverkosta, hyödyntäen Service-objektia. MySQL-tietokanta on saavutettavissa ainoastaan Kubernetes-klusterin sisällä, johon Apache-palvelu voi tarvittaessa yhdistää. Molemmille palveluille luodaan volume-objektit, jotka sallivat datan tallettamisen pysyvästi. Palvelut yhdistetään toisiinsa käyttäen Service-, label-, sekä selector-objekteja. Ratkaisun toimivuuden osoittaa phpMyAdmin, jolla voidaan hallita MySQL-tietokantoja graafisen web-käyttöliittymän avulla.

7.1 Minikuben etäkäyttö fyysisessä lähiverkossa

Minikube-alustan hallinnointi onnistuu usealla tavalla. Perinteisempi vaihtoehto on SSH-yhteyden ottaminen palvelimeen, jossa *kubectl*- sekä *minikube*-komennot mahdollistavat palveluiden luomisen sekä muokkaamisen.

Toinen opinnäytetyössä käytetty vaihtoehto on Minikube Dashboard, joka on graafinen web-käyttöliittymä Minikubelle.

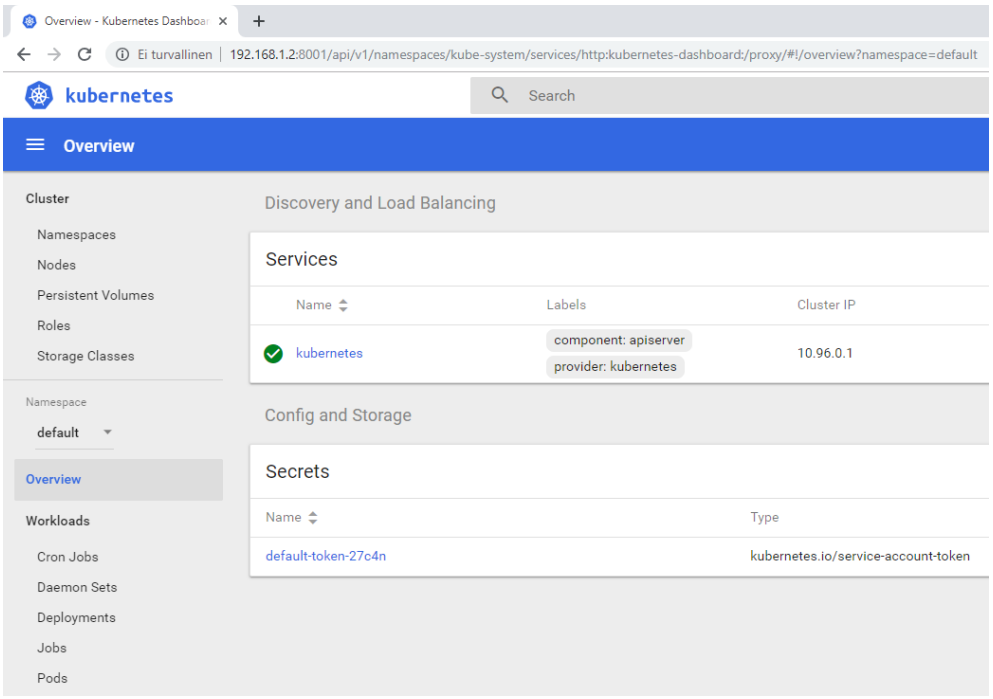
Tähän käyttöliittymään pääsy klusterin ulkopuolisesta verkosta vaatii tosin Kuberneteksen välityspalvelinominaisuuden käyttöä, joka voidaan ottaa väliaikaisesti käyttöön komennolla *kubectl proxy*. Toteutusympäristössä fyysinen palvelin sekä työasema ovat samassa lähiverkossa, joten hallinnointia yksinkertaistaa käyttöliittymään pääsy myös fyysisen työaseman selaimelta. Komento on kokonaisuudessaan:

kubectl proxy --address='0.0.0.0' --disable-filter=true

Tätä tapaa ei kuitenkaan ole suositeltavaa käyttää kuin kehitysympäristöissä, sillä *--disable-filter=true* -parametri sallii kaikkien samassa fyysisessä lähiverkossa olevien tietokoneiden yhdistämisen hallintakäyttöliittymään selaimen välityksellä. Kun välityspalvelin on käynnistetty, käyttöliittymä on saavutettavissa portissa 8001, joka on Kuberneteksen vakioportti välityspalvelinliikenteelle. Web-käyttöliittymän kokonainen osoite on:

http://palvelimen-osoite:8001/api/v1/namespaces/kube-system/services/http:kubernetes-dashboard:/proxy/

Alla olevassa kuvassa 18. on kuvakaappaus Minikube Dashboard -käyttöliittymästä. Tämä web-käyttöliittymä mahdollistaa yleisimpien *kubectl*-toimintojen suorittamisen graafisen käyttöliittymän avulla. Yksi suurimmista graafisen käyttöliittymän eduista on Kubernetes-objektien tilanteen seuraaminen visuaalisesti. Dashboard-näkymässä esimerkiksi palveluiden, podien sekä levyosioiden seuraaminen ja hallinta helpottuu merkittävästi.



Kuva 18. Kuvakaappaus Minikube Dashboard-käyttöliittymästä.

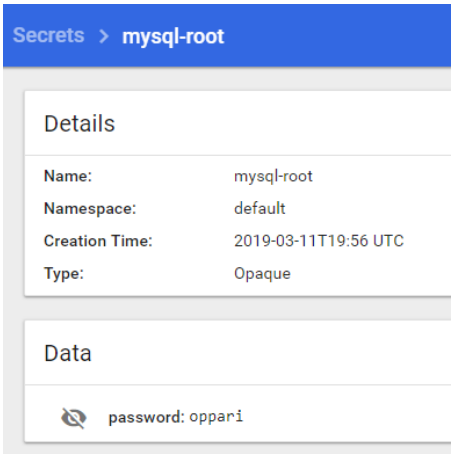
7.2 MySQL-salasanojen luonti Secret-objektilla

Ennen YAML-asennusmääritystiedostojen tekemistä, on hyvä luoda palvelun tarvittavat salasanat käyttäen Secret-objektia. Kuberneteksen Secret-objektit antavat käyttäjän säilöä näissä arkaluonteista tietoa, kuten salasanoja. Salasanojen säilyttäminen Secret-objekteissa on huomattavasti turvallisempaa kuin esimerkiksi selkokielisinä levykuvakkeissa tai konfiguraatiotiedostoissa.

Tässä tapauksessa luodaan kaksi salasanaa MySQL-tietokantaa varten. Ensimmäinen salasana toimii tietokannan root-käyttäjän salasanana ja toinen käyttöönotossa luotavan MySQL-käyttäjän salasanana. Secret-objektien luominen on yksinkertaista ja suositeltavaa jokaisessa ympäristössä. Helpoiten salasanan luominen onnistuu *kubectl*-komentoa käyttäen:

kubectl create secret generic mysql-root --from-literal=password=salasana

Ylhäällä olevassa komennossa punaisella korostetut kohdat ilmaisevat objektin nimen sekä asetetun salasanan. Toteutuksen tapauksessa loin objektit *mysql-root* ja *mysql-user*, joiden salasanat ovat *oppari.* Objektien olemassaolo voidaan varmistaa komennolla *kubectl get secrets*. Näitä salasanoja voi tarkastella selkokielisenä esimerkiksi Dashboardin Secrets-valikon kautta. Alla olevassa kuvassa 19. nähdään valitun Secret-objektin nimi, tyyppi sekä salasana selkokielisessä muodossa.



Kuva 19. Selkokielinen salasana Secrets-objektissa.

7.3 YAML-asennusmääritykset ja PersistentVolume-objektien luominen

Jotta Apache-webpalvelimelle ja MySQL-tietokantaan voidaan tallettaa pysyvästi dataa, tarvitaan PersistentVolume-objekteja. PersistentVolumeClaims-objektilla pyydetään toteutusympäristön tapauksessa Kubernetes-klusterilta asetetun määrän tallennuskapasiteettia, jonka jälkeen Kubernetes luo itsestään sen kokoisen PersistentVolume-levyosion palvelimelle. Nämä objektit tulevat säilymään podien elinkaaren yli sekä Claims-objektin voi kiinnittää halutun podin tietojärjestelmään.

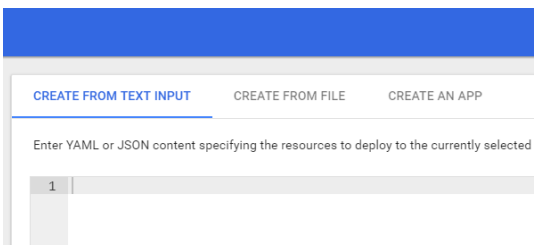
Tästä esimerkkinä voisi olla Apache-webpalvelimen pod, jonka verkkosivut sijaitsevat tiedostojärjestelmän kansiossa /var/www/html. Kaikki selaimella avattavat verkkosivut löytyvät tuosta polusta, joten tämän kansion tietojen säilyminen on hyvin tärkeää palvelun toimivuudelle. Kiinnittämällä PersistentVolumeClaims-objekti tähän tiedostojärjestelmän polkuun, voidaan taata tiedostojen säilyminen myös uudelleenkäynnistyksien yhteydessä.

Kaikkien objektien luominen Kuberneteksessä tapahtuu YAML- tai json-asennusmääritystiedostoilla. Näitä tiedostoja voidaan ajaa joko *kubectl*-komennon tai Dashboard-käyttöliittymän kautta. MySQL-tietokannan Volume-objektin luominen on toteutettu vastaavanlaisella YAML-tiedostolla opinnäytetyön toteutusympäristössä:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-claim
  labels:
    app: webserver
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

YAML-tiedostossa *kind*-arvolla määritetään objektin tyyppi, *metadata* -osion *name* nimeää objektin sekä *labels*-osion avulla voidaan antaa objektille nimikkeitä, jotka auttavat podien linkittämistä keskenään. Lopuksi *spec*-osiossa määritetään objektin asetukset; tämän objektin tapauksessa asetukset ovat *accessMode*s sekä *storage*. *AccessModes* asettaa tavan, kuinka levyosiota käytetään pod-objektin tietojärjestelmässä. *Storage* kertoo Claims-objektin pyydetyn levykapasiteetin.

Myös Apachelle luotiin samanlainen määritystiedosto, jossa vaihdettiin ainoastaan objektin nimi. Kaikki Kuberneteksen YAML-määritystiedostot mukailevat samaa syntaksia, joten useat määritykset ovat samanlaisia riippumatta objektien tyypistä. Määritystiedosto voidaan ajaa joko *kubectl*-komennolla tai Dashboardin graafisen käyttöliittymän kautta, johon pääsee käyttöliittymän oikean yläkulman *CREATE*-painikkeella. Kubectl-komento määritystiedoston ajamiseen on *kubectl create -f tiedosto*.



Kuva 20. YAML-tiedoston syöttäminen onnistuu myös Dashboardista.

7.4 **Deployment-objektit ja podien luominen**

Deployment-ohjainobjekti on yksi tärkeimmistä osista Kubernetes-alustassa. Se tarjoaa päivityksiä Pod-objekteille sekä podien replikoinnista vastuussa olevalle ReplicaSets-ohjainobjektille. Käytännössä Deployment-määritystiedostoilla luodaan ja konfiguroidaan pod-objektit. Käyttöönoton YAML-tiedosto sisältää esimerkiksi sovelluskontin levykuvan tiedot, levyosioiden asettamisen sekä kontin käyttämiä ympäristömuuttujia. Alla olevalla YAML-tiedostolla toteutusympäristöön luotiin MySQL-pod.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver-mysql
  labels:
    app: webserver
spec:
  selector:
    matchLabels:
      app: webserver
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: webserver
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_DATABASE
          value: tietokanta
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-root
              key: password
        - name: MYSQL_USER
          value: joni
        - name: MYSQL_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-user
              key: password
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mysql-persistent-storage
          mountPath: /var/lib/mysql
      volumes:
      - name: mysql-persistent-storage
        persistentVolumeClaim:
          claimName: mysql-claim
```

Kuten aiemmassa Volume-objektin luonnissa, YAML-tiedosto sisältää nimikkeitä sekä osoittajia, joilla yhdistetään esimerkiksi VolumeClaims-objektit tiettyihin podeihin. Merkittävänä osana Deployment-tiedostossa on sovelluskonttien määritys käyttäen *containers* -asetusta. Tämän objektin ollessa MySQL-tietokantaa varten on sovelluskontin levykuvaksi valittu Docker Hubin virallinen *mysql*-levykuva, jonka Kubernetes osaa hakea automaattisesti palvelimelta.

Tämä levykuva sallii myös *env*-ympäristömuuttujien käyttämisen, jolla voidaan antaa arvoja pod-objektien luontivaiheessa. Toteutusympäristön määritystiedostossa on luotu uusi tietokanta *MYSQL_DATABASE* -muuttujalla, salasanat käyttäen aiemmin luotuja *Secrets*-objekteja sekä yksi normaali käyttäjätunnus root-käyttäjän lisäksi. Lisäksi sovelluskontille on annettu sisäinen portti 3306 sekä *PersistentVolumeClaims*-objekti on osoitettu sovelluskontin tietojärjestelmän kansioon */var/lib/mysql*, missä tietokannan data sijaitsee. Kaikki tuohon polkuun tallentuvat tiedostot pysyvät tallessa Volume-objektissa, jotka ovat podeista riippumattomia.

Toisena Deployment-objektina toteutusympäristössä toimii Apache-webpalvelin, johon on lisätty tuki PHP-komentosarjakielelle dynaamista verkkosisältöä varten sekä tuki MySQL Improved-komennoille PHP:ssä.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webserver
  labels:
    app: webserver
spec:
  selector:
    matchLabels:
      app: webserver
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: webserver
        tier: frontend
    spec:
      containers:
      - image: jonihann/apache-php-mysqli:v1
        name: webserver
        ports:
        - containerPort: 80
          name: webserver
        volumeMounts:
        - name: webserver-persistent-storage
          mountPath: /var/www/html
      volumes:
      - name: webserver-persistent-storage
        persistentVolumeClaim:
          claimName: webserver-claim
```

Yllä oleva Deployment-tiedosto mukailee pitkälti MySQL-käyttöönoton periaatteita. Pod-objekti osoitetaan tiettyihin nimikkeisiin, podille määritetään sovelluskontin levykuva sekä käytettävä portti. Myös levyosion asettaminen tapahtuu samalla tavalla, joskin kontin tiedostojärjestelmän polku on nyt */var/www/html*, joka on Apachen vakiopolku verkkosisällölle.

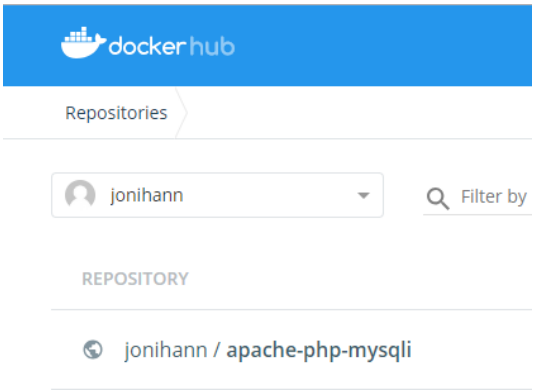
Myös PHP-ohjelmointikieleen tarvittavat komponentit ovat saatavilla omana sovelluskonttinaan Docker Hubin virallisten levykuvien kautta. Lisäksi tästä levykuvasta on mahdollista ladata versio, joka sisältää Apache-webpalvelimen valmiiksi käyttämällä esimerkiksi levykuvaa tagilla *php:7-apache*. Tämäkään levykuva ei tosin sisällä useissa tietokantaratkaisuissa pakollista MySQL Improved-ajuria. Tätä ajuria käyttämällä PHP:llä voidaan kommunikoida halutun MySQL-tietokannan kanssa. Ongelman voi ratkaista yksinkertaisesti rakentamalla oman Docker-levykuvan alla olevalla Dockerfile-tiedostolla.

FROM php:7.3.3-apache
RUN docker-php-ext-install mysqli

Tämän jälkeen voidaan luoda levykuvan normaalisti *docker build* -komennon avulla. Toteutusympäristön tapauksessa rakennettu levykuva haluttiin saataville julkisesti Docker Hubin kautta, joten nimeämisessä tuli ottaa huomioon Docker Hub -palvelun käyttäjätunnus sekä sinne luotu tietovarasto (repository). Ensin levykuvan rakentaminen suoritettiin komennolla:

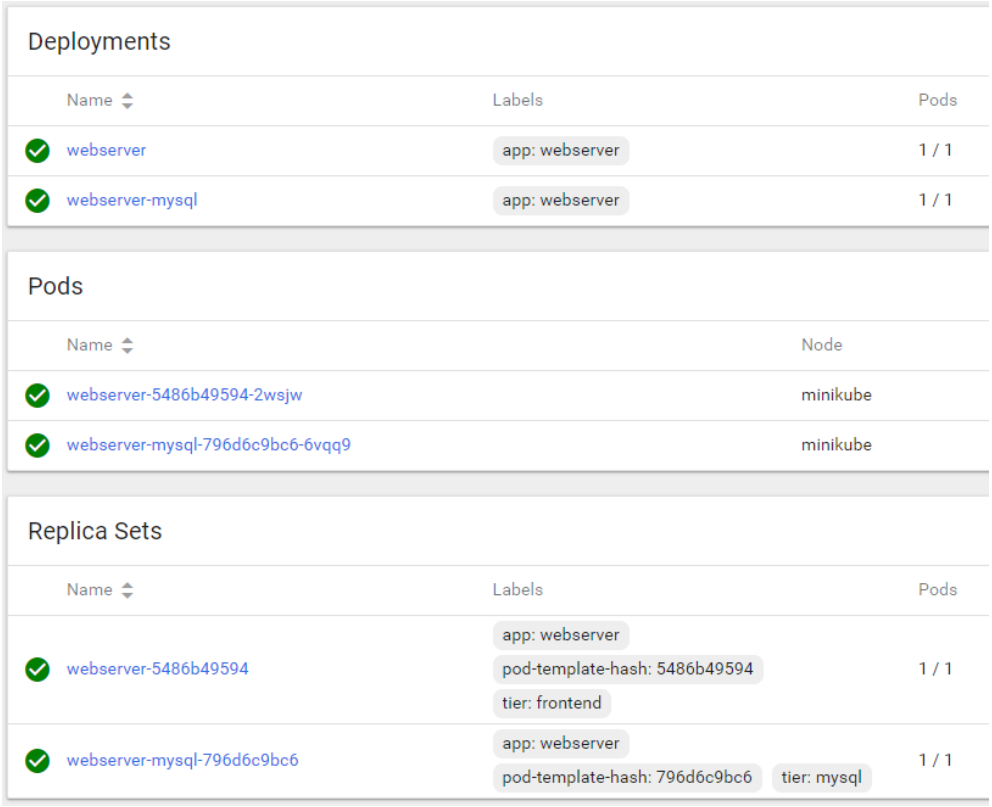
sudo docker build . -t jonihann/apache-php-mysqli:v1

Tässä tapauksessa *jonihann* on käyttäjätunnus Docker Hubissa, *apache-php-mysqli* tietovaraston nimi sekä *v1* toimii tunnisteena kyseiselle versiolle levykuvasta. Levykuvan rakentamisen jälkeen se haluttiin siirtää Docker Hubin palvelimille käyttäen *docker push* -komentoa, jota ennen kirjaudutaan *jonihann* -käyttäjätunnukselle *docker login* -komennolla. Alla olevassa kuvassa 21. nähdään Docker Hub-sivuston käyttäjänäkymä omista levykuvista palvelussa.



Kuva 21. Docker Hub -sivuston käyttäjän näkymä omista sovelluskonttien levykuvista palvelussa.

Kun luodut Deployment-määritystiedostot on ajettu, voi käyttöönoton tilannetta seurata graafisesti Kubernetes Dashboardin avulla. Overview- eli yleisnäkymästä saadaan selville Deployment-tiedostolla luotujen objektien nykyinen tila ja lukumäärä. Myös Volume-objektien tilannetta voi seurata tästä näkymästä. Toteutusympäristön tapauksessa web- sekä tietokantapodeista luotiin vain yhdet kappaleet yksinkertaistamaan toteutusta. Alla olevassa kuvassa 22. on kuvankaappaus Dashboardin yleisnäkymästä, joka ilmaisee edellä mainitut objektit.



Kuva 22. Kuberneteksen Dashboard-käyttöliittymän yleisnäkymä.

7.5 Service-objektit ja palveluiden avaaminen ulkoverkkoon

Viimeisinä Kubernetes-objekteina toteutukselle ovat Service-objektit, jotka mahdollistavat luotujen pod-objektien avaamisen joko klusterin sisäisesti tai ulkoverkkoon. Nämä objektit sallivat podien keskenvälisen kommunikoinnin samassa nodessa, sekä toteutusympäristön tapauksessa Apache-palvelimen avaamisen fyysiseen lähiverkkoon. Toteutuksessa MySQL-tietokannan pod-objekti pystyy kommunikoimaan Apache-podin kanssa, mutta ainoastaan Apache on saavutettavissa ulkoverkosta.

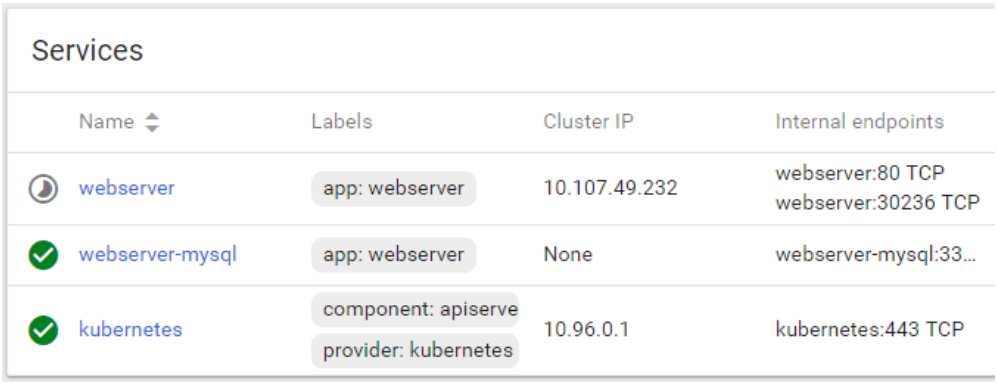
Service-objektin luominen YAML-tiedostolla vaatii muiden objektien tapaan nimikkeiden sekä valitsimien määrittämisen, jotta palvelut löytävät oikeat pod-objektit. Tiedostossa määritellään myös Service-objektin tyyppi sekä käytetty porttinumero klusterissa tai sisäverkossa. Työn toteutuksessa palvelut luotiin alla olevien YAML-tiedostojen mukaisesti.

```
apiVersion: v1
kind: Service
metadata:
  name: webserver-mysql
  labels:
    app: webserver
spec:
  ports:
    - port: 3306
  selector:
    app: webserver
    tier: mysql
  clusterIP: None
```

MySQL-podien palvelun tyypiksi asetettiin Kuberneteksen vakiona käyttämä *ClusterIP*, joka avaa palvelun ainoastaan klusterin sisäiseen verkkoon. Service-objektin porttina toimii jo aiemmin sovelluskontille määritetty 3306. Selector- ja label-arvoilla palvelu linkitetään haluttuihin podeihin.

```
apiVersion: v1
kind: Service
metadata:
  name: webserver
  labels:
    app: webserver
spec:
  ports:
    - port: 80
  selector:
    app: webserver
    tier: frontend
  type: LoadBalancer
```

Ulkoverkosta saavutettavaksi tarkoitettu Apache-webpalvelin käyttää sen sijaan *LoadBalancer*-palvelutyyppiä sisäisellä portilla 80. *LoadBalancer*-kuormantasaajaa käytetään yleisesti palveluille, joille on saatavilla esimerkiksi pilvipalvelun oma kuormantasaajaratkaisu. *LoadBalanceria* voi kuitenkin käyttää myös ilman kuormantasaajaa. Palvelu luo itselleen automaattisesti ulkomaailmaan avoimena olevan portin, johon yhdistetään. Alhaalla olevassa kuvassa 23. nähdään luodut palvelut ja niiden portit. Webserver-palvelun ulkomaailmaan avoinna oleva portti on 30236.



Kuva 23. Service-objektien listaus Kuberneteksen Dashboardissa.

Työn toteutusympäristössä oleva Kubernetes-alusta toimii VirtualBox-hypervisorin avulla omalla virtuaalikoneellaan. Tälle virtuaalikoneelle on luotu oma verkkoadapteri, joka hyödyntää osoitteenmuunnosta eli NAT-tekniikkaa. Tämä tarkoittaa, että saapuva liikenne täytyy sallia porttiohjauksella erikseen. Jotta fyysisen lähiverkon työasema saa selaimella yhteyden Apache-palveluun, avataan portti 30236 alla olevalla komennolla.

vboxmanage controlvm ”minikube” natpfl ”http,tcp,,30236,,30236”

Vboxmanage-komennolla voidaan ohjata VirtualBox-hypervisoriin liittyviä komponentteja. *Controlvm* -parametrillä konfiguroidaan tietyn virtuaalikoneen asetuksia, joka on tässä tapauksessa ”*minikube*”. *Natpfl* kuvastaa virtuaalikoneen verkkokomponentin NAT-tekniikkaa. Lopuksi komennossa annetaan porttisääntö TCP-protokollan portille 30236 nimellä HTTP.

Onnistunut porttiohjaus ja Apache-palvelun osittainen toimivuus voidaan todeta fyysisen lähiverkon työaseman selaimen avulla menemällä palvelimen osoitteeseen äsken avatulla porttinumerolla. Apache näyttää kieltoviestin liittyen juurihakemiston selaamisen estoihin, todentaen samalla palvelun osittaisen toimivuuden alla olevan kuvan 24. mukaisella tavalla.

Kuva 24. Kuvakaappaus Apache-palveluun yhdistämisestä selaimella fyysisestä ulkoverkosta.

7.6 Palvelun testaus sekä pod-objektien poistaminen

MySQL-tietokannan, PHP-moduulien ja Volume-levyosioiden toimivuuden testaus palvelimella toteutetaan käyttäen phpMyAdmin-käyttöliittymää. phpMyAdmin on selainpohjainen, palvelimella toimiva PHP-ohjelmointikielellä toteutettu graafinen käyttöliittymä MySQL-tietokantojen hallintaan.

Pääsy phpMyAdminiin ulkoverkon kautta ei ole suositeltavaa tuotantoympäristössä, mutta phpMyAdmin on yksinkertainen tapa testata LAMP-ympäristön toimivuutta. Tämän testauksen jälkeen poistan molempien palveluiden pod-objektit todentaen Volume-objektien toimivuuden. Työn toteutuksessa phpMyAdmin ladattiin projektin viralliselta verkkosivulta.

Helpoiten phpMyAdminin saa ladattua palvelimelle käyttäen komentoa *wget verkko-osoite*, jonka jälkeen pakattu kansio puretaan *tar*-komennolla. Jotta phpMyAdmin osaa yhdistää oikeaan tietokantapalveluun, tulee käyttäjän luoda *phpMyAdmin*-kansion juuressa sijaitsevasta *config.sample.inc.php* -tiedostosta kopio nimellä *config.inc.php*, jossa muutetaan palvelimen osoite *localhost*- eli paikallisesta osoitteesta muotoon *webserver-mysql*. Muutos tehdään alla olevan kuvan 25. mukaisesti *host*-asetukselle.

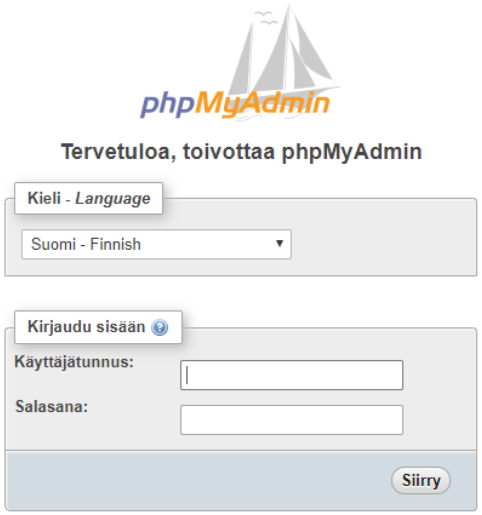
```
/* Authentication type */
$cfg['Servers'][$i]['auth_type'] = 'cookie';
/* Server parameters */
$cfg['Servers'][$i]['host'] = 'webserver-mysql';
$cfg['Servers'][$i]['compress'] = false;
$cfg['Servers'][$i]['AllowNoPassword'] = false;
```

Kuva 25. *config.inc.php* -tiedoston sisältö, johon palvelimen nimi konfiguroidaan yhdistämistä varten.

Nyt phpMyAdmin-kansio on valmiina siirrettäväksi pod-objektiin. Jotta kansio voidaan kopioida suoraan sovelluskontin */var/www* -kansioon, tulee käyttäjän selvittää Apache-podin nimi. Tämän löytää helposti esimerkiksi Dashboardin *Pods*-välilehdeltä. Kun podin tarkka nimi on selvitetty, voidaan kansio kopioida sovelluskonttiin *kubectl*-komennon *cp*-ominaisuudella alla olevan komennon mukaisesti.

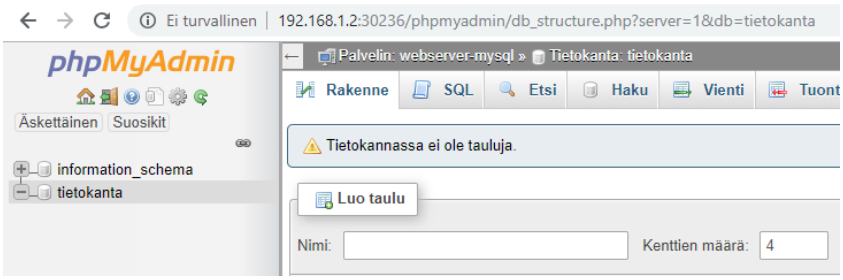
kubectl cp phpmyadmin default/webserver-podnimi:/var/www/html

Komennossa ensimmäinen punaisella korostettu kohta ilmaisee kopioitavan kansion polun ja toinen pod-objektin nimen. Loppuosan polku kertoo mihin polkuun sovelluskontin tietojärjestelmässä kansio halutaan sijoittaa. Kopioinnin onnistuessa käyttöliittymään pääsee ulkoverkon työasemalla selaimella osoitteesta *http://palvelimen-osoite:30236/phpmyadmin/*. Verkkosivu näyttää alla olevan kuvan 26. mukaiselta kirjautumisruudulta.



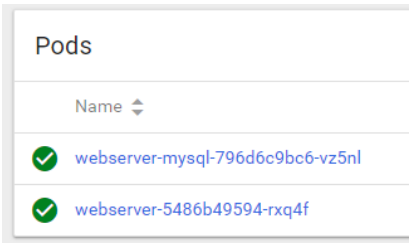
Kuva 26. phpMyAdminin käyttöliittymän kirjautumissivu.

PhpMyAdminiin kirjaudutaan aiemmin Deployment-vaiheessa luodulla MySQL-peruskäyttäjätunnuksella, joka tässä toteutuksessa oli *joni* salasanalla *oppari*. Jos yhdistäminen MySQL-podiin onnistuu, aukeaa käyttäjälle kuvan 27. mukainen phpMyAdminin käyttöliittymä, jossa vasemmassa navigointipalkissa on listattuna myös aiemmin Deployment-tiedostossa luotu *tietokanta*.



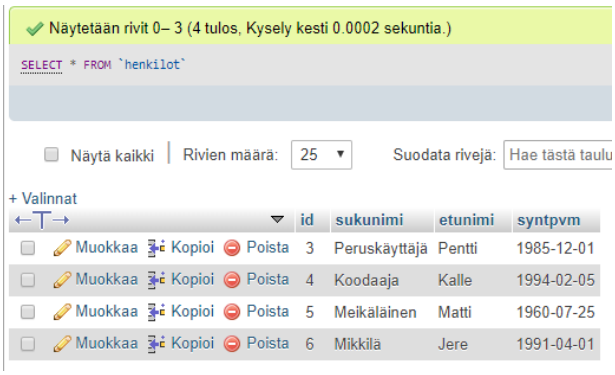
Kuva 27. Onnistunut yhteys Apache-podin kautta MySQL-tietokantaan.

Viimeisenä testinä Kubernetes-alustan LAMP-toteutuksen toimivuudesta loin tietokantaan taulukon, johon oli syötetty dataa. Tämän jälkeen poistin MySQL- sekä Apache-palvelimen pod-objektit, jotka välittömästi syntyivät uudestaan aiemman Deployment-määrittelyn ansiosta. Molemmat pod-objektit saivat uudet nimet luontivaiheessa kuvan 28. mukaisesti.



Kuva 28. Deployment-asennusmäärityksen luomat uudet pod-objektit aiempien podien poistamisen jälkeen.

Sivun uudelleenpäivittämisen jälkeenkin kaikki data, joka luotiin ennen podien poistamista on edelleen tallessa. PhpMyAdminiin kirjautuminen onnistuu sekä aiemmin luotu *henkilot* -taulu sisältöineen on saatavilla.



Kuva 29. Ennen pod-objektien poistamista luotu phpMyAdmin-asennus ja tietokantadata säilyi.

8 YHTEENVETO

Tässä opinnäytetyössä käytiin läpi tietokonevirtualisoinnin ja sovelluskonttien perusteita teoriassa sekä käytännössä. Työssä käsiteltiin myös sovelluskonttien käyttötarkoituksia sekä skaalaustapoja eri työkalujen avulla. Teoriaosuudessa esiteltiin sovelluskonttien ymmärtämiseen tarvittavat peruskäsitteet sekä niiden toimintaperiaatteet. Käytännön osuudessa toteutettiin yksinkertainen LAMP-ympäristö palvelimelle käyttäen Docker-sovelluskontteja sekä Kubernetes-alustaa sovelluskonttien skaalaamiseen.

Työn tavoitteena oli perehtyä sovelluskonttien ja niiden hallintatyökalujen toimintaperiaatteisiin sekä toteuttaa näitä tekniikoita käyttäen ympäristö web-sisällölle. Koen, että sovelluskontit ovat tärkeä aihe jokaiselle tietotekniikasta kiinnostuneelle. Varsinkin ylläpito- sekä kehitystehtävissä toimivien henkilöiden olisi hyvä perehtyä sovelluskonttien tuomiin mahdollisuuksiin omassa organisaatiossaan. Työ tarjoaa hyvän pohjan sovelluskonttitekniikoiden syvemmälle opettelulle.

Mielestäni työni aihe oli kokonaisuutena erittäin mielenkiintoinen ja samalla haastava. Teoriaosuuden taustatutkimus opetti paljon itselle jo hieman tutuista aiheista, vahvistaen tietämystäni varsinkin virtualisoinnista. Sovelluskontit olivat itselleni täysin uusi aihe, joten valtaosa ajasta kului niiden toimintaperiaatteiden oppimisessa. Kattavan teoriaopiskelun ansiosta käytäntöön siirtyminen oli huomattavasti helpompaa, kun perusasioista oli hyvä käsitys.

Käytännön osuus toi monipuolisia haasteita, esimerkiksi palveluiden avaaminen ulkoverkkoon sekä tietokannan yhdistäminen muihin palveluihin vaati paljon testausta toimiakseen halutulla tavalla. Aiemmasta Linux-osaamisesta onkin huomattavasti hyötyä työn kaltaisen ympäristön luomisessa. Ilman aiempaa kokemusta vastaavanlainen toteutus olisi vienyt huomattavasti enemmän aikaa, joten Linux-aloittelijalla täytyy olla kova halu oppia uutta onnistuakseen.

Olen varsin tyytyväinen työn lopputulokseen, vaikka työn laajuutta täytyikin rajoittaa huomattavasti, jotta sivumäärä pysyi kurissa. Varsinkin Kubernetes käsiteltiin lähinnä pintaraapaisuna, kattaen ainoastaan yleisimmät konseptit, jotta käytännön osuus saatiin toteutettua. Saavutin silti asettamani tavoitteet ja sain toteutettua toimivan LAMP-ympäristön Dockeria ja Kubernetestä käyttäen. Mielestäni työni toimii hyvänä oppaana sovelluskonteista kiinnostuneille henkilöille sekä organisaatioille.

LÄHTEET

Carter, E. (2018). *2018 Docker Usage Report.* Blogijulkaisu 29.5.2018. Haettu 2.2.2019 osoitteesta https://sysdig.com/blog/2018-docker-usage-report/

Datadog (2018). *8 surprising facts about Docker adoption.* Haettu 8.2.2019 osoitteesta https://www.datadoghq.com/docker-adoption/

Docker Inc. (2019a). *Docker overview.* Haettu 9.2.2019 osoitteesta https://docs.docker.com/engine/docker-overview/

Docker Inc. (2019b). *Get Started with Docker.* Haettu 10.2.2019 osoitteesta https://docs.docker.com/get-started/

Docker Inc. (2019c). *Docker Registry.* Haettu 10.2.2019 osoitteesta https://docs.docker.com/registry/

Docker Inc. (2019d). *Docker Hub.* Haettu 10.2.2019 osoitteesta https://www.docker.com/products/docker-hub

Docker Inc. (2019e). *How services work.* Haettu 10.2.2019 osoitteesta https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/

Docker Inc. (2019f). *About Docker CE.* Haettu 15.2.2019 osoitteesta https://docs.docker.com/install/

Docker Inc. (2019g). *Get Docker CE for Ubuntu.* Haettu 15.2.2019 osoitteesta https://docs.docker.com/install/linux/docker-ce/ubuntu/

Docker Inc. (2019h). *Overview of Docker Compose.* Haettu 25.2.2019 osoitteesta https://docs.docker.com/compose/overview/

Docker Inc. (2017). *Understanding the Business Value of Docker Enterprise Edition.* Haettu 2.2.2019 osoitteesta https://goto.docker.com/rs/929-FJL-178/images/WP_BusinessValueofDocker_06.26.2017.pdf

Docker Inc. (n. d.). *What is a Container.* Haettu 2.2.2019 osoitteesta https://www.docker.com/resources/what-container

Easttom, C. (2011). *Essential Linux Administration: A Comprehensive Guide for Beginners*. Course Technology / Cengage Learning

Gerrard, A. (2018). *Kubernetes Architecture.* Haettu 4.3.2019 osoitteesta https://blog.newrelic.com/engineering/what-is-kubernetes/

InfoWorld (2016). *Docker image layers.* Haettu 7.2.2019 osoitteesta https://www.infoworld.com/article/3077875/linux/containers-101-docker-fundamentals.html

The Kubernetes Authors (2019a). *What is Kubernetes?* Haettu 1.3.2019 osoitteesta https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

The Kubernetes Authors (2019b). *Kubernetes Components.* Haettu 1.3.2019 osoitteesta https://kubernetes.io/docs/concepts/overview/components/

The Kubernetes Authors (2019c). *Pod Overview.* Haettu 3.3.2019 osoitteesta https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/

The Kubernetes Authors (2019d). *Services.* Haettu 4.3.2019 osoitteesta https://kubernetes.io/docs/concepts/services-networking/service/

The Kubernetes Authors (2019e). *Using a Service to Expose Your App*. Haettu 4.3.2019 osoitteesta https://kubernetes.io/docs/tutorials/kubernetes-basics/expose/expose-intro/

The Kubernetes Authors (2019f). *Volumes.* Haettu 5.3.2019 osoitteesta https://kubernetes.io/docs/concepts/storage/volumes/

The Kubernetes Authors (2019g). *Picking the Right Solution.* Haettu 5.3.2019 osoitteesta https://kubernetes.io/docs/setup/pick-right-solution/

The Kubernetes Authors (2018a). *Concepts.* Haettu 3.3.2019 osoitteesta https://kubernetes.io/docs/concepts/

The Kubernetes Authors (2018b). *Labels and Selectors.* Haettu 5.3.2019 osoitteesta https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Linux.fi (n. d.). *Chroot.* Haettu 8.2.2019 osoitteesta https://www.linux.fi/wiki/Chroot

Martin, N. (2015). *A brief history of Docker Containers' overnight success.* Haettu 8.2.2019 osoitteesta https://searchservervirtualization.techtarget.com/feature/A-brief-history-of-Docker-Containers-overnight-success

Microsoft (2018). *What is Docker?* Haettu 8.2.2019 osoitteesta https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/container-docker-introduction/docker-defined

National Instruments (n.d.). *Vertailu hypervisor-ohjelmien välillä.* Haettu 26.1.2019 osoitteesta http://www.ni.com/white-paper/9629/en/

Portnoy, M. (2016). *Virtualization Essentials.* John Wiley & Sons, Incorporated. Haettu 26.1.2019 osoitteesta https://ebookcentral.proquest.com.ezproxy.hamk.fi/lib/hamk-ebooks/detail.action?docID=4644086

Rathod, H. & Townsend, J. (2014). *Virtualization 2.0 For Dummies.* John Wiley & Sons, Ltd. Haettu 26.1.2019 osoitteesta http://learn.vmware.com/30462_VSOM_Pilot_REG

Rosen, C. (2018). *Docker Swarm vs. Kubernetes: A Comparison.* Blogijulkaisu 15.10.2018. Haettu 27.2.2019 osoitteesta https://www.ibm.com/blogs/bluemix/2018/10/docker-swarm-vs-kubernetes-a-comparison/

Rouse, M. (2018). *What is container (containerization or container-based virtualization)?* Haettu 2.2.2019 osoitteesta https://searchitoperations.techtarget.com/definition/container-containerization-or-container-based-virtualization

Stackshare (2018). *Companies that use Docker & Docker Integrations.* Haettu 8.2.2019 osoitteesta https://stackshare.io/docker/in-stacks

VMWare (2006). *Virtualization Overview.* Haettu 26.1.2019 osoitteesta https://www.vmware.com/pdf/virtualization.pdf

VMWare (2007). *Understanding Full Virtualization, Paravirtualization, and Hardware Assist.* Haettu 26.1.2019 osoitteesta https://www.vmware.com/techpapers/2007/understanding-full-virtualization-paravirtualizat-1008.html

Wright, C. (2017). *Kubernetes vs. Docker Swarm Mindshare.* Haettu 4.3.2019 osoitteesta https://platform9.com/blog/kubernetes-docker-swarm-compared/