Dockerin hyödyntäminen korkeakouluopetuksessa

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Visamäki, Tietojenkäsittelyn koulutusohjelma

Kevät 2018

Timo Anttonen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutusohjelma
Visamäki

Tekijä Timo Anttonen **Vuosi** 2018

Työn nimi Dockerin hyödyntäminen korkeakouluopetuksessa

Työn ohjaaja Erkki Laine

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli löytää Hämeen ammattikorkeakoululle lähestymistapa, joka tukisi BYOD-toimintamallia ja ratkaisisi osittain palvelimien resurssien riittämättömyysongelmat sekä ajasta ja paikasta riippuvaisen opetuksen. Työssä tutkittiin, miten konttiteknologia toimii, ja miten sitä olisi mahdollista hyödyntää opetustilanteissa sekä opiskelijoiden itsenäisessä opiskelemisessa. Käytettäväksi ohjelmistoksi valittiin Docker-tuoteperhe, eikä muihin vastaaviin palveluihin otettu kantaa. Työn toimeksiantajana oli Hämeen ammattikorkeakoulussa lehtorina työskentelevä Tero Keso.

Teoriaosuuteen kerättiin aineistoa internetlähteistä, ja HAMKin e-opetuksen nykyisen tilan sekä tulevaisuuden suunnitelmien selvittämiseksi toteutettiin puolistrukturoitu haastattelu. Käytännön osuudessa käytiin läpi ohjelmiston toimintoja ja komentoja esimerkkien avulla. Työn lopussa arvioitiin ohjelmiston skaalautuvuutta ja soveltuvuutta mahdollisesti tulevaan käyttötarkoitukseen.

Työn tuloksena saatiin luotua levykuva, jota olisi mahdollista hyödyntää oppilaan omatoimisessa oppimisessa ja opetuskäytössä. Oppimisympäristökokonaisuuteen tarvittavat kontit saatiin linkitettyä toisiinsa komentotulkin ja Docker Composen avulla, ja ympäristön toimivuus saatiin todennettua testaamalla. Kontin tila onnistuttiin tallentamaan uudeksi levykuvaksi, joka saatiin siirrettyä sekä käyttöönotettua toisella koneella. Mahdollisten IP-osoiteongelmien vuoksi tutkittiin eri tapoja, miten konteille saadaan määriteltyä tietty IP-osoite ja verkko. Lopuksi onnistuttiin ottamaan käyttöön graafinen käyttöliittymä, jolla on mahdollista hallita kontteja.

Avainsanat Docker, Virtualisointi, Verkko-oppiminen

Sivut 46 sivua, joista liitteitä 2 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Visamäki

Author Timo Anttonen **Year** 2018

Subject Utilizing Docker in University of Applied Sciences

Supervisor Erkki Laine

ABSTRACT

The purpose of this thesis was to find an approach for Häme University of Applied Sciences that would support the BYOD procedure and partly solve the inadequacy of the server resources as well as decrease the time and place dependent teaching. The aims were to examine how container technology works and how it could be utilized in teaching situations and student self-study. Docker was selected as used software, and other similar services were not considered. The client of this thesis was Tero Keso who works as a lecturer at the Häme University of Applied Sciences.

Information for the theoretical part was collected from the internet sources and a semi-structured interview was conducted to clarify the current state of HAMK e-learning and future plans. In the practical part, the functions and commands of the software were researched by means of examples. At the end of the thesis, software's scalability and suitability for the intended purpose was assessed.

As a result of the thesis, Docker-image was created, and it could be utilized in the teaching and students' self-studying. The containers needed for the learning environment were linked to one another by command interpreter and Docker Compose, and the functionality of the environment was verified by testing. The state of the container was successfully saved as a new Docker-image and it was moved and deployed on another machine. The work also examines different ways how it is possible to define the container IP address, because of the possible conflict problems. Finally, a graphical user interface, which could be used to manage containers, were introduced.

Keywords Docker, Virtualization, e-learning

Pages 46 pages including appendices 2 pages

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön tavoitteena on selvittää, miten konttiteknologia toimii ja miten sitä voisi hyödyntää opetustilanteissa sekä opiskelijan itsenäisessä opiskelemisessa. Virtualisoinnin erilaiset toteuttamistavat sekä niiden yhdisteleminen on kasvanut merkittävästi yritysmaailmassa, ja konttipohjaisella virtualisoinnilla on mahdollista tehostaa työskentelyä, mutta yrityksissä ei välttämättä ole kovinkaan suurta tietotaitoa tästä aiheesta. Opinnäytetyöntekijän oppimistavoite on saada paremmat valmiudet toimia virtualisoinnin ja konttiteknologian parissa. Työssä keskitytään konttiteknologiaa hyödyntävistä ohjelmista ainoastaan Docker-tuoteperheeseen, eikä vastaavien ohjelmien paremmuuteen tai huonommuuteen oteta kantaa. Myöskään mahdollisiin laitteisto- tai lisenssivaatimuksiin ei oteta kantaa. Työn toimeksiantajana toimii Hämeen ammattikorkeakoulussa opettajana työskentelevä Tero Keso.

Hämeen ammattikorkeakoulussa ollaan siirtymässä kovaa vauhtia BYOD (Bring your own device) -toimintamalliin. Tällä pyritään saamaan opiskeleminen tehokkaammaksi ja lähemmäksi opiskelijaa sekä kuluja pienemmiksi. Opiskelijoilta vaaditaan osittain jo nyt ja varsinkin tulevaisuudessa omat kannettavat tietokoneet, joilla he voivat suorittaa annettuja tehtäviä. Tämän seurauksena erilaisille virtualisointitavoille ja -toteutuksille olisi suuri kysyntä, mikäli ne saadaan toimimaan oppilailla sujuvasti ja opettajat kokevat saavansa helpotusta opettamiseen sekä niiden tuovan siihen lisäarvoa.

Teoriaosuudessa käsitellään yleisesti virtualisointia, sen hyötyjä ja haittoja sekä erilaisia tapoja toteuttaa sitä. Lisäksi käsitellään konttiteknologian sekä Docker-ohjelmiston historiaa, arkkitehtuuria ja verrataan konttia virtuaalikoneeseen. Teoriaosan lopussa käydään läpi Hämeen ammattikorkeakoulun e-opetuksen tämän hetkinen tilanne ja tulevaisuuden näkymät virtualisoinnin sekä BYODin osalta. Teorian eri osa-alueiden sisällön ymmärtäminen on tärkeää, jotta käsitetään virtualisoinnin tuomat mahdollisuudet ja sen laajuus sekä ollaan tietoisia virtualisointitapojen eroavaisuuksista ja osataan valita käyttökohteesta riippuen paras toteutustapa. Lisäksi teoriaosa antaa hyvän pohjan lopussa tapahtuvalle pohdinnalle, jossa mietitään, kuinka hyvin Docker skaalautuu ja soveltuu käytettäväksi e-opetuksessa.

Käytännön osuudessa tutkitaan Docker-ohjelmiston toimintoja esimerkkien avulla. Työssä käydään läpi levykuvan luominen, joka sisältää Docker-tiedoston rakentamisen ja sen perusteella kuvan luomisen. Lisäksi suoritetaan levykuva kontissa ja luodaan muutosten jälkeen kontista uusi levykuva. Tutkitaan levykuvan ja tietokannan siirtämistä käyttäjältä toiselle sekä sitä, kuinka linkitetään kontteja toisiinsa, jotta ne toimivat yhdessä.

Esimerkit toteutetaan käyttäen komentotulkkia. Yksittäisten komentojen lisäksi käydään läpi myös Docker Compose, joka on työkalu usean kontin määrittelemiseen ja suorittamiseen. Lopuksi tarkastellaan vielä pintapuolisesti, minkälaisia graafisia käyttöliittymiä on konttien hallintaan, ja miten sellainen on mahdollista ottaa käyttöön. Kyseiset osuudet pyritään suorittamaan siten, että niitä voitaisiin hyödyntää samalla tavalla e-opetuksessa.

Opinnäytetyössä pyritään vastaamaan kahteen tutkimuskysymykseen. Miten konttiteknologia toimii, ja miten konttiteknologiaa voidaan hyödyntää BYOD-pohjaisessa opetuksessa tietojenkäsittelyn koulutusohjelmassa?

2 VIRTUALISOINTI

Virtualisointi-termi kuvaa yleisesti palvelupyynnön erottamista palvelun taustalla olevasta fyysisestä laitteistosta (VMware 2007). Termiä käytettiin ensimmäisen kerran jo 1960-luvulla IBM:n 360-laitteistoperheeseen kuuluvan 65-mallin yhteydessä, mutta ensimmäinen virtualisointitekniikan konkreettinen toteutus saavutettiin IBM:n toimesta CP-40-laitteella vuonna 1967. (StoreCraft n.d.)

Teknologian kehityksen edistymistä nopeutti pöytätietokoneiden ja x86-palvelimien kaupallistuminen 1980-luvulla, minkä seurauksena laitteistojen määrä yrityksissä ja kuluttajilla kasvoi huomattavasti. Pöytätietokoneiden markkinoiden kasvu johti virtualisoinnin tarpeeseen ja kehittäjät alkoivat välittömästi panostaa teknologian kehitykseen. Vuonna 1987 Locus Computing Corporation julkaisi erittäin alkeellisen Merge-virtualisointiohjelman, jonka avulla voitiin suorittaa MS-DOS-käyttöjärjestelmää SCO-UNIX-ympäristössä. (StoreCraft n.d.)

VMware sai vuonna 1998 selville, miten virtualisoida x86-arkkitehtuurin alustaa, joka vaikutti yhdessä välissä jopa mahdottomalta. Ratkaisuksi muodostui yhdistelmä binäärikäännöksiä sekä suoraan prosessoriin tehtyjä toteutuksia, joka lopulta mahdollisti usean vieraan käyttöjärjestelmän suorittamisen täysin eristyksissä yhdellä tietokoneella. Yritykset saavuttivat kehitetyllä toteutuksella suuria säästöjä, mikä nopeutti virtualisoinnin siirtymistä pöytäkoneista palvelinkeskuksiin. (VMware 2007.)

Koska kuluttajien pöytäkoneiden ja palvelimien prosessorikapasiteetti on kasvanut vuosi vuodelta, on virtualisointi osoittautunut tehokkaaksi teknologiaksi esimerkiksi ohjelmistojen kehittämiseen, testauksen yksinkertaistamiseen ja kulujen karsimiseen. Täydellinen abstraktio käyttöjärjestelmästä ja laitteiston sovelluksista sekä niiden eristäminen siirrettäviin virtuaalikoneisiin on mahdollistanut virtuaalisia infrastruktuuritoimintoja, jotka eivät yksinkertaisesti olisi mahdollisia pelkän normaalin laitteiston kanssa. (VMware 2007.)

2.1 Virtualisoinnin hyödyt

Virtualisointi tarjoaa monia etuja, joiden avulla voidaan vähentää kustannuksia ja kasvattaa työn tehokkuutta organisaatioissa. Palvelinvirtualisointitekniikan käyttöönottaminen organisaatiossa auttaa vähentämään infrastruktuurikustannuksia. Kustannusten vähenemiseen liittyviä osatekijöitä ovat myös esimerkiksi laitteistot, ympäristökustannukset, hallinnointi ja palvelininfrastruktuurin hallinta. Virtualisointi mahdollistaa esimerkiksi useiden yksittäisten työasemien käyttämisen yhdellä korkean suorituskyvyn laitteella, minkä seurauksena saavutetaan suurempi hyötykäyttöaste laitteistolla. (UK Essays 2015.)

Virtualisoinnin hyötyä kuvastaa esimerkiksi se, että jos perinteisellä tavalla toimiessa käytettäisiin viittä (5) fyysistä konetta ja saavutettaisiin yhdellä koneella ainoastaan noin 10 %:n hyötykäyttöaste. Virtualisoinnin avulla päästään yhdellä koneella jo 50 %:n hyötykäyttöasteeseen, kun koneen resurssit voidaan hyödyntää usealle työasemalle. Säästöjä saadaan lisäksi koneiden tehon käytön suhteen, pienemmällä tilan tarpeella ja tiloissa vaadittavien jäähdytysjärjestelmien pienenemisellä laitteistomäärän supistuessa. (UK Essays 2015.)

Kustannussäästöjen lisäksi virtualisointi antaa organisaatiolle paremman mahdollisuuden reagoida muuttuviin markkinaolosuhteisiin vähentämällä uusien palvelimien ja työasemien käyttöönottoon kuluvaa aikaa. Koska virtuaalikoneet ovat lähinnä tiedostoja eikä fyysisiä laitteita, voidaan niitä kopioida, korvata ja ottaa käyttöön paljon nopeammin kuin perinteisiä fyysisiä laitteita. Palvelimien konsolidaatio (consolidation) on yksi suurimmista syistä käyttää virtualisointia. Siinä palvelimien laskentaresurssit voidaan jakaa useiden sovellusten ja palveluiden välillä samanaikaisesti. Tämän avulla organisaatiot voivat poistaa useita yksittäisiä palvelimia ja maksimoida käytettävissä olevat resurssit käyttämällä useita eri sovelluksia samalla palvelimella. (UK Essays 2015.)

Fyysisen palvelimen kaatuessa uudelleenkäynnistysaika riippuu useista muuttujista. Tällaisia muuttujia ovat esimerkiksi se, onko palvelin, jossa varmuuskopiot sijaitsevat helposti saatavilla ja kuinka ajan tasalla varmuuskopio on. Virtuaalisten palvelimien avulla voidaan palauttaa kaatunut palvelin muutamassa minuutissa tuntien sijaan. Virtualisoinnissa on mahdollista käyttää tilannevedosta (snapshot), jonka avulla virtuaalipalvelin saadaan nopeasti pystyyn vähäisellä alhaallaoloajalla (downtime). (AgileIT 2015.)

Palvelinten virtuaalisella uudelleenjärjestelyllä parannetaan häiriötilanteisiin varautumista. Koska yritysten ei tarvitse virtualisoinnin ansiosta pitää enää yhtä montaa fyysistä palvelinta, voidaan ensisijaisesta palvelimesta pitää kaksoiskappale rinnakkain käytössä olevan palvelimen kanssa ja ottaa se käyttöön mahdollisten häiriötilanteiden ilmestyessä. Tällä tavalla voidaan taata tuotannon jatkuvuus ilman suurempia keskeytyksiä. (AgileIT 2015.)

Lisäksi virtualisoinnin katsotaan olevan paras ympäristö ohjelmoijille kehittää ja testata sovelluksia, kun koodi voidaan kirjoittaa ja sen jälkeen käyttää eri käyttöjärjestelmässä. Tätä kutsutaan segregaatioratkaisuksi. (UK Essays 2015.) Virtualisoinnilla pystytään vähentämään hieman myös toimittajariippuvuutta, koska tarvittavaa laitteistoa ei ole niin paljoa ja se tarjoaa valinnanvapautta, kun vaihtoehtoja alustaksi on useita. (AgileIT 2015.)

2.2 Virtualisoinnin haitat

Virtualisoinnin haitat ovat enimmäkseen sellaisia, jotka voivat ilmetä missä tahansa projektissa, jossa siirrytään uuteen tekniikkaan. Huolellisella suunnitelulla ja asiantuntevalla toteutuksella kaikki haitat voidaan kuitenkin ratkaista tai minimoida. (Milner 2015.)

Ensimmäisenä haittana organisaatioille tulee usein mahdolliset etukäteiskustannukset. Virtualisointiohjelmistoon ja mahdolliseen lisälaitteistoon voidaan joutua sijoittamaan rahaa, ennen kuin virtualisointi on mahdollista toteuttaa. Ohjelmistojen lisenssien kanssa voi ilmetä haasteita, mutta tämä ongelma on kuitenkin vähenemässä. Useimmat ohjelmistotoimittajat ovat lisänneet tukea ja alkaneet hyväksyä ohjelmistojen virtualisointia. On kuitenkin tärkeää tarkistaa ja sopia toimittajien kanssa selkeästi se, miten he näkevät ja hyväksyvät ohjelmiston käyttämisen virtualisoidussa ympäristössä. (Milner 2015.)

Virtuaaliympäristön toteuttaminen ja hallinta edellyttävät IT-henkilökunnalta virtualisoinnin asiantuntemusta, minkä vuoksi uuden järjestelmän käyttöönotto vaatii usein lisäkoulutusta. Koska virtualisointi ei ole kaikkein helpoin opittava teknologia, joillekin henkilöille uuden oppiminen saattaa muodostua liian suureksi haasteeksi. Monille yrityksille virtualisoinnin haittojen suhde hyötyihin jää kuitenkin sen verran pieneksi, että teknologiaan kannattaa siirtyä. (Milner 2015.)

3 VIRTUALISOINTITAVAT

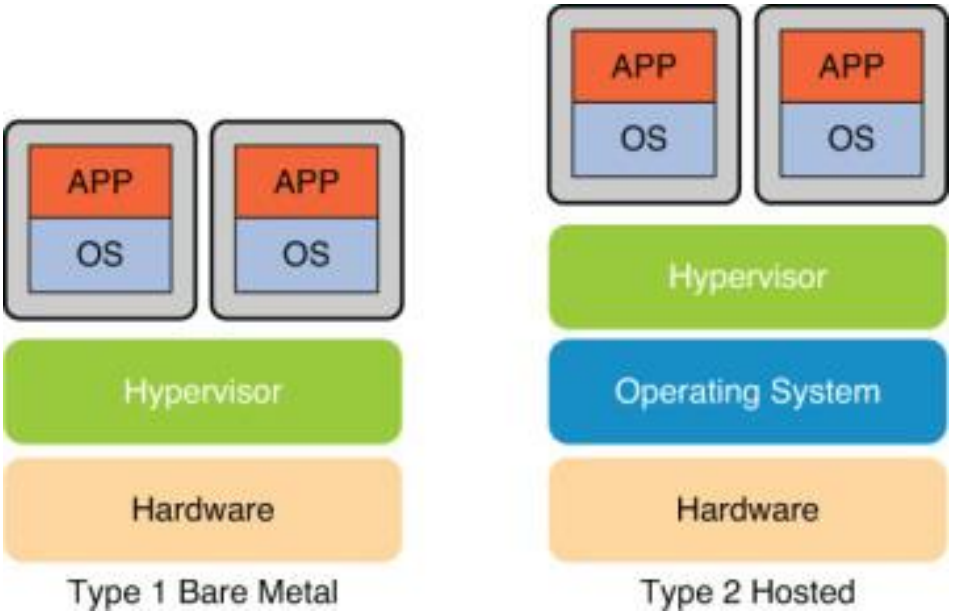
Erilaisia virtualisointiratkaisuja on olemassa useita ja ne erottuvat toisistaan toiminnallisuuksien ja teknistenratkaisujen osalta. Tässä luvussa käsitellään muutamaa yleisintä virtualisoinnin toteutustapaa.

3.1 Täysvirtualisointi

Täysvirtualisoinnissa virtualisoidun koneen käyttöjärjestelmä on kokonaan irrotettu perustana toimivasta laitteistosta virtualisointikerroksella, eli virtuaalikoneen näkemä laitteisto on näennäinen. Virtuaalikoneen käyttämä käyttöjärjestelmä ei ole tietoinen siitä, että sitä virtualisoidaan eikä se yleensä vaadi ylimääräisiä modifikaatioita asennuksen yhteydessä. Täysvirtualisoinnilla saadaan paras eristys ja turvallisuus virtuaalikoneille verrattuna muihin virtualisointiratkaisuihin. Se myös yksinkertaistaa koneiden migraatioita ja siirrettävyyttä. (VMware 2007.)

Virtualisointikerroksessa käytetään sitä varten luotua ohjelmistoa, jota kutsutaan nimellä hypervisor. Hypervisorista on olemassa kaksi eri versiota luokka 1 ja luokka 2 (Kuva 1). Luokan 1 hypervisor tunnetaan myös paljasmetalli (bare-metal) toteutuksena, koska se sijoittuu suoraan fyysisen laitteiston päälle käyttämättä mitään käyttöjärjestelmää siinä välissä. Ohjelmisto toimii alustana virtuaalikoneille ja suorassa vuorovaikutuksessa perustana olevan laitteiston, kuten palvelimen resurssien kanssa. Luokan 1 hypervisor on paljon nopeampi ja turvallisempi kuin luokan 2, koska siinä ei tule laitteiston ja sen välille ylimääräisiä kerroksia. (securitywing 2014.)

Luokan 2 toteutuksessa hypervisor toimii käyttöjärjestelmän päällä, joka on asennettu sen ja fyysisen laitteiston väliin. Etuina luokan 2 toteutuksessa on isäntäkäyttöjärjestelmän huolehtiminen kaikista laitteistoista ja laajemman tuen tarjoaminen. Lisäksi luokan 2-mallin toteuttaminen on paljon helpompaa verrattuna luokan 1-malliin. Luokan 2-mallissa on kuitenkin haittapuolena suurempi mahdollisuus vikatilanteisiin, koska isäntäkäyttöjärjestelmän häiriöt voivat vaikuttaa suuresti myös luotuihin virtuaalikoneisiin. Esimerkiksi kun isäntäkäyttöjärjestelmä tarvitsee uudelleen käynnistää, niin myös kaikki virtuaalikoneet käynnistetään uudelleen. (securitywing 2014.)



Kuva 1. Luokan 1 ja 2 hypervisorin eroavaisuus (Pearson 2014).

Hypervisorin ansiosta palvelimella voi olla useita virtuaalikoneita ja ne toimivat täysin itsenäisesti sekä tiedostamatta toistensa olemassa oloa. Jokaisella virtuaalikoneella käytetään omaa käyttöjärjestelmää, joka voidaan valita mieltymysten sekä käyttötarkoitusten mukaan. Esimerkiksi ensimmäisellä virtuaalikoneella voidaan käyttää Linux- ja toisella Windowskäyttöjärjestelmää. (HowStuffWorks n.d.)

Hypervisor seuraa palvelimen resursseja ja ohjaa niitä virtuaalikoneiden käyttöön, kun koneissa suoritetaan sovelluksia. Ohjelmisto käyttää myös itse palvelimen resursseja, kuten prosessointitehoa ja muistia, mikä pitää huomioida jaettaessa resursseja virtuaalikoneille. Jos palvelimelle luodaan liian monta virtuaalikonetta, voi resurssipula johtaa koko palvelimen sekä virtuaalikoneiden käyttämien sovellusten hidastumiseen suorituskyvyn laskiessa. (HowStuffWorks n.d.)

3.2 **Osittainen virtualisointi**

Osittaisessa virtualisoinnissa virtuaalikoneille ei yritetä jäljitellä laitteistoympäristöä, vaan pikemminkin hypervisor koordinoi koneiden pääsyä sen alla olevan palvelimen resursseihin (HP 2011). Kyseisessä virtualisointitavassa virtuaalikoneet ovat tietoisia toisistaan ja vaatimuksista, joita muut koneet asettavat palvelimelle. Tämän johdosta osittaisessa virtualisoinnissa ei tarvita yhtä paljon prosessointitehoa ja koko järjestelmä toimii yhdessä yhtenäisenä yksikkönä. (HowStuffWorks n.d.)

Osittaisen virtualisoinnin hyötynä on parempi suorituskyky verrattuna täysvirtualisointiin, koska palvelupyynnöt menevät nopeammin perille fyysiselle laitteistolle. Suurin haittapuoli on se, että vieraskäyttöjärjestelmien ydintä joudutaan usein muokkaamaan, jotta kommunikointi hypervisorin

kanssa onnistuisi. Koska kaikkien käyttöjärjestelmien ydintä ei ole mahdollista muokata, rajoittaa se valintaa käytettävien käyttöjärjestelmien osalta. (securitywing 2014.)

3.3 **Käyttöjärjestelmävirtualisointi**

Käyttöjärjestelmätason virtualisoinnissa ei käytetä ollenkaan hypervisoria, vaan virtualisointi toteutetaan isäntänä toimivassa käyttöjärjestelmässä, joka suorittaa toiminnot hypervisorin kaltaisesti. Suurin rajoitus tässä menetelmässä on se, että jokaisen virtuaalikoneen tulee käyttää samaa käyttöjärjestelmää, vaikka ne ovatkin riippumattomia toisistaan. Tätä kutsutaan homogeeniseksi ympäristöksi. (HowStuffWorks n.d.)

3.4 **Sovellusvirtualisointi**

Sovellusvirtualisoinnista käytetään myös nimitystä sovelluspalvelun virtualisointi. Sovellusvirtualisointi hämää tietokoneen toimimaan ikään kuin sovellus olisi käytössä paikallisessa tietokoneessa. Todellisuudessa se kuitenkin oikeasti toimii esimerkiksi palvelimella tai virtuaalikoneella käyttäen sen käyttöjärjestelmää. (technopedia n.d.)

Perinteisessä ohjelmistotilanteessa ohjelma asennettaisiin käyttäjän paikallisen tietokoneen kiintolevylle ja se olisi näin ollen täysin riippuvainen kyseisen koneen käyttöjärjestelmästä. Sovellusvirtualisoinnissa käytetään teknologiaa, jossa käytettävä ohjelmisto käännetään erillisen ohjelman avulla itsenäisiin paketteihin ja erotetaan käytettävästä käyttöjärjestelmästä. (Information Technology Group 2015.) Luodut ohjelmistopaketit ovat pohjimmiltaan valmiita toimintaympäristöjä, eivätkä ne näin ollen vaadi erillistä asentamista käyttäjän tietokoneelle. Virtualisoidut ohjelmat voivat toimia normaalilla tavalla asennettujen ohjelmien rinnalla sekä keskenään rinnakkain ilman ristiriitoja. (AnandTech 2008.)

Sovellusvirtualisoinnilla on mahdollista saavuttaa monenlaisia hyötyjä, mutta keskeisin niistä on kustannusten väheneminen. Yrityksellä on mahdollisuus säilyttää kaikki ohjelmistot useissa tietokoneissa ja laitteissa, mutta niiden hallinnoimiseen kuluva aika pienenee. Samaa ohjelmaa on mahdollista käyttää millä tahansa koneella riippumatta käytettävästä käyttöjärjestelmästä, ja käyttäjän on mahdollista suorittaa samanaikaisesti eri versioita ohjelmasta. (Information Technology Group 2015.)

Ohjelmiston erotus käyttöjärjestelmästä antaa yrityksille myös lisää tietoturvaa. Mikäli sovellus on vaarantunut tai saastunut tietoturvaongelman vuoksi, niin koneen käyttöjärjestelmä ja kiintolevyllä olevat tiedostot eivät välttämättä vaarannu samanaikaisesti. (Information Technology Group 2015.)

Sovellusten virtualisointi sisältää kuitenkin myös haasteita, eivätkä kaikki sovellukset sovellu virtualisointiin. Tästä esimerkkinä ovat runsaasti grafiikkaa sisältävät sovellukset, jotka saattavat häiriintyä renderöintiprosessissa. Lisäksi käyttäjät tarvitsevat kiinteän yhteyden palvelimeen sovellusten käyttämiseksi. (TechTarget 2011.)

3.5 **Sovellusten suoratoisto**

Sovelluksen suoratoisto on ”on demand”-toimitusmalli, joka hyödyntää sitä, että useimmat sovellukset vaativat vain pienen osan ohjelmakoodista toimintojen suorittamiseen. Suoratoistossa hyödynnetään reaaliaikaista suoratoisto-protokollaa (RTSP) ja sitä käytetään usein yhdessä työpöytävirtualisoinnin kanssa. (TechTarget 2009.)

Suoratoistossa virtualisoidut sovellukset toimivat käyttäjän paikallisella koneella. Kun käyttäjä käynnistää sovellusta, lähettää paikallinen kone palvelimelle pyynnön, jonka avulla tietokone osaa ladata sovelluksen käynnistämiseen tarvittavat komponentit. Tavallisesti käynnistämiseen tarvitaan vain 10 prosenttia koko sovelluksen komponenteista ja loput komponentit ladataan koneelle tarpeen mukaan. (TechTarget 2009.)

Komponentit on jaettu lohkoihin käyttötarkoituksen mukaan. Tällaisia lohkoja ovat esimerkiksi käynnistys-, ennakoivat, ja vaadittavat lohkot. Käynnistyslohkojen lähettäminen palvelimelta asiakkaalle aloitetaan, kun käyttäjä napsauttaa pikakuvaketta työpöydällä. Sovellus käynnistyy, kun käynnistyslohkot ovat saatavilla paikallisesti ja alkaa seuraavaksi lataamaan ennakoivia lohkoja. Ennakoivat lohkot on ennalta määritelty käyttäjän todennäköisten seuraavien tarpeiden mukaan. Mikäli käyttäjä napsauttaa sovelluksen toimintoa, jonka tarvittavia lohkoja ei ole vielä lähetetty, niin kyseisten lohkojen lähettäminen aloitetaan. (TechTarget 2009.)

Verkonvalvoja voi sallia lohkojen tallentamisen paikallisesti, jotta sovellus on käytettävissä, vaikka käyttäjä olisi irrotettuna verkosta. Jos verkonvalvoja asentaa sovellukseen korjaustiedoston tai uuden version, niin päivityksiä sisältävät lohkot toimitetaan seuraavan kerran, kun käyttäjä käynnistää sovelluksen. (TechTarget 2009.)

4 DOCKER

Hypervisorit ja virtuaalikoneet ovat vain yksi lähestymistapa virtualisoinnin toteuttamiseen. Konttivirtualisointitekniikka on nopeasti noussut tehokkaaksi ja luotettavaksi vaihtoehdoksi perinteiselle virtualisoinnille tarjoamalla uusia ominaisuuksia ja suhtautumistapoja kehittäjille ja datakeskusten ylläpitäjille. (TechTarget 2017.)

Docker on avoimen lähdekoodin projekti, jonka avulla sovellukset voidaan automatisoida siirrettäviksi ja itsenäisiksi ohjelmistopaketeiksi. Näistä ohjelmistopaketeista käytetään nimitystä kontti ja ne voivat toimia pilvessä tai paikallisesti. Docker on myös yritys, joka edistää ja kehittää tätä teknologiaa. Yritys toimii myös yhteistyössä pilvi-, Linux- ja Windows-toimittajien kanssa. (Microsoft 2017b.)

4.1 Konttiteknologian historia

Docker on ollut avainasemassa konttiteknologian käyttöönotossa ja yleistymisessä, mutta se ei kuitenkaan itse keksinyt konseptia kyseiselle teknologialle. Teknologian kehittymisen aikajana havainnollistetaan kuvassa 2. Ensimmäiset konttiteknologian versiot sijoittuvat jo vuodelle 1979, jolloin Unixin seitsemännessä versiossa (Unix V7) käytettiin chroot-toimintoa. Chroot on Unix-käyttöjärjestelmän toiminto, joka muuttaa sovelluksen juurihakemiston ja sen alihakemistoja näennäisesti uuteen tiedostojärjestelmään. Sovellus ei pääse käsiksi tiedostoihin määrätyn hakemistopuun ulkopuolelle, kun se suoritetaan tällaisessa muunnetussa ympäristössä. Muunnetusta ympäristöstä käytetään nimitystä chroot-vankila, ja ominaisuudella oli tarkoituksena tarjota eristetty levytila jokaiselle prosessille. (plesk 2016.)

Vuonna 2000 konteista alkoi syntyä merkittävä teknologia FreeBSD-vankiloiden ansiosta. Tarve FreeBSD-vankiloille tuli yritykseltä R&D Associates, joka toimi jaettujen alustojen palveluntarjoajana. Yrityksen omistaja Derrick T. Woolworth tahtoi luoda turvallisuuden sekä hallinnoimisen helpottamisen vuoksi selkeän erottelun asiakkaille tarjottavien ja omien käytössä olevien palveluiden välille. (plesk 2016.) FreeBSD-vankilamekanismi on käyttöjärjestelmätason virtualisointitoteutus, jonka avulla voidaan jakaa FreeBSD-pohjainen tietojärjestelmä useisiin itsenäisiin minijärjestelmiin. Näistä minijärjestelmistä käytetään nimitystä vankila. Vankilat kehittivät chroot-konseptia eteenpäin usealla tavalla. Chroot-ympäristössä prosessit on rajoitettu käyttämään ainoastaan tiettyä levyjärjestelmän osaa, johon ne voivat päästä käsiksi. Loput järjestelmän resursseista on jaettu Chroot-ympäristöissä olevien prosessien kesken. Vankilat laajensivat konseptin koskemaan myös muita resursseja virtualisoinnin avulla. (freeBSD n.d.)

Ensimmäinen kaupallinen konttiteknologia kehitettiin Virtuozzon toimesta vuonna 2001. Sitä käyttää nykyään yli 700 palveluntarjoajaa ja yritystä viiden miljoonan virtuaaliympäristön ylläpitämiseen. Virtuozzo on ollut myös merkittävä tukija ja kehittäjä lukuisissa avoimen lähdekoodin projekteissa kuten OpenVZ, CRIU, Docker ja OpenStack. (plesk 2016.)

Läpi 2000-luvun alkupuolen julkaistiin myös useita muita konttiteknologian ratkaisuja kuten Linux VServer (2001), Oraclen Solaris Containers (2004), Parallelsin OpenVZ (2005) ja Googlen Process Containers (2006). Linux VServeriä voidaan käyttää järjestelmän resurssien, kuten tiedostojärjestelmän, verkko-osoitteiden ja muistin jakamiseen. VServerille julkaistaan vielä kokeellisia päivitystiedostoja, mutta viimeinen vakaa päivitys julkaistiin vuonna 2006. Oraclen Solarin Containers mahdollisti järjestelmän resurssienhallinnan ja rajaerottelun yhdistämisen, jonka avulla pystyttiin luomaan tilannevedoksia ja kloonaamaan ZFS-tiedostojärjestelmää. OpenVZ tarjosi Linuxille käyttöjärjestelmätason virtualisointitekniikan, joka käyttää paranneltua Linux-ydintä virtualisointiin, eristämiseen ja resurssienhallintaan. Googlen Process Containers kehitettiin prosessien käyttämien resurssien, kuten muistin ja verkon eristämiseen muista prosesseista. Vuonna 2007 Process Container nimettiin uudelleen Control Groupiksi, jotta vältettäisiin sekaannus kontti-termin kanssa. (plesk 2016.)

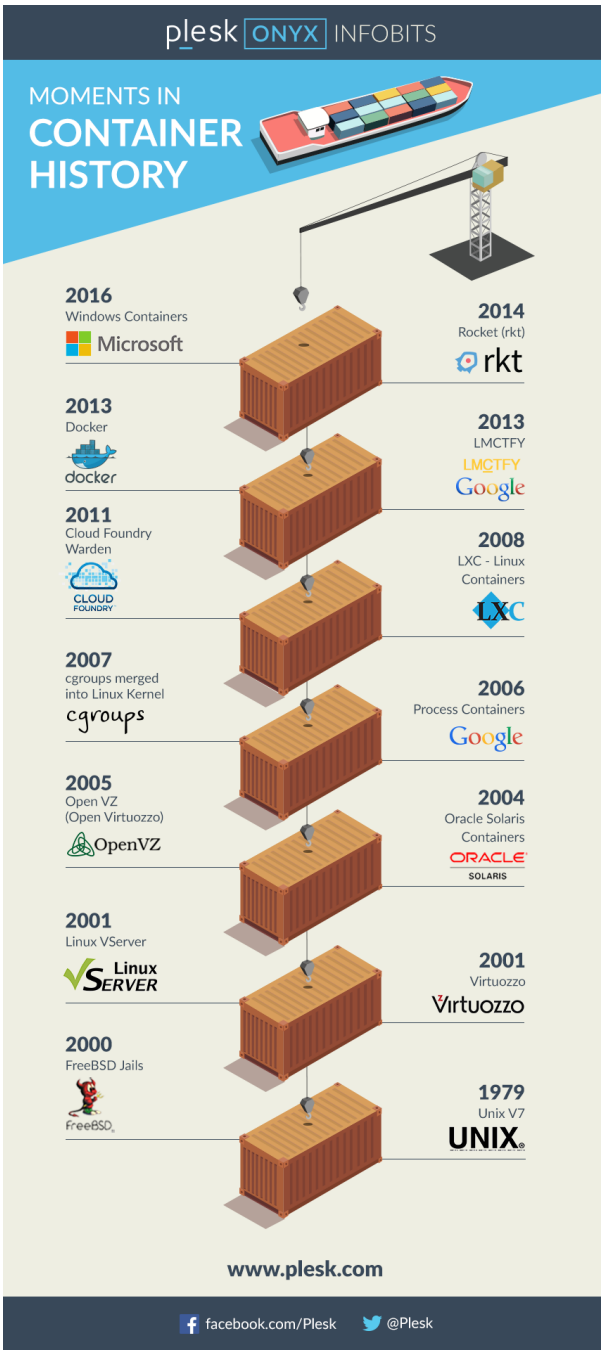
Vuonna 2008 toteutettiin ensimmäinen Linux-konttien hallintaohjelman täydellinen toteutus nimeltä Linux Containers eli LXC. Siinä hyödynnetään Control Group ja Namespace -ominaisuuksia, joita myös Dockerissa käytetään. LXC tarjosi kielituen usealle eri ohjelmointikielelle, kuten Python3:lle, Python2:lle, Lua:lle ja Ruby:lle. 2010-luvulla kehitettiin vielä CloudFoundryn Warden (2011) ja Googlen LMCTFY (2013), mutta vasta vuonna 2013 konttiteknologia alkoi tulla suosituksi, kun Docker julkisti oman versionsa teknologiasta. (plesk 2016.)

Docker julkaistiin dotCloudin toimesta avoimen lähdekoodin projektina vuonna 2013. Se tukeutuu Linux-ytimen ominaisuuksiin, kuten Namespace ja Control Group (cgroup), joita käytetään resurssien eristämiseen ja sovellusten paketoimiseen tarvittavien riippuvuuksien kanssa. Tämä riippuvuuksien pakkaaminen mahdollistaa sovelluksen toimimisen sen odotusten mukaisesti eri käyttöjärjestelmissä ja helpottaa siirrettävyyttä huomattavasti. Tämän johdosta kehittäjä voi kirjoittaa sovelluksen millä tahansa ohjelmointikielellä ja siirtää sen helposti laitteesta toiseen riippumatta käytettävästä käyttöjärjestelmästä. Juuri helpon siirrettävyyden johdosta kehittäjät ja järjestelmien ylläpitäjät kiinnostuivat Dockerista. (TechTarget 2015.)

Opetusohjelma Dockerin käytöstä julkaistiin elokuussa 2013, ja kuukausi sen jälkeen Docker sanoi, että sitä oli kokeillut 10 000 kehittäjää. Vuoden kuluessa Red Hat ja Amazon lisäsivät kaupallista tukea Dockerille, vaikka Dockerin johtohenkilöt varoittivat käyttäjiä vielä olemaan käyttämättä tuotetta tuotannossa. Kun Docker julkaisi 1.0-versionsa kesäkuussa 2014,

oli sitä ladattu jo 2,75 miljoonaa kertaa, ja nyt määrä on jo yli 100 miljoonaa. Dockerin odottamaton menestys on herättänyt sen verran huomiota, että konttivirtualisointiin on kehitetty jo useita kilpailevia lähestymistapoja. Yksi tällainen on CoreOS:n vuonna 2014 kehittämä Rocket (rkt). (TechTarget 2015.)

Vuonna 2015 Microsoft ryhtyi lisäämään konttitukea Windows-pohjaisille sovelluksille nimellä Windows Containers. Microsoft ilmoitti vuonna 2016, että julkaistu Windows Server 2016 -ohjelmisto suorittaa jo natiivisti konteja Docker Enginen avulla Windowsissa. (plesk 2016.)



Kuva 2. Konttiteknologian kehitys (plesk 2016).

4.2 **Käyttökohteet ja hyödyntäminen**

Sovellusten laittamisella konttiin on mahdollista saavuttaa monia yksittäisiä hyötyjä, mutta koska käyttötapausten välillä on usein vahva yhteys toisiinsa, voidaan Dockerista hyötyä monella tavalla tuotannossa niiden yhteisvaikutuksen ansiosta. Käyttötapausten suhteita ja niiden kytkeytymistä toisiinsa tuodaan esille kuvassa 3. (Container Solutions 2015.)

Kontit ja isäntäjärjestelmä jakavat ytimen, mutta muuten konteissa pyörivät sovellukset on eristetty isäntäjärjestelmästä hyvin. Sovellukset, joita ei aikaisemmin ollut mahdollisuutta suorittaa samassa järjestelmässä verkkoasetusten, prosessientilan tai tiedostojärjestelmän ristiriitojen takia, voivat nyt jakaa fyysisen tai virtuaalikoneen riippumatta siitä, ovatko ne täysin erilaisia sovelluksia tai koostuvatko ne useista eri instansseista. (Container Solutions 2015.)

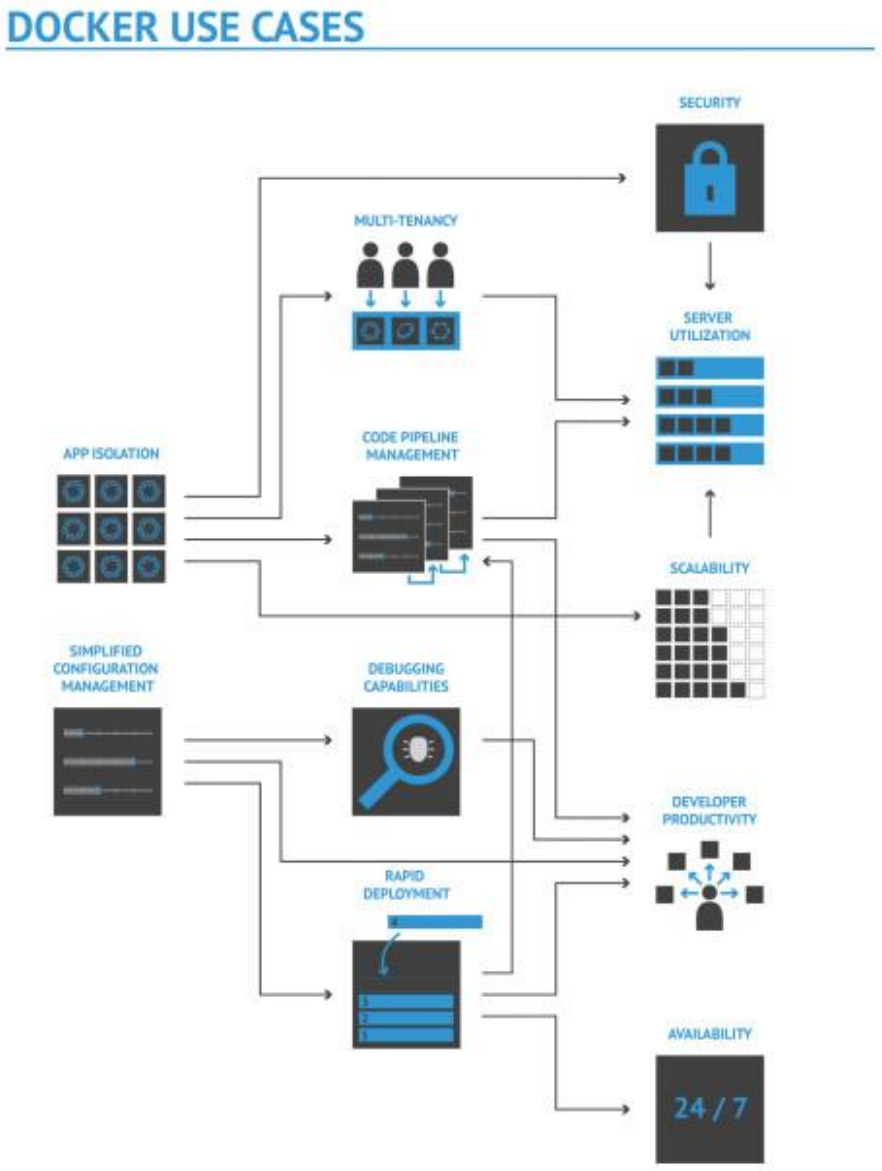
Konteissa olevat vakioasetukset tai aikaisemmin määritellyt asetukset voidaan määritellä uudestaan massatuotantona tai pelkästään yhtä konttia kohden. Tämä nopeuttaa koko prosessia tuotannossa, kun konteissa voidaan tarjota eri vaiheissa erilaisia asetuksia eikä itse levykuvaa tarvitse muokata. Konttien kanssa on mahdollista käyttää myös erillisiä palveluja kuten etcd:tä tai Vault:ia, jotka liittyvät salausavaimien hallintaan. (Container Solutions 2015.)

Eristettyjen sovellusten avulla mahdollistetaan korkeampi laitteistojen resurssien käyttäminen. Tästä voi olla keskisuurelle tai suurelle yritykselle huomattavia ekologisia ja taloudellisia hyötyjä. Suuntaviivana voidaan pitää 5—10 kertaa suurempaa keskimääräistä käyttöastetta laitteistolla, mikäli kontteja käytetään yhdessä resurssien hallintatyökalujen kanssa. (Container Solutions 2015.)

Kehittäjien tuottavuus ja virheenkorjausmahdollisuudet kasvavat, kun sama sovellusympäristö voidaan toistaa useassa eri paikassa. Dockerin erityisen hyödyllinen ominaisuus on mahdollisuus tallentaa kontin tila commit-toiminnolla uuteen levykuvaan ja suorittaa sekä tutkia sitä myöhemmin eri toimintaympäristössä. Kehittäjä pääsee nopeasti käynnissä olevan kontin sisään käyttämään virheenkorjaus- ja muita kehitystyökaluja, kuitenkaan vaarantamatta tavanomaista vaikeasti palautettavaa testausympäristöä. Käyttökelvoton kontti voidaan nopeasti poistaa ja alkuperäinen versio palauttaa, jolloin säästyy huomattavasti aikaa. (Container Solutions 2015.)

Innovaatiot vaativat usein nopeita testauksia ja käyttöönottoja. Dockerin tapa jakaa levykuvia ja varastoida niitä paikallisesti nopeuttaa sovelluksen käyttöönottoa ja mahdollistaa erittäin nopean tehtävän suorittamisen. Mahdollisuus käynnistää kontteja millisekunneissa verrattuna virtuaalikoneiden sekunteihin tai minuutteihin luo huomattavan eron sovellusten

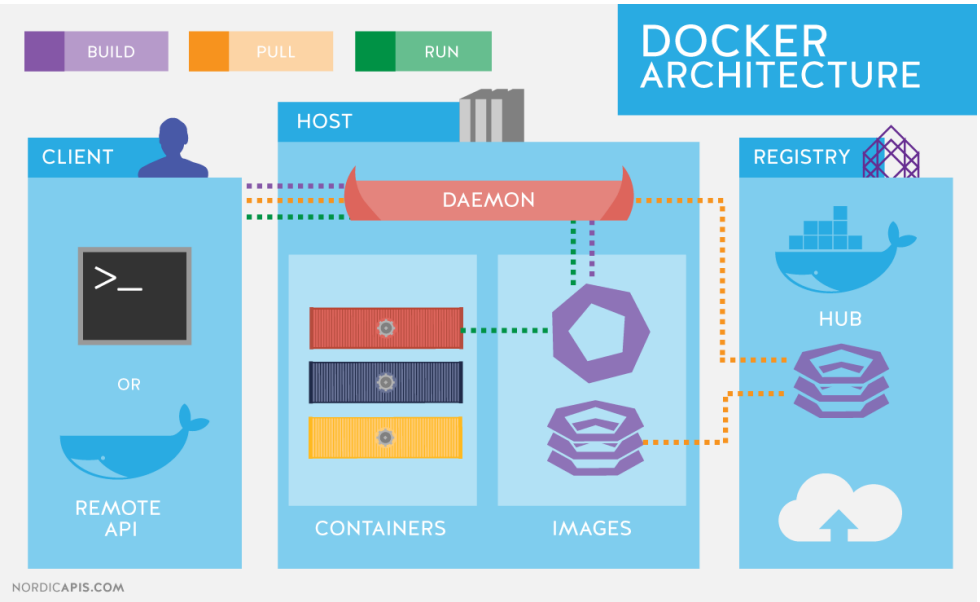
saatavuuteen tilanteissa, joissa laitteisto vikaantuu, sovellus kaatuu tai operaattorin tarvitsee resetoida palvelin. (Container Solutions 2015.)



Kuva 3. Dockerin käyttötapauksia ja niiden kytkeytyminen toisiinsa (Container Solutions 2015).

4.3 **Arkkitehtuuri**

Docker käyttää client-server arkkitehtuurimallia, joka sisältää Docker-asiakasohjelman, -palvelun ja -rekisterin, joilla jokaisella on oma roolinsa Dockerin toiminnassa. Yhdessä nämä komponentit muodostavat Dockerin arkkitehtuurikokonaisuuden (Kuva 4). (Docker 2018.)



Kuva 4. Dockerin arkkitehtuuri (ApacheBooster 2017).

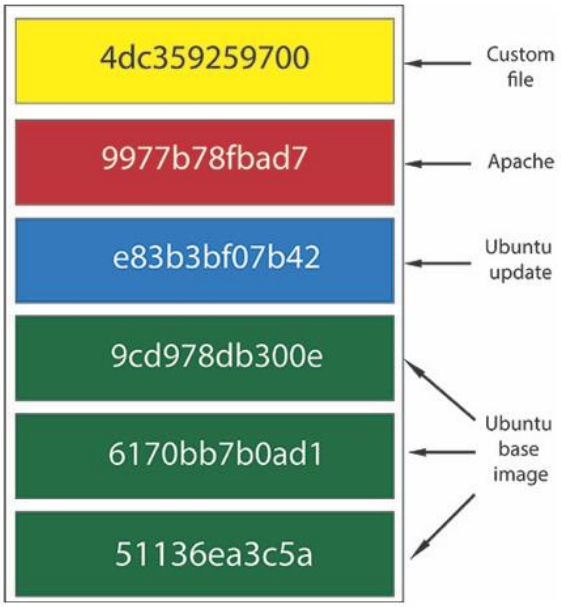
4.4 Palvelu ja asiakasohjelma

Docker-palvelu (Docker-daemon tai Docker Engine) toimii isäntäkoneella ja vastaa konttien sekä levykuvien rakentamisesta, suorittamisesta, lataamisesta ja jakamisesta. Käyttäjä ei voi olla suoraan vuorovaikutuksessa palvelun kanssa vaan siihen vaaditaan avuksi Docker-asiakasohjelma (Docker-client). Se toimii käyttäjän ja palvelun välisenä rajapintana, joka käsittelee käyttäjän komennot ja kommunikoi edestakaisin palvelun kanssa. Näin käyttäjä voi käyttää esimerkiksi komentotulkkia komentojen suorittamiseen. (Eduonix 2016.)

4.5 Levykuva

Docker-levykuva (Docker-image) on ainoastaan luettavassa muodossa oleva tiedosto, joka sisältää tarvittavat tiedot kontin luomiseen. Tarvittavia tietoja ovat muun muassa kaikki riippuvuussuhteet sekä käyttöönotto- ja suoritusasetukset, jotka tarvitaan kontin suorittamista varten. Usein levykuva on koostettu pohjakuvasta ja sovelluskerroksista, jotka päällekkäisinä kerroksina muodostavat lopulta kontin tiedostojärjestelmän (Kuva 5). (Microsoft 2017a.)

Levykuvan rakentamiseen käytetään Docker-tiedostoa (Dockerfile), joka on tekstitiedosto. Se sisältää kaikki konfigurointitiedot ja komennot, joita tarvitaan levykuvan kokoamiseen. Tiedoston avulla palvelu osaa automaattisesti rakentaa levykuvan, kun käyttäjä syöttää rakentamiseen tarkoitetun komennon. Jokaisella rakennuskerralla tiedosto luo uuden kerroksen levykuvaan ja ottaa mukaan ainoastaan tehdyt muutokset alemmista kerroksista. Tämän vuoksi levykuvat ovat niin pieniä ja nopeita verrattuna muihin virtualisointiteknologioihin. (InfoWorld 2016.)



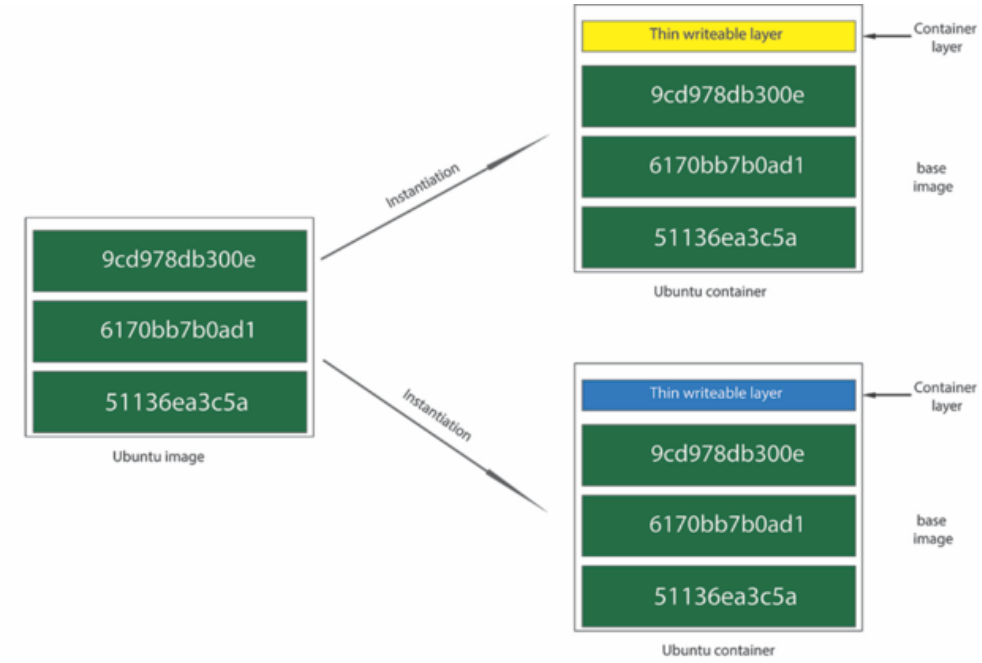
Kuva 5. Docker-levykuvan sovelluskerrokset (InfoWorld 2016).

4.6 **Kontti**

Docker-kontti (Docker-container) on levykuvan pohjalta luotu sovelluksen ajoympäristö, ja sitä hallinnoidaan sovellusliittymän tai komentoliittymän avulla. Samasta levykuvasta peräisin olevat kontit ovat identtisiä sovelluskoodin ja riippuvuuksien suhteen, mutta toisin kuin pelkästään luettavassa tilassa olevat levykuvat, sisältävät kontit kirjoitettavan kerroksen (container layer). Kirjoittamisen mahdollistava kerros sijaitsee sovelluskerroksista päällimmäisenä, pelkästään luettavissa olevien kerrosten päällä. Kaikki tehdyt muutokset kontin suorittamisen aikana tallennetaan tähän kirjoitettavaan kerrokseen. Tämän kerroksen ansiosta samaa levykuvaa voidaan hyödyntää useassa kontissa, ja ne voivat käytettäessä erota toisistaan loppujen lopuksi huomattavasti (Kuva 6). (InfoWorld 2016.)

Kun käynnissä oleva kontti poistetaan, myös kirjoitettava kerros poistetaan samaan aikaan. Ainoa tapa säilyttää tehdyt muutokset on käyttää Docker commit -toimintoa ennen kontin poistamista. Kyseinen toiminto luo uuden levykuvan käynnissä olevasta kontista, joka sisältää aiemman levykuvan kerrokset sekä kirjoitettavan kerroksen, johon muutokset ovat tapahtuneet. Uusi levykuva eroaa muutoksien takia levykuvasta, jota suoritettiin alun perin kontissa. Tämä helpottaa versionhallintaa sekä mahdollistaa alkuperäisen levykuvan käytön uudestaan ilman kontissa tehtyjä muutoksia. (InfoWorld 2016.)

Dockerissa käytetään Linux-ytimen Namespace-ominaisuutta kontin eristämiseen toisista konteista ja isäntäjärjestelmästä. Ominaisuus myös mahdollistaa konteille omat IP-osoitteet. Palvelu käyttää cgroup (control groups) -ominaisuutta järjestelmän käytettävissä olevien resurssien jakamiseen eri konteille. Cgroupin avulla on mahdollista myös määritellä yksittäisen kontin käytettävissä oleva muisti. (Medium 2017.) Kontteja on mahdollista määritellä ja käynnistää useita samanaikaisesti Docker Composen avulla. Siinä hyödynnetään YAML-tiedostoa asetuksien määrittelemiseen ja konttien linkittämiseen. (Microsoft 2017a.)



Kuva 6. Docker-levykuva ja käynnissä olevat kontit omilla kirjoitettavilla kerroksilla (InfoWorld 2016).

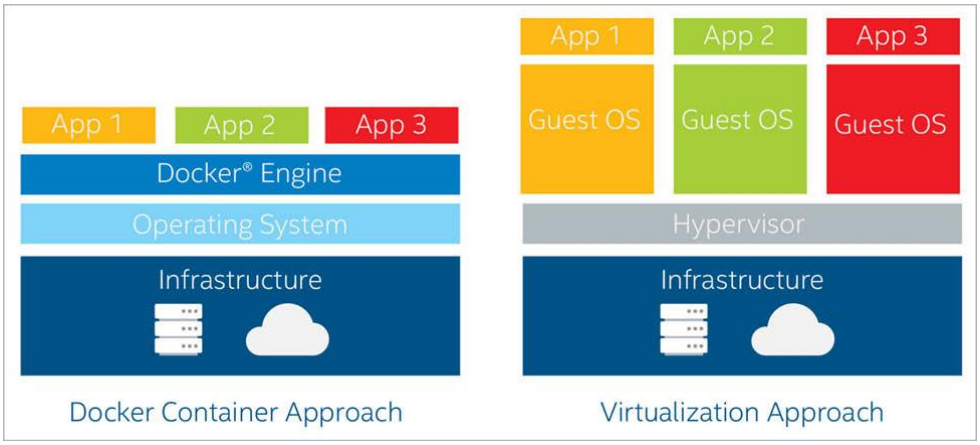
4.7 **Rekisteri**

Docker-rekisteri (Docker registry) on paikka, johon levykuvat voidaan julkaista ja tallentaa. Rekisteri voi sijaita pilvessä tai paikallisesti ja se voi olla julkinen tai yksityinen sekä vain organisaation tai tiettyjen käyttäjien saatavilla. Se sisältää yleiset ohjelmointirajapinnat, joiden avulla käyttäjät voivat luoda, julkaista, etsiä, ladata ja hallita levykuvia. (InfoWorld 2016.)

Docker Hub on julkinen pilvipohjainen levykuvarekisteri, jota hallinnoi Docker. Se tarjoaa tuen levykuvan etsimiseen ja jakeluun sekä toimii oletushakupaikkana ja -julkaisualustana. Docker hub sisältää myös virallisia kuvia, jotka Docker on sertifioinut. Nämä kuvat ovat tunnettujen julkaisijoiden luomia, joita ovat esimerkiksi Canonical, Red Hat ja MongoDB. Virallisia kuvia on mahdollista käyttää omien kuvien ja sovellusten rakentamisen perustana. (InfoWorld 2016.)

5 KONTTI VERRATTUNA VIRTUAALIKONEESEEN

Pohjimmiltaan kontit ovat virtualisointitekniikkaa ja niillä on paljon yhteistä virtuaalikoneiden kanssa. Ne eroavat toisistaan kuitenkin siinä, missä virtualisointikerros sijaitsee ja miten järjestelmän resursseja käytetään (Kuva 7). Hypervisor virtualisoi fyysisen laitteiston komponentteja, kuten verkkokorttia, kiintolevyä, grafiikkakorttia ja muistia. Näitä virtualisoituja osia voidaan sitten käyttää luotaessa virtuaalikoneita, joihin asennetaan käyttöjärjestelmä aivan kuin se asennettaisiin fyysiseen laitteistoon. Laitteiston komponenttien virtuaalisointi mahdollistaa fyysisen koneen monipuolisen käytön sekä suuren hyötykäyttöasteen. Yksittäinen fyysinen tietokone voi suorittaa potentiaalisesti kymmeniä eri käyttöjärjestelmiä samanaikaisesti ja käyttää sovelluksia näiden virtuaalisten koneiden sisällä. Kontit taas siirtävät abstraktion käyttöjärjestelmätasolle asti. Virtuaalisten komponenttien sijaan kontit tarjoavat virtuaalisen ytimen. Virtuaaliydin tarjoaa kaikki rajapinnat, joita sovellukset käyttävät esimerkiksi tiedostojärjestelmän ja muistihallinnan suorittamiseen. (Wintellect 2017.)



Kuva 7. Kontti- ja virtuaalikonejärjestelmien rakenteellinen ero (OpenSource 2017).

Kontit ovat tuotannossa huomattavasti tehokkaampia kuin virtuaalikoneet. Kontteja voidaan luoda ja hävittää lähes välittömästi, jos niitä verrataan huomattavasti raskaampiin virtuaalikoneisiin. Docker-levykuva voi olla niin pieni kuin 12 kilotavua, kun taas virtuaalikoneet ovat tyypillisesti kooltaan gigoissa jo pelkästään vieraskäyttöjärjestelmän takia. Sen vuoksi kontteja voi olla koneella huomattavasti enemmän kuin virtuaalikoneita. (C2 Labs n.d.)

Perinteisestä virtuaalikoneesta kontit eroavat myös siinä, että ne koostuvat kerroksista. Tämä lähestymistapa helpottaa olemassa olevan kontin muokkaamista ja uudelleenrakentamista, koska se muuttaa vain tiettyjä kerroksia eikä koko kuvaa. Tämä helpottaa laajennettavuutta ja lyhentää käyttöönottoaikaa. Vastaavasti virtuaalikoneet ovat luonteeltaan erittäin raskaita ja vaativat koko kuvan päivittämistä. Tämä vaikeuttaa nopeaa ja

ketterää päivitystä. Virtuaalikoneiden suuren koon vuoksi niiden liikuttelemiseen vaaditaan myös kontteja enemmän kaistaleveyttä. (C2 Labs n.d.)

Nämä tehokkuuden parannukset ja levykuvien tiheys vaikuttavat monien mielikuviin siten, että kontit tekevät virtuaalikoneista vanhanaikaisia. Kontit eivät kuitenkaan tarjoa yhtä hyvää eristystä, kuin luokan 1 hypervisor. Kontit jakavat käyttöjärjestelmän eli kaikki kontit toimivat yhden käyttöjärjestelmän päällä. Tämä on erittäin tehokasta, sillä jokainen kontti on pohjimmiltaan vain eristetty käynnissä oleva prosessi. Kuitenkin jokainen kontti jakaa muistin muiden konttien kanssa. Tämä luo tietoturvariskin, jota ei ole hypervisorin kanssa, sillä siinä käytetään erillistä käyttöjärjestelmää ja sille annettuja laitteistoresursseja. (C2 Labs n.d.)

Kontit ja virtuaalikoneet voivat olla samanaikaisesti samassa datakeskusympäristössä, ja usein molempia tekniikoita pidetäänkin toisiaan täydentävinä. Sovellusarkkitehtien ja datakeskusten ylläpitäjien saatavilla olevan työkalusarjan laajeneminen tarjoaa uusia etuja ja mahdollisuuksia työtehtävien toteuttamiseen. (TechTarget 2017.)

6 HAMKIN E-OPETUKSEN TILANNE JA TULEVAISUUS

Hämeen ammattikorkeakoulun BYODin ja virtualisoinnin tämänhetkisen tilanteen sekä tulevaisuuden näkymien selvittämiseksi haastateltiin puolistrukturoidusti tietojenkäsittelyn lehtorina toimivaa Tero Kesoa. Keso on toiminut opettajana HAMKissa 2,5 vuotta ja sitä ennen koulutuslaitoksen tietohallinnossa kahdeksan vuotta. Viimeisimpänä toimenkuvana tietohallinnossa hän toimi IT-vastaavana ja oli mukana edellisen HAMKin virtuaaliympäristön Lab Managerin käyttöönotossa. Keso on ollut myös aktiivisesti mukana kehittämässä HAMKin nykyistä vCommander-virtuaaliympäristöä. (Keso 2018.)

6.1 Virtualisointi

Tällä hetkellä HAMKissa käytetään VDI-tekniikkaa, ja pääasiallisena alustana sille toimii VMware. VMwaren päällä käytetään vCommander-hallintakäyttöliittymää virtuaalikoneiden hallinnoimiseen ja resurssien jakamiseen. Toteutettu virtuaaliympäristö on oleellinen osa tietojenkäsittelyn opetusta, mutta sitä käytetään myös epäsäännöllisesti muillakin koulutusohjelmilla. (Keso 2018.)

Perusongelmana siinä on se, että kun virtualisoidaan koneita, niin ovat ne usein kooltaan huomattavan suuria ja raskaita, minkä vuoksi joudutaan varaamaan suuret määrät levytilaa sekä resursseja palvelimelta. Pahimmassa tapauksessa virtuaalikoneella on graafinen käyttöliittymä ja kaikkea muuta tarpeetonta. Tämä on johtanut siihen, että palvelinten suorituskyky alkaa loppua. (Keso 2018.)

Koska virtuaalikoneiden suoritustehon tarpeet kasvavat vuosi vuodelta, niin vaaditaan palvelimelta aina vain enemmän resursseja. Taustajärjestelmää on kuitenkin mahdollista kasvattaa vain tiettyyn pisteeseen asti. Ratkaisuna taustajärjestelmän kasvattaminen ei myöskään tue HAMKin yhtä periaatetta, joka on BYOD (luku 6.2). Opiskelijoilla on kannettavissa tietokoneissaan teknologian kehittymisen myötä entistä enemmän laskentatehoa, minkä ansiosta opiskelijat voivat itse tuottaa opiskelemiseen tarvittavia palveluita. Opiskelijoiden itse tuottamilla palveluilla mahdollistettaisiin heidän työskentelynsä ajasta ja paikasta riippumatta. (Keso 2018.)

Nykyinen virtuaaliympäristö on tällä hetkellä opettajariippuvainen, koska opiskelija ei itse pysty luomaan omia virtuaalisia työtiloja. Ongelmia ja viivästyksiä tulee, mikäli opettaja unohtaa tilata työtilan opiskelijalle käytettäväksi. Tilaaminen itsessään on myös liian raskas ja pitkä prosessi, koska tilaaminen tarvitsee tehdä kaksi (2) viikkoa aikaisemmin ennen opetustilannetta. Siihen tarvitsee lisäksi määritellä mitä kaikkea siellä voi opiskella sekä tarvittavat verkkoasetukset. Opetusta suunniteltaessa tulee myös huomioida mahdolliset muut käynnissä olevat kurssit, joilla käytetään virtuaalikoneita, koska mikäli virtuaalikoneita on samaan aikaan käytössä

muillakin kursseilla, niin palvelimen käsittelemä kuorma kasvaa nopeasti liian suureksi. Nämä kaikki yhdessä vievät huomattavan määrän opettajan käytettävissä olevasta ajasta sekä palvelinten resursseista. (Keso 2018.)

Opiskelijan näkökulmasta nykyinen toimintaympäristö on myöskin hieman monimutkainen, koska vCommander toimii tällä hetkellä vain oppilaitoksen sisäverkossa. Ennen kuin työtilaa päästään käyttämään, niin opiskelijan tarvitsee ottaa etätyöpöytäyhteys erilliselle HAMKin tarjoamalle virtuaalikoneelle ja suorittaa sen avulla omaa virtuaalikonetta tai käyttää oppilaitoksen tarjoamaa VPN-yhteyttä päästäkseen sisäverkkoon. Molemmat tavat vievät oppilaan oman koneen ja oppilaitoksen palvelimien resursseja. Resurssien riittämättömyyden ja ajasta sekä paikasta riippuvaisen opetuksen ongelmiin pyritään oppilaitoksessa löytämään sopivia metodeja kehittämällä sekä uudistamalla virtualisointitapoja ja -ratkaisuja. (Keso 2018.)

6.2 **BYOD-toimintamalli**

BYOD on lyhenne sanoista bring your own device. Sillä tarkoitetaan toimintamallia, missä työntekijät tuovat ja käyttävät työpaikalla työn tekemiseen omia tietoteknisiä laitteitaan, kuten kannettavaa tietokonetta, älypuhelinta tai tablettia yritysten tarjoamien laitteiden sijaan (webopedia n.d.). BYOD-strategiassa on etuja, kuten työntekijöiden tyytyväisyyden lisääminen mahdollisuudella työskennellä joustavammin, kustannussäästöt laitteiston ja niiden ylläpidon pienenemisestä sekä tuottavuuden paraneminen työntekijöiden ollessa onnellisempia ja nopeampia omilla laitteillaan. Toimintamallin yleistyminen kuitenkin pakottaa IT-osastot ottamaan huomioon ja kehittelemään käytäntöjä, kuinka yrityksen dataa on mahdollista käyttää laitteilla, joihin sillä on vain vähän tai ei laisinkaan vaikutusmahdollisuutta. Käytäntöjä kehitellään esimerkiksi tilanteisiin, joissa työntekijän laite menetetään, varastetaan tai muuten vaarannetaan ja mitkä ovat turvatoimet siitä tulevien uhkien ehkäisemiseksi. (techradar 2015.)

Opiskelussa hyödynnetään nykyään todella usein digitaalisia työvälineitä ja ne ovat luonnollinen osa oppimisprosessia. Tietokoneita, tablet-laitteita ja älypuhelimia käytetään tiedonhakuun, tiedon tuottamiseen ja yhteydenpitoon opiskelijoiden sekä ohjaajien välillä. Opiskelijat käyttävät omia laitteita myös oppimistilanteiden ja tulosten taltioinnissa. Pyrkimyksenä on siirtää oppilaita ulos luokkahuoneista, sinne mihin kulloinenkin opetustilanne vaatii, kuten kasvihuoneelle, metsään tai muuhun käytännön työn mahdollistavaan paikkaan. (HAMK n.d.)

Syksystä 2015 lähtien HAMKissa on ollut käytäntönä, että opiskelijat voivat hyödyntää omia laitteitaan oppimisessa (HAMK n.d.). Kokonaan BYOD-toimintamalliin siirtyminen on ajoitettava niin, että opettamisen laatu ei heikkene eikä keskeydy. Nyt on muutaman vuoden ajan lisätty ja täsmennetty opiskelijoiden sisäänpääsykirjeisiin tietoa siitä, että he tulevat tarvitsemaan omia laitteitaan opiskelun suorittamiseksi. Tällä hetkellä aletaan

olemaan siinä tilanteessa, että voidaan turvallisemmin siirtyä BYODin toteuttamisessa eteenpäin, kun uudet opiskelijat ovat varmasti tietoisia vaatimuksista laitteiston suhteen. Kuitenkaan pelkästään opiskelijoiden valmius BYODiin ei riitä. Tulee ottaa huomioon, että valmius toimintamallin käyttöönottoon on myös tiloissa, opetusmateriaaleissa ja opetustekniikassa. (Keso 2018.)

Kesällä 2018 on tulossa isoja muutoksia ATK-luokkien ja niiden koneiden suhteen. Koneiden määrää tullaan vähentämään runsaasti, minkä seurauksena siirrytään entistä enemmän BYOD-toimintamalliin. (Keso 2018.)

7 DOCKERIN SOVELTAMINEN

Tässä luvussa kerrotaan, miten Dockeria soveltamalla pyrittiin etsimään ratkaisua resurssien riittämättömyyteen ja ajasta sekä paikasta riippuvaiseen opetukseen. Kuinka oppilaitoksen olisi mahdollista tarjota opiskelijalle uusi työkalu itsenäiseen opiskelemiseen ja harjoittelemiseen, sekä miten opetustilanteissa käytettäviä tehtäviä voitaisiin vaihtoehtoisesti suorittaa Dockerilla. Esimerkkiympäristönä käytettiin LAMP-kokonaisuutta, joka sisältää käyttöjärjestelmän (Linux), verkkopalvelinohjelman (Apache), tietokantapalvelun (MySQL) ja ohjelmointikielituen (PHP). Tutkimusympäristössä käytettiin Linux- sekä Windows-käyttöjärjestelmää, jotta saatiin kokeiltua ohjelmistoa molemmissa ympäristöissä mahdollisesti tulevaa käyttötarkoitusta varten.

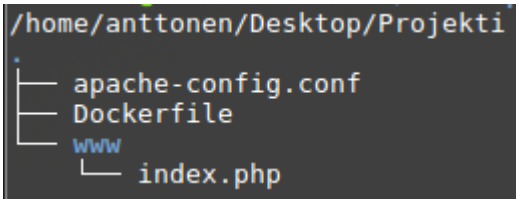
7.1 Levykuvan luominen

Ensimmäisenä tavoitteena oli luoda Docker-tiedostoa käyttämällä oma levykuva, joka sisältäisi Apache-verkkopalvelinohjelman ja PHP-ohjelmointikielituen. Tällä oli tarkoituksena demonstroida, miten opiskelijoille voitaisiin tarjota koulun räätälöimiä levykuvia itsenäiseen opiskeluun ja tehtävien tekemiseen. Havainnollistamisen ja tutkimisen vuoksi työssä ei käytetty valmiita Docker-rekisteristä ladattavia levykuvia, jotka sisältäisivät Apachen ja PHP: n. Usein on kuitenkin ajan ja kustannuksien säästämiseksi järkevää tutkia, onko halutusta palvelusta jo tehty toimiva levykuva.

Levykuvia rakennettaessa valitaan monesti valmis pohjakuva, jonka päälle aletaan luomaan omaa levykuvaa. Pohjakuvaksi on mahdollista valita erilaisia vaihtoehtoja, ja usein tiettyä tarkoitusta varten onkin jo valmiiksi tehty kuvia, joita kannattaa käyttää. Tällaisia ovat esimerkiksi Dockerin sertifioimat levykuvat. Levykuvien tulisi olla mahdollisimman pieniä, joten niiden pitäisi sisältää vain tarvittavat sovelluspaketit eikä mitään ylimääräistä. Pohjakuvassa käytettävään käyttöjärjestelmään tulee kiinnittää erityistä huomiota, koska sillä voidaan vaikuttaa suuresti levykuvan lopulliseen kokoon sekä toiminnallisuuksiin. On myös mahdollista luoda itse aivan tyhjästä levykuva ilman pohjakuvaa, mutta usein se ei ole tarpeellista tai kannattavaa.

Kuvia räätälöitäessä omiin tarpeisiin yhtenä hyvänä vaihtoehtona käytettäväksi pohjakuvaksi ja käyttöjärjestelmäksi olisi esimerkiksi Linuxin Alpine versio, joka on huomattavasti pienempi kooltaan verrattuna Linuxin Ubuntuun. Tässä työssä käytettiin kuitenkin Ubuntua sen tutun ympäristön ja laajan tuen takia. Opetuskäyttöön otettavat levykuvat tulisi olla mahdollisimman pieniä niiden helpomman siirreltävyyden vuoksi. Oppilaita on ryhmissä paljon, ja mikäli jokainen palauttaa opettajalle kooltaan ison levykuvan, niin rasittaa se turhaan palautuksiin käytettävää palvelua, esimerkiksi palvelinta tai pilvipalvelua.

Oman levykuvan rakentaminen aloitettiin luomalla tutkimusympäristössä käytettävän Linux Mint -koneen työpöydälle Projekti-kansio, jonka sisälle sijoitettiin levykuvan luomiseen tarvittavat ja käytettävät tiedostot (kuva 8). Nämä tiedostot olivat Dockerfile, apache-config.conf ja www-kansion sisällä oleva index.php. Dockerfile-tekstitiedosto sisältää kaikki konfigurointitiedot ja komennot, joita käytetään halutun levykuvan aikaan saamiseksi (kuva 9). Tiedoston avulla docker-palvelu osaa automaattisesti rakentaa levykuvan, kun sille syötetään rakentamiskomento. Apache-config.conf-tiedosto sisältää Apachen määrityksiä (kuva 11) ja index.php on PHP-ohjelmointikielinen tiedosto, jolla korvataan Apachen vakio aloitussivu (kuva 10). Aloitussivuna voisi olla esimerkiksi ohjeet opiskelijalle siitä, miten kyseinen ympäristö toimii, ja kuinka harjoituksia sekä tehtäviä kuuluisi tehdä.



Kuva 8. Kansiorakenne levykuvan rakentamisessa.

Docker-tiedoston laatiminen aloitettiin tutkimalla mitä eri parametrit tekevät ja tarkoittavat. Mallia omalle kuvalle haettiin valmiiksi tehdyistä levykuvista, joiden avulla pyrittiin oppimaan syntaksi ja saamaan parempi käsitys toiminnoista. Tiedostoa jouduttiin muokkaamaan ja testaamaan muutaman kerran, ennen kuin se saatiin valmiiksi ja toimimaan halutulla tavalla. Rakentamiseen käytettiin Notepadqq-ohjelmistoa, joka ladattiin aloitettaessa koneelle. Lopullinen Docker-tiedosto koostui useasta eri parametrista, joita tarvittiin halutun levykuvan aikaansaamiseksi. Käytetyt parametrit olivat:

- FROM
- MAINTAINER
- LABEL
- RUN
- ENV
- CMD
- EXPOSE
- COPY.

Docker-tiedoston alussa valitaan FROM-parametrilla se, mitä pohjakuvaa käytetään. Valitun pohjakuvan päälle määritellään tarvittavat paketit ja muut määritykset, joita käytetään omassa levykuvassa. Työssä käytettiin pohjakuvana Ubuntua sen tutun ympäristön ja laajan tuen takia.

MAINTAINER on ei-suoritettava määritys, jota käytetään kertomaan, kuka on tehnyt ja ylläpitää levykuvaa sekä mihin voi ottaa yhteyttä. Tässä tapauksessa siihen kirjoitettiin työn tekijän nimi ja koulun sähköpostiosoite.

Levykuvaan on mahdollista lisätä metadataa nimikkeiden (labels) avulla, jotka auttavat kuvien organisoinnissa. Pitää kuitenkin huomioida, että jokainen uusi nimike lisää uuden kerroksen kuvaan ja näin ollen kasvattaa sen kokoa. Tässä tapauksessa kerrottiin, että levykuvassa on pohjakuvana Ubuntu ja se sisältää Apachen sekä PHP:n opiskelua varten.

RUN on yksi levykuvan rakennusvaihe, joka suorittaa komennon rakentamisen aikana. Parametri suorittaa komennon olemassa olevan kerroksen päälle ja luo uuden kerroksen komentotuloksen kanssa. RUN-parametreja on usein tiedostossa useampi, joiden avulla levykuvan kerrokset rakennetaan. Esimerkiksi tässä tapauksessa Apache: n ja PHP: n asentaminen oli ennakkoedellytys ennen sovelluksen suorittamista. Parametrin avulla voidaan suorittaa tarvittavat komennot sovellusten asentamiseksi pohjakuvan päälle.

ENV-parametrin avulla on mahdollista määritellä ympäristömuuttujia. Työssä annettiin ympäristömuuttuja DEBIAN_FRONTEND=noninteractive, jotta levykuvan luomisen aikana ei käyttäjältä kysytä määrityksiä, vaan valitaan oletusasetukset automaattisesti. Tiedostossa määritellään ympäristömuuttujilla myös Apachen asetuksia.

Työssä käytettiin myös CMD-parametria. Suurin ero CMD- ja RUN-parametrin välillä on se, että CMD ei suorita mitään levykuvan rakennusaikana. Se vain määrittää kuvaan halutun komennon, joka suoritetaan oletuksena, kun rakennettu levykuva käynnistetään. Docker-tiedostossa voi olla vain yksi CMD-parametri. Jos siihen lisätään useampi, niin vain viimeinen tulee voimaan. Levykuvaan määritellyt CMD-komennot on mahdollista yliajaa komentotulkissa konttia käynnistettäessä.

EXPOSE-parametri toimii dokumentaationa kuvan rakennuttajan ja kontin ylläpitäjän välillä. Sen tarkoituksena on näyttää portit, joita olisi tarkoitus käyttää tuotannossa. Jotta portti saadaan oikeasti käyttöön, niin tulee kontin käynnistysvaiheessa -p -parametrilla määritellä avattavat portit.

ADD- ja COPY-parametrit ovat toiminnallisesti hyvin pitkälti samankaltaisia, mutta yleisesti COPY on suositellumpi. Tämä johtuu siitä, että se on selkeämpi. COPY tukee vain paikallisten tiedostojen kopiointia konttiin, kun taas ADD:illa on ominaisuuksia, kuten tar-tiedostojen automaattinen purkaminen ja URL-tuki, jotka tuovat mahdollisuuksia, mutta eivät ole ilmiselviä. Työssä käytettiin COPY-parametria esimerkiksi apache-config.conf-tiedoston siirtämiseen.

Levykuvaan on mahdollista luoda isäntäjärjestelmästä levyjakoja VOLUME-parametrin avulla. Mahdollisimman yksinkertaisen demonstroimisen vuoksi jakamista ei kuitenkaan käsitelty työssä.

```
#Base-image
FROM ubuntu:latest

#Maintainer
MAINTAINER Timo Anttonen <timo.anttonen@student.hamk.fi>

#Description
LABEL Description="Ubuntu with apache and php7.0 for e-learning, thesis project"

#Ubuntun update
RUN apt-get update -y

#This is the anti-frontend. It never interacts with you  at  all,
#and  makes  the  default  answers  be used for all questions.
ENV DEBIAN_FRONTEND=noninteractive

#Packages
RUN apt-get install -y \
    apache2 \
    php7.0 \
    php7.0-bz2 \
    php7.0-cgi \
    php7.0-cli \
    php7.0-common \
    php7.0-curl \
    php7.0-dev \
    php7.0-enchant \
    php7.0-fpm \
    php7.0-gd \
    php7.0-gmp \
    php7.0-imap \
    php7.0-interbase \
    php7.0-intl \
    php7.0-json \
    php7.0-ldap \
    php7.0-mcrypt \
    php7.0-mysql \
    php7.0-odbc \
    php7.0-opcache \
    php7.0-pgsql \
    php7.0-phpdbg \
    php7.0-pspell \
    php7.0-readline \
    php7.0-recode \
    php7.0-snmp \
    php7.0-sqlite3 \
    php7.0-sybase \
    php7.0-tidy \
    php7.0-xmlrpc \
    php7.0-xsl \
    libapache2-mod-php7.0

# Enable apache mods.
RUN a2enmod php7.0
RUN a2enmod rewrite

# Update the PHP.ini file, enable <? ?> tags and quieten logging.
RUN sed -i "s/short_open_tag = Off/short_open_tag = On/" /etc/php/7.0/apache2/php.ini
RUN sed -i "s/error_reporting = .*$/error_reporting = E_ERROR | E_WARNING | E_PARSE/" /etc/php/7.0/apache2/php.ini

# Manually set up the apache environment variables
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2
ENV APACHE_PID_FILE /var/run/apache2.pid

# Copy this repo into place.
COPY www /var/www/site

# Update the default apache site with the created config.
COPY apache-config.conf /etc/apache2/sites-enabled/000-default.conf

# Remove APT files
RUN apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/*

# Expose apache.
EXPOSE 80

#Execute container foreground
CMD /usr/sbin/apache2ctl -D FOREGROUND
```

Kuva 9. Valmis Docker-tiedosto.

```
<? echo "<p>Tässä voisi olla ohjeet</p>"; ?>
```

Kuva 10. Index.php-tiedosto www-kansiosta, joka määriteltiin Apachen aloitussivuksi.

```
<VirtualHost *:80>
  ServerAdmin timo.anttonen@student.hamk.fi
  DocumentRoot /var/www/site

  <Directory /var/www/site/>
      Options Indexes FollowSymLinks MultiViews
      AllowOverride All
      Order deny,allow
      Allow from all
  </Directory>

  ErrorLog ${APACHE_LOG_DIR}/error.log
  CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Kuva 11. Apache-config.conf-tiedosto.

Kun tiedostot oltiin saatu valmiiksi annettiin Projekti-kansiossa ollessa komentotulkille levykuvan rakennuskomento *sudo docker build -t ”apache_php:updated”* . . Onnistuneen rakentamisen jälkeen saatiin teksti Successfully built IMAGE ID. Koneella olevat levykuvat tarkistettiin komennolla *sudo docker images* ja huomattiin, että oma levykuva oli nyt käytettävissä.

```
anttonen@anttonen-K53U ~ $ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
apache_php          updated             53fb03dbeee3        8 days ago          511MB
```

Kuva 12. Luodun kuvan tarkistaminen levykuvista.

Levykuva on tämän jälkeen mahdollista käynnistää konttiin useilla eri määrityksillä. Yhtenä esimerkkinä voidaan käyttää komentoa *sudo docker run -it IMAGE ID /bin/bash*, jolloin päästään kontin sisälle käyttämään komentotulkkia. Kontista voidaan poistua ensin painamalla näppäinyhdistelmää Ctrl + p ja sen jälkeen Ctrl + q, jolloin kontti jää taustalle ajoon. Mikäli kontti halutaan sammuttaa samalla kun sen sisältä poistutaan, niin käytetään näppäinyhdistelmää Ctrl + d. Uudelleen kontin sisälle menemiseen on vaihtoehtoina esimerkiksi komennot exec ja attach. Komennoilla on hieman eri käyttötarkoitukset, vaatimukset ja toiminnallisuudet, joihin ei tässä työssä syvennytä. Kontteja voidaan poistaa rm-komennolla ja levykuvia rmi-komennolla esimerkiksi *sudo docker rmi IMAGE ID*.

7.2 **Konttien linkittäminen**

Toimivan ympäristön saamiseksi tarvitaan usein useampaa sovellusta. Koska Dockerin perusideana on sisällyttää yhteen konttiin vain yksi sovellus, joudutaankin sen vuoksi melkein aina kontteja linkittämään toisiinsa, jotta haluttu kokonaisuus saadaan aikaseksi. Linkittämistä tutkittiin luomalla LAMP-kokonaisuus, joka sisältää aiemmin luodun levykuvan, jossa on Apache ja PHP sekä Dockerin sertifioimat MySQL- ja phpMyAdmin-levykuvat.

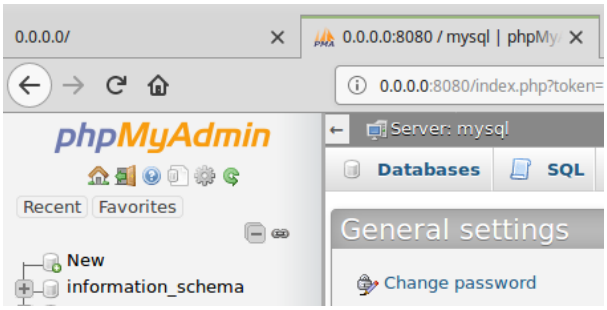
Ensimmäiseksi alettiin tutkimaan, miten komentotulkin avulla voidaan yksittäisiä kontteja käynnistää ja linkittää toisiinsa. Linkityksien jälkeen testattiin ympäristökokonaisuuden toimivuus. Tämän jälkeen käytiin myös läpi, miten Docker Compose-työkalulla voidaan YAML-tiedostoa käyttämällä käynnistää ja linkittää useita kontteja kerralla.

Ympäristön luominen aloitettiin käynnistämällä MySQL-tietokantapalvelu. Komennossa määriteltiin --name parametrillä tulevan kontin nimi, joka tässä tapauksessa oli mysql. Palvelun pääkäyttäjän salasana määriteltiin ympäristömuuttujalla –e MYSQL_ROOT_PASSWORD=qwerty1. Parametrin -d avulla kontti saatiin taustalle suoritettavaksi, ja viimeisenä määriteltiin levykuva, joka suoritetaan eli Dockerin sertifioima MySQL ja siitä viimeisin versio. Komennoksi muodostui *sudo docker run --name mysql -e MYSQL_ROOT_PASSWORD=qwerty1 -d mysql:latest*.

Seuraavaksi käynnistettiin aiemmassa vaiheessa luotu apache_php:updated-levykuva ja linkitettiin se toimimaan käynnistetyn mysql-kontin kanssa. Parametrillä -p 80:80 määriteltiin kontti toimimaan portissa 80. Tällä portilla päästiin selaimen kautta näkemään Apache-palvelimen index.php-tiedoston näkymä. Kuten MySQL-palvelua käynnistettäessä määriteltiin --name parametrillä kontin nimi, joka tässä tapauksessa oli apache_php. Parametrillä --link määriteltiin kontti toimimaan yhdessä mysql-kontin kanssa. Loput komennosta määräytyi samalla tavalla, kuten käynnistettäessä mysql-konttia. Parametri -d suorittaa kontin taustalla ja lopuksi levykuva, joka kontissa suoritetaan eli apache_php:updated. Komennoksi muodostui *sudo docker run -p 80:80 --name apache_php --link mysql:mysql -d apache_php:updated*.

Viimeisenä käynnistettiin phpMyAdmin-palvelu ja linkitettiin myös se toimimaan mysql-kontin kanssa. PhpMyAdmin määriteltiin toimimaan portissa 8080 parametrillä -p 8080:80. Nimeksi määriteltiin phpmyadmin, ja levykuvana käytettiin Dockerin sertifioimaa versiota. Suurin ero apache_php levykuvan käynnistyskomentoon tuli ympäristömuuttujasta -e PMA_HOST=mysql, jolla määriteltiin MySQL-palvelua isännöivän kontin käyttäminen tietokantapalveluna. Komennoksi muodostui *sudo docker run -p 8080:80 --name phpmyadmin --link mysql:mysql -d -e PMA_HOST=mysql phpmyadmin/phpmyadmin*.

Kun komennot oli suoritettu, tarkistettiin seuraavaksi, että kontit olivat varmasti käynnissä komennolla *sudo docker ps -a*. Komento näyttää kaikki kontit ja ilman -a parametria voidaan tarkistaa pelkästään käynnissä olevat kontit. Selaimen kautta tarkastettiin, että phpMyAdmin-palvelun avulla voidaan hallinnoida tietokantapalvelua. Siirryttiin selaimessa osoitteeseen 0.0.0.0:8080, josta päästiin phpMyAdminin kirjautumissivulle. Tunnusten syöttämisen jälkeen päästiin kirjautumaan palveluun ja voitiin todeta kohdasta Server: mysql, että phpMyAdminin tietokantapalvelimena toimii käynnistetty mysql-kontti (kuva 13).

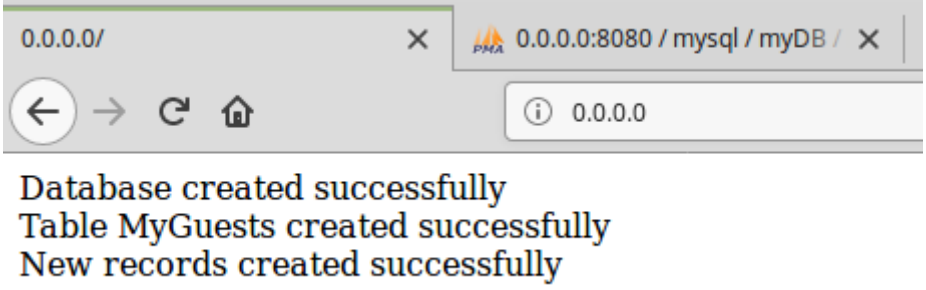


Kuva 13. phpMyAdminin hallintapaneelinäkymä, josta selviää käytetty tietokantapalvelin.

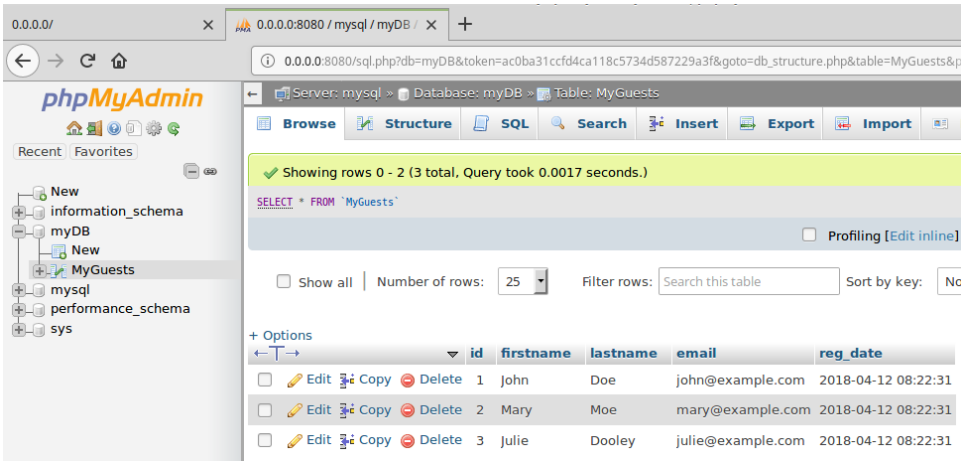
Apache_php- ja mysql-kontin välinen toimivuus tarkistettiin luomalla mysql-konttiin tietokanta, taulu ja tauluun tietoja. Tämän jälkeen taulun tiedot haettiin näkymään Apache-palvelimelle. Molempiin toimintoihin käytettiin erillisiä php-tiedostoja, joiden avulla komennot suoritettiin. Php-tiedostoihin käytettiin valmiita pohjia, joita hieman muokattiin vastaamaan vaadittuja tarpeita.

Testauksessa siirrettiin ensimmäiseksi isäntäkoneelta tiedosto Apache-palvelimelle. Tiedostossa otettiin tietokantayhteys mysql-konttiin, ja sen jälkeen lisättiin myDB-tietokanta, MyGuests-taulu ja hieman tietoja (liite 1). Tiedoston siirtäminen tapahtui komennolla *sudo docker cp /home/anttonen/Desktop/php-tiedostot/sql/index.php apache_php:/var/www/site*.

Komennot suoritettiin päivittämällä verkkosivu selaimessa. Kun komennot olivat menneet läpi, saatiin ilmoitus onnistuneista suorituksista (kuva14). Tiedostossa on myös tarkistuslauseke, joka ilmoittaa käyttäjälle, mikäli komennot eivät menneet läpi sekä mahdollisen virheilmoituksen. PhpMyAdminin hallintapaneelin kautta voitiin todeta, että komennot saatiin suoritettua onnistuneesti (kuva 15). Tietokannan tietoja on myös mahdollista tarkastella käynnistämällä uusi kontti ja linkittämällä se luotuun mysql-konttiin. Tämän tilapäisen tarkistuskontin avulla kirjaudutaan mysql-kontin tietokantapalveluun, jossa voidaan komentotulkin avulla tutkia tai muokata tehtyjä muutoksia. Kontti saatiin suoritettua komennolla *sudo docker run -it –link mysql:mysql --rm mysql sh -c 'exec mysql -h"$MYSQL_PORT_3306_TCP_ADDR" -P"$MYSQL_PORT_3306_TCP_PORT" -uroot -p"qwerty1"'*.

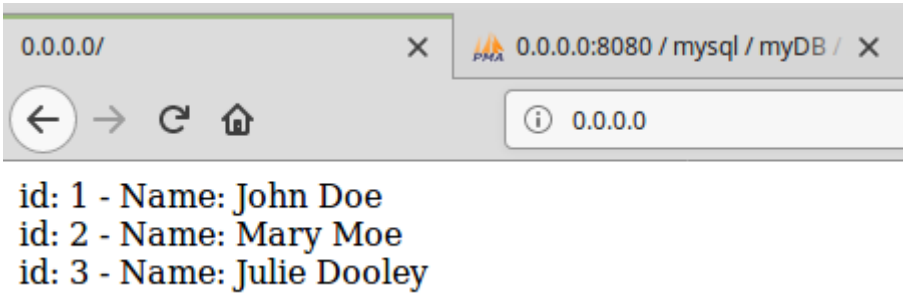


Kuva 14. Selainnäkymä komentojen onnistuneet suorittamisen jälkeen.



Kuva 15. phpMyAdmin näkymä, kun komennot on onnistuneesti suoritettu.

Tietojen luomisen jälkeen siirrettiin Apache-palvelimelle toinen php-tiedosto, joka korvasi aikaisemmin siirretyn tiedoston (liite 2). Tällä haettiin tietokantakyselyllä mysql-kontista tiedot näkymään verkkopalvelimelle. Aikaisempaan siirtokomentoon vaihtui pelkästään siirrettävän tiedoston tiedostopolku. Siirtämiseen käytettiin komentoa *sudo docker cp /home/anttonen/Desktop/php-tiedostot/datan_haku/index.php apache_php:/var/www/site*. Kun verkkosivu taas päivitettiin, niin tietokannan tiedot saatiin näkyviin (kuva 16). Testauksen jälkeen voitiin todeta yhteyden ja ympäristön vastaavan tavoitteita.



Kuva 16. Selainnäkymä, kun toinen tiedosto on siirretty ja tiedot saatu haettua tietokantapalvelusta.

Sama ympäristö pystytettiin myös käyttämällä Docker Compose-työkalua. Docker Compose on erillinen työkalu Dockeriin, joka tarvitsee asentaa koneelle Docker-palvelun asentamisen jälkeen. Asennus suoritettiin Dockerin virallisten internetsivujen ohjeiden mukaan.

Kun asennus oli saatu suoritettua, aloitettiin Composen käyttämän YAML-tiedoston rakentaminen käyttäen Notepadqq-ohjelmistoa. Tiedoston alkuun määriteltiin tiedoston versio parametrilla version: ’2’. Työssä käytettiin versiota kaksi, vaikka versio kolme oli jo julkaistu, koska siihen vaikutti löytyvän enemmän ohjeita ja selityksiä. Uusimpaan versioon on usein paranneltu composen toiminnallisuuksia ja sen takia onkin yleisesti parempi

käyttää uusinta versiota. Versioiden välillä syntaksi ja toiminnallisuudet saattavat vaihtua, joten tiedostoa luodessa tai versiota vaihdettaessa on kannattavaa käydä tarkistamassa Dockerin sivuilta tapahtuneet muutokset ongelmien välttämiseksi.

Seuraavaksi määriteltiin käynnistettävät palvelut services-parametrin alle. Käynnistettävien palveluiden alle määriteltiin käytettävä levykuva image-parametrillä. Esimerkiksi phpMyAdmin-palveluun käytettiin samaa dockerin sertifioimaa levykuvaa kuin aikaisemminkin, joka merkattiin tiedostoon muotoon image: phpmyadmin/phpmyadmin:latest. Tiedostossa määriteltiin kontin nimi parametrilla container_name. Restart-parametrilla voidaan päättää kontin uudelleenkäynnistyskäytäntö ja kontteja voidaan määritellä käynnistettäväksi tietyssä järjestyksessä depends_on-parametrilla. Ympäristöä luodessa haluttiin, että ensimmäisenä käynnistetään tietokantapalvelu ja vasta sitten muut palvelut. Tämä saatiin toteutettua lisäämällä Apachen ja phpMyAdminin alle parametri depends_on: - mysql.

Tiedostoon on mahdollista lisätä ympäristömuuttujia environment-parametrilla. PhpMyAdmin-palvelun alle lisättiin sama PMA_HOST: mysql, kuin komentotulkilla komentoja suoritettaessa, mutta myös PMA_USER: root ja PMA_PASSWORD: qwerty1, jolloin kirjautuminen phpMyAdmin-palveluun saatiin tapahtumaan suoraan ilman kyselyitä. Lopuksi määriteltiin käytettävät portit parametrilla ports. Esimerkiksi phpMyAdminille laitettiin käytettäväksi sama portti kuin komentotulkin kautta ympäristöä luodessa eli 8080. Kun tarvittavat tiedot oli saatu lisättyä tiedostoon, niin tallennettiin se koneelle yml-tiedostomuotoon. Valmistunut tiedosto käy ilmi kuvasta 17.

```
version: '2'
services:
    mysql:
        image: mysql:latest
        container_name: mysql
        restart: always
        environment:
            MYSQL_ROOT_PASSWORD: qwerty1
        ports:
            - 3306:3306
    webserver:
        image: apache_php:updated
        container_name: apache_php
        restart: always
        depends_on:
            - mysql
        ports:
            - 80:80
    phpmyadmin:
        image: phpmyadmin/phpmyadmin:latest
        container_name: phpmyadmin
        restart: always
        depends_on:
            - mysql
        environment:
            PMA_HOST: mysql
            PMA_USER: root
            PMA_PASSWORD: qwerty1
        ports:
            - 8080:80
```

Kuva 17. Docker-compose.yml-tiedosto.

Kontit käynnistettiin olemalla kansiossa, jossa YAML-tiedosto sijaitsee, komennolla *sudo docker-compose up -d*. Käynnistettäessä kontteja luodaan ympäristölle automaattisesti myös oma verkko, jonka alle kontit lisätään. Konttien käynnistyttyä suoritettiin samat testaukset, kuin komentotulkin kautta kontteja käynnistettäessä.

Verkko ja kontit on mahdollista poistaa komennolla *sudo docker-compose down*, mutta tulee huomioida, että silloin kontit nimenomaan poistetaan eikä vain sammuteta, minkä vuoksi tehdyt muutokset katoavat. Kontit on mahdollista sammuttaa ilman niiden poistamista komennolla *sudo docker-compose stop*. Silloin ne voidaan käynnistää uudelleen muokattujen tietojen kanssa tai ottaa tiedot talteen. Käynnistäminen uudelleen tapahtuu komennolla *sudo docker-compose start*.

7.3 **Levykuvan tallentaminen, siirtäminen ja käyttöönottaminen**

Tarkoituksena oli testata ja demonstroida, kuinka oppilaan olisi mahdollista tehdä ja harjoitella opettajan antamia tehtäviä ja sen jälkeen palauttaa levykuva sekä tietokanta opettajalle tarkistettavaksi. Siirtämisen testaukseen käytettiin Linux Mint- ja Windows10-käyttöjärjestelmää, jotta saatiin todennettua toimivuus eri käyttöjärjestelmien välillä. Tietokannan palauttamista varten ladattiin koneille mySQL Community Server.

Testaaminen aloitettiin Linux-koneella käynnistämällä ja linkittämällä keskenään aiemmin luotu apache_php-levykuva sekä dockerin sertifioimat mySQL- ja phpMyAdmin-levykuvat. Seuraavaksi luotiin tietokanta ja sen alle tietoja käyttämällä aiemmin luotua php-tiedostoa. Kun tietokanta oli valmis, siirrettiin apache_php-konttiin php-tiedosto, joka hakee tiedot mysql-kontista. Vaiheet toteutettiin komentotulkkia käyttämällä samalla tavalla kuin tehtiin luvussa 7.2 Konttien linkittäminen.

Tietokantaan muutettiin alkuperäiset taulukontiedot toisenlaiseksi käyttäen phpMyAdminin hallintapaneelia. Tällä pyrittiin demonstroimaan, miten oppilas voisi tehdä annettua tehtävää, joka sisältäisi tietokannan (kuva 18). Sertifioitu MySQL-levykuva käyttää datan sijoittamisessa levyjakoa paikalliselle tietokoneelle, jonka vuoksi tietokannat ja niiden data eivät sijaitse kontissa vaan paikallisella tietokoneella. Tämän vuoksi kontista ei voida tehdä commit-komennolla uutta levykuvaa, joka sisältäisi opiskelijan tekemät tietokantamuutokset. On hyvä tiedostaa, että mysql-kontin tietokantaan tehdyt tiedot häviävät, mikäli kontti poistetaan. Tehtävän palauttamista varten tarvitsee tietokanta ottaa erikseen talteen koneelle ja siirtää omana tiedostonaan opettajan koneelle, jossa se voidaan ottaa uudestaan käyttöön. Siirtämistä varten otettiin tietokannasta tiedot talteen koneelle komennolla *sudo mysqldump -uroot -pqwerty1 -h 172.17.0.2 -P 3306 myDB > /home/anttonen/Desktop/timon_mysql.sql*.

+ Options

	id	firstname	lastname	email	reg_date
Edit Copy Delete	1	Aku	Ankka	aku@thesis.com	2018-04-19 06:34:45
Edit Copy Delete	2	Mikki	Hiiri	mikki@thesis.com	2018-04-19 06:35:15
Edit Copy Delete	3	Hessu	Hopo	hessu@thesis.com	2018-04-19 06:35:43
Edit Copy Delete	4	Teppo	Tulppu	teppo@thesis.com	2018-04-19 06:35:43

Kuva 18. Esimerkki oppilaan tekemistä muutoksista.

Tiedostojen pysymistä testattiin myös erillisellä tekstitiedostolla, joka luotiin apache_php-konttiin. Aluksi mentiin käynnissä olevan kontin sisälle komennolla *sudo docker exec -ti apache_php bash,* jolla saatiin kontin sisällä komentotulkki käyttöön. Kontissa sisällä ollessa luotiin tekstitiedosto esimerkki.txt komennolla *cat > esimerkki.txt* ja kirjoitettiin siihen tekstiä. Lopuksi poistuttiin tiedostosta ja kontista painamalla kaksi kertaa näppäinyhdistelmää Ctrl + d.

Kun tarvittavat muutokset apache_php-konttiin ja tietokantaan oli tehty sekä toimivuus testattu, niin tallennettiin kontin tila uudeksi levykuvaksi komennolla *sudo docker commit apache_php apache_php:timo*. Komennolla *sudo docker images* tarkistettiin, että levykuva varmasti löytyy. Seuraavaksi levykuvasta luotiin tar-tiedosto, jotta se oli mahdollista siirtää toiselle koneelle. Tämä suoritettiin komennolla *sudo docker save -o /home/anttonen/Desktop/apache_php_timo.tar apache_php:timo* . Vielä ennen siirtämistä muokattiin luodun tar-tiedoston oikeuksia, jotta myöhemmässä vaiheessa ei tulisi ongelmia. Tämä tehtiin käyttäen chmod-komentoa.

Levykuvan ja tietokannan siirtäminen toiselle koneelle toteutettiin käyttämällä Google Drive -palvelua, joka on Googlen tarjoama pilvipalvelu, johon on mahdollista tallentaa tiedostoja. Tiedostot siirrettiin Linux-koneelta Googlen pilveen ja ladattiin Windows10-koneella palvelusta koneelle.

Kun tiedostot olivat latautuneet pilvipalvelusta koneelle, niin pakattu levykuva voitiin ottaa käyttöön komennolla *docker load -i "C:\Users\Anttonen\Desktop\Oppari\Siirto\apache_php_timo.tar"*. Load-komennon suorittamisen jälkeen tarkastettiin, että apache_php:timo näkyi levykuvissa komennolla *docker images*.

Konttien käynnistämiseen käytettiin Windows10-koneella composea, jota varten siirrettiin Linux-koneelta aiemmin käytetty YAML-tiedosto. Tiedostoon tehtiin pieni muutos helpottamaan tietokannan käyttöönottoa. Siihen lisättiin mysql kohdan alle ympäristömuuttuja MYSQL_DATABASE: myDB (kuva 19). Tällöin mysql-kontin käynnistyessä luodaan tietokanta nimeltä myDB. Tämä helpottaa siten, että ei tarvitse erikseen käydä luomassa kyseistä tietokantaa, kun oppilaan tietokantaa otetaan opettajalla käyttöön. Konttien käynnistäminen ja linkittäminen suoritettiin komennolla *docker-compose up -d*. Docker lataa tarvittavat levykuvat dockerin rekisteristä, mikäli niitä ei ole aikaisemmin ladattu paikalliselle koneelle.

```
environment:
    MYSQL_ROOT_PASSWORD: qwerty1
    MYSQL_DATABASE: myDB
```

Kuva 19. Lisätty ympäristömuuttuja.

Kun kontit olivat käynnissä, otettiin oppilaan tekemä tietokanta käyttöön ja testattiin toimivuus. Aluksi siirryttiin ladatun ja puretun mySQL Community Serverin bin-kansioon, joka tässä tilanteessa sijaitsi polussa *C:\Users\Anttonen\Desktop\Oppari\Siirto\mysql\mysql-8.0.11-winx64\bin*. Kansiossa ollessa otettiin tietokanta mysql-konttiin käyttöön antamalla komento *cmd /c '.\mysql -uroot -pqwerty1 -h localhost -P 3306 myDB < "C:\Users\Anttonen\Desktop\Oppari\Siirto\timon_mysql.sql"'*.

Toimivuus todennettiin phpMyAdminin hallintapaneelin kautta, että tietokannassa näkyy oppilaan muuttamat tiedot. Sen lisäksi tarkistettiin, että Apachen aloitussivulle onnistutaan hakemaan nyt siirretyn tietokannan tiedot (kuva 20).

```
id: 1 - Name: Aku Ankka
id: 2 - Name: Mikki Hiiri
id: 3 - Name: Hessu Hopo
id: 4 - Name: Teppo Tulppu
```

Kuva 20. Siirretyn tietokannan onnistunut tietojen hakeminen.

Lopuksi mentiin vielä apache_php-konttiin sisälle komennolla *docker exec -ti apache_php bash* ja tarkistettiin, että esimerkki.txt löytyy ja se aukeaa sekä se, että oppilaan suorittamat komennot löytyvät historiasta. Historia tarkistettiin antamalla komento *history*. Historian tarkistaminen auttaa opettajaa mahdollisten ongelmatilanteiden selvittämisessä, ja sen avulla on mahdollista saada parempi yleiskäsitys oppilaan tehtävien suorittamisesta.

7.4 **Konttien IP-osoitteiden muuttaminen**

IP-osoitteiden ja verkkoavaruuksien kanssa joudutaan tuotannossa todella usein tekemisiin, joten on hyvä tietää, miten kontin IP-osoite tarkastetaan, ja miten sille on mahdollista määritellä tietty osoite. IP-osoitteen tarkastaminen suoritettiin käyttämällä komentotulkkia ja kontin osoitteen määritteleminen tehtiin kontin käynnistämisen yhteydessä sekä Docker Composen avulla, jotta saatiin selville molemmat toteutustavat.

Konttien IP-osoitteiden muuttamista aloitettiin tutkimalla, miten ja mitä tietoja käynnissä olevasta kontista on mahdollista saada. Tarkoituksena oli selvittää mikä on kontin IP-osoite ja minkä nimisessä verkossa se sijait-

see. Tätä varten käynnistettiin apache_php:updated-levykuva samalla komennolla kuin aikaisemminkin eli *sudo docker run -p 80:80 --name apache_php -d apache_php:updated*. Sen jälkeen tarkistettiin käynnistetyn kontin IP-osoite ja verkko komennolla *sudo docker inspect apache_php*. Komento antaa listan kontin tiedoista ja sieltä löytyy Networks-kohta, josta nähdään verkon nimi ja kontin ”IPAddress” eli IP-osoite. Kyseisen kontin IP-osoite oli kuvan 21 mukaisesti 172.17.0.2 ja se kuului bridge-nimiseen verkkoon. Tämän avulla voitiin todeta myöhemmässä vaiheessa osoitteen onnistunut määritteleminen.


```
"Networks": {
    "bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "921fb5cd77a96fc18d482c6ad9674672556dae3880eb1ac11497188cab22ca66",
        "EndpointID": "48c7ec793b04c22bfa60a9dfc5af26c38c11580ed72f34172dc0dd1e0bb122a2",
        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:11:00:02"
    }
}
```


Kuva 21. Käynnissä olevan kontin verkko ja IP-osoite .

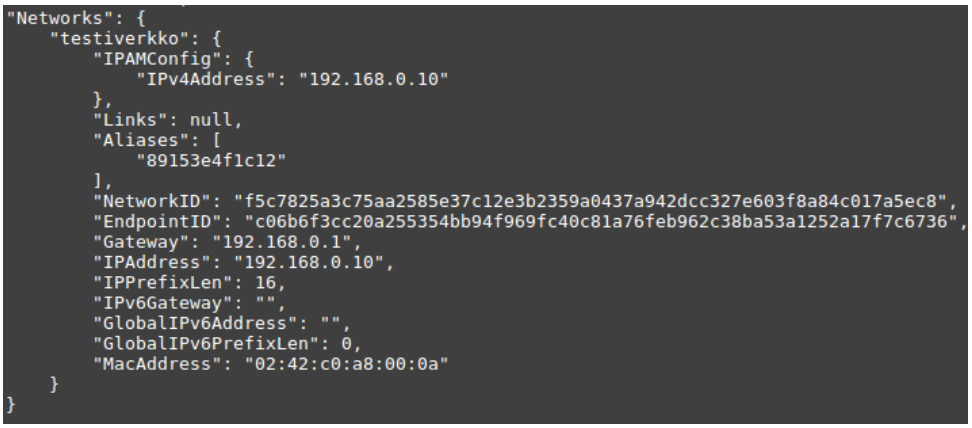
Docker määrittelee automaattisesti konteille IP-osoitteet. Jos halutaan määritellä osoite itse, niin tarvitsee aluksi luoda oma verkko. Verkon luomisen jälkeen voidaan kontti lisätä haluamalla osoitteella kyseiseen verkkoon.

Työssä luotiin oma verkko, jolle annettiin nimeksi testiverkko ja osoiteavaruudeksi valittiin 192.168.0.0/16. Verkko luotiin komennolla *sudo docker network create --subnet=192.168.0.0/16 testiverkko*. Olemassa olevat verkot on mahdollista tarkistaa komennolla *sudo docker network ls* (kuva 22).

```
anttonen@anttonen-K53U ~ $ sudo docker network create --subnet=192.168.0.0/16 testiverkko
f5c7825a3c75aa2585e37c12e3b2359a0437a942dcc327e603f8a84c017a5ec8
anttonen@anttonen-K53U ~ $ sudo docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
921fb5cd77a9        bridge              bridge              local
e18a744f0e7f        compose_default     bridge              local
8d742bbfc928        host                host                local
55997df6937a        none                null                local
f5c7825a3c75        testiverkko         bridge              local
```

Kuva 22. Verkon luominen ja olemassa olevien verkkojen tarkastaminen.

Nyt oli mahdollista käynnistää kontti omaan testiverkkoon halutulla IP-osoitteella. Aikaisempaan käynnistyskomentoon tarvitsi lisätä parametrit --net ja --ip, joista --net määrittelee verkon, johon kontti liitetään ja --ip kontin IP-osoitteen. Kontti käynnistettiin komennolla *sudo docker run --net testiverkko --ip 192.168.0.10 --name apache_php -p 80:80 -d apache_php:updated*. Lopuksi tarkistettiin kontin IP-osoite aikaisemminkin käytetyllä komennolla *sudo docker inspect apache_php*, että kontilla on nyt haluttu IP-osoite, ja että se sijaitsee testiverkossa (kuva 23). Kontille valitulla IP-osoitteella päästiin myös Apachen verkkopalvelimelle.



```
"Networks": {
    "testiverkko": {
        "IPAMConfig": {
            "IPv4Address": "192.168.0.10"
        },
        "Links": null,
        "Aliases": [
            "89153e4f1c12"
        ],
        "NetworkID": "f5c7825a3c75aa2585e37c12e3b2359a0437a942dcc327e603f8a84c017a5ec8",
        "EndpointID": "c06b6f3cc20a255354bb94f969fc40c81a76feb962c38ba53a1252a17f7c6736",
        "Gateway": "192.168.0.1",
        "IPAddress": "192.168.0.10",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:c0:a8:00:0a"
    }
}
```


Kuva 23. Käynnistetyn kontin verkkoasetukset.

Kun oli saatu selvitettyä, miten yksittäiselle kontille voidaan määritellä haluttu IP-osoite, niin alettiin tutkimaan, miten YAML-tiedoston avulla on mahdollista määritellä useille konteille oma verkko ja osoitteet. Pohjana käytettiin aikaisemmin luotua tiedostoa, joka rakennettiin vastaamaan haluttua ympäristöä.

Tiedoston loppuun määriteltiin aluksi oman verkon luomiseen tarvittavat tiedot networks-parametrin alle. Verkon nimeksi annettiin testi, joka kuitenkin muuttuu muotoon compose_testi Dockerin lisätessä siihen etuliitteen. Seuraavaksi määriteltiin käytettävä ajuri. Bridge valittiin sen helppokäyttöisyyden ja helpon ymmärtämisen vuoksi. Ajuri luo isäntäjärjestelmän sisälle yksityisen verkon, jonka avulla kontit voivat kommunikoida keskenään. Ulkoinen pääsy kontteihin määritellään käytettävillä porteilla. Docker varmistaa automaattisesti verkon eristämisen muista verkoista, etteivät verkkojen yhteydet mene keskenään sekaisin. IPAMin (IP Address Management) ajuriksi valittiin oletus eikä muihin vaihtoehtoihin tutustuttu tässä työssä. Osoiteavaruudeksi määriteltiin 10.0.0.0/8 ja yhdyskäytäväksi 10.0.0.1.

Jokaiselle palvelulle annettiin ennalta päätetyt osoitteet, esimerkiksi phpMyAdmin-palvelulle määriteltiin käytettäväksi 10.0.0.13 IP-osoite. Konttien nimien perään lisättiin compose-tunniste ja tietokantapalvelun pääkäyttäjän salasana vaihdettiin helpottamaan testausta. Lopullinen YAML-tiedosto ilmenee kuvasta 24.

```
version: '2'
services:
    mysql:
        image: mysql:latest
        container_name: mysql_compose
        restart: always
        environment:
            MYSQL_ROOT_PASSWORD: compose
        ports:
            - 3306:3306
        networks:
            testi:
                ipv4_address: 10.0.0.11
    webserver:
        image: apache_php:updated
        container_name: apache_php_compose
        restart: always
        depends_on:
            - mysql
        ports:
            - 80:80
        networks:
            testi:
                ipv4_address: 10.0.0.10

    phpmyadmin:
        image: phpmyadmin/phpmyadmin:latest
        container_name: phpmyadmin_compose
        restart: always
        depends_on:
            - mysql
        environment:
            PMA_HOST: mysql_compose
            PMA_USER: root
            PMA_PASSWORD: compose
        ports:
            - 8080:80
        networks:
            testi:
                ipv4_address: 10.0.0.13
networks:
  testi:
    driver: bridge
    ipam:
      driver: default
      config:
      - subnet: 10.0.0.0/8
        gateway: 10.0.0.1
```

Kuva 24. Oman verkon ja IP-osoitteiden määrittelemiseen käytetty YAML-tiedosto.

Kontit käynnistettiin samalla komennolla kuin aikaisemminkin eli *sudo docker-compose up -d*. Tiedostopolkua ei tarvittu, koska käynnistyskomento annettiin tiedoston sijaitsevassa kansiossa. Kuvasta 25 voidaan huomata, että komennon antamisen jälkeen Docker luo verkon ja kontit sekä antaa ilmoituksen, kun kontit ovat käyttövalmiina.

```
anttonen@anttonen-K53U ~/Desktop/Compose $ sudo docker-compose up -d
Creating network "compose_testi" with driver "bridge"
Creating mysql_compose ... done
Creating apache_php_compose ... done
Creating phpmyadmin_compose ... done
```

Kuva 25. Verkon luominen ja konttien käynnistys käyttäen YAML-tiedostoa.

Verkkoa ja sen sisältämiä kontteja on mahdollista tutkia komennolla *sudo docker network inspect compose_testi*. Sen avulla tarkastettiin, että kontit ovat saaneet halutut IP-osoitteet, ja että ne sijaitsevat luodun verkon alla (kuva 26). Verkkopalvelimelle ja phpMyAdmin-palveluun oli nyt mahdollista siirtyä selaimella käyttäen niille annettuja IP-osoitteita. Ympäristön

toimivuus testattiin samalla tavalla kuin luvussa 7.3 konttien linkittäminen. Salasanojen muokkaamisen takia oli myös testaamiseen käytettyihin PHP-tiedostoihin vaihdettava salasanat.

```
anttonen@anttonen-K53U ~/Desktop/Compose $ sudo docker network inspect compose_testi
[
    {
        "Name": "compose_testi",
        "Id": "f8813964ae495787ee230971ef9606671da31dc35e74159a8c5e5aed35e096bb",
        "Created": "2018-05-07T12:03:29.693407715+03:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.0.0.0/8",
                    "Gateway": "10.0.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "Containers": {
            "207e7361c7fcbd5fe501ca6b2d347047a6eba46a58cc8535835452f3c2d5726c": {
                "Name": "phpmyadmin_compose",
                "EndpointID": "7b35a78abfb5432b8cacdb1ba4eaf3ac4e2e326f63787ae55d4272acf60df08
5",
                "MacAddress": "02:42:0a:00:00:0d",
                "IPv4Address": "10.0.0.13/8",
                "IPv6Address": ""
            },
            "7ee570bff4ccc955545e81932c4bc03ccc850b51d21153076cb4f9988389b64b": {
                "Name": "apache_php_compose",
                "EndpointID": "13d7c5498941a31ba849d8f1040795670fbf7afd490feeae31d968653ed6981
c",
                "MacAddress": "02:42:0a:00:00:0a",
                "IPv4Address": "10.0.0.10/8",
                "IPv6Address": ""
            },
            "ab2894e4e7364d6244319ad75b526d1531f4dd30a7f39f16ccb06c53859cf218": {
                "Name": "mysql_compose",
                "EndpointID": "c1806ada516f5665be0542091bf81127e892e7844413245ff1b596ff1657a2a
3",
                "MacAddress": "02:42:0a:00:00:0b",
                "IPv4Address": "10.0.0.11/8",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

Kuva 26. Luotu verkko ja sen sisältämät kontit.

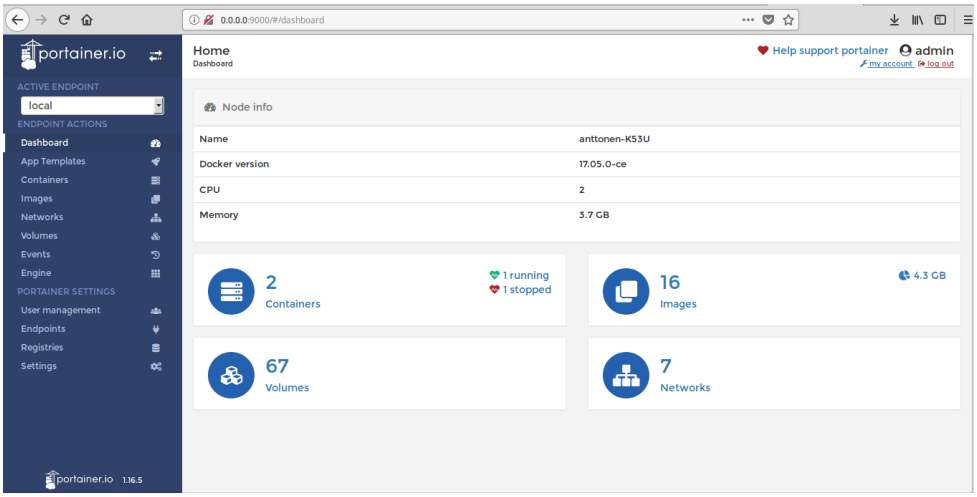
7.5 Graafinen käyttöliittymä

Konttien hallintaan on saatavilla ulkopuolisilta palveluntarjoajilta erilaisia graafisia käyttöliittymiä. Tarkoituksena oli selvittää pintapuolisesti, minkälaisia palveluita on tarjolla, kuinka tällainen käyttöliittymä voidaan ottaa käyttöön ja kuinka selkeä se on. Vaihtoehtoja tutkittiin selailemalla internetiä. Käyttöliittymiä löytyi paikallisesti asennettavina paketteina ja konteissa suoritettavina palveluina. Osa näistä oli avoimen lähdekoodin lisenssillä tuotettuja. Käyttöliittymiä löytyi myös klusterien luomiseen ja hallinnointiin sekä levykuvien ja konttien kehitykseen.

Käyttöönotettavaksi palveluksi valikoitui Portainerin tarjoama käyttöliittymä. Se on kehitetty avoimen lähdekoodin lisenssillä, joka mahdollistaa muokkausten tekemisen ohjelmaan, mikäli sille on tarvetta. Käyttöliittymä suoritetaan kontissa, joten paikalliselle koneelle ladataan siihen tarvittava

levykuva, kun käynnistyskomento suoritetaan komentotulkissa. Käyttöliittymä saatiin käyttöön komennolla *sudo docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock -v portainer_data:/data portainer/portainer*.

Komennon suorittamisen jälkeen graafinen käyttöliittymä toimii omassa kontissaan ja sitä oli mahdollista käyttää selaimen kautta. Tässä tapauksessa palveluun päästiin osoitteella 0.0.0.0:9000. Kun palvelu käynnistettiin ensimmäisen kerran, niin tarvitsi luoda pääkäyttäjätunnukset. Sen jälkeen tuli valita, käytetäänkö palvelua ympäristössä, jossa Portainer on käynnissä eli paikallisesti vai käytetäänkö sitä etänä olevan ympäristön hallinnointiin. Testauksessa käytettiin vain paikallista vaihtoehtoa. Sen jälkeen avautui kuvan 27 mukainen hallintapaneeli, joka osoittautui selkeäksi. Yksittäisiä kontteja oli mahdollista luoda nopeasti ja monipuolisilla asetuksilla, ja myös muut hallinnointimahdollisuudet vaikuttivat ymmärrettäviltä. Vaikka graafisia käyttöliittymiä on tarjolla, niin eivät ne kuitenkaan poista käyttäjältä vaadittua tietotaitoa Dockerista. Tämä johtuu siitä, että hallintapaneelin kautta kontteja ja ympäristöjä luotaessa ovat käytössä samat parametrit kuin komentotulkin kautta toimittaessa.



Kuva 27. Portainerin hallintapaneeli.

8 YHTEENVETO

Työn tavoitteena oli Dockeria soveltamalla tutkia, miten konttiteknologia toimii ja sen sopivuutta sekä skaalautuvuutta opetuskäyttöön, jotta ajasta ja paikasta riippuvainen opetus sekä palvelimien resurssipula helpottuisi. Tutkimusta tehdessä kävi ilmi, että ohjelmisto on erittäin skaalautuva ja sitä on mahdollista hyödyntää monella eri tavalla. Kun perustietotaito ohjelmasta on hallussa, voidaan sen jälkeen hyvinkin nopeasti pystyttää ympäristöjä ja saada tehtyä haluttuja asioita. Työn aikana Dockeria käytettiin Linux- sekä Windows-ympäristössä, ja molempien todettiin sopivan käytettäväksi. Tämä helpottaa molempia osapuolia, kun voidaan valita ympäristö mieltymysten ja osaamisen mukaan. Tekijän työn aikana saamien kokemuksien perusteella tultiin siihen tulokseen, että Dockeria voitaisiin sen tämänhetkisessä tilanteessa käyttää jo osaan opetustehtävistä, mutta joissain tilanteissa perinteinen virtuaalikone toimisi paremmin.

Dockerin ottaminen opetuskäyttöön vaatii paljon pohjatyötä, joka tulee olla tehtynä valmiiksi, ennen kuin siitä saadaan kaikki potentiaalinen hyöty irti. Tällaista pohjatyötä ovat esimerkiksi levykuvien luominen ja räätälöiminen käyttötarkoitusta varten sekä oppilaiden ja opettajien ohjeistus ohjelmiston onnistuneeseen käyttöön. Opetuskäyttöä varten tulisi luoda työskentelyä helpottavia ja nopeuttavia käytäntöjä esimerkiksi tietokantojen nimeämiseen.

Työn tuloksena saatiin luotua levykuva, jota olisi mahdollista hyödyntää oppilaan omatoimisessa oppimisessa ja opetuskäytössä. Oppimisympäristökokonaisuuteen tarvittavat kontit saatiin linkitettyä toisiinsa komentotulkin ja Docker Composen avulla, ja ympäristön toimivuus saatiin todennettua testaamalla. Kontin tila onnistuttiin tallentamaan uudeksi levykuvaksi ja se saatiin siirrettyä sekä käyttöönotettua toisella koneella. Mahdollisten IP-osoite ongelmien vuoksi tutkittiin eri tapoja, miten konteille saadaan määriteltyä tietty IP-osoite ja verkko. Lopuksi onnistuttiin ottamaan käyttöön graafinen käyttöliittymä, jolla on mahdollista hallita kontteja.

Suurimmat ongelmat työn aikana keskittyivät syntaksien selvittämiseen ja tietokantaongelmiin. Esimerkiksi työn aikana MySQL:n viimeisin versio muuttui 8.0-versioksi, joka aiheutti ongelmia työn lopussa tarkastettaessa komentojen toimivuuksia. PhpMyAdmin-palvelun ja tietokantapalvelun sisältävät kontit eivät enää toimineet keskenään uudistetun salauksen johdosta. Ilmestyneeseen ongelmaan ei enää lähdetty etsimään ratkaisua, vaan lopputestaukset suoritettiin korvaamalla MySQL:n 8.0-versio 5.7.22-versiolla, jota myös alun perin työssä käytettiin. Muutos toteutettiin niin, että levykuvan määräämiin parametreihin, joissa käytettiin mysql:latest-määritystä muutettiin muotoon mysql:5.7.22, jolloin koneelle ladattiin ja otettiin käyttöön vanhempi toimiva versio.

Kirjoittajan mielestä opinnäytetyön lopputulos on onnistunut, vaikkakaan siitä ei tullut käytännön työn osalta niin syventävä, kuin tekijä olisi itse halunnut. Työn tilaaja oli lopputulokseen tyytyväinen ja sitä mieltä, että työ antaa hyvän perustan oppilaitokselle jatkaa ohjelmiston ottamista käyttöön opetuksessa. Tutkimuskysymyksiin onnistuttiin vastaamaan teorian ja käytännön avulla. Tekijä oppi hyvin paljon ohjelmiston käyttämisestä, koska aikaisempaa kokemusta siitä ei juurikaan ollut. Tulevaisuudessa oppilaitoksen ja työn tekijän on molempien mahdollista hyödyntää tutkimuksen tuloksia tuotannossa.

Tutkimus toimii hyvänä pohjana projektille tarjoten perustiedot ja -toiminnallisuudet Dockerista. Seuraavaksi projektia voisi jatkaa eteenpäin esimerkiksi toteuttamalla johonkin tiettyyn opetettavaan tehtävään räätälöity oppimisympäristö. Työssä käytettyjen esimerkkien avuksi voisi tulevaisuudessa myös kehitellä esimerkiksi skriptejä, jotka nopeuttaisivat opettajien työtä heidän tarkastellessaan oppilaiden palautuksia.

LÄHTEET

AgileIT (2015). Why Virtualize? Haettu 31.1.2018 osoitteesta https://www.agileit.com/news/why-virtualize/

AnandTech (2008). Application Virtualization. Haettu 22.1.2018 osoitteesta https://www.anandtech.com/show/2456/2

ApacheBooster (2017). docker architecture. Haettu 23.1.2018 osoitteesta http://apachebooster.com/kb/what-is-a-docker-container-for-beginners/docker-architecture/

C2 Labs (n.d.). Docker – The Difference Between Containers and Virtual Machines (VMs). Haettu 6.2.2018 osoitteesta http://c2labs.com/docker-the-difference-between-containers-and-virtual-machines-vms/

Container Solutions (2015). Docker Use Cases. Haettu 23.1.2018 osoitteesta http://container-solutions.com/docker-use-cases/

Docker (2018). Docker overview. Haettu 7.2.2018 osoitteesta https://docs.docker.com/engine/docker-overview/

Eduonix (2016). Learn about Docker and its Main Components. Haettu 7.2.2018 osoitteesta https://www.eduonix.com/blog/software-development/learn-about-docker-and-its-main-components/

freeBSD (n.d.). Handbook. Haettu 23.1.2018 osoitteesta https://www.freebsd.org/doc/handbook/jails.html

HAMK (n.d.). BYOD – OMAT LAITTEET OPPIMISESSA. Haettu 5.2.2018 osoitteesta http://www.hamk.fi/ohjeita/digitaalisuus/ohjelmat-ja-oppaat/byod/Sivut/default.aspx

HowStuffWorks (n.d.). How Server Virtualization Works. Haettu 23.1.2018 osoitteesta https://computer.howstuffworks.com/server-virtualization2.htm

HP (2011). Virtualization For Dummies. Haettu 23.1.2018 osoitteesta https://ssl.www8.hp.com/de/de/pdf/virtuallisation_tcm_144_1147500.pdf

Information Technology Group (2015). What is Application Virtualization, and Why Do You Need It? Haettu 22.1.2018 osoitteesta https://www.itgct.com/what-is-application-virtualization-and-why-do-you-need-it/

InfoWorld (2016). Docker fundamentals. Haettu 23.1.2018 osoitteesta https://www.infoworld.com/article/3077875/linux/containers-101-docker-fundamentals.html

Keso, T. (2018). Tietojenkäsittelyn lehtori. Hämeen ammattikorkeakoulu. Haastattelu 12.2.2018.

Medium (2017). Docker Overview – A Complete Guide. Haettu 23.1.2018 osoitteesta https://medium.com/@xenonstack/docker-overview-a-complete-guide-43decd218eca

Microsoft (2017a). Docker terminology. Haettu 23.1.2018 osoitteesta https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/container-docker-introduction/docker-terminology

Microsoft (2017b). What is Docker? Haettu 23.1.2018 osoitteesta https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/container-docker-introduction/docker-defined

Milner (2015). The Advantages and Disadvantages of Virtualization. Haettu 26.1.2018 osoitteesta http://milner.com/company/blog/technology/2015/07/14/the-advantages-and-disadvantages-of-virtualization

OpenSource (2017). Docker is emerging as the future of application delivery. Haettu 7.2.2018 osoitteesta http://opensourceforu.com/2017/09/docker-is-emerging-as-the-future-of-application-delivery/

Pearson (2014). Introduction to Oracle Databases on Virtual Infrastructure. Haettu 7.2.2018 http://www.pearsonitcertification.com/articles/article.aspx?p=2256453

plesk (2016). A BRIEF HISTORY OF CONTAINERIZATION. Haettu 23.1.2018 osoitteesta https://www.plesk.com/blog/business-industry/infographic-brief-history-linux-containerization/

securitywing (2014). Types of Virtualization Technology – Advantages vs. Disadvantages. Haettu 7.2.2018 osoitteesta https://securitywing.com/types-virtualization-technology/

StorageCraft (n.d.). The History of Virtualization. Haettu 1.2.2018 osoitteesta https://www.storagecraft.com/blog/infographic-history-virtualization/

techradar (2015). What is BYOD and why is it important? Haettu 5.2.2018 osoitteesta http://www.techradar.com/news/computing/what-is-byod-and-why-is-it-important-1175088

TechTarget (2009). application streaming. Haettu 22.1.2018 osoitteesta http://searchvirtualdesktop.techtarget.com/definition/application-streaming

TechTarget (2011). app virtualization (application virtualization). Haettu 22.1.2018 osoitteesta http://searchvirtualdesktop.techtarget.com/definition/app-virtualization

TechTarget (2015). A brief history of Docker Containers' overnight success. Haettu 24.1.2018 osoitteesta http://searchservervirtualization.techtarget.com/feature/A-brief-history-of-Docker-Containers-overnight-success

TechTarget (2017). Containers vs. VM: What's the difference? Haettu 6.2.2018 osoitteesta http://searchservervirtualization.techtarget.com/answer/Containers-vs-VMs-Whats-the-difference

tehcnopedia (n.d.). Application Virtualization. Haettu 22.1.2018 osoitteesta https://www.techopedia.com/definition/573/application-virtualization

UK Essays (2015). The History of Virtualization Information Technology Essay. Haettu 22.1.2018 osoitteesta https://www.ukessays.com/essays/information-technology/the-history-of-virtualization-information-technology-essay.php

VMware (2007). Understanding Full Virtualization, Paravirtualization, and Hardware Assist. Haettu 19.1.2018 osoitteesta https://www.vmware.com/techpapers/2007/understanding-full-virtualization-paravirtualizat-1008.html

webopedia (n.d.). BYOD – bring your own device. Haettu 5.2.2018 osoitteesta https://www.webopedia.com/TERM/B/BYOD.html

Wintellect (2017). Choosing Between Containers and Virtual Machines. Haettu 6.2.2018 osoitteesta https://www.wintellect.com/choosing-containers-virtual-machines/

PHP-tiedosto tietokannan luomiseen ja sisällön lisäämiseen Liite 1

```php
 <?php
$servername = "mysql";
$username = "root";
$password = "qwerty1";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error() ."<br>");
}



// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
    echo "Database created successfully" ."<br>";
} else {
    echo "Error creating database: " . mysqli_error($conn) ."<br>";
}

mysqli_close($conn);

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error() ."<br>");
}



// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";

if (mysqli_query($conn, $sql)) {
    echo "Table MyGuests created successfully" ."<br>";
} else {
    echo "Error creating table: " . mysqli_error($conn) ."<br>";
}



$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if (mysqli_multi_query($conn, $sql)) {
    echo "New records created successfully" ."<br>";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn) ."<br>";
}


mysqli_close($conn);
?>
```

PHP-Tiedosto tietokannan tietojen hakemiseen Liite 2

```php
<?php
$servername = "mysql";
$username = "root";
$password = "qwerty1";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
    // output data of each row
    while($row = mysqli_fetch_assoc($result)) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}

mysqli_close($conn);
?>
```