Haaga-Helia
ammattikorkeakoulu Oy

Wordpress-sovelluksen käyttöönotto ja ylläpito Docker-teknologialla

Juha-Matti Ohvo

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2017

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t)
Juha-Matti Ohvo

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
Wordpress-sovelluksen käyttöönotto ja ylläpito Docker-teknologialla

Sivu- ja liitesivumäärä
28+4

Opinnäytetyön tavoitteena oli suorittaa Wordpress-sovelluksen käyttöönotto Docker-teknologialla. Lisäksi asensin Wordpressin virtuaalipalvelinympäristöön ja vertailin käyttöönoton nopeutta ja helppoutta Dockeriin verrattuna. Toteutin työssä myös ylläpitotoimenpiteen siirtämällä Wordpress-sovelluksen toiselle Docker-palvelimelle. Aloitin opinnäytetyön joulukuussa 2016 ja työstin sitä kevään 2017 ajan.

Opinnäytetyö koostuu opinnäytetyön tietoperustasta, teknisestä toteutuksesta ja yhteenvedosta. Tietoperustassa on tarkemmin selitettynä opinnäytetyössä käytetyt teknologiat. Teknisessä toteutuksessa suoritin opinnäytetyön teknisen osuuden raportoimalla tekemiseni, jotka sisältävät myös pohdintoja. Opinnäytetyön lopussa tein yhteenvedon, jossa pohdin Dockerin hyötyjä virtuaalitietokoneympäristöön verrattuna.

Opinnäytetyössä oli käytössä neljä virtuaalipalvelinta, jotka vuokrasin DigitalOcean-palvelusta. Näistä kaksi toimi Docker-palvelimina, yksi sovelluspalvelimena WordPressia varten ja yksi tietokantapalvelimena. Palvelininfrastruktuuri on kuvattuna tarkemmin liitteessä 3.

Opinnäytetyön aikana havaitsin Dockerin osalta paljon hyötynäkökohtia. Suurimmat ongelmat liittyivät työn aikana tiedontallentamiseen Docker-ympäristössä. Minulla ei ollut työn alussa lainkaan kokemusta Dockerista ja yksi työn tavoitteena oli oppia Dockerin perusteet ja saada Wordpress asennettua Docker-teknologiaa käyttäen.

Asiasanat
Docker, WordPress, virtuaalipalvelin, ylläpito, Linux

Sisällys

Keskeiset käsitteet

Apache
Avoimen lähdekoodin web-palvelinohjelmisto, jota käytetään http-palvelimena esimerkiksi verkkosovelluksia varten.

apt
Debian-pohjaisissa Linux-käyttöjärjestelmäjakeluissa käytettävä paketinhallintaohjelma. Apt on käytössä Ubuntu-käyttöjärjestelmässä, jota käytetään tässä projektissa.

cgroups
Linux-kernelin, eli ytimen, ominaisuus, joka vastaa Docker-konttien resurssien käytöstä, esimerkiksi laskentatehosta ja muistista.

Docker-kontti
Sovellus ja kaikki sovelluskomponentit, eli ohjelmistokirjastot ja palvelut, sisältävä paketointi, joka ajetaan Docker-palvelussa.

Docker-palvelu
Isäntätietokoneella toimiva Docker-sovellus, jonka Docker-kontti vaatii toimiakseen.

Hypervisor
Virtuaalitietokoneiden ajamiseen vaadittava ohjelma, joka asennetaan isäntätietokoneen ja virtuaalitietokoneen väliin.

LAMP-sovelluspino
Linuxista, Apache web-palvelimesta, MySQL-tietokantapalvelusta ja PHP-ohjelmointikielestä koostuva sovelluspino ohjelmistojen kehittämistä varten.

MySQL
Tietokantapalvelinohjelmisto tiedon tallentamista varten.

PHP
Web-pohjaisissa sovelluksissa käytetty ohjelmointikieli.

SSH

Suojattu verkkoliikenneprotokolla, joka mahdollistaa salattujen verkkoyhteyksien muodostamisen.

Tiedoston luku-, kirjoitus, suoritusoikeudet
Unix-käyttöjärjestelmissä (Linux- ja Mac-tietokoneet) tiedoston lukuoikeus mahdollistaa tiedoston sisällön lukemisen ja kirjoitusoikeus tiedoston muokkaamisen. Suoritusoikeus tarkoittaa, että tiedosto eli ohjelma voidaan ajaa. Tiedostojen oikeuksilla parannetaan tietoturvaa. Esimerkiksi konfiguraatiotiedostoille on tapana antaa suoritus- ja lukuoikeudet, jotta kyseinen tiedosto voidaan ajaa, mutta lukuoikeuksien poistamisella voidaan estää tiedoston muokkaaminen.

Virtuaalitietokone (VM)
Emuloitu tietokone, joka asennetaan fyysisen tietokoneen päälle. Virtuaalikone toimii samalla tavalla kuin fyysinen tietokone, mutta käyttää isäntätietokoneen, eli käyttöjärjestelmän resursseja.

WordPress
PHP-ohjelmointikielellä kirjoitettu avoimen lähdekoodin sisällönhallintajärjestelmä.

1 Johdanto

Sovelluskehittäjän on kehitysvaiheessa pidettävä huolta siitä, että kehitettävä sovellus toimii testiympäristön lisäksi varsinaisessa tuotantoympäristössä. Järjestelmän ylläpitäjän on puolestaan huolehdittava, että tuotantoympäristö on optimoitu sovellusta varten niin, että sovelluksen toimivuuden kannalta vaadittavat ohjelmistokirjastot ja komponentit ovat asennettuna tuotantoympäristöön. Kehitettävä sovellus ei välttämättä toimi, jos tuotantoympäristö poikkeaa testiympäristöstä. (Docker docs 2017g)

Yksi tapa ratkaista edellä mainittu ongelma on paketoida sovellus ja sen toiminnan kannalta vaadittavat komponentit Docker-tietokoneeseen, joka kutsutaan Docker-kontiksi. Docker-kontti ajetaan Docker-palvelussa, joka asennetaan erikseen palvelimelle ja se vastaa Docker-konttien käynnistämisestä ja ajamisesta. Tässä tapauksessa palvelinympäristö vaatii vain Docker-palvelun, joten järjestelmän ylläpitäjän ei tarvitse huolehtia palvelimelle asennettavista sovelluskomponenteista ja -kirjastoista. Kehittäjä voi puolestaan Docker-kontin luontivaiheessa päättää, mitä komponentteja Docker-konttiin paketoidaan. (Docker docs 2017g)

Opinnäytetyössä toteutin Wordpress-sovelluksen käyttöönoton Docker-teknologialla ja havainnollistin kuvitteellisen yrityksen tilannetta, jossa yritys tuottaa asiakkaalle Wordpress-palvelun. Toteutin Wordpress-sovelluksen käyttöönoton kahdella tavalla; sekä asentamalla Wordpressin ja sen vaatimat sovelluskomponentit manuaalisesti virtuaalipalvelimelle, että tekemällä käyttöönoton Docker-teknologialla. Projektissa ei keskitytä Wordpressin toimintaan tarkemmin, vaan ainoastaan sen käyttöönottoon. Opinnäytetyössä tarkasteltiin Dockerin hyötyjä verrattuna virtuaalitietokoneympäristöön.

Projektin alussa minulla ei ollut aiempaa Docker-osaamista ja opinnäytetyön aihe valittiin sillä perusteella, että halusin oppia Dockerin perusteet ja selvittää syitä, miksi kyseessä on hyvin suosittu teknologia nykypäivän ohjelmistokehityksessä. Tavoitteenani oli myös havainnollistaa sen hyödyt sovelluksen käyttöönotossa ja saada Wordpress-sovellus toimimaan Docker-ympäristössä.

Työtä varten vuokrasin neljä virtuaalipalvelinta DigitalOcean-palvelusta; kaksi Ubuntu Server 16.04.2 -palvelinta Wordpress-sovellusten käyttöönottoa varten, yksi palvelin Docker-sovelluksen siirtoa varten, johon asennetaan CentOS 7 -palvelinkäyttöjärjestelmä ja Docker-palvelu. Neljättä palvelinta käytettiin MySQL-tietokantapalvelimena Wordpress-sovelluksen tiedontallentamista varten.

2 Tietoperusta

Tässä kappaleessa on esiteltynä tarkemmin projektissa käyttämistäni tekniikoista. Lisäksi kappaleessa on selitettynä Dockerin arkkitehtuuri, Dockerin historiaa ja Docker-teknologian toiminta. Kappaleen tavoitteena on selventää Dockerin perusteet niin, että lukija ymmärtää teknisessä toteutuksessa käytettäviä termejä ja teknologioita.

2.1 WordPress

WordPress on avoimen lähdekoodin sisällönhallintajärjestelmä, joka on suosittu ja yleisesti käytetty julkaisualusta esimerkiksi blogisivustoissa. WordPressin julkaisu tapahtui vuonna 2003 ja siitä on tällä hetkellä julkaistu versionumero 4.7.4. Se on kirjoitettu PHP-ohjelmointikielellä ja tiedontallentamista varten se käyttää MySQL-tietokantapalvelua. WordPressin suosion taustalla on sen helppokäyttöisyys, muokattavuus ja sen päälle on mahdollista kehittää omia lisäosia. (Wordpress.org 2017)

2.2 Docker

Docker on avoimeen lähdekoodiin perustuva ohjelmisto, jota käytetään sovelluksen ja sen käyttämien sovelluskirjastojen ja komponenttien paketoimiseen. Se on tällä hetkellä suosituin ja käytetyin konttiteknologia. Docker mahdollistaa sovelluksen paketoimisen Docker-konttiin, joka on isäntäpalvelimesta eristetty sovelluksen ajoympäristö. Dockeria käytetään esimerkiksi ohjelmistoympäristön automatisoitua käyttöottoa varten. (Docker.com 2017)

2.3 Dockerin historia

Docker perustuu konttiteknologiaan, joka sai alkunsa vuonna 2008, kun Solomon Hykes perusti dotCloudin. DotCloudin ideana oli kehittää sovelluskehitysalusta, joka tarjoaisi laajan tuen useammalle ohjelmointikielelle. Tämä antaisi antaisi sovelluskehittäjille mahdollisuuden valita kehitettävälle sovellukselle sille sopivimman ohjelmointikielen. Vuonna 2010 dotCloud otti osaa Y Combinatorin kehitysohjelmaan, jonka seurauksena se sai uusia yhteistyökumppaneita ja alkoi kiinnostaa myös sijoittajia.

Sijoittajien mielenkiinnosta huolimatta dotCloud päätti vuonna 2013, että sen teknologian kehittämistä jatketaan tulevaisuudessa kehittäjäyhteisön panostuksella. Dockerin alkeellisemmat versiot olivat käytännössä kehitetty Linuxin omien LXC-konttien ympärille, joita käytetään virtuaaliympäristöjen luomiseen. Kehitys oli kuitenkin erittäin nopeaa ja Dockerin versionumero 0.1 julkaistiin maaliskuussa vuonna 2013 ja vain 15 kuukautta myö-

hemmin siitä oli julkaistu versio 1.0. DotCloud vaihtoi myös kehityksen myötä nimensä Docker Inc:ksi.

Suuret yritykset, kuten Google, Amazon ja DigitalOcean, ryhtyivät pian tarjoamaan omissa pilvipalveluissaan suoran tuen Docker-teknologialle. Tällöin sovelluskehittäjät voisivat kehittää sovelluksensa Docker-teknologiaa käyttäen ja ajaa sen Docker-palvelussa riippumatta siitä, millä ohjelmointikielellä sovellus on kehitetty. Kyseinen liiketoimintamalli kuvaa hyvin dotCloudin alkuperäistä ideaa riippumattomasta palvelinalustasta.

Dockerin kehitys jatkui hurjana ja vuonna 2014 Dockerista julkistettiin Docker Swarm -niminen ohjelma Docker-konttien provisiointia varten ja lähes samaan aikaan CoreOS julkaisi oman konttien ajamiseen tarkoitetun ohjelman. Solomon Hykes ja CoreOS:n työntekijä Alex Polvi julkaisivat Open Container Initiative -hankkeen, jonka tarkoituksena olisi kehittää yleiset standardit konttiteknologioita ja ajoympäristöjä varten. Konttiteknologia ja erityisesti Docker, ovat tämän jälkeen kasvattaneet suosiotaan sovelluskehittäjien keskuudessa. (Mouat 2016, sivut 8-10)

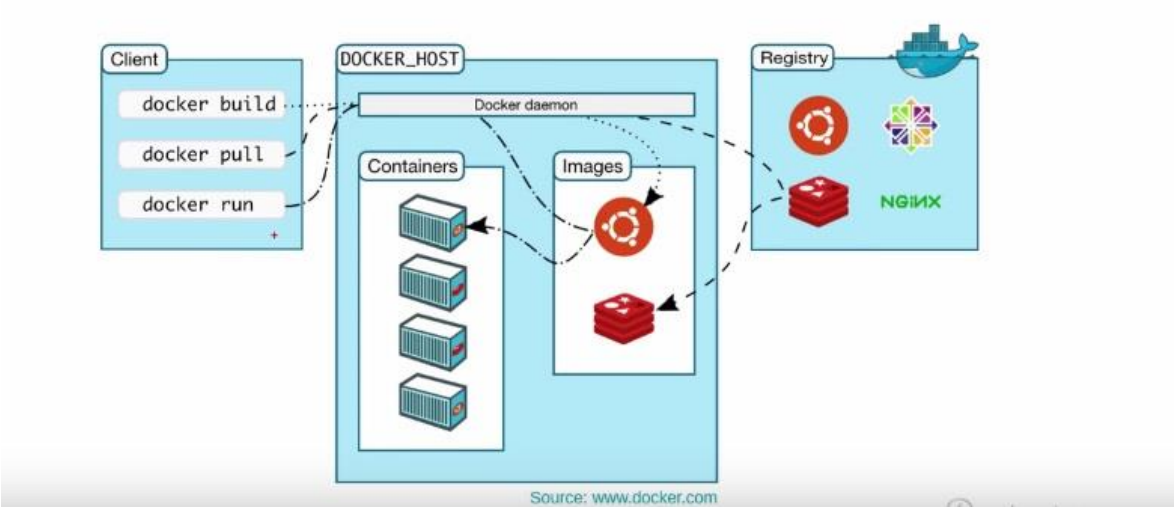
2.4 Dockerin käyttökohteet

Sovelluskehityksessä sovelluksen siirtäminen tuotantopalvelimelle voi olla haastavaa, jos kehitettävän sovelluksen testi- ja tuotantoympäristö poikkeavat toisistaan. Tuotantoympäristö voi poiketa testiympäristöstä esimerkiksi käyttöjärjestelmän ja asennettujen sovelluskomponenttien osalta. Esimerkiksi vuokrattu web-palvelin voi olla valmiiksi valmisteltu palveluntarjoajan toimesta ja tällöin käyttäjälle ei ole annettu pääkäyttäjän oikeuksia palvelimelle, joten sovelluskomponenttien asentaminen ei ole asiakkaan toimesta mahdollista. Muun muassa suomalaisessa Sigmatic web-hotellissa on kyseinen järjestely, jota olen käyttänyt itse henkilökohtaisissa Wordpress-projekteissani.

Docker-kontit pyrkivät poistaa ongelmat sovelluksen siirtovaiheessa testiympäristöstä tuotantopalvelimelle. Docker-kontti pysyy muuttumattomana luomisen jälkeen ja täten se helpottaa sovelluskehittäjän työtä, koska se toimii aina samalla tavalla ympäristöstä riippumatta. Edellä mainitussa web-hotellin tapauksessa riittäisi vain, että palveluntarjoajan tarjoamassa palvelimessa on asennettuna Docker-palvelu. (Docker.com 2017)

2.5 Dockerin arkkitehtuuri

Dockerin arkkitehtuuri koostuu useammasta kokonaisuudesta, joista jokaisella on oma rooli Dockerin toiminnassa.



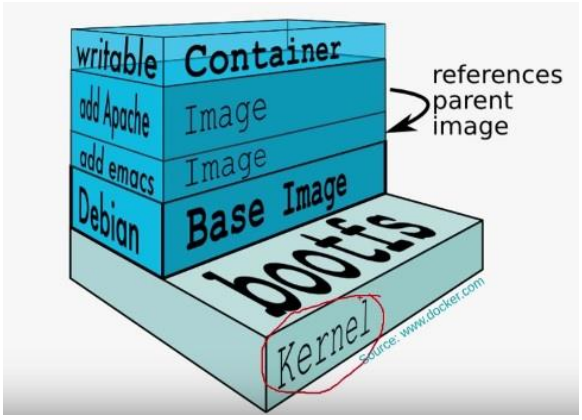
Kuva 1, Docker-palvelun arkkitehtuuri, lähde: https://www.docker.com

2.5.1 Docker-palvelu

Docker-palvelulla (docker daemon) tarkoitetaan palvelinkäyttöjärjestelmän päälle asennettavaa palvelua, joka vastaa konttien luomisesta ja ajosta sekä konttien verkkoyhteyksistä ja tallennustilasta. Isäntätietokone vastaa Docker-palvelun käynnistämisestä ja toiminnasta. *Docker-asiakas* (Docker client) on käyttäjän ja Docker-palvelun välinen rajapinta, joka mahdollistaa komentojen antamisen Docker-palvelulle esimerkiksi komentorivikäyttöliittymältä. (Docker overview 2017a)

2.5.2 Docker-image

Docker image on Docker-kontin perusta. Docker image koostuu useammasta *sovelluskerroksesta* (layer) ja yhdessä kerroksessa kuvataan yksi *Dockertiedosto* (Dockerfile). Esimerkiksi yhteen docker imageen voidaan paketoida LAMP-sovelluspino ja yksittäinen komponentti vastaa yhtä dockertiedostoa, eli kerrosta, ja näiden kokonaisuudesta muodostuu docker image. (Docker docs 2017b)

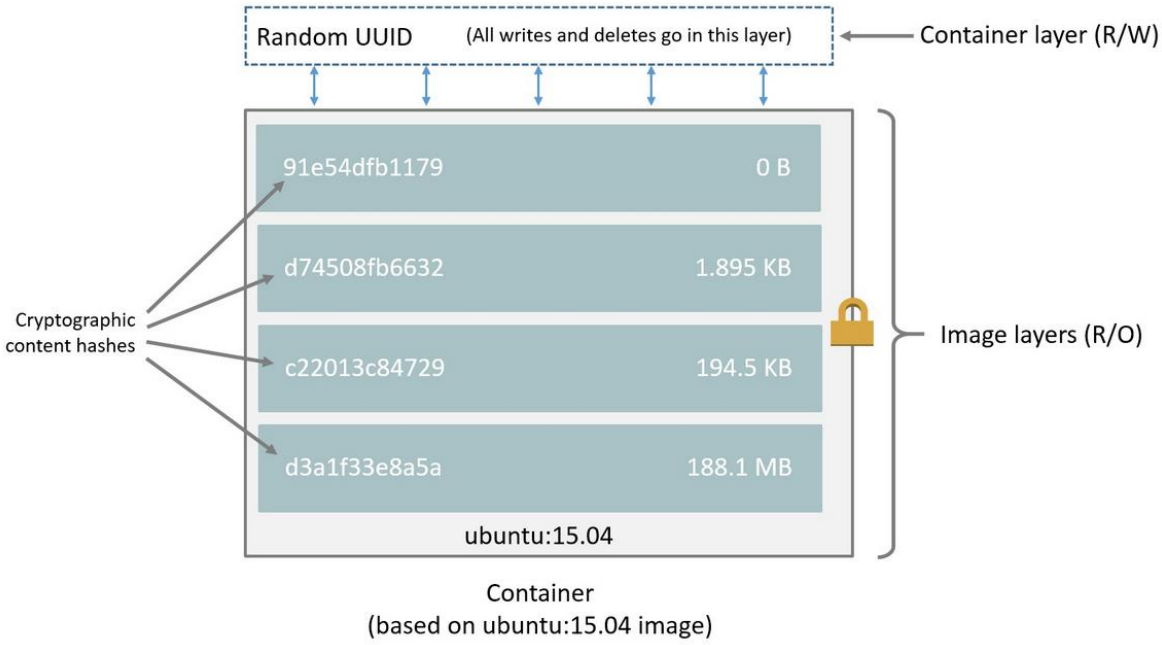


Kuva 2. Docker-image kuvattuna, lähde: https://www.docker.com

Docker-image luodaan *base imagen* päälle, joka sisältää käyttöjärjestelmän tiedostojärjestelmän. Base imagen päälle lisätään muut kerrokset ja lopuksi tästä luodaan valmis Docker-image. Docker-imagella ei ole määriteltyä *tilaa* (state) ja se ei koskaan muutu. Docker-imagelle on täten määrätty vain lukuoikeus, eli kun image ajetaan kontissa, siihen ei tapahdu muutoksia sovelluksen ajon aikana ja sen jälkeen. (Docker docs 2017b)

2.5.3 Docker-kontti

Docker-kontti (docker container), on Docker-imagen pohjalta ajettava varsinainen instanssi eli sovelluksen ajoympäristö, jolla on oma yksityinen ip-osoite ja tallennustila. Docker-konttiin luodaan *konttikerros* (container layer), johon kaikki kontin luomisen jälkeen tehtävät muutokset tehdään, esimerkiksi kontin sisälle asennuksen jälkeen muokatut tiedostot ja asennettavat ohjelmistot. Docker imagen kerroksista poiketen konttikerroksessa on käyttäjällä kaikki tiedosto-oikeudet. (Docker docs 2017a)



Kuva 3. Docker-kontti kuvattuna, lähde: https://docs.docker.com

Docker käyttää isäntätietokoneen kernelin ominaisuuksia Docker-konttien toiminnassa. Docker-konttien resurssien käytöstä vastaa isäntätietokoneen *cgroups* (control groups). Cgroups-ominaisuus myös vastaa Docker-konttien pysäyttämisestä (freeze ja unfreeze). Namespace-ominaisuus puolestaan eristää Docker-kontin tiedostojärjestelmän, nimen, käyttäjät, verkkoyhteyden ja prosessit isäntätietokoneesta. (Docker overview 2017)

2.5.4 Docker-rekisteri

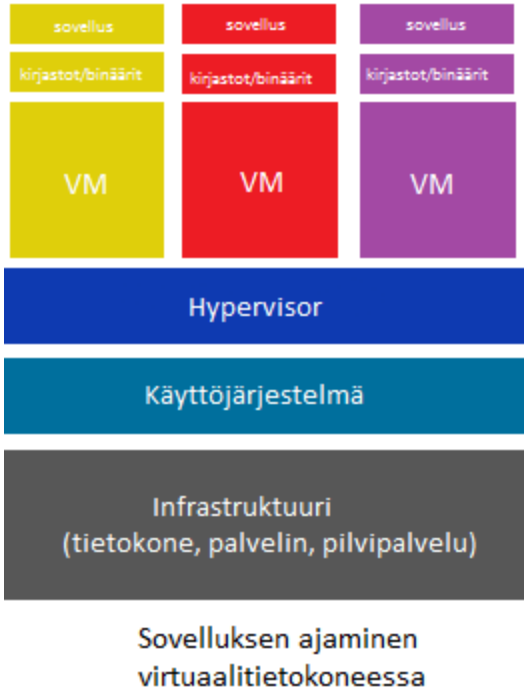
Docker-rekisteriä käytetään Docker-imagejen tallentamista ja levittämistä varten. Rekisterin kautta on mahdollista ladata valmiita Docker-imageja ja tallentaa omia imageja. Docker käyttää oletuksena Docker Hub -rekisteriä, jonka kautta on ladattavissa tuhansia valmiita Docker-imageja ja myös virallisia Docker-imageja, esimerkiksi WordPressin virallinen image. Yritykset käyttävät usein omia Docker-rekistereitään virallisten ja sensitiivistä dataa sisältävien Docker-imagejen tallentamista varten. Oman rekisterin ylläpitäminen myös poissulkee tarpeen ladata Docker-imageja suoraan internetistä. (Mouat 2016)

3 Dockerin ja virtuaalitietokoneen erot

Docker ja virtuaalitietokone poikkeavat toisistaan esimerkiksi toiminnan kannalta. Suurimmat erot liittyvät muun muassa niiden suorituskykyyn ja toimintaan, esimerkiksi resurssien käyttäminen. Tässä kappaleessa olen selittänyt tarkemmin, kuinka Docker-ympäristö poikkeaa virtuaalitietokoneen toiminnasta ja mitä eroja näillä kahdella on.

3.1 Virtualisoidun sovelluspalvelun heikkoudet

Alla olevassa kuvassa on kuvattuna, kuinka virtuaalitietokoneessa ajettava sovellus toimii.



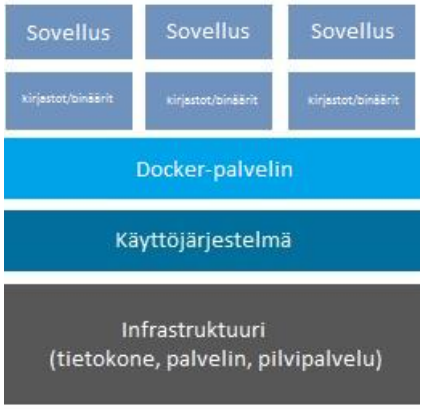
Kuva 4. Virtuaalitietokoneympäristö kuvattuna

Virtuaalitietokone, kuten mikä tahansa tietokone, vaatii toimiakseen infrastruktuurin, esimerkiksi fyysisen tietokoneen, palvelimen tai pilvipalvelimen. Infrastruktuuri käsittää tietokoneen laskentatehon, eli sisältää komponentit, jotka tietokone vaatii toimiakseen (prosessori, muisti, tallennustila). Virtuaalitietokone toimii hypervisorin päällä. Hypervisor on ohjelma, jolla virtuaalitietokoneita hallitaan ja se mahdollistaa niiden kommunikoinnin isäntätietokoneen käyttöjärjestelmän kanssa. Yleisimmät käyttöjärjestelmät ovat Microsoft Windows, Linux-käyttöjärjestelmät ja OSX. (VmWare.com 2017)

Virtuaalitietokone käyttää vain sille varattuja resursseja, jonka täytyy riittää käyttöjärjestelmän ja sovelluksen ajamista varten. Tässä tapauksessa jokaiselle toiminnassa olevalle virtuaalitietokoneelle on varatattava tarpeeksi resursseja sekä käyttöjärjestelmää, että sen päällä pyörivää sovellusta varten, jotta ne toimivat ilman huomattavia ongelmia. (Mouat 2016, s. 4-5)

Sovelluksen ajaminen virtuaalitietokoneessa käyttää isäntäkoneen resursseja, koska kokonaisen käyttöjärjestelmän ajaminen vaatii tarpeeksi muistia, tallennustilaa ja laskentatehoa. Esimerkiksi Ubuntu Server 16.04.1 64-bittinen palvelinkäyttöjärjestelmä vaatii 300 megahertsiä prosessoritehoa, 256 megatavua muistia, vähintään 1,5 gigatavua tallennustilaa ja tämän lisäksi itse sovelluspalvelu vaatii lisää resursseja toimiakseen (Ubuntu.com 2017). Tällöin muutaman sovelluspalvelimen ajaminen virtualisointina kasvattaa resurssien käytön suureksi.

3.2 Dockerin hyödyt verrattuna virtuaalitietokoneeseen



Kuva 5. Docker-ympäristö kuvattuna

Docker-sovelluksen vieminen tuotantokäyttöön on nykypäivänä helpompaa, koska suurimmat pilvipalvelutarjoajat, esimerkiksi Amazon, Google ja Microsoft, tarjoavat suoran tuen Docker-sovelluksille. Docker-palvelu tarjotaan valmiiksi asennettuna vuokrattavalle virtuaalipalvelimelle. Luotu Docker-kontti on samanlainen riippumatta siitä, missä ympäristössä se ajetaan, joten testiympäristössä toimiva Docker-konttiin asennettu sovellus toimii myös tuotantokäytössä. (Docker.com 2017)

3.2.1 Tietoturva

Useamman sovelluksen asentaminen yhdelle virtuaalitietokoneelle ei ole hyvien käytäntöjen mukaista. Tietomurron yhteydessä, jossa yhdessä sovelluksessa olevan tietoturva-aukon kautta tietomurtaja pääsee palvelimelle käsiksi, altistaa se toiset samalla palveli-

mella toimivat sovelluksen tietoturvanuhan alaisiksi. Useamman sovelluksen vaatiessa oman virtuaalipalvelimen infrastruktuuri vaatisi lisäksi huomattavan määrän resursseja. Dockerissa kontti on eristettynä itse isäntäpalvelimesta, joten Docker-konttiin tapahtuvassa mahdollisessa tietomurrossa hyökkäyksen pinta-ala on täten pienempi. Lisäksi useamman Docker-kontin ajaminen yhdellä palvelimella on turvallista, koska myös konti on eristetty toisistaan. Docker-konteissa myös ajetaan vain itse sovellus, joten konttiin ei tarvittaessa asenneta turhia ohjelmia, joka myös pienentää potentiaalista hyökkäyspinta-alaa. (Docker security 2017)

3.2.2 Suorituskyky

Docker-konttiin ei ole asennettuna virtuaalitietokoneen tapaan kokonaista käyttöjärjestelmää, vaan se sisältää vain käyttöjärjestelmän tiedostojärjestelmän. Täten Docker-kontin uudelleenkäynnistämisen voi tehdä hyvin nopeasti. Perinteisessä virtuaalitietokoneessa uudelleenkäynnistäminen vaatii palvelinkäyttöjärjestelmän uudelleenkäynnistämisen, joka kestää kauemmin kuin Docker-kontin uudelleenkäynnistys. Lisäksi jos Docker-palvelimella toimii useampi palvelu, niin muut palvelut ovat yhä saatavilla, vaikka yksi palvelu joudutaan ottamaan väliaikaisesti pois käytöstä. (Mouat 2016, s. 4-5)

4 Wordpress-ympäristön valmistelu

Projektin asetelmana on kuvitteellinen yritys, joka toteuttaa Wordpress-sovelluksen käyttöönoton Docker-teknologialla. Työssä toteutetaan Wordpressin käyttöönotto kahdella tavalla; asentamalla se sekä virtuaalitietokonealustalle, että Docker-palvelimelle. Työssä vertaillaan vaiheittain näitä kahta tapaa toteuttaa Wordpressin käyttöönotto ja samalla pyritään huomioimaan hyödyt ja haitat Dockerin ja virtuaalipalvelinympäristön välillä. Työssä toteutetaan myös MySQL-tietokantapalvelun asentaminen Docker-konttiin ja erilliselle virtuaalipalvelimelle.

Työssä on käytössä neljä virtuaalipalvelinta ja palvelininfrastruktuuri on kuvattu tarkemmin liitteessä 3. Työssä palvelimiin viitataan niiden nimillä.

DOCKER1
Docker-palvelin, johon on asennettu Ubuntu Server 16.04.2 LTS 64-bittinen palvelinkäyttöjärjestelmä. Palvelimelle asensin Docker-palvelu ja lisäksi asensin Wordpress ja MySQL Docker-kontteihin.

DOCKER2
Toinen Docker-palvelin, jota käytin Docker-sovelluksen siirtämistä varten. Palvelimelle asensin CentOS 7 -palvelinkäyttöjärjestelmä, koska työssä on tarkoitus testata Dockerin toimintaa eri käyttöjärjestelmäympäristössä. Siirsin DOCKER1-palvelimella toimivan Wordpress-sovelluksen tälle palvelimelle.

VIRTUAALI
Tälle palvelimelle asensin LAMP-sovelluspinon ja Wordpressin käyttämättä Dockeria. Palvelimelle on asennettu Ubuntu Server 16.04.2 LTS 64-bittinen palvelinkäyttöjärjestelmä.

DATABASE
Tietokantapalvelin, johon on asennettu Ubuntu Server 16.04.2 LTS 64-bittinen palvelinkäyttöjärjestelmä ja MySQL-tietokantapalvelu. Käytin palvelinta Wordpress-sovelluksen tallentamaa tietoa varten.

Tässä työssä palvelimilla annettavat komennot on esitetty *kursivoituna*.

4.1 Palvelinympäristön käyttöönotto

Vuokrasin projektissa käyttämiäni virtuaalipalvelimet DigitalOcean-palvelusta. Yksittäisellä virtuaalipalvelimella on 512 megatavua keskusmuistia ja 20 gigatavua tallennustilaa. Hallinnoin virtuaalipalvelimia etäyhteydellä Putty-etähallintaohjelmalla, joka muodostaa SSH-yhteyden palvelimiin.

4.2 Docker-palvelimen valmistelu

DOCKER1-palvelin on tässä vaiheessa otettu käyttöön ja palvelimen käyttöjärjestelmä ja ohjelmistot on päivitetty ajan tasalle. Dockerin asentaminen vaatii, että palvelimen kernelin versio on 3.10 tai uudempi ja palvelinkäyttöjärjestelmän arkkitehtuuri on 64-bittinen (Mouat 2016, sivu 13).

Kirjauduin palvelimelle SSH-yhteydellä ja asensin Docker-ohjelman. Asensin tämän käyttäen ”curl”-ohjelmaa, joka mahdollistaa verkko-osoitteen sisällön lataamisen tekstiformaattina. Ensiksi ”curl” lataa palvelimelle verkko-osoitteen sisällön, joka on Dockerin asennustiedosto ja tämän jälkeen se suoritetaan palvelimella shell-skriptitiedostona, joka asentaa Dockerin palvelimelle. (Mouat 2016, sivu 14)

Asennustiedoston ajoin seuraavalla komennolla.

$ curl -sSL https://get.docker.com | sh

Asennus on ajettu loppuun, jonka jälkeen tarkistin, että Docker asennettiin palvelimelle ja samalla tarkistin sen versionumeron. Kuvasta 5 näkyy, että asennustiedosto asensi Dockerista version 17.03.0-ce.

```
oppari1@opparidocker1:~$ docker --version
Docker version 17.03.0-ce, build 60ccb22
oppari1@opparidocker1:~$
```

Kuva 5, Dockerin version tarkistus

Dockerin käyttäminen vaatii Unix-käyttäjältä pääkäyttäjäoikeudet, mutta lisäämällä käyttäjä ”docker”-nimiseen Unix-ryhmään Dockerin käyttäminen onnistuu ilman pääkäyttäjäoikeuksia. Kyseinen ryhmä luodaan Dockerin asennuksen yhteydessä. (Mouat 2016, sivu 15)

Lisäsin DOCKER1-palvelimen käyttäjän ”docker”-nimiseen ryhmään.

$ sudo usermod -aG docker oppari1
$ logout

Uloskirjautumisen jälkeen kirjauduin uudelleen DOCKER1-palvelimelle ja uudelleenkirjautumisen jälkeen ajoin Dockerin virallisen ”hello-world” -nimisen Docker-imagen, joka on kehitetty Docker-palvelun toimivuuden testaamista varten. Ajoin Dockerissa ”run”-komennon, jonka parametriksi määritellään Docker-imagen nimi, josta Docker-kontti luodaan. (Docker docs 2017g)

$ docker run hello-world

Komento loi uuden Docker-kontin ”hello-world” -nimisestä Docker-imagesta. Kyseistä imagea ei löytynyt paikallisesti DOCKER1-palvelimelta, joten Docker lataa sen palvelimelle automaattisesti Docker Hub -oletusrekisteristä. Luotu Docker-kontti suorittaa vain komennon, joka tulostaa kuvan mukaisen tekstin. (Docker docs 2017g)

```
oppari1@opparidocker1:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
78445dd45222: Pull complete
Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7
Status: Downloaded newer image for hello-world:latest

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

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

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

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

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/

oppari1@opparidocker1:~$
```

Kuva 6, ”hello-world” Docker-imagen testaaminen

Tässä vaiheessa Docker on asennettu DOCKER1-palvelimelle ja testasin Docker-kontin luomista onnistuneesti. Seuraavaksi poistin Docker-kontin ”hello-world” DOCKER1-palvelimelta, koska sille ei ollut testauksen jälkeen enää käyttöä. Ennen kuin Docker-kontti voidaan poistaa, täytyy se ensiksi sulkea (Docker docs 2017c).

$ docker stop hello-world
$ docker rm hello-world

4.3 Virtuaalitietokoneen valmistelu

Virtuaalipalvelin VIRTUAALI on tässä vaiheessa otettu käyttöön ja sille on asennettu DOCKER1-palvelimen tapaan kaikki päivitykset. Virtuaalipalvelimelle asennetaan tässä työvaiheessa LAMP-sovelluspino. Apache ja PHP asennetaan VIRTUAALI-palvelimelle ja MySQL-tietokantapalvelu asennetaan DATABASE-palvelimelle.

4.3.1 Apachen asentaminen VIRTUAALI-palvelimelle

Asensin Apache web-palvelimen käyttäen Ubuntun ”apt”-paketinhallintaohjelmaa.

$ sudo apt-get install apache2 -y

Asennus on ajettu loppuun ja ilman virheilmoituksia. Apachesta on nyt asennettuna versionumero 2.4.18.

Apachessa oletuksena verkkosivujen juurihakemisto on ”/var/www/html”. Tämä hakemistopolku vaatii pääkäyttäjäoikeudet ja tämä ei ole hyvän käytännön mukaista, koska verkkosivujen muokkaaminen ei onnistuisi tavalliselta käyttäjältä. Apachessa on moduuli ”userdir”, joka mahdollistaa verkkosivujen juurihakemiston luomisen jokaiselle käyttäjälle heidän kotihakemistoonsa. Tällöin käyttäjällä on oikeudet muokata kyseistä hakemistopolkua ja muut käyttäjät eivät puolestaan voi muokata toisten käyttäjien kotihakemistossa olevia tiedostoja. (Apache docs 2017)

Otin käyttöön käyttäjien kotihakemistot VIRTUAALI-palvelimen Apachessa ja tämän jälkeen käynnistin kyseisen palvelun uudelleen.

$ a2enmod userdir
$ sudo service apache2 restart

4.4 PHP:n asentaminen virtuaalipalvelimelle

Asensin PHP:n Ubuntun apt-paketinhallintaohjelmalla ja Asensin projektia varten PHP:stä version php7.0: n ja PHP-moduulin Apachea varten.

$ sudo apt-get install php7.0 libapache2-mod-php7.0 -y

Tämän jälkeen muokkasin Apachen php-konfiguraatiotiedostoa (liite 4) niin, että Apache tukee tämän jälkeen PHP-ohjelmointikieltä ja toimenpiteen jälkeen käynnistin Apachen uudelleen, koska muokattuani asennustiedostoa muutokset eivät muuten astuisi voimaan.

Käynnistin Apachen uudelleen.

$ sudo service apache2 restart

4.5 MySQL-palvelimen valmistelu

Wordpress-sovellus vaatii tietokantapalvelimen tiedon tallentamista varten, joten MySQL-asentaminen on tehtävä ennen Wordpressin asentamista. Projektissa Docker-konttiin asentamaani Wordpressiä varten tein myös MySQL-palvelulle oman Docker-kontin. Vertasin Docker-kontiin asennetun MySQL-palvelun ja virtuaalitietokoneelle asennetun tietokantapalvelimen eroja käyttöönoton ja ylläpidon näkökulmasta.

4.5.1 MySQL:n asentaminen virtuaalitietokoneelle

Asensin MySQL:n DATABASE-palvelimelle Ubuntun apt-paketinhallintaohjelmalla ja päivitin apt-pakettivaraston paketit ennen asentamista.

$ sudo apt-get update
$ sudo apt-get install mysql-server-5.7

Asennuksen yhteydessä MySQL:lle luodaan pääkäyttäjän root salasana ja tämän jälkeen asennus tapahtui onnistuneesti.

Kirjauduin DATABASE-palvelimelle pääkäyttäjänä ja loin uuden tietokannan ”wordpress” ja käyttäjän ”yllapito”. Annoin luodulle käyttäjälle kaikki oikeudet luotuun tietokantaan. MySQL:n asentamisen ja käyttäjän ja taulukon luomisen jälkeen tietokantapalvelin on valmisteltu. Palomuurista avasin lopuksi TCP-portin 3306, koska MySQL-tietokantapalvelu käyttää kyseistä TCP-verkkoliikenneporttia tietokantayhteyksiä varten (MySQL Documentaion 2017).

4.5.2 MySQL:n asentaminen Docker-kontiin

MySQL:stä on Wordpressin tapaan luotu valmis Docker-image, jota käytin tässä projektissa luodakseni MySQL:stä Docker-kontin. MySQL:n Docker-imagen latasin DOCKER1-palvelimelle Docker Hub -rekisteristä.

$ docker pull mysql

Ladattuani MySQL:n Docker-imagen loin siitä Docker-kontin, jonka nimeksi annoin ”wpdatabase”.

$ docker run --name wpdatabase -d mysql

Docker-kontti käynnistyy automaattisesti heti luomisen jälkeen (Docker docs 2017b). Luomani Docker-kontti näkyi DOCKER1-palvelimella, mutta se ei ollut luomisen jälkeen päällä. Tarkastelin ”wpdatabase”-kontin lokitietoja, joista pyrin etsimään tälle syytä.

$ docker logs wpdatabase

```
error: database is uninitialized and password option is not specified
  You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
```

Kuva 7, mysql-kontin virheilmoitus

Kuvassa 7 näkyvän virheilmoituksen mukaisesti luodulle MySQL Docker-kontille on määriteltävä yksi virheilmoituksen mukaisista ympäristömuuttujista (enviroment variable). Ympäristömuuttuja on Dockerissa Docker-kontin ulkopuolelle tallennettava parametri, jota kontti käsittelee kontin ajon yhteydessä. Esimerkiksi kirjautumistunnukset ja konfiguraatiotiedostot on hyvän käytännön mukaisesti tallennettava ympäristömuuttujiin, koska näin niitä voi käyttää yhä uudelleen, vaikka kontti poistettaisi (Docker docs 2017d).

Poistin ensiksi Docker-kontin ”wpdatabase” ja loin se uudelleen samalla nimellä käyttäen yhtä ympäristömuuttujaa. ”Wpdatabase”-kontti täytyi ensiksi pysäyttää, ennen kuin pystyin poistaa sen.

$ docker stop wpdatabase
$ docker rm wpdatabase

Loin ”wpdatabase”-kontin uudelleen. Annoin MYSQL_ROOT_PW -ympäristömuuttujan parametriksi MySQL-palvelun pääkäyttäjän salasanan.

$ docker run -e MYSQL_ROOT_PW=[salasana] -d mysql:5.7

Kirjauduin sisälle ”wpdatabase”-konttiin ja loin uuden MySQL-tietokannan ”wordpress”.

Sain asennettua MySQL-tietokantapalvelun Docker-konttiin, mutta en huomannut tässä työvaiheessa käytännön hyötyjä, koska asentaminen ei poikennut virtuaalitietokoneelle asennettavasta MySQL-palvelusta. En koe myöskään tietokantapalvelun ajamista tietoturvallisena, koska konttia ei voi rakentaa tai päivittää ilman tiedon menettämisen riskiä. RedHat:kin suosittelee tiedon tallentamisen kontin ulkopuolelle (Redhat Developer Program 2017).

MySQL:n asentaminen Docker-konttiin suorituskyvyn kannalta voi olla ongelmallista suurten tietokantapalvelun järjestelmävaatimusten takia. Projektissa tietokantapalvelua käyttää vain kerralla yksi WordPress-sovellus, joka ei käytä resursseja samalla tavalla kuin sovellus, joka tekee satoja tai jopa tuhansia samanaikaisia tietokantakyselyitä.

5 WordPressin asentaminen

Tässä työvaiheessa toteutin WordPressin asentamisen sekä virtuaalipalvelinympäristöön, että Docker-konttiin. VIRTUAALI-palvelimelle on tässä vaiheessa asennettu Apache web-palvelin ja PHP-ohjelmointikieli. Aloitin ensimmäiseksi WordPressin asentamisen VIRTUAALI-palvelimelle ja tämän jälkeen asensin Wordpressin Docker-konttiin ja lopuksi vertailin Dockerin hyötyjä virtuaalipalvelinympäristöön verrattuna.

5.1 WordPressin asennus VIRTUAALI-palvelimelle

Wordpress vaatii php7.0-mysql -nimisen moduulin mahdollistaakseen MySQL-tietokannan käsittelyn PHP-skripteillä, kun esimerkiksi blogiteksti tallennetaan tietokantaan. Asensin VIRTUAALI-palvelimelle php7.0-mysql -moduulin.

$ sudo apt-get install php7.0-mysql

Wordpressin asennuksen aloitin lataamalla asennuspaketin tar-paketointiformaatissa VIRTUAALI-palvelimelle. Tämän jälkeen purin asennuspaketin VIRTUAALI-palvelimelle luodun käyttäjän kotihakemistoon ”public_html” -hakemiston alle.

$ wget http://wordpress.org/tar.gz -P ~/public_html
$ tar xf latest.tar.gz

Avasin internetselaimella VIRTUAALI-palvelimelle ladatun Wordpress-sovelluksen kirjoittamalla osoiteriville http://VIRTUAALI:80, jossa palvelimen nimi on sen ip-osoite ja TCP-portin numero, jota Wordpress käyttää kyseisellä palvelimella.

Selaimella avautui WordPressin asennusohjelma. Wordpressin asentamisvaiheessa määritellään WordPress-sovelluksen käyttämä tietokantapalvelin ja Wordpress-sovelluksen pääkäyttäjän tunnus, salasana ja sovelluksen käyttämä nimi. Käytin WordPressia varten DATABASE-palvelinta tietokanta-asetuksissa. Lopuksi ajoin asennustiedoston loppuun ja sain asennettua Wordpressin VIRTUAALI-palvelimelle.

5.2 WordPressin asentaminen Docker-konttiin

Seuraavaksi asensin Wordpressin Docker-konttiin luomalla uuden Docker-kontin Wordpressin virallisesta Docker-imagesta. Wordpressin Docker-imagen sisältämät sovelluskomponentit ja -kirjastot ovat eritelty erikseen liitteessä 2.

5.2.1 WordPress-kontin luominen imagesta

Wordpressin kehittäjät ovat itse kehittäneet valmiin Docker-imagen WordPressistä, jota käytin tässä työssä. Latasin DOCKER1-palvelimelle WordPressin virallisen Docker-imagen.

$ docker pull wordpress

Loin ladatusta WordPressin Docker-imagesta uuden Docker-kontin, jonka nimesin ”wordpress2”:ksi. Muodostetaan sen ja ”wpdatabase”-kontin välille siltaava yhteys. Anooin komennon yhteydessä parametrina WordPressin käyttämän tietokannan salasanan ja verkko-osoitteen, joka on ”wpdatabase”-kontin IP-osoite. WordPress-sovellus toimii julkisessa TCP-portissa 80, johon tuleva ulkoinen verkkoliikenne ohjataan Docker-kontin omaan TCP-porttiin 80.

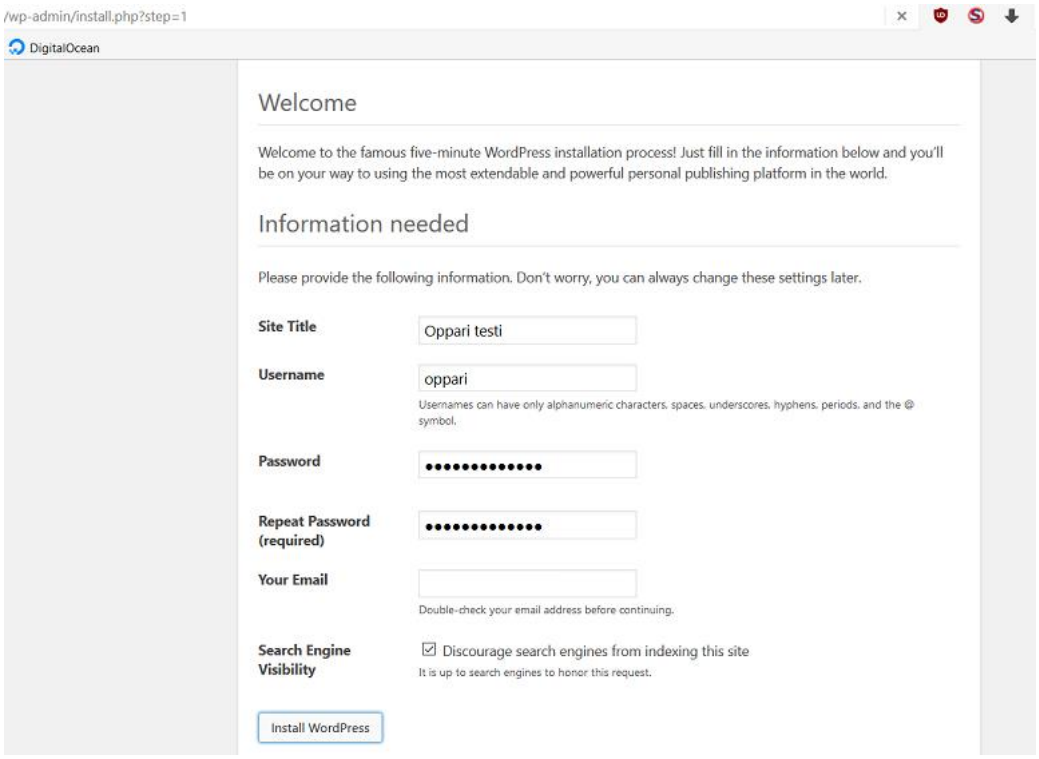
$ docker run -e WORDPRESS_DB_PASSWORD=[tietokannan salasana] -e WORDPRESS_DB_HOST=172.17.0.2 --name wordpress2 --link wpdatabase:mysql -p 80:80 -d wordpress

Luotuani Docker-kontin testasin, että Wordpress-sovellus toimii julkisessa verkossa. Muodostin internet-selaimella yhteyden DOCKER1-palvelimen verkko-osoitteeseen ”http://DOCKER1:80”, jossa palvelimen nimen tilalle kirjoitettiin sen IP-osoite.

WordPressin asennussivusto aukesi selainnäkymässä, joten luomani Docker-kontti WordPressia varten toimi. Docker-imagen lataaminen DOCKER1-palvelimelle kesti alle minuutin ja ”wordpress2”-kontin luominen kesti noin kolme sekuntia. Lataamastani Wordpressin Docker-imagesta olisin voinut luoda palvelimen resurssien puitteessa loputtomasti samankaltaisia Docker-kontteja, joten oma havaintoni on, että Dockerilla sovellusympäristön luominen on erittäin nopeaa.

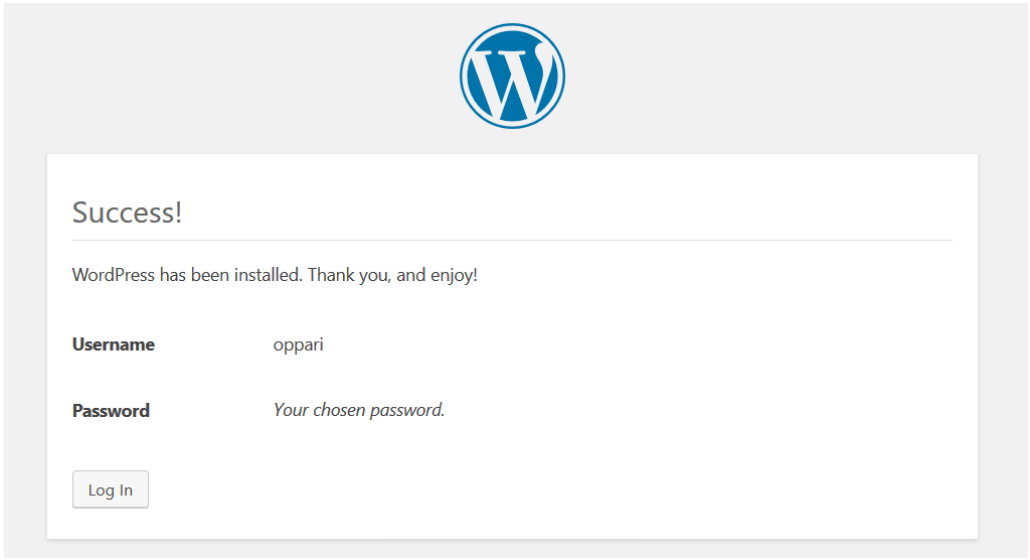
Suoritin WordPressin asennuksen loppuun. Määritin kuvan 8 mukaiset asetukset ”wordpress2”-kontissa toimivalle WordPress-sovellukselle. Asennuksessa sähköpostiosoi-

te on piilotettu tietoturvan takia, koska asennuksessa määritellään olemassa oleva sähköposti.



Kuva 8. WordPressin asennussivusto

Asennuksen ajettua sain ilmoituksen, että WordPress on asennettu onnistuneesti.

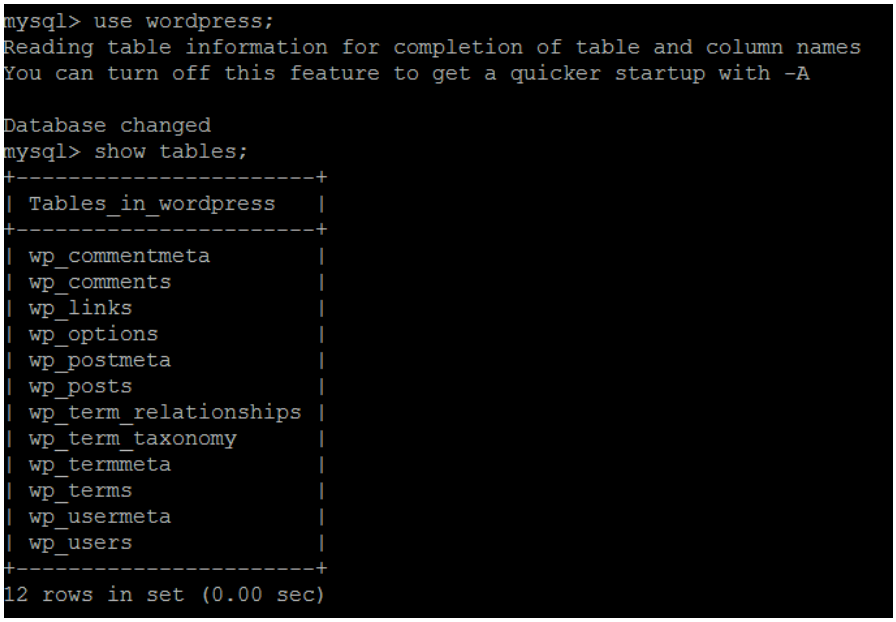


Kuva 9. Wordpressin ilmoitus onnistuneesta asennuksesta

Lopuksi tarkistin, että Wordpress-asennus on tallennettu ”wpdatabase”-kontissa toimivalle MySQL-tietokantaan. Muodostin yhteyden ”wpdatabase”-konttiin DOCKER1-palvelimella.

$ docker exec -ti wpdatabase bash

Kirjauduin ”wpdatabase”-kontissa MySQL-palveluun root-käyttäjänä ja hain kaikki taulukot tietokannasta ”wordpress”.



Kuva 10. ”wpdatabase” Docker-kontin ”wordpress”-tietokannan taulukot

5.3 MySQL Docker-kontin heikkoudet

MySQL on myös asennettu onnistuneesti Docker-konttiin ja Wordpress-sovelluksen data on onnistuneesti tallennettu ”wpdatabase”-konttiin. MySQL:n ajaminen Docker-kontissa ei tuonut sovelluksen käyttöönoton kannalta hyötyä sen sijaan, että tietokanta asennettaisi DATABASE-palvelimelle. Ainoa huomioni oli, että MySQL-tietokantapalvelun käyttöönottaminen Dockerilla oli nopeampaa kuin DATABASE-palvelimella käyttöönottamani MySQL-tietokantapalvelu.

Lisäksi Docker-konttiin tallennettu data muodostaa riskin tiedonsäilömisen kannalta. Yksi Dockerin hyvistä puolista on sovellusten nopea käyttöönotto ja Docker-kontin tuhoaminen tai uudelleenrakentaminen pitää onnistua mahdollisimman vaivattomasti (RedHat 2016). Dockerissa on täten käytettävä *tallennusvolyymia* (data volume), joka luodaan kontin tiedostojärjestelmän ulkopuolelle. Tallennusvolyymi toimii samalla periaatteella kuin ympäristömuuttujat, mutta parametrien sijaan sitä käytetään tiedon tallentamista varten. Esimerkiksi Docker-kontin luomisen jälkeen luodut asennustiedostot eivät säily, jos kontti tuhotaan, mutta tallentamalla ne datavolyymiin vältytään tältä. (Docker docs 2017e)

Virtuaalipalvelimelle täytyi asentaa useampi ohjelma, jotka ovat välttämättömiä Wordpressin toiminnan kannalta. Docker-palvelimella täytyy vain asentaa Docker-palvelu. Kuvitteellinen yritys voi täten ajaa Docker-palvelulla Wordpressin lisäksi muita sovelluksia ilman, että palvelimelle on asennettava useampi ohjelma tai ohjelmistokirjasto sovelluksen toimivuutta varten.

Wordpress-sovellus toimii Docker-kontissa, joka on eristetty ympäristö isäntäpalvelimesta. Tällöin voi olettaa, että Wordpress-sovelluksen jouduttua verkkohyökkäyksen kohteeksi itse isäntäpalvelimeen ei kohdistu suoraa uhkaa. Docker-kontin poistaminen ja uudelleenluominen olivat myös nopeita prosesseja, joten mahdollisen hyökkäyksen yhteydessä.

5.4 Wordpress-kontin asentaminen erilliselle tietokantapalvelimelle

Suoritin vielä toisen Wordpress-sovelluksen asentamisen Docker-konttiin käyttäen tiedontallentamista varten DATABASE-palvelimen MySQL-tietokantaa Docker-konttiin asentamani MySQL-tietokantapalvelun sijaan. Pyrin tällä ratkaisemaan Docker-kontissa toimivan Wordpress-sovelluksen tiedontallentamiseen liittyvät ongelmat, jotka koskevat Docker-konttiin asentamaani MySQL-tietokantapalvelua.

5.4.1 Wordpress-kontin valmisteleminen

Loin DATABASE-palvelimelle tietokannan ”wordpressdocker” ja annoin luotuun tietokantaan kaikki oikeudet MySQL:n käyttäjälle ”yllapito”. DOCKER1-palvelimella loin Docker-kontin ”wordpress3” uutta Wordpress-sovellusta varten. Tällä kertaa ei linkitetä konttia ”wpdatabase”-kontiin, vaan määritellään Wordpress-sovellus käyttämään DATABASE-palvelinta tiedon tallentamista varten. Wordpress-sovellus käyttää ulkoista TCP-porttia 8888.

$ docker run --name wordpress3 -p 8888:80 -d wordpress

MySQL:n etäyhteyttä varten Docker-kontissa on oltava asennettuna ”mysql-client” -moduuli, joka mahdollistaa etäyhteyden muodostamisen MySQL-palvelimelle. Wordpressin virallinen Dockerfile ei sisällä mysql-client -moduulia (Docker hub 2017a), joten minun täytyi asentaa se erikseen luomaani Docker-konttiin. Tässä vaiheessa yritin asentaa kontin sisällä moduulin apt-paketinhallintaohjelmalla, mutta Docker-palvelimen keskusmuisti loppui kesken komennon ajon ja täten komentoa ei voitu suorittaa loppuu.

Ratkaisin tämän ongelman luomalla Wordpressista uuden imagen, johon asensin lisäksi tarvitsemani ”mysql-client” -moduulin. Loin ensimmäiseksi DOCKER1-palvelimelle käyttä-

jän kotihakemistoon ”wordpress-custom” -hakemisto ja sinne loin tiedoston ”Dockerfile”. Tiedoston sisältö on esiteltynä seuraavan sivun alussa.

 # Dockerfilen sisältö
FROM library/wordpress
RUN apt-get update && apt-get install mysql-client -y

Dockerfile lataa ensiksi virallisen Wordpress-imagen ja sen jälkeen ”apt-get” -komennolla asentuu ”mysql-client”. Prosessin lopuksi Docker rakentaa valmiin Docker-imagen, joka sisältää nyt virallisen Wordpress-imagen lisäksi myös ”mysql-clientin”.

Rakensin Docker-imagen luomani Dockerfilesta ”docker build” -komennolla.

$ docker build ~/wordpress-custom/Dockerfile

```
oppari1@opparidocker1:~/wordpress-custom$ docker build .
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM library/wordpress
 ---> 2d81af9ee904
Step 2/3 : CMD apt-get update
 ---> Running in 5ae906a90b17
 ---> 601cca5db33d
Removing intermediate container 5ae906a90b17
Step 3/3 : CMD apt-get install mysql-client
 ---> Running in 9cbacc0a89a1
 ---> b1cc1a38700b
Removing intermediate container 9cbacc0a89a1
Successfully built b1cc1a38700b
```

Kuva 11. Docker imagen luominen dockerfilesta.

```
oppari1@opparidocker1:~/wordpress-custom$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              b1cc1a38700b        20 seconds ago      401MB
wordpress           latest              2d81af9ee904        11 days ago         401MB
mysql               latest              9e64176cd8a2        11 days ago         407MB
ubuntu              latest              f7b3f317ec73        11 days ago         117MB
ubuntu              xenial              f7b3f317ec73        11 days ago         117MB
ubuntu              trusty              302fa07d8117        3 weeks ago         188MB
alpine              latest              4a415e366388        2 months ago        3.99MB
```

Kuva 12. Luomani Docker image (<none>).

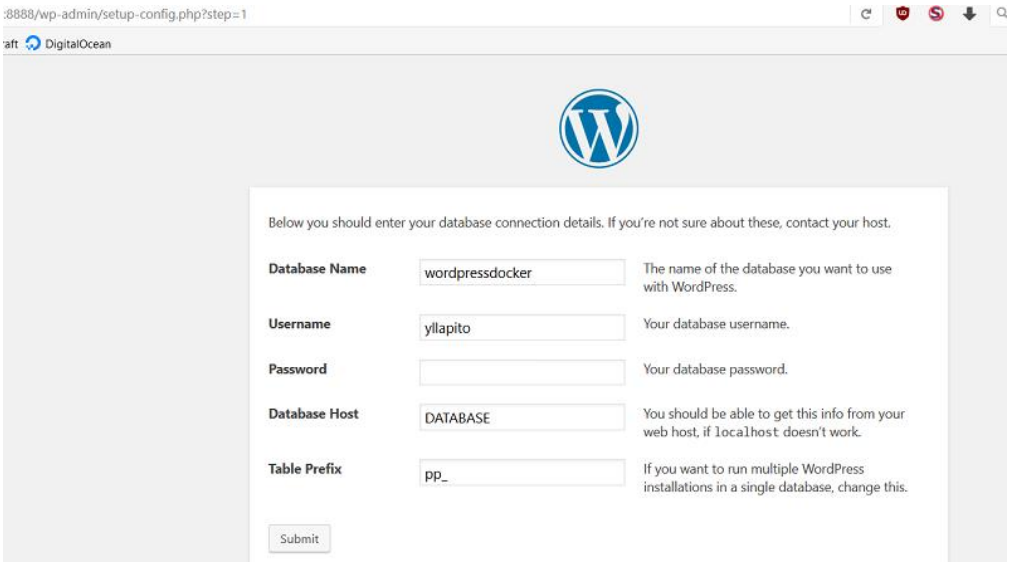
Docker-imagen luominen onnistui ja sen id:ksi, eli tunnisteeksi, muodostui kuvan 12 mukaisesti b1cc1a38700b. Poistin aikaisemman luomani Docker-kontin ”wordpress3”:n.

$ docker stop wordpress3
$ docker rm wordpress3

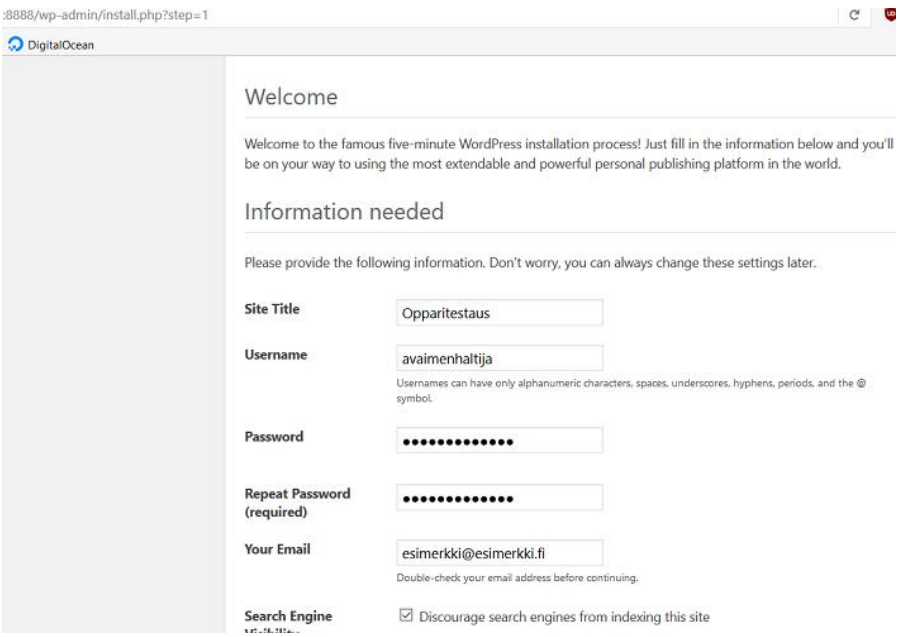
Tämän jälkeen loin uuden Docker-kontin luomastani Docker-imagesta.

$ docker run –name wordpress3 -p 8888:80 -d b1cc1a38700b

Avasin kontin luomisen jälkeen selaimella DOCKER1-palvelimen osoitteeseen ”http://DOCKER1:8888”. Tällä kertaa minun täytyi määritellä tietokantapalvelimen asetukset manuaalisesti. Tietokantamäärityksien jälkeen syötin kuvan 13 mukaiset tiedot WordPressin asennusohjelman tekstikenttiin ja suoritin tämän jälkeen asennuksen loppuun.



Kuva 13. Wordpressin tietokantamäärittelyt asennussivustolla.



Kuva 14. Wordpressin asennussivusto.

Asennus suoritettiin onnistuneesti, joten onnistuin asentamaan Wordpressin Docker-konttiin ja kyseisen sovelluksen määrittelin käyttää ulkoista DATABASE- tietokantapalvelinta. Lopuksi kirjauduin sisälle ”wordpress3”-kontissa toimivaan Wordpress-sovellukseen luomallani käyttäjällä ”avaimenhaltija”.

5.5 Docker-kontin siirto toiselle Docker-palvelimelle

Käytin projektissa toista Docker-palvelinta DOCKER2, johon asensin käyttöjärjestelmäksi CentOS 7 64-bittisen palvelin käyttöjärjestelmän. Tämän työvaiheen tarkoituksena on tarkastella toteutustapaa, kuinka Docker-kontti siirretään DOCKER1-palvelimelta DOCKER2-palvelimelle. Tässä tilanteessa CentOS 7 -palvelin on päivitetty ja palomuurissa kaikki yhteydet on estetty. Avataan TCP-portit 4232 Dockeria ja 80 Wordpressiä varten. Asennetaan Docker palvelimelle samalla tavalla, kuten DOCKER1-palvelimella asennus suoritettiin.

DOCKER1-palvelimella toimivan Wordpress-sovelluksen blogitekstit, luodut sivut ja kommentit voi siirtää toiseen sovellukseen tekemättä mitään tietokantapalvelimen puolella. Wordpressissä on vakiona ”import” ja ”export” -toiminnot. ”Export”-toiminto tallentaa blogitekstit, luodut sivut ja kommentit erilliseen .xml -tiedostoon, jonka voi taas viedä toiseen Wordpress-sovellukseen ”Import”-toiminnolla.

Käytin Wordpressin Export-toimintoa DOCKER1-palvelimella ”wordpress3”-kontissa toimivassa Wordpress-sovelluksessa ja latasin xml-tiedoston paikalliselle työasemalle. Tämän jälkeen kirjauduin DOCKER2-palvelimella toimivaan WordPress-sovellukseen ja Import-toiminnolla latasin xml-tiedoston Wordpress-sovellukseen. Tämän jälkeen ”wordpress3”-kontissa toimivan WordPress-sovelluksen blogikirjoitukset ja sivut siirtyivät ”wordpress10”-kontissa toimivaan WordPressiin.

En siis varsinaisesti siirtänyt Wordpress-sovellusta DOCKER1-palvelimelta DOCKER2-palvelimelle. RedHatin esittelemien käytäntöjen mukaisesi (RedHat 2016) loin DOCKER2-palvelimella uuden Docker-kontin käyttäen aiemmin luomaani Docker-imagea WordPressistä, joten täten luotu ympäristö oli täysin samanlainen aikaisempaan verrattuna. Tässä työvaiheessa kuvitteelisen yrityksen ei myös tarvitse huolehtia DOCKER2-palvelimen sovelluskomponenteista, vaan minulle riitti vain ja ainoastaan, että asensin Docker-palvelun kyseiselle palvelimelle.

6 Johtopäätökset

Opinnäytetyöni tarkoituksena oli toteuttaa Wordpress-sovelluksen käyttöönotto Dockerilla, jonka sain lopulta toimimaan. Docker-kontin luominen oli erittäin nopeaa verrattuna virtuaalitietokoneeseen. Esimerkiksi uuden virtuaalitietokoneen luominen valmiina olevasta levykuvasta kestää huomattavasti pidempään kuin uuden Docker-kontin luominen. Dockerissa tarvitsin vain sovellusta varten luodun ajoympäristön, eli Docker-kontin, WordPressia varten ja tämän luominen kesti yhteensä noin minuutin verran. Lisäksi Docker-kontin ympäristö oli valmiiksi konfiguroitu, joten minun ei täytynyt tehdä valmisteltavia toimenpiteitä Wordpress-sovellusta varten.

Projektin aikana huomasin, että MySQL:n suorittaminen Docker-kontissa ei tuonut suurta hyötyä. Projektin kannalta riittävää oli, että tieto saatiin varastoitua tietokantapalvelimelle ja tietokantaan saadaan yhteys myös sovelluksen siirron jälkeen. Docker kuitenkin selvästi lisää sovelluksen tietoturvaa, koska Docker-kontti eristää sovelluksen itse isäntäpalvelimesta.

Projektin aikana syntyi lukuisia jatkokehitysideoita, joita en voinut selvittää tässä työssä. Docker-kontit mahdollistivat Wordpress-sovelluksen nopean luomisen ja muokkaamisen, kuten tein ”mysql-clientin” tapauksessa, mutta työssä en testannut Docker-kontin suorituskykyä kuormitustestin aikana. Lisäksi jatkokehitysideoina projektin aikana syntyi, kuinka tiedontallentamisen ja täten myös tietokantapalvelimen ylläpitämisen tulee tehdä Docker-kontissa hyvien tietoturvakäytäntöjen mukaisesti ja useamman kuin kahden Docker-kontin käyttöönotto ja hallinta, eli kokonaisen Docker-infrastruktuurin hallinta.

Projektin aikana opin Dockerin perusteet ja ymmärrän nyt paremmin sen tuottamat hyödyt. Ymmärrän miksi Wordpressin tapaisten, pienten sovelluksien asentaminen virtuaalitietokoneelle ei ole nykyään kannattavaa. Projektin aikana ymmärsin konttiteknologian tuottamat hyödyt erityisesti suorituskyvyssä ja käyttöönotossa.

Lähteet

Adrian Mouat. 2016. Using Docker: Developing and Deploying Software with Containers. Yhdysvallat. O'Reilly.

Apache docs. 2017. Luettavissa: https://httpd.apache.org/docs/2.4/howto/public_html.html. Luettu 15.4.2017.

Docker.com. 2017. Luettavissa: https://www.docker.com/what-docker. Luettu 5.4.2017.

Docker docs. 2017a. Luettavissa: https://docs.docker.com/get-started/#setup. Luettu: 8.4.2016.

Docker docs. 2017b. Luettavissa: https://docs.docker.com/engine/reference/run/. Luettu 12.4.2017.

Docker docs. 2017c. Luettavissa: https://docs.docker.com/engine/reference/commandline/rm/. Luettu: 12.4.2017

Docker docs. 2017d. Luettavissa: https://docs.docker.com/docker-cloud/getting-started/deploy-app/6_define_environment_variables/. Luettu 15.4.2017.

Docker docs. 2017e. Luettavissa: https://docs.docker.com/engine/tutorials/dockervolumes/. Luettu: 10.4.2017.

Docker overview. 2017. Luettavissa: https://docs.docker.com/engine/docker-overview/#the-docker-client. Luettu 17.3.2017

Docker security. 2017. Luettavissa: https://docs.docker.com/engine/security/security/. Luettu: 19.3.2017.

Docker hub. 2017a. Luettavissa: https://hub.docker.com/_/mysql/. Luettu 10.3.2017.

MySQL Documentation. 2017. Luettavissa: https://dev.mysql.com/doc/refman/5.7/en/connecting.html. Luettu 14.3.2017

Redhat Developer Program. 2016. Luettavissa: https://developers.redhat.com/blog/2016/02/24/10-things-to-avoid-in-docker-containers/. Luettu: 18.3.2017.

Ubuntu.com. 2017. Luettavissa: https://help.ubuntu.com/community/Installation/SystemRequirements. Luettu: 19.3.2017

VmWare.com. 2017. Luettavissa: http://www.vmware.com/solutions/virtualization.html. Luettu: 18.3.2017

WordPress.org. 2017. Luettavissa: https://wordpress.org/about/. Luettu 10.3.2017.

Liitteet

Liite 1. Docker-komennot ja parametrit

run
Luodaan uusi kontti Docker-imagesta.

--name
Kontille määriteltävä nimi

-d
Docker-imagen nimi tai id, jota käytetään Docker-kontin luomiseen

-p
Docker-kontille määritettävät TCP-porttiasetukset,

pull
Lataa Docker-imagen. Parametriksi määritellään Docker-imagen nimi.

rm
Poistaa Docker-kontin. Kontti täytyy ensiksi sammuttaa.

stop
Sammuttaa Docker-kontin.

images
Näyttää paikallisella palvelimella olevat Docker-imaget.

ps
Näyttää käynnissä olevat Docker-kontit

-a
Näyttää kaikki olemassa olevat Docker-kontit, niin käynnissä olevat, kuin suljetut kontit.

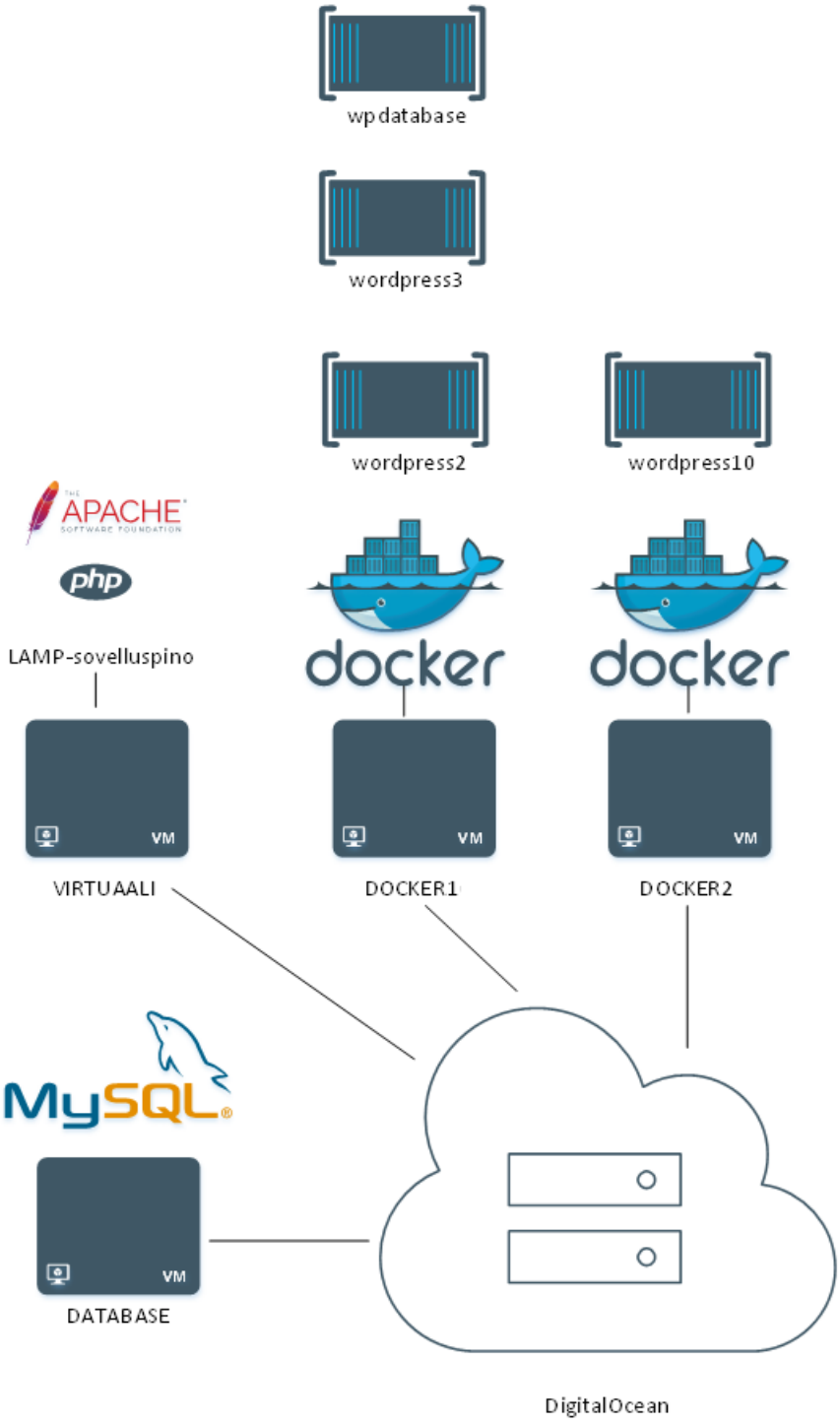
build
Rakentaa Docker-imagen.

Liite 2. Wordpressin Docker-imagen komponentit

Supported tags and respective `Dockerfile` links

- `4.7.5-apache`, `4.7-apache`, `4-apache`, `apache`, `4.7.5`, `4.7`, `4`, `latest`, `4.7.5-php5.6-apache`, `4.7-php5.6-apache`, `4-php5.6-apache`, `php5.6-apache`, `4.7.5-php5.6`, `4.7-php5.6`, `4-php5.6`, `php5.6` (*php5.6/apache/Dockerfile*)
- `4.7.5-fpm`, `4.7-fpm`, `4-fpm`, `fpm`, `4.7.5-php5.6-fpm`, `4.7-php5.6-fpm`, `4-php5.6-fpm`, `php5.6-fpm` (*php5.6/fpm/Dockerfile*)
- `4.7.5-fpm-alpine`, `4.7-fpm-alpine`, `4-fpm-alpine`, `fpm-alpine`, `4.7.5-php5.6-fpm-alpine`, `4.7-php5.6-fpm-alpine`, `4-php5.6-fpm-alpine`, `php5.6-fpm-alpine` (*php5.6/fpm-alpine/Dockerfile*)
- `4.7.5-php7.0-apache`, `4.7-php7.0-apache`, `4-php7.0-apache`, `php7.0-apache`, `4.7.5-php7.0`, `4.7-php7.0`, `4-php7.0`, `php7.0` (*php7.0/apache/Dockerfile*)
- `4.7.5-php7.0-fpm`, `4.7-php7.0-fpm`, `4-php7.0-fpm`, `php7.0-fpm` (*php7.0/fpm/Dockerfile*)
- `4.7.5-php7.0-fpm-alpine`, `4.7-php7.0-fpm-alpine`, `4-php7.0-fpm-alpine`, `php7.0-fpm-alpine` (*php7.0/fpm-alpine/Dockerfile*)
- `4.7.5-php7.1-apache`, `4.7-php7.1-apache`, `4-php7.1-apache`, `php7.1-apache`, `4.7.5-php7.1`, `4.7-php7.1`, `4-php7.1`, `php7.1` (*php7.1/apache/Dockerfile*)
- `4.7.5-php7.1-fpm`, `4.7-php7.1-fpm`, `4-php7.1-fpm`, `php7.1-fpm` (*php7.1/fpm/Dockerfile*)
- `4.7.5-php7.1-fpm-alpine`, `4.7-php7.1-fpm-alpine`, `4-php7.1-fpm-alpine`, `php7.1-fpm-alpine` (*php7.1/fpm-alpine/Dockerfile*)
- `cli-1.1.0`, `cli-1.1`, `cli-1`, `cli`, `cli-1.1.0-php5.6`, `cli-1.1-php5.6`, `cli-1-php5.6`, `cli-php5.6` (*php5.6/cli/Dockerfile*)
- `cli-1.1.0-php7.0`, `cli-1.1-php7.0`, `cli-1-php7.0`, `cli-php7.0` (*php7.0/cli/Dockerfile*)
- `cli-1.1.0-php7.1`, `cli-1.1-php7.1`, `cli-1-php7.1`, `cli-php7.1` (*php7.1/cli/Dockerfile*)

Liite 3. Projektin palvelininfrastukstuuri



Liite 4. PHP-tuen salliminen Apache web-palvelimella

```
<FilesMatch ".+\.ph(p[3457]?|t|tml)$">
    SetHandler application/x-httpd-php
</FilesMatch>
<FilesMatch ".+\.phps$">
    SetHandler application/x-httpd-php-source
    # Deny access to raw php sources by default
    # To re-enable it's recommended to enable access to the files
    # only in specific virtual host or directory
    Require all denied
</FilesMatch>
# Deny access to files without filename (e.g. '.php')
<FilesMatch "^\.ph(p[3457]?|t|tml|ps)$">
    Require all denied
</FilesMatch>

# Running PHP scripts in user directories is disabled by default
#
# To re-enable PHP in user directories comment the following lines
# (from <IfModule ...> to </IfModule>.) Do NOT set it to On as it
# prevents .htaccess files from disabling it.
#<IfModule mod_userdir.c>
#    <Directory /home/*/public_html>
#        php_admin_flag engine Off
#    </Directory>
#</IfModule>
```