LAMK
Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

DOCKERIN HYÖDYNTÄMINEN OHJELMISTOKEHITYKSESSÄ

LAHDEN AMMATTIKORKEAKOULU
Tradenomi (AMK)
Tietojenkäsittely
Syksy 2019
Ville Aunola

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Aunola, Ville	Opinnäytetyö, AMK	Syksy 2019
	Sivumäärä	
	36	

Työn nimi

Dockerin hyödyntäminen ohjelmistokehityksessä

Tutkinto

Tradenomi (AMK)

Tiivistelmä

Opinnäytetyössä käsiteltiin virtualisoituja kehitysympäristöjä ja vertailtiin, kuinka konttiteknologiaa hyödyntävä Docker eroaa perinteisistä virtuaalikoneista. Opinnäytetyön tavoitteena oli tutkia, miten Docker vaikuttaa kehitysympäristön tehokkuuteen verrattuna perinteiseen virtualisointiin.

Tutkimus oli luonteeltaan kvalitatiivinen. Teoriaosassa esiteltiin käsitteitä liittyen kehitysympäristön luontiin, Dockeriin sekä virtuaalikoneisiin. Tehokkuuden mittaamisesta kerättiin aineistoa, joka perustui vastaaviin tutkimuksiin Dockerin tehokkuudesta.

Työn tutkimusosuus toteutettiin suunnittelutieteellisen tutkimuskehyksen avulla. Tutkimuksen lähtökohtana oli selvittää Dockerin käyttöön liittyvien hyötyväitteiden, kuten helppo- ja nopeakäyttöisyyden, todenperäisyys. Tutkimuksessa suunniteltiin artefakti, joka sisälsi vaiheet hyötyväitteiden testaamiseen ja jonka avulla pystyttiin vertailemaan kehitysympäristön tehokkuutta.

Artefaktin suunnittelussa hyödynnettiin aikaisemmissa tutkimuksissa käytettyjä mittausmenetelmiä ja tuloksia, joita verrattiin myös testistä saatuihin tuloksiin testin luotettavuuden selvittämiseksi. Suoritetulla testillä saatiin mitattua tuloksia, joilla pystyttiin analysoimaan, miten Docker vaikuttaa kehitysympäristön tehokkuuteen.

Dockerin vaikutus kehitysympäristön tehokkuuteen koostuu sovellusten siirtämisen ja muokkaamisen helppoudesta sekä suorituskyvystä, joka on havaittavissa erityisesti sovellusten nopeina käynnistymisaikoina.

Asiasanat

Docker, virtualisointi, kehitysympäristö

Abstract

Author(s)	Type of publication	Published
Aunola, Ville	Bachelor's thesis	Autumn 2019
	Number of pages	
	36	

Title of publication

Benefits of Docker in software development

Name of Degree

Bachelor of Business Administration

Abstract

The thesis explores virtualized development environments and compares how Docker and its container technology differs from traditional virtual machines. The purpose of the thesis was to examine how Docker affects the effectiveness of the development environment compared to traditional virtualization.

The theoretical part of the thesis introduces concepts that are related to creating a virtualized development environment and metrics that can be used to measuring the effectiveness of the development environment.

The study was conducted using design science research method. The goal was to study the claimed advantages of Docker such as ease of use. As a result of the study, an artifact was created. The artifact contained steps to test each claim and produced values that were used to analyze the effectiveness of the development environment.

The study was a qualitative research where the results and measuring metrics of previous studies were used to guide the designing of the artifact and evaluate its validity. The artifact provided values that were used to analyze how docker affects the effectiveness of development environment.

The study results show that Docker adds effectiveness in the development environment in areas of portability, maintainability and performance which is noticeable in fast system start up time.

Keywords

Docker, virtualization, development environment

SISÄLLYS

LYHENTEET JA TERMIT

Apache	Apache HTTP Server on avoimeen lähdekoodiin perustuva HTTP-palvelinohjelma
Benchmark	Suorituskykytesti, jolla kerätään vertailtavia tuloksia
Docker	Virtualisointialusta sovelluskonttien luontiin ja ylläpitoon
Docker-kuva	Malli, jonka pohjalta sovelluskontti rakennetaan
Dockerfile	Tiedosto, jolla määritellään Docker-kuvan rakenne
GitHub	Verkkopalvelu, joka tarjoaa versionhallintaohjelmiston ohjelmistokehitysprojekteille
MySQL	Tietokantaohjelmisto, jota käytetään erityisesti web-palveluiden tietokantana
PHP	Ohjelmointikieli, jota käytetään erityisesti web-palvelinympäristöissä
VirtualBox	Virtualisointialusta virtuaalikoneiden luomiseen
Vagrant	Ohjelmisto virtuaalikoneiden ympäristön luontiin ja hallintaan
WordPress	Verkkosivujen teossa käytettävä sisällönhallintaohjelmisto

1 JOHDANTO

Ohjelmistokehityksessä muutoksiin pitää pystyä reagoimaan nopeasti. Kehitysympäristön pitäisi olla siis helppo- ja nopeakäyttöinen, jotta ohjelmointikin olisi helppoa ja nopeaa. Sovelluksen muokkaaminen ja päivittäminen tuotantoympäristöön pitäisi olla myös helppoa ja vaivatonta. Näin ei kuitenkaan aina ole, varsinkaan jos tuotantoympäristö ja kehitysympäristö eivät vastaa toisiaan. Viime vuosina konttiteknologian suosio on kasvanut ja Docker on tullut kehittäjien tietoisuuteen. Konttiteknologialla sovellus ja sen suorittamiseen tarvittavat tiedostot pakataan kontteihin, jotka toimivat kaikissa konttiteknologiaa tukevissa ympäristöissä.

Tässä opinnäytetyössä tarkastellaan tarkemmin konttiteknologiaa ja sitä, miten se eroaa virtualisoinnista. Tarkastelu tapahtuu Docker-ohjelmistolla, jonka avulla suoritetaan normaaleja ohjelmistokehityksen vaiheita kuvitteellisessa yritysympäristössä. Huomio kiinnittyy erityisesti siihen, kuinka hyvin Dockerin kotisivuilla olevat väitteet sovelluskehityksen merkittävästä nopeutumisesta, sovellusten siirrettävyydestä sekä hallinnan helpottumisesta pitävät paikkansa.

1.1 Tausta

Tekijä tutustui Dockeriin ensimmäistä kertaa työharjoittelujaksolla, jossa sitä käytettiin muun muassa verkkosivujen kehittämisessä, ja huomasi, kuinka helppoa kehitysympäristön pystyttäminen Dockerin avulla oli. Docker vaikutti hyödylliseltä ohjelmalta, joten siihen perehtyminen opinnäytetyön muodossa herätti kiinnostusta.

Toimiva kehitysympäristö on tärkeä osa ohjelmistotuotantoa, joten työkaluja sen tehostamiseen ja ohjelmoijien työn helpottamiseen on hyvä tutkia ja analysoida. Dockerin ja sen hyödyntämä konttiteknologia vaikuttavat tuovan paljon hyötyjä kehitysympäristöön, joten on hyvä tutkia tarkemmin, millaisia.

Tämä työ tarjoaa tietoa aloitteleville ohjelmoijille, joilla ei ole vielä kokemusta virtualisoinnin käytöstä, sekä yrityksille, jotka ovat pohtimassa Dockerin käyttöön siirtymistä, jotta he voivat analysoida, onko Docker soveltuva heidän kehitysympäristöönsä ja projektiensa tarpeisiin.

1.2 Tavoite

Tämän opinnäytetyön tavoitteena on tutkia hyötyjä, joita Docker tuo ohjelmistokehitykseen. Tutkimuksessa käydään läpi Dockerin ja virtuaalikoneiden käyttöä kehitysympäristössä ja testataan, miten Docker vaikuttaa kehitysympäristön tehokkuuteen. Tutkimuksella

kerätään tietoa Dockerin hyödyistä ja mahdollisista haitoista verrattuna virtuaalikoneeseen, samalla perehdytään tarkemmin Dockeriin ja sen käyttämään teknologiaan. Tutkimuksessa käydään läpi Dockerin kotisivuilta sekä muista lähteistä löytyneitä väitteitä Dockerin tuomista hyödyistä ja todistetaan niiden paikkansapitävyys. Tutkittavat väitteet ovat

- Tuottavuus paranee, sillä projektien aloitus ja lähdekoodin ohjelmointi on helppoa ja nopeaa.
- Palveluiden toimittamisen nopeus kolminkertaistuu.
- Muutosten teko ja ongelmien ratkaiseminen nopeutuu. (Docker Inc. 2019a.)
- Docker käyttää resursseja tehokkaasti, ja samalla tietokoneella voi pyörittää useita kontteja samanaikaisesti (Rad, Bhatti & Ahmadi 2017, 5).

Opinnäytetyö vastaa tutkimuskysymyksiin ”Miten Docker vaikuttaa kehitysympäristön tehokkuuteen verrattuna perinteiseen virtualisointiin?” sekä ”Miten kehitysympäristön tehokkuutta ja sen muutoksia voidaan mitata?”. Kehitysympäristöllä viitataan tässä työssä ohjelmistokokonaisuuteen, johon kuuluvat paikallisesti asennetut lähdekoodieditori, GitHub-versionhallinta sekä virtualisoitu Apache, MySQL, PHP -ympäristö. Tehokkuudella tarkoitetaan sitä, kuinka nopeasti käyttäjä pystyy suorittamaan tehtäviä kehitysympäristössä.

Tutkimus on kvalitatiivinen eli laadullinen tutkimus. Tutkimuksessa kerätään numeerista dataa, jota käytetään apuna kehitysympäristön tehokkuuden analysoimisessa ja pohdittaessa Dockerin vaikutusta kehitysympäristön tehokkuuteen. Mittarit kerättävälle datalle määritellään testin suunnitteluvaiheessa, ja ne perustuvat aineistoon, jota kerätään Dockerin tehokkuutta käsittelevistä tutkimuksista.

1.3 Tutkimusmenetelmä

Opinnäytetyössä hyödynnetään suunnittelutieteellistä tutkimuskehystä. Se on ongelmanratkaisumalli, jota käytetään yleensä tietojärjestelmien suunnittelussa ja arvioimisessa. Suunnittelutieteellisen tutkimuksen tavoitteena on kehittää artefakti, jolla tarkoitetaan suunnittelun ja arvioinnin avulla kehitettyä käsitteistöä, mallia, menetelmää tai toteutusta. Artefaktin pitää olla innovatiivinen eli ratkaista ongelma, johon ei vielä ole ratkaisua, tai ratkaista olemassa oleva ongelma tehokkaammin. Artefaktin tavoitteena on tuottaa hyötyä ja uutta tietoa ratkaistavaan ongelmaan liittyen. (Hevner, March, Park & Ram 2004, 3–5.)

Suunnittelutieteellinen tutkimus koostuu kahdesta prosessista: rakentamisesta ja arvioimisesta. Rakentamisella tarkoitetaan artefaktin rakentamista tietoperustan avulla. Tietoperustaa edellytetään suunnittelutieteellisen tutkimuksen suorittamiseen. Se koostuu muun muassa teorioista, käsitteistä, malleista, menetelmistä ja metodologiasta. Ympäristöön

kuuluvat ihmiset, organisaatio sekä sen käyttämä teknologia. Ympäristö määrittelee tutkimukselle tavoitteet, joihin tutkimus pyrkii vastaamaan mahdollisimman hyvin. Arvioinnilla tarkoitetaan artefaktin arvioimista tietoperustan metodologian suuntaviivojen mukaisesti. Suunnittelutieteellinen tutkimus on luonteeltaan iteratiivista. Artefaktin arvioinnista saatuja tuloksia hyödynnetään artefaktin rakentamisessa. Tarpeeksi monen iterointikierroksen jälkeen artefakti täyttää kaikki ongelmalle määritetyt vaatimukset, jolloin se on valmis. Arvioinnista saatavat tulokset lisätään tietoperustaan seuraavaa tutkimusta varten. (Hevner ym. 2004, 5–8.)

Onnistuneen suunnittelutieteelliseen tutkimuksen saavuttamiseksi on luotu seitsemän kohdan ohjeistus, jota seuraamalla saavutetaan hyödyllinen tutkimus. Ohjeistukset on listattu taulukossa 1. (Hevner ym. 2004, 10.)

Taulukko 1. Suunnittelutieteellisen tutkimuksen ohjeistukset (Hevner ym. 2004, 10)

Ohjeistuksen numero	Kuvaus
1. Artefaktin suunnittelu	Suunnittelutieteellisen tutkimuksen on tuotettava hyödynnettävissä oleva artefakti. Artefakti on joko käsitteistö, malli, menetelmä tai toteutus.
2. Ongelman merkittävyys	Suunnittelutieteellisen tutkimuksen tavoitteena on luoda teknologiapohjaisia ratkaisuja tärkeisiin ja ajankohtaisiin liiketoiminnallisiin ongelmiin.
3. Arvioinnin suunnittelu	Artefaktin hyödyn, laadun ja tehokkuuden täsmällisyys täytyy havainnollistaa hyvin toteutetun arviointimenetelmän avulla.
4. Tutkimuksen hyöty	Suunnittelutieteellisen tutkimuksen on tuotettava ongelma-alueeseen selkeää ja todennettavissa olevaa hyötyä.
5. Tutkimuksen täsmällisyys	Suunnittelutieteellinen tutkimus perustuu täsmällisten menetelmien käyttöön artefaktin rakentamis- ja arviointivaiheissa.

6. Etsintäprosessin suunnittelu	Tehokkain artefakti etsitään vertailemalla käytettävissä olevia keinoja noudattaen kaikkia ympäristön lakeja.
7. Tutkimuksen raportointi	Suunnittelutieteellisen tutkimuksen tulokset täytyy välittää teknologia- ja johtoryhmille.

Suunnittelutieteellisen tutkimuksen tuottamista, esittämistä ja arvioimista helpottaakseen Peffers ja muut (2008) määrittelivät suunnittelutieteelliselle tutkimukselle kuuden kohdan aktiviteettilistauksen, joka tarjoaa mallirakenteen tutkimuksen toteuttamiseen. Aktiviteetit ovat listattuna taulukossa 2. Taulukon ensimmäinen sarake sisältää aktiviteetin, toinen sarake määrittelee ohjeet aktiviteetin suorittamiseen ja kolmas sarake liittää tietoperustan aktiviteettiin. (Geerts 2011, 145.)

Taulukko 2. Suunnittelutieteellisen tutkimuksen aktiviteetit (Geerts 2011, 145)

Aktiviteetti	Kuvaus	Tietoperusta
Ongelman tunnistaminen ja motivaatio	Määrittele tutkimusongelma ja perustele ratkaisu.	Ongelman merkityksellisyyden ja nykyisen ratkaisun heikkouksien ymmärtäminen
Ratkaisun tavoitteiden määrittely	Määrittele kriteerit, jotka tutkimusongelman ratkaisun pitää täyttää.	Tieto mikä on mahdollista toteuttaa. Tieto menetelmistä, teknologioista ja teorioista, jotka auttavat tavoitteiden määrittelyssä.
Suunnittelu ja toteutus	Luo artefakti, joka sisältää tutkimuksessa havaittuja hyötyjä.	Sovella menetelmiä, teknologioita ja teorioita luodaksesi artefakti, joka ratkaisee ongelman.

Toteutuksen demonstrointi	Todista, että artefakti toimii ratkaisemalla ongelman esiintymiä.	Tieto käyttää artefaktia ongelman ratkaisemiseen.
Tulosten arviointi	Tarkkaile ja mittaa, kuinka hyvin artefakti tukee tavoitteita, vertailemalla sitä saatuihin tuloksiin.	Tieto mittareista ja arviointitekniikoista
Kommunikointi	Raportoi tutkimuksen lopputulokset tutkijoille sekä muille asiaankuuluville ryhmille.	Tieto tutkimusyhteisön kulttuurista ja toimintatavoista

1.4 Tutkimusmenetelmän soveltaminen

Tämän opinnäytetyön rakenne vastaa luvussa 1.3 esitetyn kuuden kohdan aktiviteettejä, jotka ovat listattu taulukossa 2. Aktiviteetit käydään läpi seuraavasti:

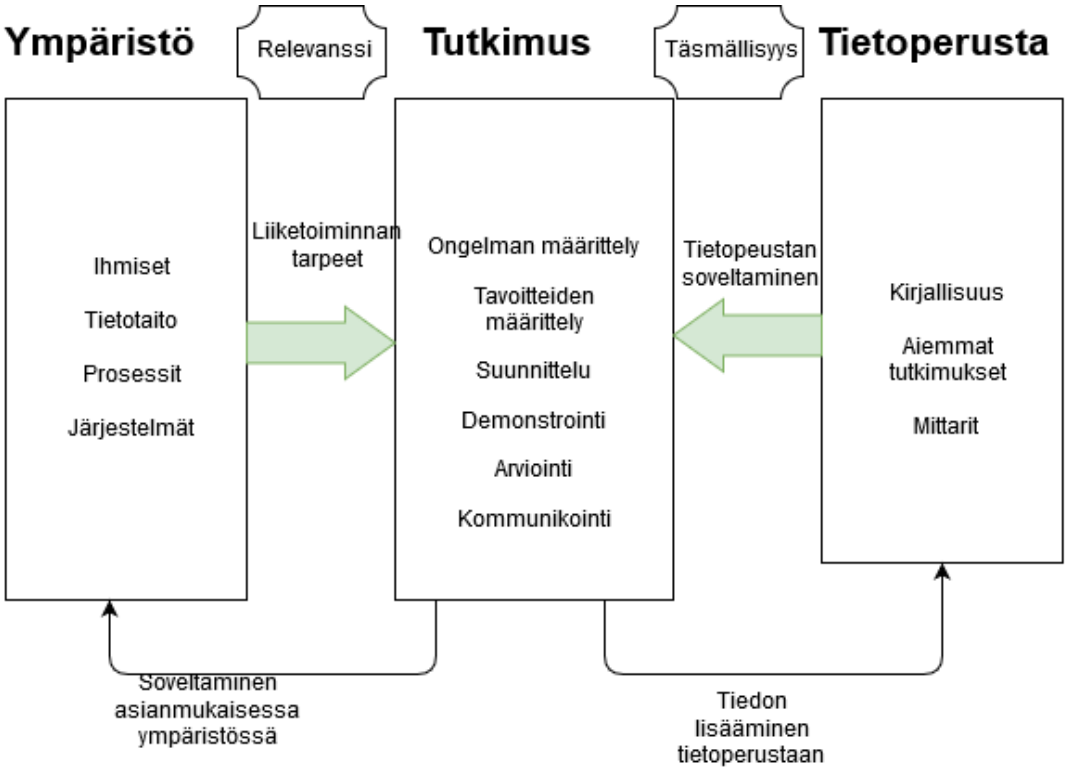
- Tutkimuksen ongelma ja ratkaisun määrittely käydään läpi luvussa 4.2.
- Kriteerit, jotka tutkimusongelman tulee täyttää, käydään läpi luvussa 4.3.
- Ratkaisun suunnittelu tapahtuu luvussa 4.4.
- Artefaktin toiminta testataan luvussa 4.5.
- Artefaktia arvioidaan luvussa 4.7 pohjautuen luvussa 4.6 esitettyihin tuloksiin ja luvussa 4.3 määritettyihin kriteereihin.

Tutkimusongelmana on selvittää, miten kehitysympäristöön tehokkuutta ja sen muutoksia voidaan mitata. Tietoperustaan kerätään teoriaa liittyen kehitysympäristöjen virtualisointiin, virtuaalikoneiden ja konttiteknologian hyviin ja huonoihin puoliin, Dockeriin ja virtuaalikoneisiin liittyviin käsitteisiin sekä tehokkuuden mittaamiseen.

Tutkimuksen tavoitteena on tuottaa vertailun avulla pätevää tietoa Dockerin tehokkuudesta virtuaalikoneisiin verrattuna. Tavoitteeseen pääsemistä varten suunnitellaan testausprosessi, jossa vertaillaan Dockeria virtuaalikoneisiin. Testin rakenne suunnitellaan tietoperustaan kertyneen teorian pohjalta liittyen Dockerin hyötyihin.

Tutkimuksessa tuotettava artefakti on testi itsessään, josta saatavilla tuloksilla pystytään ratkaisemaan tutkimusongelma ”Kuinka Docker vaikuttaa kehitysympäristön tehokkuuteen verrattuna perinteiseen virtualisointiin?”. Artefaktia eli testiä arvioidaan lopuksi perustuen siihen, kuinka hyvin sillä saatiin kerättyä tietoa tehokkuudesta.

Tutkimuksen menetelmäkehys rakentuu ympäristöstä, tutkimuksesta sekä tietoperustasta, joka on esitelty kuvassa 1.



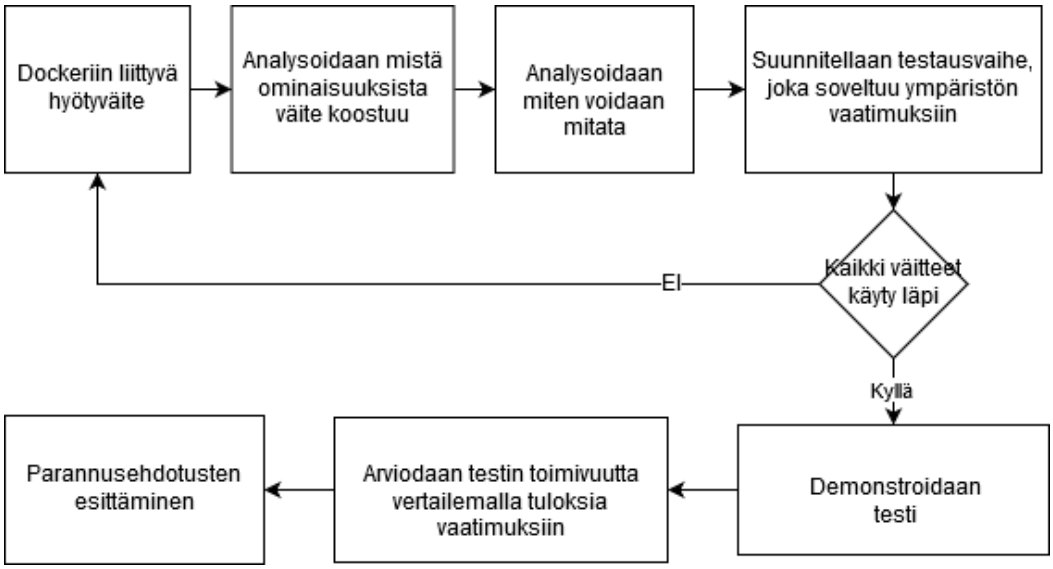
Kuva 1. Tietojärjestelmätutkimuksen menetelmäkehys tässä opinnäytetyössä (Hevner ym. 2004, 7)

Ympäristöön kuuluva yritys on juuri liiketoimintansa aloittanut pienyritys, joka tuottaa verkkosivustoja asiakkaille. Työntekijöillä on pohjatietoa virtuaalikoneiden käytöstä, mutta he eivät ole vielä tutustuneet Dockeriin tarkemmin. Työntekijät haluavat saada yhtenäiset kehitysympäristöt, jotka toimivat niin työpaikan tietokoneissa kuin myös heidän henkilökohtaisissa kotitietokoneissaan. Ympäristö määrittelee tutkimukselle tarpeen, joka tässä tapauksessa on luoda testi, jonka avulla yrityksen työntekijät voivat helposti suorittaa kehitysympäristöjen vertailua ja analysointia.

Tietoperusta rakentuu virtualisointiin liittyvistä käsitteistä, kuten virtuaalikoneista ja konttiteknologiasta, joiden avulla voidaan suunnitella testattavia asioita. Tehokkuuden mittaamisesta kerätään tietoa, jotta testissä voidaan käyttää muissa tutkimuksissa hyväksyttyjä mittareita ja tulokset ovat luotettavia.

Itse tutkimus koostuu artefaktin suunnittelusta perustuen ympäristön määrittämiin vaatimuksiin ja tietoperustaan kerättyjen mittarien toimintaan. Artefaktin toiminta demonstroidaan, minkä jälkeen arvioidaan, kuinka hyvin se vastaa ympäristön asettamia tavoitteita ja tietoperustaan kerättyjen mittausmenetelmien tuloksia. Tämän jälkeen pohditaan, miten artefaktia voitaisiin vielä parantaa.

Artefaktin suunnittelu- ja arviointiprosessi on kuvattu kuvassa 2.



Kuva 2. Artefaktin suunnittelu- ja arviointiprosessi

2 OHJELMISTOKEHITYS

2.1 Kehitysympäristö

Ohjelmistotuotanto jakaantuu yleensä ohjelmiston määrittelyyn, suunnitteluun, ohjelmointiin, testaukseen, integrointiin ja ylläpitoon. Ohjelmistotuotanto tapahtuu yleensä projektityönä, jolloin prosessiin osallistuu useita henkilöitä. Ohjelmointivaiheessa ohjelmoijien käytössä olevat ohjelmistot muodostavat kehitysympäristön. Tarvittavat ohjelmat määrittyvät projektin tarpeiden mukaisesti. Kehitysympäristön on syytä olla nopea ja helppokäyttöinen, jotta ohjelmoijat pystyvät jakamaan ja muokkaamaan lähdekoodia helposti eikä ohjelmointivaiheesta aiheudu ylimääräistä häiriöitä. (Schmidt 2013, 34–35.)

Yksinkertaisimmillaan kehitysympäristö voi olla lähdekoodieditori, versionhallinta ja ohjelmointikielen kääntäjä, joka kääntää koodin suoritettavaan muotoon. Esimerkiksi verkkosivustoa kehittäessä ympäristö voi olla Visual Studio Code -lähdekoodieditori, GitHub-versionhallinta, johon sivustoon tehtävät muokkaukset tallennetaan, sekä Apache-serveri MySQL ja PHP-tuella, jotka muuttavat verkkosivun lähdekoodin selaimen ymmärtämään muotoon. (Stephens 2015, 146–148.)

2.2 Virtualisointi

Virtualisoinnilla tarkoitetaan sitä, että jokin fyysinen komponentti muutetaan virtuaaliseksi. Esimerkiksi palvelinympäristössä fyysisessä laitteessa suoritetaan yhden käyttöjärjestelmän sijasta useita virtuaalikoneita, joista jokainen pyörittää omaa käyttöjärjestelmäänsä. (Portnoy 2016, 11.)

Virtualisointi sai alkunsa jo 1960-luvulla. Gerald J. Popek ja Robert P. Goldberg julkaisivat vuonna 1974 artikkelin ”Formal Requirements for Virtualizable Third Generation Architectures”, jossa määriteltiin virtuaalikoneiden vaatimukset, jotka ovat voimassa vielä tänäkin päivänä. Virtuaalikoneen on pystyttävä virtualisoimaan kaikki laitteiston resurssit. Hypervisor on ohjelmisto, joka tarjoaa ympäristön, jossa virtuaalikone voi toimia ja se hallitsee virtuaalikoneiden ja laitteiston välisiä vuorovaikutuksia. Sen tulee kyetä eristämään virtuaalikone niin että vain se hallitsee sitä. Sen pitää myös kyetä virtualisoimaan laitteisto niin että se on täysin identtinen alkuperäisen laitteiston kanssa ja suorituskyvyn on vastattava fyysistä vastiketta. (Portnoy 2016, 1–3.)

Hypervisorit jaetaan tyypin 1 ja tyypin 2 hypervisoreihin. Tyypin 1 hypervisor toimii suoraan laitteiston päällä ilman, että sen välissä on käyttöjärjestelmää, tehden siitä tehokkaamman ja turvallisemman kuin tyypin 2 hypervisor. Tyypin 2 hypervisorit ovat puolestaan ohjelmistoja, jotka toimivat käyttöjärjestelmän päällä. Ne käyttävät samoja resursseja

kuin isäntäkone, joten ne ovat raskaampia. Niiden asentaminen on kuitenkin helppoa, sillä laitteiston resurssit ovat määritelty valmiiksi isäntäkoneelle. Ne eivät ole kuitenkaan yhtä turvallisia kuin tyypin 1 hypervisorit, sillä kaikki isäntäkäyttöjärjestelmään vaikuttavat virhetilat vaikuttavat myös virtuaalikoneisiin. (Portnoy 2016, 23–26.)

Virtualisointia käytetään eniten servereiden yhteydessä, sillä se mahdollistaa useamman virtuaalikoneen suorittamisen samanaikaisesti yhdellä serverillä, jonka ansiosta sähkönkulutusta, jäähdytyskuluja sekä uusien konesalien käyttöönottoa voidaan vähentää. Servereiden virtualisoinnin ohella virtualisoinnilla on monia muitakin käyttötarkoituksia, yhtenä niistä on kehitysympäristön virtualisointi, joka mahdollistaa yhteneväisen ympäristön luonnin kaikkien kehittäjien kesken ja helpottaa sovellusten kehitystyötä ja testausta. (Portnoy 2016, 9–19.)

Yksinkertaisin ratkaisu kehitysympäristön virtualisoimiseen on luoda virtuaalikone, johon asennetaan haluttu ympäristö, käyttöjärjestelmä ja ohjelmat. Sen jälkeen se voidaan kopioida kaikille ohjelmoijille. Virtuaalikoneet sisältävät oman käyttöjärjestelmänsä, kaikilla käyttöjärjestelmään kuuluvilla kirjastoilla ja tiedostoilla. Jokainen virtuaalikone käyttää samoja resursseja kuin isäntäkone, joten niiden pyörittäminen on raskasta. (Portnoy 2016, 25–26.)

Dockerin hyödyntämää konttiteknologiaa on kuvailtu sovelluskehityksen ja virtualisoinnin mullistajaksi, sillä kontit toimivat suoraan käyttöjärjestelmän päällä eikä ne tarvitse erillisiä käyttöjärjestelmiä, mikä tekee niistä nopeita ja kevyitä suorittaa (Kotilainen 2017).

2.2.1 Virtuaalikoneet

Perinteisellä virtualisoinnilla tarkoitetaan virtuaalikoneen luomista. Virtuaalikoneelle on määritelty käyttöjärjestelmä ja resurssit, joita se voi käyttää. Virtuaalikoneet voivat käyttää eri käyttöjärjestelmiä ja pyöriä samanaikaisesti samalla serverillä. Virtuaalikoneet rakentuvat virtuaalisesta levykuvasta sekä määritystiedostosta, jolla määritellään virtuaalikoneen käytössä olevat resurssit, kuten prosessori, muisti ja tallennusasemat. Virtuaalikone toimii identtisesti fyysiseen vastikkeeseen verrattuna. (Portnoy 2016, 37–39.)

Virtuaalikoneiden avulla voidaan luoda kaikille kehittäjille yhtenäinen kehitysympäristö, joka sisältää kaikki tarvittavat ohjelmat projektiin liittyen. Ympäristö voidaan varmuuskopioida ottamalla siitä tilannekuva, joka tallentaa virtuaalikoneen sen hetkisen tilan. Tilannekuvat ovat palautuspisteitä, joiden avulla virtuaalikone voidaan palauttaa täsmälleen siihen tilaan missä se oli kuvanottohetkellä. Virtuaalikoneen voi myös siirtää toiseen käyttöjärjestelmään, jolloin projektia voi työstää niin kotitietokoneella kuin työtietokoneellakin. Siirtäminen tapahtuu tuonti- ja vientityökalujen avulla. Tällöin virtuaalikoneesta luodaan

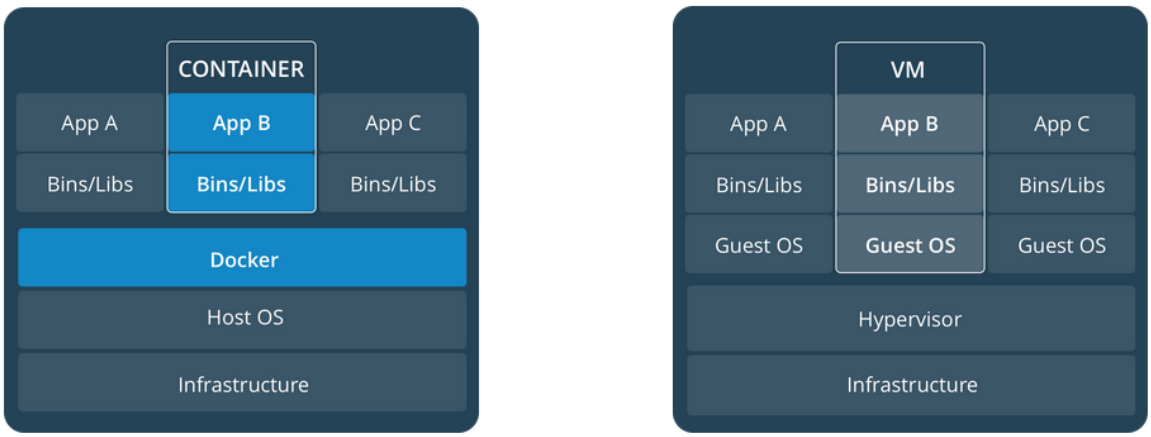
OVF-tiedosto (Open Virtualization Format), joka voidaan siirtää virtuaalikoneita luovaan ohjelmaan, kuten Oraclen VirtualBoxiin, joka luo virtuaalikoneen OVF-tiedoston avulla. (Oracle VM VirtualBox 2019, 17–21.)

Oraclen VirtualBox on tyypin 2 hypervisor, jonka avulla voidaan luoda virtuaalikoneita isäntäkäyttöjärjestelmän päälle. Lähtökohtaisesti tyypin 2 hypervisorin avulla suoritetut sovellukset ovat hitaampia, sillä ne sisältävät omat käyttöjärjestelmänsä sekä niiden suorittamiseen vaadittavat kirjastot ja tiedostot. Niiden pyörittäminen on raskasta, sillä ne käyttävät samoja resursseja kuin isäntäkäyttöjärjestelmä. (Portnoy 2016, 23–26.)

Virtuaalikoneiden luontia ja hallintaa voidaan helpottaa Vagrant-ohjelmiston avulla, joka on suunniteltu hallinnoimaan virtuaalikoneiden ympäristöjä. Vagrantilla voidaan automatisoida virtuaalikoneiden luomisprosessi, jolloin niitä ei tarvitse asentaa manuaalisesti. Virtuaalikoneen asetukset määritellään vagrantfilen avulla, joka suoritetaan komentorivillä komennolla “vagrant up”. Vagrant välittää tiedot VirtualBoxille, joka vastaa virtuaalikoneen luomisesta. Vagrantfile on helppo jakaa ja sen avulla voidaan luoda kehittäjille identtiset ympäristöt. (Peacock 2013, 15–24.)

2.2.2 Konttiteknologia ja Docker

Viime vuosina konttiteknologia on kasvattanut suosiotaan. Se hyödyntää käyttöjärjestelmätason virtualisointia, joka tehdään nimensä mukaisesti käyttöjärjestelmätasolla. Kontit pyörivät käyttöjärjestelmän ytimen päällä, ilman virtuaalikoneiden luomisen tarvetta, joten ne vaativat virtuaalikoneita vähemmän resursseja isäntäkoneelta. Virtuaalikoneilla suoritetuille sovelluksille pitää kaikille määrittää omat käyttöjärjestelmät, jotka käyttävät runsaasti isäntäkäyttöjärjestelmän resursseja ja hidastavat sitä. (Krochmalski 2016, 8.)



Kuva 3. Konttiteknologian ja virtuaalikoneiden ero (Docker Inc 2019b)

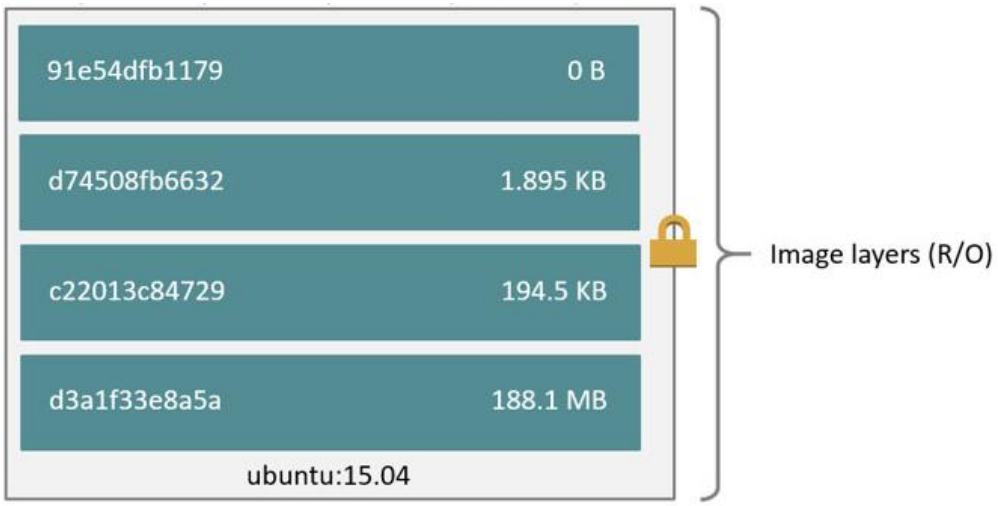
Kontit ovat eristettyjä isäntäkäyttöjärjestelmästä sekä muista konteista, ja niillä on omat tiedostorakenteensa sekä ympäristömuuttujat. Kontit ovat myös hyvin kevyitä ja niiden käynnistäminen on nopeaa. Konttiin on pakattu kaikki sovelluksen ajamiseen tarvittava tieto ja niitä voidaan siirrellä ja hallita helposti. (Krochmalski 2016, 8.)

Docker on avoimeen lähdekoodiin perustuva sovellus, jonka avulla voidaan kehittää, jakaa ja ajaa sovelluksia. Se julkaistiin vuonna 2013 ja se on ollut konttiteknologian yksi tärkeimmistä kehittäjistä. Docker on luonut standardit konteille sekä niiden hallintaan liittyviin työkaluihin sekä tehnyt konttien käytöstä suosittua. (Krochmalski 2016, 6–7.)

Dockerin perusideana on pakata sovellus ja kaikki sen suorittamiseen tarvittavat tiedot yhteen pakettiin, jota kutsutaan kontiksi. Kontin avulla ajettu sovellus toimii aina samanlailla riippumatta ympäristöstä, jossa sitä suoritetaan. Dockerin avulla voidaan kehittää sovelluksia ilman, että omalle koneelle tarvitsee asentaa mitään projektissa tarvittavia ympäristöjä, kuten esimerkiksi NodeJs-ympäristöä. (Krochmalski 2016, 7.)

Kontit rakennetaan Docker-kuvien avulla, jotka ovat kerroksista koostuvia kirjoitussuojattuja malleja. Ne sisältävät kaiken tiedon, jota sovelluksen suorittamisessa tarvitaan. Jokaisen kuvan rakentaminen aloitetaan pohjakuvasta, kuten esimerkiksi Ubuntun tai PHP:n peruskuvasta. Pohjakuvan päälle lisätään uusia kerroksia Dockerfilen ohjeistuksen mukaisesti. Ohjeistus voi sisältää komentoja, tiedostojen tai kansioiden lisäämistä tai ympäristömuuttujan luomista. Näistä ohjeistuksista sekä pohjakuvasta muodostuu valmis Docker-kuva. (Krochmalski 2016, 45.)

Dockerfile on tekstitiedosto, jolla määritellään ohjeet, miten Docker-kuva rakennetaan. Kun Docker-kuva rakennetaan, jokainen Dockerfilen rivi lisätään siihen uutena kirjoitussuojattuna kerroksena, joista muodostuu lopuksi valmis Docker-kuva. Dockerfile alkaa FROM ohjeistuksella, jolla määritellään pohjakuva, jonka päälle Docker-kuva rakennetaan. (Docker Inc 2019c.)



Kuva 4. Docker-kuva rakentuu kerroksista, jotka ovat pinottu päällekkäin (Docker Inc 2019c)

Docker käyttää Docker-kuvien rakentamisessa union filesystem -tiedostojärjestelmää, joka vastaa kerrosten yhdistämisestä yhtenäiseksi kuvaksi. Se mahdollistaa myös Dockerin nopean kuvien rakentamisen. Jos Dockerfileen tekee muutoksia, vain muutosta vastaava kerros päivitetään, jolloin koko kuvaa ei tarvitse rakentaa uudelleen. (Docker Inc 2019d.)

Docker compose on työkalu, jolla voidaan määritellä ja suorittaa useamman kontin sovellus. Docker compose -tiedostoon listataan kaikki sovelluksen tarvitsevat palvelut ja niiden väliset riippuvuudet. Docker compose suoritetaan komennolla “docker compose up”. Tämä käynnistää sovelluksen ja kaikki siihen linkitetyt kontit. (Docker Inc 2019e.)

Docker Hub on paikka, johon Docker-kuvia voidaan tallentaa ja mistä ne ladataan. Docker Hub on julkinen jakelualusta, jota kaikki voivat käyttää Docker-kuvien jakamiseen ja lataamiseen. Docker-kuvia voi tallentaa joko julkiseen tai yksityiseen tietovarastoon. (Krochmalski 2016, 103.)

3 TEHOKKUUDEN MITTAAMINEN

Tehokkuudella tarkoitetaan tässä opinnäytetyössä, kuinka nopeasti käyttäjä pystyy suorittamaan tehtäviä kehitysympäristössä.

Ohjelmistojen yhteydessä tehokkuutta mitataan yleensä benchmark-työkalujen avulla, jotka simuloivat ohjelmalle useita samanaikaisia käyttäjiä ja rasittavat palvelua keräten samalla tuloksia palvelun suorituskyvystä.

3.1 Tehokkuuden mittaaminen suorituskyvyn avulla

Suurteholaskennassa palvelun tehokkuutta määritellessä voidaan käyttää muun muassa Linpack, Stream ja RandomAccess benchmark-työkaluja. Linpack testaa prosessorin tehoa ratkaista monimutkaisia lineaarisia yhtälöryhmä, Stream testaa muistin kaistanleveyttä ja RandomAccess testaa muistin nopeutta. Felter, Ferreira, Rajamony ja Rubio tutkivat virtuaalikoneiden ja Dockerin tehokkuutta mittaamalla laskentanopeuksia, muistin käyttöä, tiedonsiirtonopeutta ja tietokannan nopeutta. Heidän saamiensa tulosten mukaan Docker oli tehokkaampi kaikissa testeissä. (Felter, Ferreira, Rajamony & Rubio 2015, 5–15.)

Paikallisella koneella suoritettavien konttien ja virtuaalikoneiden käyttämien resurssien mittaamiseen voidaan käyttää yksinkertaisempia menetelmiä, kuten prosessorin ja muistin suorituskykyä sekä tiedonluku- ja kirjoitusnopeutta. Zheng ja Thain tutkivat kuinka Dockerin käyttöönoton saa toimimaan tehokkaimmin työnkulkujärjestelmissä ja hyödynsivät yksinkertaisia benchmark-työkaluja arvioidakseen yksittäisellä tietokoneella pyöriviä kontteja. Nämä benchmark-työkalut koostuivat Sysbench-työkalusta, jolla voi mitata prosessorin ja muistin suorituskykyä, netperf-työkalusta, jolla voidaan mitata tiedonkulkua ja bonnie++-työkalusta, jolla voidaan mitata kiintolevyn tiedonluku- ja kirjoitusnopeutta. Testien perusteella he vahvistivat olettamuksen, että konttien suorituskyky oli lähes identtinen isäntäkoneen suorituskykyyn verrattuna. (Zheng & Thain 2015, 5.)

Rad, Bhatti ja Ahmadi tutkivat Dockerin ja virtuaalikoneiden suorituskykyä yksinkertaisimmilla mittareilla, kuten keskimääräisenä käynnistymisaikana ja laskentatehona. Käynnistymisajan mittaamista varten he käynnistivät Docker-kuvan 20:llä eri serverillä saadakseen käynnistymisajan keskiarvon. Laskentanopeuksia he mittasivat suorittamalla Pythonin avulla 100 000 ja 200 000 työvaihetta ja mittaamalla suorittamiseen kuluneen ajan. Testi toistettiin 100 kertaa. Näiden testien tulosten pohjalta voitiin todeta, että Docker-ympäristön käynnistymisaika oli lähes kuusi kertaa nopeampi kuin virtuaalikoneella. Laskentanopeuksilla ei ollut hirveästi eroa, mutta Docker suoriutui niistäkin virtuaalikonetta

nopeammin. Lisäksi he testasivat virtuaalikoneen ja konttiteknologian avulla asennetun Docker-ympäristön tietokantakyselyiden suorittamisnopeutta hyödyntäen JMeter-työkalua. Kyselyitä suoritettiin 10 000 ja näiden tulosten pohjalta virtuaalikoneelle asennettu Docker suoriutui testistä huomattavasti nopeammin. (Rad ym. 2017, 5–7.)

Muita hyödyllisiä työkaluja tehokkuuden mittaamiselle ovat mpstat, iostat, docker stat, cAdvisor, Prometheus ja Grafana. Näiden avulla voidaan mitata muun muassa prosessorin käyttöastetta, benchmark-testien suoritusaikoja, tiedonsiirtonopeutta ja kiintolevyn kirjoitusnopeutta. Docker stat on Dockerin komento, joka näyttää muun muassa konttien prosessorin ja muistin käyttöasteen sekä tiedonkulun määrän. Mpstat on puolestaan vastaava komento Linux-järjestelmille, joten sitä voidaan käyttää virtuaalikoneissa. (Casalicchio & Perciballi 2017, 2–4.)

Tehokkuudella voidaan tarkoittaa myös ohjelmiston energiatehokkuutta, joka tarkoittaa suhdelukua ohjelmiston tuottaman työn hyödyn ja sen kuluttaman energian välillä. Tehokkuutta voidaan mitata kokonaisuutena tai erikseen jokaiselle ohjelmiston osalle. Tavoitteena on pienentää ohjelmistojen tai niiden osien energiankulutusta. (Johann, Dick, Naumann & Kern 2012, 2–3.)

3.2 Tehokkuuden mittaaminen ohjelmiston laatuominaisuuksien avulla

Ohjelmiston tehokkuutta voidaan mitata myös sen laadukkuuden kautta. Kun ohjelmiston laadukkuus paranee, myös tehokkuus paranee. Ohjelmiston laatutekijöihin kuuluvat funktionaalisuus, luotettavuus, käytettävyys, suorituskyky, ylläpidettävyys ja siirrettävyys. (Berander, Damm, Eriksson, Gorschek, Henningsson, Jönsson, Kågström, Milicic, Mårtensson, Rönkkö & Tomaszewski 2005, 13–15.)

Funktionaalisuus sisältää yleiset ominaisuudet ohjelmiston toiminnalle, joilla voidaan arvioida, kuinka hyvin ohjelmisto soveltuu sille suunniteltuun tehtävään, kuinka tarkasti ohjelmisto toteuttaa tehtävänsä, kuinka hyvä tietoturva ohjelmistossa on, kuinka hyvin se toimii järjestelmien kanssa, johon se on suunniteltu ja kuinka hyvin se mukautuu vallitseviin lakeihin ja standardeihin (Berander ym. 2005, 15).

Luotettavuus ilmaisee, toimiiko ohjelmisto, kuten sen on tarkoitettu. Luotettavuutta voidaan arvioida ohjelmiston virheherkkyydellä, virheiden sietokyvyllä sekä virheistä toipumiskyvyllä. Nämä voidaan laskea mittaamalla, kuinka usein virheitä tapahtuu ja kuinka kauan kestää, että ne saadaan korjattua. (Berander ym. 2005, 15.)

Käytettävyys ilmaisee, kuinka helposti ohjelmistoa pystyy käyttämään. Sitä voidaan mitata ohjelmiston ymmärrettävyyden, opittavuuden, käytön tehokkuuden sekä miellyttävyyden

avulla. (Berander ym. 2005, 15). Ymmärrettävyyttä voidaan mitata virheprosenttina jakamalla suoritettujen tehtävien lukumäärä, jotka poikkeavat normaalista toimintatavasta, tehtävien kokonaismäärällä. Opittavuutta voidaan laskea jakamalla ensimmäisen kriittisen tehtävän suorittamiseen kulunut aika kokonaisajalla, joka järjestelmä oli käytössä. Käytön tehokkuus voidaan laskea jakamalla suoritettujen tehtävien lukumäärällä kokonaistehtävien lukumäärällä. (Albertao, Xiao, Tian, Lu, Zhang & Liu 2010, 5.)

Suorituskyky ilmaisee, kuinka ohjelmisto toimii suhteessa kulutettuihin resursseihin. Suorituskykyä voidaan mitata vasteaikana tehtävän suorittamiseen tai tehtävän suorittamiseen tarvitulla resurssien käytöllä. Suorituskyvyn mittareita käytiin läpi luvussa 3.1.(Berander ym. 2005, 15.)

Ylläpidettävyys sisältää mittarit, sille kuinka helposti ohjelmistoon pystyy tekemään muutoksia. Sitä voidaan arvioida analysoitavuudella, muokattavuudella, vakautena sekä testattavuutena. (Berander ym. 2005, 15). Ohjelmiston muokattavuutta voidaan mitata ajallisesti, mittaamalla kuinka kauan aikaa kuluu muutosten tekemiseen ja niiden testaamiseen. Lähdekoodin ylläpidettävyyttä voidaan mitata vertailemalla ohjelmiston koodirivien määrää ennen muokkausta ja muokkauksen jälkeen. (Riaz, Mendes & Tempero 2009, 5-9.) Ohjelmiston vakautta voidaan mitata epävakausarvolla. Epävakausarvo on väliltä 0-1, jossa 0 tarkoittaa komponenttia, jota pystyy muokkaamaan ilman, että se vaikuttaa muihin komponentteihin. Arvo 1 tarkoittaa, että komponentti on täysin epävakaa, joka tarkoittaa sitä, että muutokset muissa komponenteissa vaikuttavat myös kyseiseen komponenttiin. Epävakausluku lasketaan jakamalla komponentin riippuvuudet muihin komponentteihin riippuvuuksien kokonaismäärällä. (Albertao ym. 2010, 4.)

Siirrettävyydellä tarkoitetaan sitä, kuinka helposti ohjelmisto kykenee toimimaan uudessa ympäristössä. Sitä voidaan analysoida mukautuvuutena, asennuskelpoisuudella sekä korvattavuutena. (Berander ym. 2005, 15.)

4 TUTKIMUS

4.1 Työprosessi

Tutkimus aloitettiin keräämällä tietoa virtualisointiin sekä tehokkuuden mittaamiseen liittyvistä käsitteistä (luvut 2.2 ja 3). Näiden tietojen pohjalta alettiin toteuttamaan suunnittelutieteellistä tutkimusta luvussa 1.4 kuvatun mallin mukaisesti.

4.2 Ongelman tunnistaminen ja motivaatio

Docker ja konttiteknologia ovat vielä uudehkoja käsitteitä, eikä kaikilla yrityksillä ole vielä tietoa niiden käytöstä. Yritykset saattavat miettiä Dockerin käyttöön siirtymistä, mutta eivät tiedä siitä vielä tarpeeksi ja pohtivat, onko siihen siirtyminen kannattavaa.

Yritysten ongelmana on epätietoisuus siitä, miten Docker lisää kehitysympäristön tehokkuutta. Ratkaisuna tähän ongelmaan on luoda testi, jolla voidaan arvioida Dockerin ja virtuaalikoneiden avulla luotujen kehitysympäristöjen tehokkuutta. Tätä varten pitää ensiksi tutustua virtualisointiin, Dockeriin ja virtuaalikoneisiin sekä selvittää, miten tehokkuutta voidaan mitata.

4.3 Ratkaisun vaatimukset

Suunnitellun testin tulee koostua kohdista, joilla voidaan todistaa Dockeriin liittyvien hyötyjen paikkansapitävyys. Testin tulee olla tarpeeksi yksinkertainen, jotta kohdeyrityksen henkilöstö voi suorittaa sen omilla tietokoneillaan. Testin tuloksia vertailemalla pitää saada pätevää tietoa Dockerin ja virtuaalikoneiden eroista. Pätevä tieto saadaan käyttämällä tietoperustaan kerättyjä tehokkuuden mittareita. Testissä tulisi käydä läpi myös käsitteitä sekä työkaluja, joita voidaan hyödyntää ohjelmiston kehitysprosessin aikana.

Hyötyväitteet, joita testissä käydään läpi, koostuvat Dockerin kotisivuilla listattuihin väitteisiin sekä tutkimukseen Dockerin tehokkuuteen liittyen:

- Tuottavuus paranee, sillä projektien aloitus ja lähdekoodin ohjelmointi on helppoa ja nopeaa.
- Palveluiden toimittamisen nopeus kolminkertaistuu.
- Muutosten teko ja ongelmien ratkaiseminen nopeutuu. (Docker Inc. 2019a.)
- Docker käyttää resursseja tehokkaasti, ja samalla tietokoneella voi pyörittää useita kontteja samanaikaisesti (Rad ym. 2017, 5).

4.4 Ratkaisun suunnittelu

Testin suunnittelussa käydään luvussa 4.3 esitetyt hyötyväitteet läpi pohtien, mistä laatuominaisuuksista ne koostuvat, mitkä ovat niiden tärkeimmät laatuominaisuudet ja millaisella mittarilla niitä kannattaisi arvioida.

Ensimmäisenä voidaan analysoida väitettä ”Tuottavuus paranee, sillä projektien aloitus ja lähdekoodin ohjelmointi on helppoa ja nopeaa”. Tästä väitteestä voidaan selvimpinä ominaisuuksina erottaa käytettävyys, suorituskyky, siirrettävyys sekä luotettavuus.

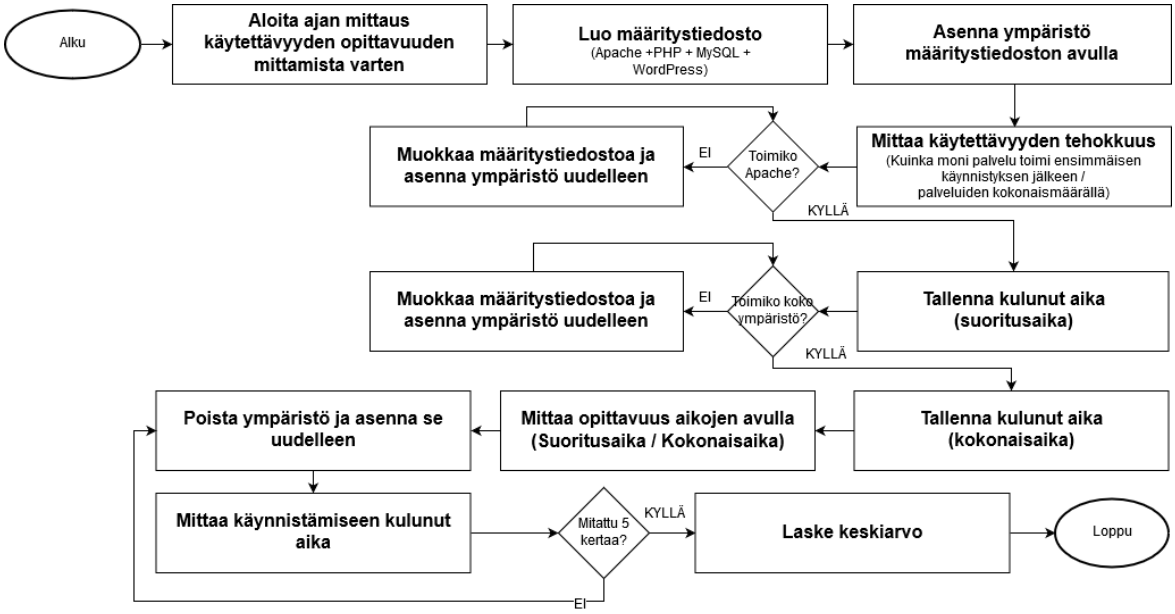
Käytettävyys ilmenee väitteessä projektien aloittamisen ja ohjelmoinnin helppoutena. Suorituskyky ilmenee väitteessä projektien aloittamisen ja ohjelmoinnin nopeutena. Projektin aloittaminen voi olla myös helppoa johtuen komponenttien siirrettävyydestä sekä ohjelmiston luotettavuudesta.

Koska kohdeyrityksen työntekijöillä ei ole vielä tietoa Dockerin käytöstä, voidaan pitää perusteltuna, että testin ensimmäisessä vaiheessa keskitytään vain uuden projektin aloittamiseen ja täten mitataan käytettävyyttä ja suorituskykyä. Käytettävyyttä voitaisiin arvioida muun muassa opittavuuden, tehokkuuden, ymmärrettävyyden sekä miellyttävyyden avulla. Koska kehitysympäristö luodaan määritystiedoston pohjalta, voidaan hyödyllisimpänä käytettävyyden mittarina pitää tehokkuutta, joka voidaan laskea jakamalla suoritetut tehtävät tehtävien kokonaismäärällä. Lisäksi voidaan hyödyntää opittavuutta, jolloin mitataan aika ensimmäisen kriittisen tehtävän suorittamiseen ja jaetaan se järjestelmän kokonaiskäyttöajalla. Näiden mittarien avulla saadaan testattua virtualisointitekniikoiden käytettävyyden helppoutta. Miellyttävyyttä ja ymmärrettävyyttä ei ole tarpeellista analysoida, sillä ne eivät vaikuta siihen, kuinka nopeasti ympäristön saa määriteltyä ja käynnistettyä. Suorituskykyä voidaan mitata projektin aloittamisen nopeutena mittaamalla käynnistymisaika, joka kuluu käynnistyskomennosta siihen, että ympäristö on käynnistetty. Esimerkiksi laitteiston resurssien käyttöä ei ole vielä aiheellista mitata, sillä nopeus näkyy käyttäjälle ensisijaisesti käynnistymiseen kuluvana aikana.

Koska yritys tuottaa verkkosivuja, voidaan sopivana lähtökohtana testille pitää kehitysympäristöä, joka koostuu Apache, PHP ja MySQL -palveluista, joiden avulla voidaan kehittää WordPress-sivustoja. Hyödyllisenä testinä voitaisiin pitää myös verkkokauppa-alustojen asennusta, mutta koska WordPress on yksi yleisimpiä verkkosivujen teossa käytettäviä julkaisujärjestelmiä, voidaan testata, kuinka helposti WordPress-ympäristön saa pystytettyä.

Ensimmäiseksi testin kohdaksi voidaan määrittää rakenne, joka alkaa määritystiedoston luomisella, johon määritellään Apache, PHP, MySQL sekä WordPress. Samalla mitataan

aikaa, jota hyödynnetään käytettävyyden oppimisen mittaamiseen. Kun määritystiedosto on valmis, asennetaan ympäristö sen avulla. Tämän jälkeen mitataan käytettävyyden tehokkuus jakamalla toimivien palveluiden lukumäärä palveluiden kokonaismäärällä. Apache, PHP, MySQL ja WordPress lasketaan kaikki erillisiksi palveluiksi, jolloin kokonaismäärä on neljä. Apachen toimivuutta voidaan pitää ensimmäisenä kriittisenä tehtävänä, joten sen toimiessa kirjataan ylös aika, joka kului sen toimintakuntoon saamisessa. Tämän jälkeen muokataan määritystiedostoa, kunnes kaikki muutkin palvelut toimivat. Sen jälkeen kirjataan ylös kokonaisaika, joka kului toimivan ympäristön luomiseen, jonka jälkeen voidaan laskea käytettävyyden opittavuuden arviointiluku. Kun ympäristö on saatu toimimaan oikein, se poistetaan ja asennetaan uudelleen viisi kertaa. Jokaisella käynnistyskerralla mitataan käynnistymisaika, joiden perusteella lasketaan keskiarvo, jota voidaan verrata kehitysympäristöjen välillä. Testin ensimmäinen vaihe on havainnollistettu kuvassa 5.

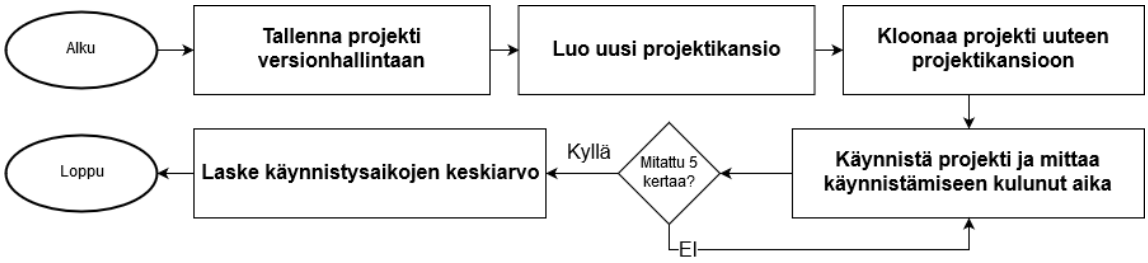


Kuva 5. Käytettävyyden ja nopeuden testausrakenne

Seuraavaksi analysoidaan väitettä ”Palveluiden toimittamisen nopeus kolminkertaistuu”. Toimittamisen nopeutuminen kolminkertaiseksi voidaan ajatella liittyvän koko ohjelmistokehitysprosessiin kuluvaan aikaan, jolloin sen arvioiminen olisi liian suuritöistä. Yksinkertaistettuna väitteen voidaan ajatella viittaavan ohjelmiston vaivattomaan siirtämiseen kehityskoneiden sekä tuotannon välillä. Tärkein ominaisuus on tällöin siirrettävyys. Myöskin luotettavuus on tärkeä ominaisuus, jotta ohjelmisto toimii myös siirtämisen jälkeen oikein. Siirrettävyydelle on haasteellista määritellä testiin soveltuvaa mittaria, joten voidaan hyödyntää suorituskyvyn mittareita ja mitata, kuinka nopeasti versionhallintaan päivitetyn projektin saa käynnistettyä omalle koneelle. Oletetaan, että ympäristö löytyy jo valmiiksi

omalta koneelta, joten se täytyy vain päivittää versionhallinnassa olevalla projektilla ja käynnistää. Samalla voidaan arvioida luotettavuutta mittaamalla mahdollisten virhetilanteiden määrä ja niiden korjaamiseen tarvittava aika.

Testin seuraava kohta alkaa projektin tallentamisella versionhallintaan. Tämän jälkeen luodaan uusi projektikansio, johon versionhallintaan tallennettu projekti kloonataan. Kloonattu projekti käynnistetään ja sen käynnistymisaika mitataan viidesti ja tulosten avulla lasketaan käynnistymisaikojen keskiarvo. Testin vaihe on havainnollistettu kuvassa 6.

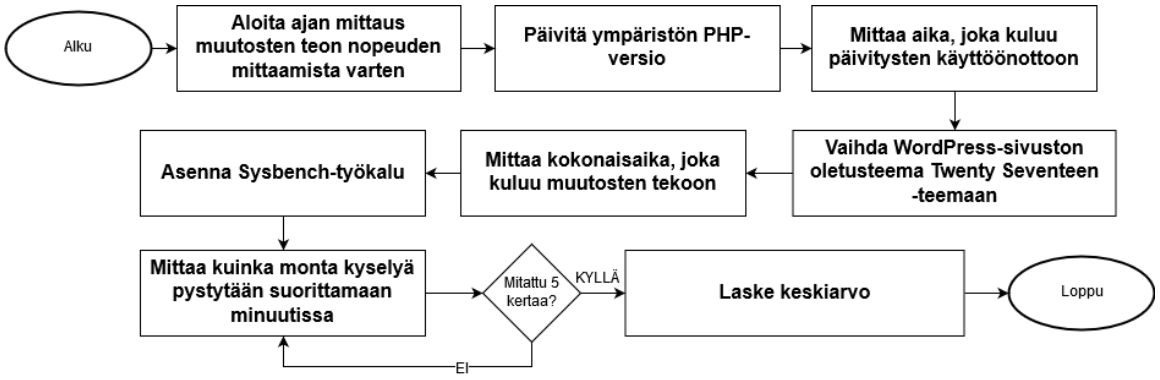


Kuva 6. Siirrettävyyden testausrakenne

Seuraavaksi käsitellään väitettä ”Muutosten teko ja ongelmien ratkaiseminen nopeutuu”. Väitteestä voidaan hahmottaa tärkeimpänä ominaisuutena ylläpito. Sitä voidaan mitata kokonaisaikana, joka kuluu muutosten suorittamiseen sekä vertailemalla, kuinka monta riviä koodia tarvitsee kirjoittaa muutosten lisäämiseen. Kohdeyritys voi saada esimerkiksi tehtäväksi päivittää sivuston PHP-version ja sen jälkeen tehdä sivustolle pieniä muutoksia asiakkaan tarpeen mukaisesti. Myöskin suorituskyvyn voidaan ajatella liittyvän väitteeseen, sillä ohjelmiston pitää jaksaa pyöriä vaivattomasti, jotta sen käyttö on tehokasta. Suorituskykyä voidaan tässä tapauksessa testata MySQL-rasitustestillä. Kohdeyritys tuottaa verkkosivuja, joten esimerkiksi WordPress-sisällönhallintajärjestelmä tarvitsee tietokannan toimiakseen. Tällöin voidaan testata, kuinka tehokkaasti voidaan työskennellä, jos joskus tarvitsee työskennellä projektissa, jossa tarvitsee tehdä paljon tietokantakyselyitä. Työkaluja, joilla mittauksen voi tehdä on monia, kuten JMeter, Prometheus tai Sysbench. Tutustumalla eri vaihtoehtoihin Sysbench-työkalu vaikutti tarpeeksi yksinkertaiselta ja helposti käytettävältä, jotta testauksen voi suorittaa helposti.

Testin seuraava kohta voidaan aloittaa PHP-version päivittämisellä. Tällöin määritystiedostoon tehdään muutoksia, joilla PHP-versio saadaan päivitettyä. Muutosten teon jälkeen käynnistetään ympäristö ja mitataan, kuinka kauan aikaa kuluu, että muutokset päivittyvät ympäristöön ja ympäristö käynnistyy. Tämän jälkeen kirjaudutaan WordPress-sivustolle ja muutetaan oletusteema ja kirjataan ylös koko muutosprosessissa kulunut aika. Tämän jälkeen asennetaan Sysbench-työkalu, jonka avulla suoritetaan MySQL-suorituskykytestaus. Suorituskykytestauksessa mitataan, kuinka monta tietokantakyselyä sovellus

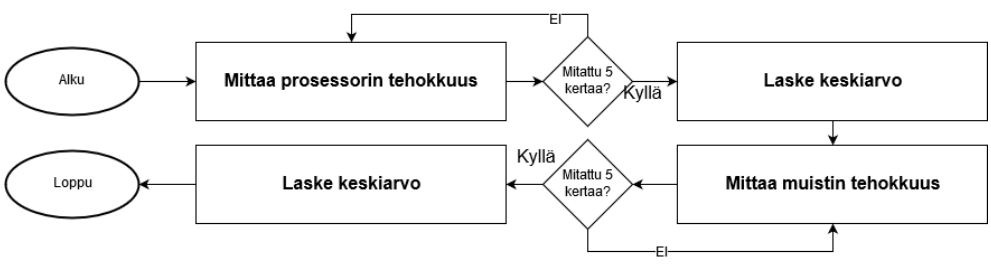
suorittaa minuutissa. Mittausyksikkönä pidetään kyselyä per sekunti. Mittaus toistetaan viidesti ja mittaustuloksista lasketaan keskiarvo. Testin rakenne on havainnollistettu kuvassa 7.



Kuva 7. Ylläpidon ja sovelluksen suorituskyvyn testausrakenne

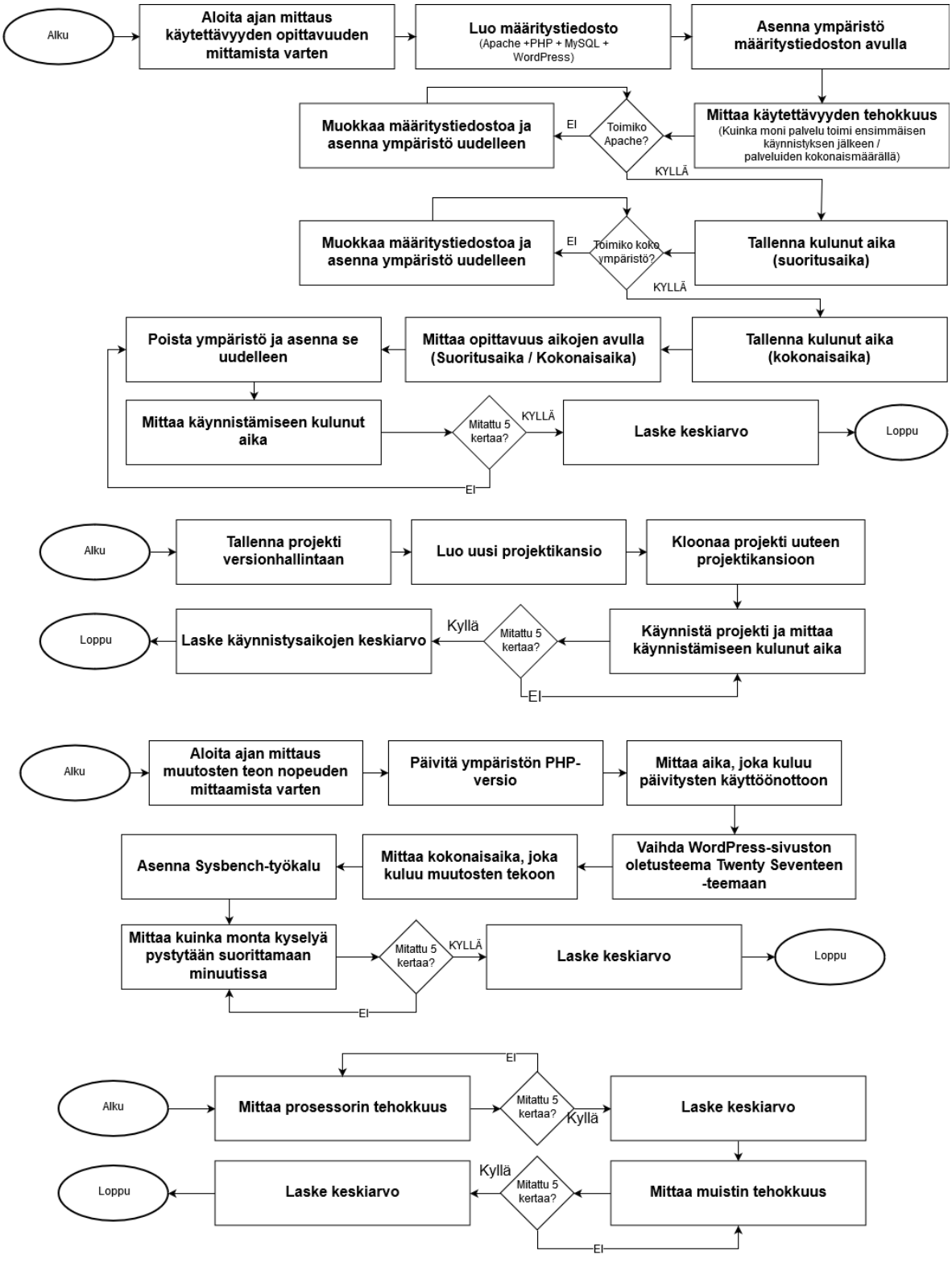
Viimeisenä analysoidaan väitettä ”Docker käyttää resursseja tehokkaasti, ja samalla tietokoneella voi pyörittää useita kontteja samanaikaisesti”. Tämä väite liittyy pelkästään suorituskykyyn. Jos koneella voi pyörittää useita kontteja samanaikaisesti, voidaan päätellä, että prosessoria ja muistia ei rasiteta paljoa ja ne voivat toimia tehokkaasti. Prosessorin ja muistin tehokkuutta voidaan mitata usealla benchmark-työkalulla, mutta koska testi pitäisi olla helposti toteutettavissa, valitaan mahdollisimman helppo työkalu. Tässäkin voidaan valita Sysbench-työkalu, koska se on jo asennettuna ympäristöön edellisen testikohdan jäljiltä. Mittauksia voitaisiin tehdä myös hyödyntämällä Dockerin omaa docker stats komentoa, mutta näitä tuloksia ei pystytä tallentamaan helposti mihinkään. Hyödyllisinä työkaluina voidaan myös pitää Prometheusta ja Grafanaa, joilla pystyisi tallentamaan tietoa tasaisin väliajoin, mutta niiden käyttäminen vaatisi liian pitkää tutustumista.

Viimeisenä testin vaiheena voidaan pitää prosessorin ja muistin tehokkuuden mittaamista rasitustestillä. Molemmat mittaukset suoritetaan Sysbench-työkalun avulla. Testi kestää minuutin ja sillä lasketaan, kuinka monta toimintoa prosessori ja muisti käsittelevät minuutissa. Mittausyksikkönä pidetään toimintoa per sekunti. Testi toistetaan viisi kertaa ja tuloksista lasketaan keskiarvo. Testin kohta on havainnollistettu kuvassa 8.



Kuva 8. Suorituskyvyn testausrakenne

Koko testin rakenne on esitetty vaiheittain kuvassa 9.

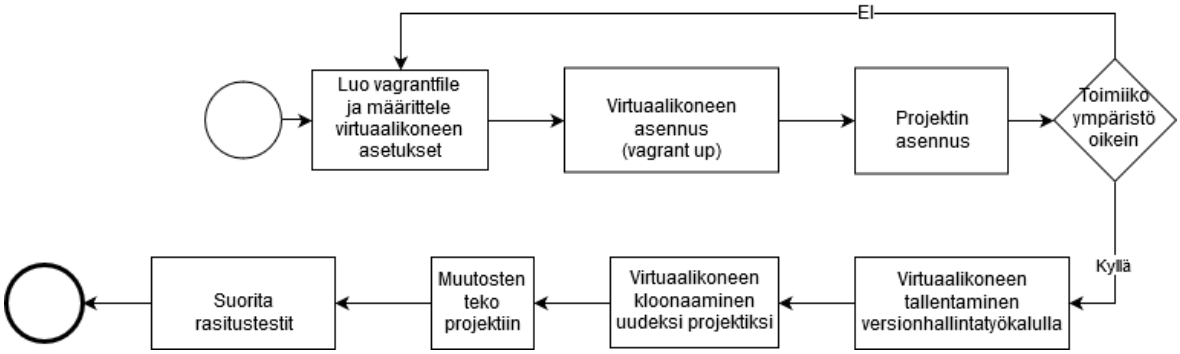


Kuva 9. Testin rakenne vaiheittain

4.5 Ratkaisun demonstrointi

Ratkaisun demonstroinnissa toteutetaan suunniteltu testi ensin virtuaalikoneella ja sen jälkeen Dockerilla. Testistä saatavia tehokkuuden mittaustuloksia käytetään virtuaalikoneiden ja Dockerin tehokkuuden vertailemiseen.

4.5.1 Kehitysympäristö virtuaalikoneella



Kuva 10. Prosessikaavio kehitysympäristön luomisen vaiheista Vagrantin ja VirtualBoxin avulla

Kehitysympäristön luominen virtuaalikoneella alkoi virtuaalikoneen käyttöjärjestelmän asentamisella. Virtuaalikone asennettiin VirtualBox- ja Vagrant-ohjelmistojen avulla. Asennusprosessi aloitettiin komentorivin avulla siirtymällä projektikansioon ja kirjoittamalla komento ”`Vagrant init ubuntu/xenial64`”. Komento loi vagrantfile-tiedoston, jossa käyttöjärjestelmän asentamiseen oli määritelty valmiiksi boksi nimeltään ubuntu/xenial64. Virtuaalikoneen käyttöjärjestelmänä toimi siis Ubuntu.

Vagrantfileen piti lisätä vielä muita määrityksiä, jotta ympäristö toimi oikein. Suurin osa määrityksistä oli lisätty vagrantfileen automaattisesti, mutta ne olivat kommentoituina, eli ne eivät olleet heti käytössä. Käyttöönotetut määritykset olivat

```
config.vm.network "forwarded_port", guest: 80, host: 8080

config.vm.synced_folder ".", "/var/www/html"

config.vm.provision "shell", path:”script.sh”.
```

Ensimmäinen määritys määritteli porttiasetukset, joita tarvittiin verkkosivuston näkemiseen selaimella. Sivusto toimi osoitteessa localhost:8080. Toisella määrityksellä määriteltiin synkronoitu kansio isäntäkoneen ja virtuaalikoneen välille. Virtuaalikoneen /var/www/html kansio synkronoitiin projektikansion kanssa. Tällöin jos virtuaalikoneella teki muutoksia kyseisessä kansiossa, ne synkronoitiin heti myös isäntäkoneen kansioon ja päinvastoin.

Viimeisenä piti määrittää virtuaalikoneessa tarvittavat ohjelmat. Ne määriteltiin vagrant-fileen shell-skriptin avulla. Projektikansioon luotiin uusi tiedosto nimeltään script.sh, jonka polku lisättiin Vagrantfileen. Script.sh tiedostoon kirjoitettiin komennot Apachen, PHP:n, MySQL:n sekä WordPressin asentamiselle. Tämän jälkeen virtuaalikoneelle oli määritelty kaikki tarpeellinen, joten se voitiin käynnistää komentorivillä komennolla ”`Vagrant up`”.

Kun virtuaalikone oli luotu, mentiin selaimella osoitteeseen localhost:8080 ja todettiin, että Apachen asentaminen onnistui. Sen jälkeen kirjauduttiin virtuaalikoneen komentotulkkiin komennolla ”`vagrant ssh`”. WordPressin asentamista varten tarvitsi MySQL-tietokantaan luoda ”wordpress” taulu. Se luotiin kirjautumalla tietokantaan komennolla ”`mysql -u root -p`”, jonka jälkeen annettiin salasana, joka oli määritelty script.sh tiedostoon MySQL:n asentamisen yhteydessä. Taulu luotiin komennolla ”`CREATE DATABASE wordpress;`”. Tämän jälkeen voitiin isäntäkoneen projektikansiossa avata WordPressin wp-config-sample.php tiedosto ja muokata se vastaamaan MySQL:n asetuksia, jonka jälkeen se tallennettin nimellä wp-config.php. Sen jälkeen WordPress asennettiin ohjatun asennusnäkymän mukaisesti osoitteessa localhost:8080/wordpress.

Kun ympäristö oli asennettu onnistuneesti, siirryttiin mittaamaan virtuaalikoneen asentamiseen kuluvaa aikaa. Ensiksi virtuaalikone poistettiin komennolla ”`Vagrant destroy`” ja virtuaalikoneen tarvitsema boksi poistettiin komennolla ”`vagrant box remove ubuntu/xenial64`”. Tämän jälkeen suoritettiin komento ”`vagrant up`”, joka asensi virtuaalikoneen uudestaan.

Projektin jakaminen suoritettiin luomalla uusi tietovarasto GitHub-versionhallinnalla, johon projekti tallennettiin. Tämän jälkeen isäntäkoneelle luotiin uusi kansio, johon projekti kloonattiin komennolla ”`git clone`”. Tämän jälkeen projekti käynnistettiin ja mitattiin, kuinka kauan kestää, että ympäristö on pystyssä. Koska virtuaalikone jouduttiin tässäkin vaiheessa asentamaan uudelleen, mitattiin myös sammutetun virtuaalikoneen uudelleenkäynnistämiseen kuluva aika. Kun virtuaalikone oli asennettu, se sammutettiin komennolla ”`vagrant halt`”, jonka jälkeen se käynnistettiin uudelleen ja mitattiin käynnistymisaika.

Virtuaalikoneen muokattavuutta testattiin muokkaamalla PHP-versio 7.1 versioon 7.2, jonka jälkeen vaihdettiin WordPress-sivuston teema. PHP-versio muokattiin vaihtamalla script.sh tiedostossa kaikkien asennettujen PHP-versioiden ja lisäosien versionumero 7.2:een. Tämän jälkeen kirjoitettiin komentoriville komento ”`Vagrant provision`”, joka teki muutokset virtuaalikoneeseen.

Virtuaalikoneen suorituskykyä testattiin Sysbench-työkalulla, joka asennettiin virtuaalikoneeseen komennolla ”`sudo apt-get install sysbench`”. Ensimmäisenä testattiin,

kuinka monta MySQL-kyselyä virtuaalikoneella pystyi suorittamaan minuutissa. Tätä varten piti luoda ”testi” tietokanta, johon kyselyitä suoritettiin suorituskykytestin aikana. Tietokanta luotiin kirjautumalla MySQL-tietokantaan komennolla ”`mysql -u root -p`”, jonka jälkeen se luotiin komennolla ”`CREATE DATABASE testi;`”. Tämän jälkeen tietokantaan lisättiin rivejä komennolla

> ”`sysbench --test=oltp --oltp-table-size=10000 --db-driver=mysql --mysql-db=testi --mysql-user=root --mysql-password=salasana123 prepare`”.

Kun rivit oli luotu, voitiin siirtyä suorituskykytestiin, joka aloitettiin komennolla

> ”`sysbench --test=oltp --oltp-table-size=10000 --db-driver=mysql mysql-db=testi --mysql-user=root --mysql-password=salasana123 --max-time=60 --oltp-read-only=on --max-requests=0 --num-threads=1 run`”.

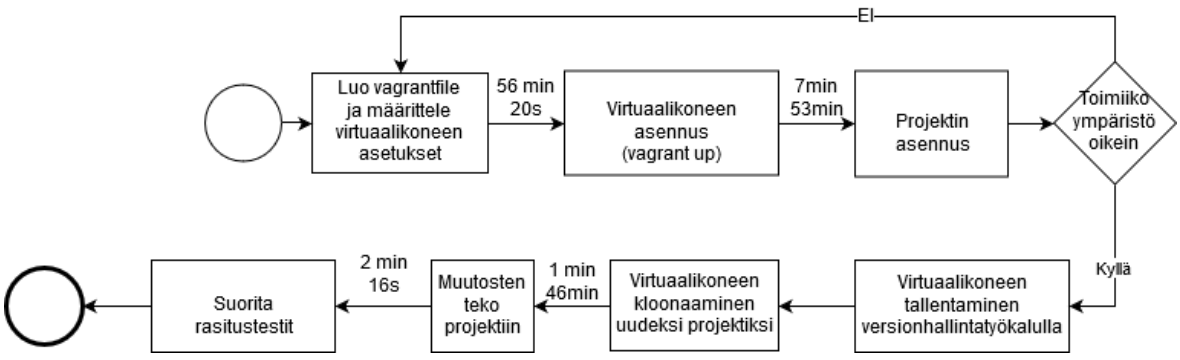
Viimeisenä vaiheena mitattiin prosessorin ja muistin tehokkuutta. Prosessorin tehokkuutta mitattiin komennolla

> ”`sysbench --test=cpu --num-threads=1 --max-time=60 --cpu-max-prime=50000 run`”.

Muistin tehokkuutta mitattiin komennolla

> ”`sysbench --test=memory --num-threads=1 --max-time=60 run`”.

4.5.2 Tehokkuuden mittaustulokset virtuaalikoneella



Kuva 11. Kehitysympäristön luomisen vaiheiden nopeus Vagrantin ja VirtualBoxin avulla

Ensimmäisenä testattiin virtuaalikoneen käytettävyyden tehokkuutta ja opittavuutta. Apache-serveri oli asennettu 15 minuutissa ja 23 sekunnissa. PHP ja WordPress eivät kuitenkaan toimineet vielä ensimmäisen asennusyrityksen jälkeen, joten käytettävyyden

tehokkuus oli vain 2/4. Virheiden korjaamisessa ja WordPressin asentamisessa meni tämän jälkeen vielä 41 minuuttia, joten kokonaisaika oli 56 minuuttia 29 sekuntia. Opittavuuden vertailuluku oli tällöin 923s/3389s = 0,27.

Virtuaalikoneen asentamisen keskiarvoinen kesto oli 7 minuuttia 53 sekuntia. Viisi kertaa suoritetun testin tulokset olivat 370, 400, 360, 599 ja 697 sekuntia.

Sammutetun virtuaalikoneen käynnistämisen keskiarvoinen kesto oli 1 minuutti 46 sekuntia. Viisi kertaa mitattujen käynnistymisaikojen tulokset olivat 119, 97, 103, 100 ja 111 sekuntia. Neljännellä kerralla testaus keskeytyi, sillä virtuaalikone meni vikatilaan eikä käynnistynyt. Tätä aikaa ei kirjattu tuloksiin.

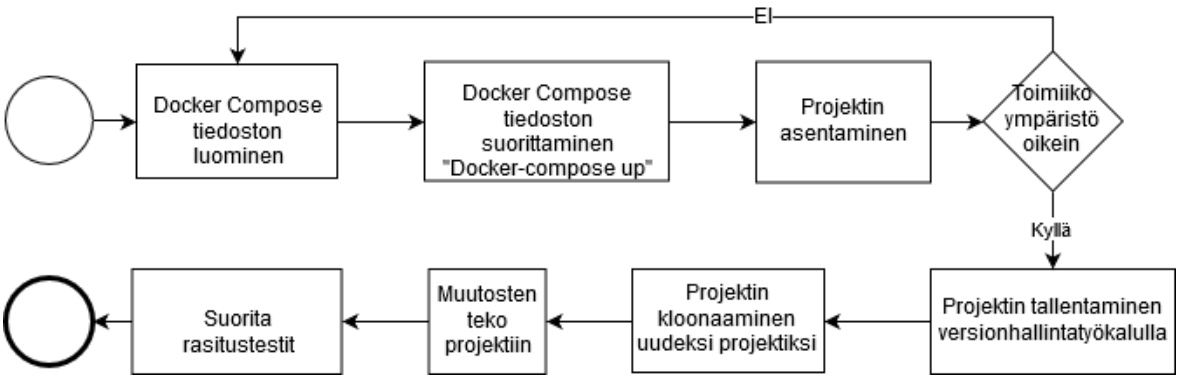
PHP-version muokkaamisessa ja teeman vaihtamisessa meni yhteensä 2 minuuttia 16 sekuntia. PHP-version päivittämisessä kesti virtuaalikoneella 54 sekuntia.

Minuutin kestäneen MySQL-rasitustestin aikana virtuaalikone käsitteli keskimääräisesti 6754 kyselyä sekunnissa. Viisi kertaa suoritetun testin aikana virtuaalikone käsitteli 6490, 5706, 8033, 7487 ja 6058 kyselyä sekunnissa.

Minuutin kestäneen prosessorin rasitustestin aikana virtuaalikone suoritti keskimäärin 98 toimintoa sekunnissa. Viidesti toteutetun testin aikana virtuaalikone suoritti 94,101, 96, 100 ja 100 toimintoa sekunnissa.

Minuutin kestäneen muistin rasitustestin aikana virtuaalikone käsitteli keskimäärin 1602435 toimintoa sekunnissa. Viidesti toteutetun testin aikana virtuaalikone suoritti 1618062, 1621169, 1532908, 1623426 ja 1616611 toimintoa sekunnissa.

4.5.3 Kehitysympäristö Dockerilla



Kuva 12. Prosessikaavio kehitysympäristön luomisen vaiheista Dockerilla

Kehitysympäristön luominen Dockerilla alkoi docker-compose tiedoston luomisella, johon listattiin kaikki sovelluksen tarvitsevat palvelut. Tiedostoon piti määritellä WordPressin ja

MySQL:n kuvat, joille löytyi viralliset kuvat Docker Hubista. Docker-compose tiedostossa määriteltiin kullekin palvelulle kuva (image), josta palvelu koostuu, synkronoidut kansiot (volumes) konttien ja isäntätietokoneen välillä, ympäristömuuttujat (environment) sekä portit (ports). Tämän jälkeen ajettiin komentorivillä komento ”`Docker-compose up –d`”, joka latasi kuvat rekisteristä ja rakensi kuvien avulla sovellukselle kontit.

Kun kontit olivat käynnistyneet, mentiin selaimella osoitteeseen localhost:8081, jossa asennettiin WordPress ohjatun asennusnäkymän mukaisesti.

Ympäristön käynnistymisaika mitattiin ensin kaatamalla kontit komennolla ”`docker-compose down`”, jonka jälkeen poistettiin kaikki ladatut kuvat ja kontit komennolla ”`docker system prune -a`”. Tämän jälkeen ympäristö pystytettiin uudelleen komennolla ”`docker-compose up -d`” ja mitattiin käynnistymisaika.

Tämän jälkeen projektille luotiin uusi tietovarasto GitHub-versionhallinnalla, johon projekti tallennettiin. Tämän jälkeen luotiin uusi kansio, johon projekti kloonattiin komennolla ”`git clone`”. Seuraavaksi projekti käynnistettiin ja mitattiin, kuinka kauan kestää, että ympäristö on pystyssä. Projektissa tarvittavat Docker-kuvat löytyivät jo koneelta, sillä alkuperäinen ympäristö luotiin niiden avulla. Komennolla ”`Docker-compose up -d`” käynnistettiin projekti ja mitattiin käynnistymisaika, jonka jälkeen ympäristö kaadettiin komennolla ”`Docker-compose down`”.

Seuraavaksi päivitettiin ympäristön PHP-versio ja kirjauduttiin WordPressiin ja vaihdettiin sivuston oletusteema. PHP-version sai vaihdettua lisäämällä WordPress-kuvan perään kaksoispisteen ja lisäämällä WordPressin Docker Hubista löytyvän tunnistetiedon kuvaan. PHP-version voi päivittää 7.1:stä 7.2:een seuraavalla kuvalla: ”wordpress:5.3.0-php7.2-apache”.

Sysbench-työkalun asentamiseksi piti ensin kirjautua WordPress-kontin komentotulkkiin komennolla ”`docker exec -it wp_wordpress_1 bash`”. Tämän jälkeen se asennettiin komennoilla

”`apt-get update`”

”`apt-get upgrade`”

”`curl -s https://packagecloud.io/install/repositories/akopytov/sysbench/script.deb.sh | bash`”

”`apt -y install sysbench`”.

Seuraavaksi piti MySQL-tietokantaan luoda tietokanta nimeltä sbtest. Tietokanta luotiin kirjautumalla MySQL-kontin komentotulkkiin komennolla ”`docker exec -it wp_db_1 bash`”, jonka jälkeen se luotiin komennolla ”`CREATE DATABASE sbtest;`” Tämän jälkeen kirjauduttiin takaisin WordPress-kontin komentotulkkiin ja syötettiin komento

```
”sysbench --db-driver=mysql --mysql-host=wp_db_1 --mysql-port=3306
--threads=1 --mysql-user=root --mysql-password=somewordpress --
time=60 oltp_read_only --table_size=10000 --tables=1 prepare”,
```

jonka jälkeen suoritettiin MySQL:n rasitustesti komennolla

```
”sysbench --db-driver=mysql --mysql-host=wp_db_1 --mysql-port=3306
--threads=1 --mysql-user=root --mysql-password=somewordpress --
time=60 oltp_read_only --table_size=10000 --tables=1 run”.
```

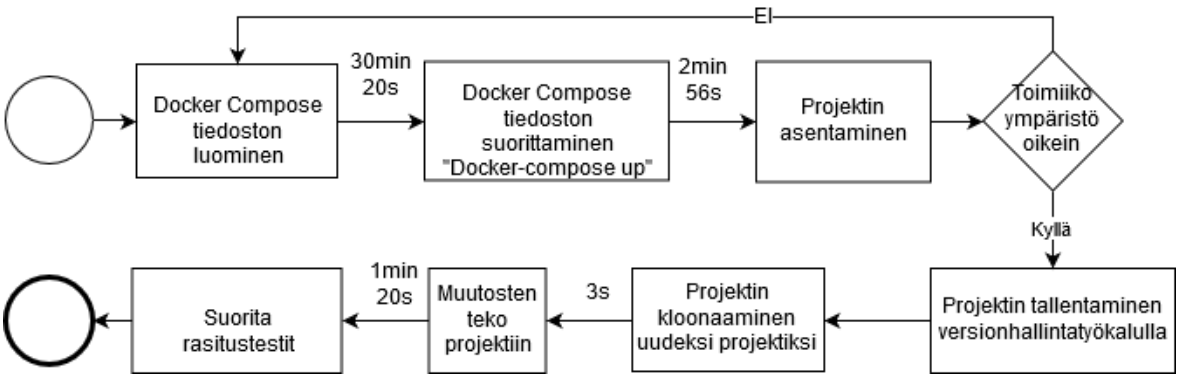
Lopuksi mitattiin prosessorin ja muistin tehokkuus. Prosessorin tehokkuutta mitattiin komennolla

```
”sysbench --test=cpu --num-threads=1 --max-time=60 --cpu-max-
prime=50000 run”.
```

Muistin tehokkuutta mitattiin komennolla

```
”sysbench --test=memory --num-threads=1 --max-time=60 run”.
```

4.5.4 Tehokkuuden mittaustulokset Dockerilla



Kuva 13. kehitysympäristön luomisen vaiheiden nopeus Dockerin avulla

Ensimmäisenä testattiin Dockerin käytettävyyden tehokkuutta ja opittavuutta. WordPressin Docker-kuva sisältää jo itsessään Apachen ja PHP:n, joten niitä ei tarvinnut määritellä erikseen. Sivusto oli pystyssä 15 minuutissa ja 15 sekunnissa. MySQL-serveri ei toiminut

vielä ensimmäisen asennuksen jälkeen, joten tehokkuussuhde oli 3/4. Virheen ratkaisemisessa meni 15 minuuttia 5 sekuntia, joten kokonaisaika oli 30 minuuttia 20 sekuntia. Opittavuuden vertailuluku oli tällöin 915s / 1820s = 0,50.

Ympäristön asennuksen keskiarvoinen kesto oli 2 minuuttia 56 sekuntia. Käynnistymisajat olivat viiden testin aikana 179, 178, 178, 177 ja 172 sekuntia.

Ympäristön uudelleenkäynnistämisessä kesti keskiarvoisesti 3 sekuntia. Kaikkien viiden testin käynnistymisaika oli 3 sekuntia.

PHP-version päivittäminen ja WordPress-teeman vaihtaminen onnistui 80 sekunnissa. PHP-version päivittämisessä kesti Dockerilla 44 sekuntia.

Minuutin kestäneen MySQL-rasitustestin aikana Docker käsitteli keskimääräisesti 7319 kyselyä sekunnissa. Viisi kertaa suoritetun testin aikana Docker käsitteli 7285, 7333, 7147, 7418 ja 7415 kyselyä sekunnissa.

Minuutin kestäneen prosessorin rasitustestin aikana Docker suoritti keskimäärin 96 toimintoa sekunnissa. Viidesti toteutetun testin aikana Docker suoritti 93, 94, 98, 97 ja 98 toimintoa sekunnissa.

Minuutin kestäneen muistin rasitustestin aikana Docker käsitteli keskimäärin 2909114 tapahtumaa sekunnissa. Viidesti toteutetun testin aikana Docker suoritti 2787354, 2988474, 2845610, 2964460 ja 2959676 toimintoa sekunnissa.

4.6 Tulosten vertailu ja analysointi

Ensimmäisenä testissä käsiteltiin käytettävyyden tehokkuutta ja opittavuutta. Opittavuutta mitattiin jakamalla Apachen asentamiseen kulunut aika koko ympäristön asentamiseen kuluneella ajalla. Tehokkuutta mitattiin jakamalla palveluiden onnistuneet asennukset palveluiden kokonaismäärällä. Tulokset on ilmoitettu taulukossa 3.

Taulukko 3. Käytettävyyden tulokset

Käytettävyys	Virtuaalikone	Docker
Opittavuus	923s / 3389s = 0,27	915s / 1820s = 0,50
Tehokkuus	2 / 4 = 0,5	3 / 4 = 0,75

Kehitysympäristön luominen virtuaalikoneella kesti paljon kauemmin kuin Dockerilla, sillä virtuaalikoneella kaikki tarvittavat ohjelmat piti asentaa yksitellen. Tuloksista voidaan myös

huomata, että määritystiedoston luominen kesti yhtä kauan sekä virtuaalikoneella, että Dockerilla, mutta määritystiedoston muokkaaminen täysin toimivaksi kesti virtuaalikoneella lähes kaksi kertaa kauemmin kuin Dockerilla. Dockerin käyttöä helpottaa Docker Hubissa Docker-kuville löytyvät ohjeet, miten ympäristö määritellään, mitä muita versioita kuvasta on saatavilla ja mitä muita kuvia tarvitaan, jotta ympäristö toimii. Dockerin opittavuutta helpottaa docker compose -tiedoston selkeä rakenne, jossa kaikki tarvittavat ohjelmat listataan ja määritellään samoin. Käytettävyyden tehokkuuteen virtuaalikoneella vaikuttaa kuinka hyvin käyttäjä hallitsee Linux-käyttöjärjestelmän, sillä kaikki määritykset tehdään Linuxin komentorivikomentojen avulla. Dockerilla haasteita voi aiheuttaa palveluiden liittäminen toisiinsa, esimerkiksi verkkokauppaa asentaessa pitää MySQL-serverin käyttämiseen tietää localhost osoitteen sijasta MySQL-kontin nimi, jotta yhdistäminen tietokantaan onnistuu.

Ympäristön suorituskykyä mitattiin käyttäjälle näkyvänä ympäristön käynnistymisaikana. Käynnistymisaikoja mitattiin projektin eri vaiheissa. Ensimmäinen aika mitattiin projektia aloittaessa, jolloin kaikki ohjelmat on ladattava ja asennettava ympäristöön. Toinen aika mitattiin projektia jatkaessa, jolloin ympäristö oli jo asennettuna. Kolmas aika mitattiin, kun ympäristöön tehtiin muutoksia. Tällöin mitattiin, kuinka kauan kuluu, että muutokset päivittyvät ympäristöön. Tuloksista laskettiin keskiarvot, jotka ovat ilmoitettu sekunteina taulukossa 4.

Taulukko 4. Ympäristön käynnistymisajat projektin eri vaiheissa

Käynnistymisajat (keskiarvo)	Virtuaalikone	Docker
Projektin aloittaminen	473	176,8
Projektin jatkaminen	106	3
Projektin muokkaaminen	106 + 54	44

Projektin aloittaminen oli huomattavasti nopeampaa Dockerilla kuin virtuaalikoneella. Dockerin hyödyt tulevat selvästi esille, kun Docker-kuvat on ladattu valmiiksi koneelle ja toimiva ympäristö on saatu asennettua. Projekti käynnistyy lähes saman tien ja sen muokkaaminen on myös erittäin nopeaa. Virtuaalikoneen uudelleenkäynnistämisessä kuluu peräti 35 kertaa kauemmin. Tämä on tärkeä huomio tehokkuuden kannalta, etenkin jos työpäivän aikana joutuu työstämään useita eri projekteja. Tällöin Docker nopeuttaa kehitysympäristöä, sillä se poistaa ympäristön käynnistymiseen kuluvaa odotusaikaa reilusti. Virtuaalikoneen käynnistymisajat olivat myös paljon epätasaisempia ja käynnistyminen ei

aina edes onnistunut, sillä virtuaalikone voi mennä myös vikatilaan käynnistyessä. Tällöin menetetään vielä enemmän työskentelyaikaa. Docker-kuvien rakentaminen kerroksittain tulee hyvin esille projektia muokatessa, jolloin muutokset kohdistuvat vain siihen kuvaan ja siihen kerrokseen, jota on muokattu. Kaikki muut Docker-kuvat ja niiden kerrokset pysyvät koskemattomina, jolloin projektin muokkaaminen on myös nopeaa. Virtuaalikone pitää olla käynnissä, jos siihen haluaa tehdä muutoksia, joka hidastaa muutosten tekoa. Nopeiden käynnistymisaikojen takia Docker tehostaa erityisesti projektien jakamista ja muokkaamista.

Laitteiston suorituskykyä mitattiin Sysbench-työkalun avulla. Testaukset kohdistuivat tietokannan, prosessorin ja muistin tehokkuuteen. Tulokset ovat esitelty taulukossa 5.

Taulukko 5. Laitteiston suorituskyky

	Virtuaalikone	Docker
SQL kyselyä sekunnissa	6754	7319
Prosessorin tekemät toiminnot sekunnissa	98	96
Muistin tekemät toiminnot sekunnissa	1602435	2909114

Sysbench-työkalun avulla saadut tulokset olivat melko samanlaisia virtuaalikoneen ja Dockerin välillä. Suurin ero on havaittavissa muistin toiminnassa. Dockerin pitäisi kuitenkin käyttää resursseja tehokkaammin kuin virtuaalikoneen, joten myös prosessorin käyttö pitäisi olla tehokkaampaa. Myöskin Dockerin komennolla ”docker stats” näkee, ettei Docker käytä prosessoria yhtä paljon kuin virtuaalikone, jonka prosessorinkäyttöä voi seurata komennolla ”mpstat”.

4.7 Johtopäätökset

Testin avulla saatiin kerättyä vertailtavissa olevaa dataa Dockerin ja virtuaalikoneen tehokkuuteen vaikuttavista tekijöistä. Käytettävyyden mittareilla saatiin kerättyä tietoa siitä, kuinka helppoa projektin aloitus on mittaamalla ympäristön pystyttämiseen tarvittua aikaa, jonka avulla voitiin arvioida kykyä oppia, miten määritystiedosto luodaan. Käytettävyyden mittareita ei voida pitää pätevinä mittareina, sillä tulokset vaihtelevat jokaisen henkilön

kohdalla eikä niillä pysty saamaan samoja tuloksia uudestaan, sillä ohjelmiston käyttötaito kasvaa ohjelmistoa käyttäessä.

Ympäristön käynnistymisajoilla saatiin kuitenkin todistettua ympäristön pystyttämisen nopeus. Tutkimuksen avulla kerätyt käynnistymisajat projektin eri vaiheista tukevat väitteitä ”tuottavuus paranee, sillä projektien aloitus ja lähdekoodin ohjelmointi on helppoa ja nopeaa” sekä ”muutosten teko ja ongelmien ratkaiseminen nopeutuu”. Käynnistymisaikojen mittausta voidaan pitää pätevänä mittarina arvioitaessa tehokkuutta, myös Rad ja muut mittasivat käynnistymisaikoja tutkiessaan Dockerin tehokkuutta (Rad ym. 2017, 6).

Väitettä ”Palveluiden toimittamisen nopeus kolminkertaistuu” lähdettiin myös mittaamaan nopeutena jakaa projekti ja asentaa se omalle koneelle, koska tarkoitus oli suunnitella helposti toteutettava testi. Tämän kohdan voisi testissä kuitenkin korvata oikean elämän esimerkillä ohjelmistokehitysprojektista, joka on toteutettu joko virtuaalikoneella tai täysin paikallisesti. Tällöin saataisiin hyvä lähtökohta jakaa projekti osiin ja arvioida, mitkä ohjelmiston laadun ominaisuudet liittyivät mihinkin projektin vaiheeseen. Tämän jälkeen voitaisiin projektin ydinkohdat lisätä testin runkoon, jonka avulla saataisiin mitattua konkreettisesti, kuinka paljon tehokkaammin projektin olisi saanut vietyä loppuun Dockerilla ja miten muuten se olisi vaikuttanut projektin kulkuun. Tällöin voitaisiin myös ottaa paremmin kantaa väitteeseen, että toimittamisen nopeus kolminkertaistuu.

Väitettä ”Docker käyttää resursseja tehokkaasti, ja samalla tietokoneella voi pyörittää useita kontteja samanaikaisesti” lähdettiin testaamaan prosessorin ja muistin rasitustesteillä. Prosessorin ja muistin käyttö sekä MySQL-tietokannan tehokkuus ovat hyviä mittareita, joita on käytetty myös muissa tutkimuksissa. Felter ja muut saivat paljon vertailukelpoista dataa hyödyntäen laskentatehon, muistin ja MySQL-tietokannan tehokkuuden mittareita (Felter ym. 2015). Prosessorin ja muistin rasitustesteillä saatiin vertailtavissa olevia tuloksia, mutta tulokset eivät vaikuta oikeilta verrattuna muihin tutkimuksiin. Tästä syystä pitäisi testin rakennetta muuttaa mittaamaan rasitusta useammilla eri rasitustestin arvoilla. Hyödyllistä olisi myös kerätä tietoa resurssien käytöstä ohjelmistoa käyttäessä. Tätä varten pitäisi tutustua Prometheuksen ja Grafanan käyttöön, sillä niillä pystytään keräämään tasaisin väliajoin tietoa resurssien käytöstä. Testin piti olla myös yksinkertainen ja helposti toteutettavissa, joten tätä varten pitäisi kerätä enemmän tietoa siitä, miten Prometheus ja Grafana toimivat, sekä luoda kunnolliset ohjeet niiden käyttöön, jotta testin voi suorittaa helposti.

5 YHTEENVETO

Opinnäytetyön tarkoitus oli selvittää, miten Docker vaikuttaa kehitysympäristön tehokkuuteen verrattuna perinteiseen virtualisointiin. Tutkimuksen lähtökohtana oli todistaa Dockerin käyttöön liittyvien hyötynäkökulmien, kuten nopeus, helppous, muokattavuus ja tehokkaan resurssien käytön, todenperäisyys. Todistus tapahtui suunnittelemalla testi, joka sisälsi ohjelmiston kehitysprojektin vaiheita, jotka toteuttamalla saatiin kerättyä numeerista dataa, jota voitiin käyttää tehokkuuden arvioimiseen.

Testin suunnittelemisessa hyödynnettiin ohjelmistojen laatuominaisuuksia, joiden avulla pohdittiin mistä Dockerin hyötynäkökulmat koostuvat ja miten niitä voitaisiin mitata. Kaikille näkökulmille sopivan mittarin löytäminen osoittautui haastavaksi. Suunnittelussa päädyttiin hyödyntämään lähinnä käytettävyyttä, muokattavuutta, ympäristön käynnistymisaikoja sekä laitteiston suorituskykyä.

Testin oli tarkoitus olla yksinkertainen ja helposti toteutettavissa. Tästä syystä myös mitattavat asiat pidettiin yksinkertaisina ja helposti havaittavina. Myöskin suorituskyvyn mittaamiseen yritettiin löytää mahdollisimman helposti asennettavissa ja käytettävissä oleva työkalu. Suorituskyvyn mittaamisella ei saatu kuitenkaan niin paljoa tietoa kuin olisi tarvittu tulosten kunnolliseen analysoimiseen. Laadukkaampia työkaluja olisi ollut, mutta niiden käyttöönotto oli liian monimutkaista, joten niihin olisi pitänyt perehtyä erittäin hyvin. Opinnäytetyön tavoitteena oli tutustua Dockeriin ja virtuaalikoneisiin, joten ajallisesti olisi ollut ongelmallista tutustua vielä vaativiin benchmark-työkaluihin.

Suunniteltua testiä voidaan pitää onnistuneena, sillä sen avulla saatiin hyvä ensivaikutelma Dockerin tehokkuudesta ja se antaa myös hyvin tietoa muista Dockerin hyödyistä. Testistä on hyötyä ainakin niille, jotka eivät ole Dockerista vielä tietoisia. Tekijä oppi uusia asioita Dockerin sekä virtuaalikoneiden käytöstä työtä tehdessä, mikä oli yksi asia, joka motivoi tekemään opinnäytetyön tästä aiheesta.

Testistä saatujen tulosten perusteella voidaan todeta, että Docker lisää kehitysympäristön tehokkuutta siirrettävyyden, muokattavuuden sekä suorituskyvyn osa-alueilla.

Laadullista tutkimusta voidaan arvioida validiteetin ja reliabiliteetin avulla. Validiteetilla tarkoitetaan tutkimuksen pätevyyttä eli onko tutkimuksesta saatavat tulokset ja tehdyt päätelmät oikeita. Reliabiliteetti tarkoittaa tutkimustulosten pysyvyyttä eli jos testi toistetaan, saadaanko samat tulokset. (KvaliMOTV 2019a; KvaliMOTV 2019b).

Tutkimusta varten kerättiin muista Dockeriin liittyvistä tutkimuksista tehokkuuden mittareita ja työkaluja niiden mittaamiseen. Suunnittelutieteellisen tutkimuskehyksen mukaisesti

testin jokaiselle kohdalle suunniteltiin sopiva mittari perustuen muista tutkimuksista saatuihin tietoihin. Suorituskyvyn mittareita voidaan pitää pätevinä, mutta pelkästään niiden avulla oli vaikeaa suunnitella jokaista testin kohtaa, joten mittaamisen avuksi otettiin ohjelmiston laatumallin kriteerit. Näiden avulla ei pystytty mittaamaan yhtä päteviä tuloksia, mutta niillä saatiin kuitenkin vertailulukuja, joita hyödynnettiin tehokkuuden analysoimisessa. Täten tutkimusta ja sen päätelmiä voidaan pitää pätevinä.

Tutkimustulokset eivät ole kuitenkaan pysyviä. Samalla tietokoneella tehtävät laitteiston suorituskykytestit antavat suunnilleen samoja tuloksia, mutta muulloin tuloksiin vaikuttavat käyttäjän taidot sekä ympäristö, jossa testi suoritetaan.

Opinnäytetyössä käytetyissä lähteissä korostui Dockerin käytön yleistyminen varsinkin pilvipalveluissa. Jatkotutkimusmahdollisuuksia löytyisi esimerkiksi, tutkimalla miten pilvipalvelussa käytettäviä sovelluskontteja voi hallita tehokkaasti, tai tutkia eri sovelluskonttien hallintaohjelmia ja vertailla niiden ominaisuuksia ja eroja.

6 LÄHTEET

Albertao, F., Xiao J., Tian, C., Lu, Y., Zhang, K & Liu, C. 2010. Measuring the Sustainability Performance of Software Projects [viitattu 6.11.2019]. Saatavissa: https://domino.research.ibm.com/library/cyberdig.nsf/papers/3CB7BE1573BE3D0C852577520057A7EB/$File/rc25019.pdf

Berander, P., Damm, L., Eriksson, J., Gorschek, T., Henningsson, K., Jönsson, P., Kågström, S., Milicic, D., Mårtensson, F., Rönkkö, K. & Tomaszewski, P. 2005. Software quality attributes and trade-offs [viitattu 13.11.2019]. Saatavissa: https://www.researchgate.net/profile/Jeanette_Eriksson/publication/238700270_Software_quality_attributes_and_trade-offs_Authors/links/543fa4550cf2f3e82851eef5/Software-quality-attributes-and-trade-offs-Authors.pdf

Casalicchio, E. & Perciballi, V. 2017. Measuring docker performance: What a mess!!! [viitattu 6.11.2019]. Saatavissa: http://www.diva-portal.org/smash/get/diva2:1107544/FULLTEXT01.pdf

Docker Inc. 2019a. Why Docker? [viitattu 2.10.2019]. Saatavissa: https://www.docker.com/why-docker

Docker Inc. 2019b. Orientation and setup [viitattu 17.11.2019]. Saatavissa: https://docs.docker.com/get-started/

Docker Inc. 2019c. About storage drivers [viitattu 18.7.2019]. Saatavissa: https://docs.docker.com/storage/storagedriver/

Docker Inc. 2019d. Use the OverlayFS storage driver [viitattu 19.7.2019]. Saatavissa: https://docs.docker.com/storage/storagedriver/overlayfs-driver/

Docker Inc. 2019e. Overview of Docker Compose [viitattu 20.7.2019]. Saatavissa: https://docs.docker.com/compose/

Felter, W., Ferreira, A., Rajamony, R., Rubio, J. 2015. An Updated Performance Comparison of Virtual Machines and Linux Containers [viitattu 6.11.2019]. Saatavissa: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.471.9242&rep=rep1&type=pdf

Geerts, G. 2011. A design science research methodology and its application to accounting information systems research. International Journal of Accounting Information Systems 12, 142-151

Hevner, A., March, S., Park, J. & Ram, S. 2004. Design Science in Information Systems Research [viitattu 20.10.2019]. Saatavissa: https://www.researchgate.net/publication/201168946_Design_Science_in_Information_Systems_Research

Johann, T., Dick, M., Naumann, S. & Kern, E. 2012. How to measure energy-efficiency of software: Metrics and measurement results [viitattu 1.11.2019]. Saatavissa: https://www.researchgate.net/profile/Timo_Johann/publication/254040409_How_to_measure_energy-efficiency_of_software_Metrics_and_measurement_results/links/5541e8500cf2b790436beb84/How-to-measure-energy-efficiency-of-software-Metrics-and-measurement-results.pdf

Kotilainen, S. 2017. Koodi sujahtaa konttiin – sovellusten kehittäminen mullistuu. Tivi [viitattu 17.11.2019]. Saatavissa: https://www.tivi.fi/uutiset/koodi-sujahtaa-konttiin-sovellusten-kehittaminen-mullistuu/7931ccac-1cd7-3c40-b338-be25469cf1dd

Krochmalski, J. 2016. Developing with Docker. Birmingham: Packt Publishing Ltd.

KvaliMOTV. 2019a. Validiteetti [viitattu 18.11.2019]. Saatavissa: https://www.fsd.uta.fi/menetelmaopetus/kvali/L3_3_1.html

KvaliMOTV. 2019b. Reliabiliteetti [viitattu 18.11.2019]. Saatavissa: https://www.fsd.uta.fi/menetelmaopetus/kvali/L3_3_2.html

Oracle VM VirtualBox. 2019. User Manual [viitattu 19.9.2019]. Saatavissa: http://download.virtualbox.org/virtualbox/UserManual.pdf

Peacock, M. 2013. Creating Development Environments with Vagrant. Birmingham: Packt Publishing Ltd.

Portnoy, M. 2016. Virtualization Essentials. Indianapolis: John Wiley & Sons, Incorporated.

Rad, B., Bhatti, H. & Ahmadi, M. 2017. Introduction to Docker and Analysis of its Performance [viitattu 5.11.2019]. Saatavissa: https://www.researchgate.net/profile/Babak_Bashari_Rad/publication/318816158_An_Introduction_to_Docker_and_Analysis_of_its_Performance/links/5b6c3b76a6fdcc87df700fa6/An-Introduction-to-Docker-and-Analysis-of-its-Performance.pdf

Riaz, M., Mendes, E. & Tempero, E. 2009. A Systematic Review of Software Maintainability Prediction and Metrics [viitattu 13.11.2019]. Saatavissa: https://www.rose-hulman.edu/class/csse/csse575/Resources/maintainabilityMeas05314233.pdf

Schmidt, R. 2013. Software Engineering: Architecture-driven Software Development. Burlington: Morgan Kaufmann.

Stephens, R. 2015. Beginning Software Engineering. Indianapolis: John Wiley & Sons, Incorporated.

Zheng, C. & Thain, D. 2015. Integrating Containers into Workflows: A Case Study Using Makeflow, Work Queue, and Docker [viitattu 5.11.2019]. Saatavissa: http://ccl.cse.nd.edu/research/papers/wq-docker-vtdc15.pdf