Opinnäytetyö AMK

Tieto-ja viestintätekniikka

Kevät 2020

Niko Mikkola

LAMP-PALVELINYMPÄRISTÖN TOTEUTUS RASPBIAN-KÄYTTÖJÄRJESTELMÄLLE

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ AMK TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto-ja viestintätekniikka

2020 | 54 sivua, 7 liitesivua

Niko Mikkola

LAMP-PALVELINYMPÄRISTÖN TOTEUTUS RASPBIAN-KÄYTTÖJÄRJESTELMÄLLE

LAMP-ohjelmistopino on maailman laajimmin käytetyin avoimen lähdekoodin palvelinympäristö. Se on yksi verkon alkuperäisistä avoimen lähdekoodin ohjelmistopaketeista, jota monet kehittäjät käyttävät edelleen verkkosivujen kehityksessä. LAMP-ohjelmistopino on eräänlainen verkkokehityksen dinosaurus, jota tuhannet yritykset käyttävät ja ylläpitävät työssään. Opinnäytetyön tarkoituksena oli LAMP-ohjelmistoympäristön asennus Rasbian käyttöjärjestelmälle soveltaen Dockerin hyödyntämiä DevOpsin toimintaperiaatteita. Työn tarkoituksena oli selvittää LAMP-ohjelmistopinon toteutuksessa käytettyjen avoimen lähdekoodin ohjelmistojen toiminta. Näihin ohjelmiin kuuluvat Apache HTTP-palvelin, MySQL tai MariaDB-relaatiotietokanta ja PHP-palvelinpuolen skriptikieli, jotka yhdessä Linux-jakelun kanssa muodostavat LAMP-palvelinympäristön.

Docker-ympäristöä käytettiin DevOpsin työkaluna toteuttamaan helposti päivitettävä vaihe ja kehitysympäristö. Docker-ympäristöllä toteutettu kehitysympäristö mahdollisti työssä DevOpsin mukaiset jatkuvan kehityksen ja automaation menetelmät. Työssä syvennytään Dockerin tarjoamiin ratkaisuihin kehitysympäristön toteutusta varten ja esittämään yleiskuva tuotannossa vaadituista ratkaisuista.

ASIASANAT:

LAMP, Linux, Apache, PHP, MySQL, DevOps, Docker

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2020 | 54 pages, 7 pages in appendices

Niko Mikkola

LAMP WEB SERVER IMPLEMENTATION ON RASPBIAN OPERATING SYSTEM

The LAMP software stack is the most widely used open source server environment. It is one of the original open source software packages on the internet that many of the developers still use in website development. The LAMP software stack is sort of like web development dinosaur that is still used and maintained by thousands of companies all over the world. The main purpose of the thesis was to implement and demonstrate the complete installation process of the LAMP software stack on the Raspbian operating system, while applying operating principles defined in DevOps. The secondary purpose of the work is to deepen reader's understanding of the softwares used in open source LAMP stack. The software included in the Lamp stack were Apache http-server, MySQL/MariaDB relational database. These software together with a chosen linux distro and PHP server-side scripting language make up a LAMP server environment.

The Docker environment used in project was implemented as a DevOps tool to create an easily updatable development environment which supports the principles of continuous development and automation. The result of the software stack implementation was locally running LAMP environment. The thesis work also delves briefly into topic of various methods of Docker environment implementation and presents an overview of the solutions required in the project. The work demonstrates consistent methods of implementing dynamic web-pages using the Docker environment.

KEYWORDS:

LAMP, Linux, Apache, MySQL, PHP, DevOps, Docker

SISÄLLYSLUETTELO

LIITTEET

KUVAT

SANASTO

Apache	Internetin suosituin http-palvelin, joka perustuu avoimeen lähdekoodiin
Agile	Ketterä ohjelmistokehityksen toimintamalli, joka on joukko ohjelmistotuotannossa käytettävistä menetelmistä.
DevOps	Toimintamalli, joka kuvaa ihmisten, prosessien ja ohjelmistotuotteiden välistä liittoa. DevOps on lyhenne ”Dev”-ja ”Ops”-sanoista, jotka viittaavat kehittäjä-ja operaatiotiimeihin. Oleellisia DevOpsin käytäntöjä ovat ketterä suunnittelu, jatkuva integrointi ja jatkuva kehitys.
Docker	Konttimoottori, joka luo kontteja ajettavan käyttöjärjestelmän päälle automatisoiden sovellusten käyttöönoton vaiheita.
FHS	Tiedostojärjestelmähierarkiastandardi, joka määrittelee Linux-jakeluiden hakemistorakenteen ja hakemistorakenteiden sisällön
GNU GPL	GPL tai General Public License on yleisin ilmainen ohjelmistolisenssi, jonka on kirjoittanut Richard Stallman GNU-projektin vapaan ohjelmiston säätiöstä. Lisenssin avulla kuka tahansa voi vapaasti käyttää, muokata ja levittää ohjelmistoja.
LAMP	LAMP-ohjelmistopino on joukko avoimen lähdekoodin ohjelmistoja, jotka yhdessä muodostavat verkkosovellusten kehittämiseen vaaditun palvelinympäristön. LAMP tulee sanoista: Linux, Apache, MySQL/MariaDB, PHP/PEARL/PYTHON.
Lean	Ajattelutapa, joka keskittyy tuotannossa syntyneiden jätteiden vähentämiseen ja asiakasarvojen lisäämiseen. Lean on prosessivirta, jossa toteutetaan vain asiakasarvoa lisääviä toiminnallisuuksia.

MAMP	Ohjelmistokokoelma, jota käytetään paikallisen web-palvelimen hallintaan. Paketti on lyhenne kokoelman sisältämistä komponenttien alkukirjaimista, jotka ovat Mac OS X käyttöjärjestelmä, Apache, MySQL ja PHP.
MySQL	Avoimen lähdekoodin relaatiotietokannan hallintajärjestelmä, jossa käytetään Structured Query Language kyselykieltä
PDCA	PDCA-sykli on ongelman ratkaisu ja kehittämismalli
PHP	Avoimen lähdekoodin palvelinpuolen skriptikieli, jota käytetään verkkosovellusten kehittämiseen.

1 JOHDANTO

LAMP-ohjelmistopino on tunnetuin sovelluspino maailmassa. Sitä käytetään yleisesti erilaisten dynaamisten verkkosivujen ylläpitämiseen ja toteuttamiseen. LAMP-ohjelmistopinosta tekee erityisen se, että se koostuu täysin avoimen lähdekoodin ohjelmista, johon kuuluu ohjelmien lisäksi Linux-käyttöjärjestelmä, Apache http-palvelin ja MySQL-relaatiotietokanta. Ohjelmistopinossa käytetään vielä PHP-ohjelmistokieltä, joka yhdessä edellä mainittujen ohjelmien kanssa muodostavat LAMP-lyhenteen. Avoimen lähdekoodin toteutuksen ansiosta palvelinympäristön ja loppukäyttäjien välistä vuorovaikutusta on helppo muokata kehittäjien tarpeiden mukaan. Lähestulkoon kaikki palvelintarjoajat käyttävät jotain LAMP-ohjelmistopinon muunnelmaa työssään, minkä vuoksi kehittäjille on tarjolla laaja tuki ja dokumentaatio vastaavista toteutuksista.

Opinnäytetyön tavoitteena oli toteuttaa ja tehdä havaintoja LAMP-ohjelmistopinon asennuksesta Raspbian-käyttöjärjestelmälle hyödyntäen DevOps-toimintamallin periaatteita. DevOps on ohjelmistokehityksen kulttuuri, jota käytetään tavanomaisesti tuotantokehityksessä parantamaan ketterän kehityksen suhdetta kehityksen ja muun IT-toiminnan operaatioiden välille. DevOps on yleistynyt merkittävästi viime vuosina ohjelmistokehityksessä, minkä vuoksi toimintamallia haluttiin soveltaa työssä Docker-konttisovelluksen avulla. Linux-jakeluna käytettiin Debian-käyttöjärjestelmään perustuvaa Raspbian-käyttöjärjestelmää, joka ajettiin Raspberry Pi-isäntäkoneella. Kokonaisuudessaan opinnäytetyöllä pyritään antamaan yleiskuva LAMP-ohjelmistopinossa käytetyistä ohjelmista, DevOps-toimintamallin periaatteista, sekä itse kehitysympäristön asentamisesta ja konfiguroinnista. Tämän lisäksi työllä hahmotetaan lyhyesti Raspberry pi-isäntäkoneella toteutetun LAMP-palvelinympäristön toteutuksen hyödyt vastaavaan virtuaalipalvelimen toimintaan.

Opinnäytetyö suoritettiin itsenäisesti eikä työllä ollut varsinaista toimeksiantajaa. Työ rajattiin siten, että LAMP-ohjelmistopino asennettiin Docker-ja docker-compose ohjelmia hyväksikäyttäen paikallisesti yhdelle Rasbian-isäntäkoneelle. Työssä käytettiin konttien orkesterointiin docker-compose-ohjelmaa osoittamaan normaalien komentorivillä määriteltävien konttien ajon sijaan modulaarisempi, jatkokehityksen kannalta siisti paketti. Työn tarkoituksena ei ole toteuttaa täydellistä DevOps-toimintamallin mukaista ratkaisua, vaan osoittaa tapoja ja metodeja, joilla palvelinympäristö voidaan toteuttaa. Tämän vuoksi toteutuksessa määritetyjen konttien orkesteroinnissa ei esimerkiksi hyödynnetty

docker-swarm tai kubernetes ratkaisua, vaikka molemmat metodit tarjoavat docker-composen tapaan tavan määrittää ja integroida sovelluksia pyörittävät docker-kontit docker ympäristöön.

2 DEVOPS

Nykypäivän ohjelmistokehitys noudattaa tärkeitä tuotannon elinkaaren vaiheita, jotka ovat välttämättömiä monen nykypäivän ohjelmistoyrityksen toiminnalle. Perinteisen ohjelmistokehityksen vaiheisiin kuuluu muun muassa suunnittelun, analysoinnin, toteutuksen, testauksen ja toimituksen vaiheet. Yritykset, jotka perustuvat ohjelmistosovellusten ja tuotteiden valmistukseen valitsevat tavanomaisesti omaa tuotannon elinkaaren toimintoja tukevan kehitysmallin. Näihin toimintamalleihin kuuluvat muun muassa vesiputous, spiraali, V-malli ja Agile, joista Agile kuvaa parhaiten nykypäivän ohjelmistoyritysten vaatimaa kehityssuuntaa. Devops-toimintamalli voidaan nähdä viimeisimpänä lisäyksenä edellä mainittuihin toimintamalleihin ja se on pitkään ollut ohjelmistokehittäjien suosiossa erilaisten digitaalisten palveluiden kehityksessä. Mutta mitä tarkoittaa sana DevOps? Tässä luvussa käydään läpi toimintamallin alkuperä, syyt sen yleistymiseen, sekä sen periaatteet tuotantokehityksessä.

2.1 Historia

Jotta voidaan ymmärtää DevOps periaatteita, tarvitsee myös ymmärtää Lean-ajattelumalli, johon toimintamalli perustuu. Lean-ajattelu alkoi 1910-luvulla, jolloin Henry Ford ja hänen autovalmistamonsa aloitti Ford Model T tuotannon. Henry Ford halusi keskittyä parhaan mahdollisen tuotteen tekemiseen pyrkimällä tavoittelemaan metodeja, joilla voitaisiin minimoida jätteet tuotannossa ja maksimoida työntekijöiden toimintatehokkuus. Yhtiön tavoitteeksi hän ei halunnut asettaa työntekijöiden työpanoksen lisäämistä, vaan hän halusi yhtiön kokonaisuudessaan toimivan järkevämmin. Hänen edellä oleva toimintamalli yksittäisten osien integrointiin, inhimilliseen työhön ja liikkuvaan kuljetukseen oli ensimmäinen esimerkki Lean-ajattelun käytöstä käytännön tuotekehityksessä (Sixsigmadaily, 2017).

Vaikka Ford sai aikaiseksi useita uusia tekniikoita, jotka johtivat ennen pitkään Lean-menetelmiin, olivat ne puutteellisia yhdessä osa-alueessa. Tämä puute oli nykyaikaisten tuotantoyritysten vaatima variaatio liiketoiminnassa (Sixsigmadaily, 2017). Tämän vuoksi toimintamallia kehitettiin ja hienosäädettiin jälkikäteen Kiichiro Toyoda ja Taiichi Ohnon johdolla 1930-luvulta alkaen aina toisen maailmansodan loppuun (Sharma, 2017, s. 38). Kiichiro Toyoda ja Taiichi Ohno saivat vaikutteita tekemiinsä muutoksiinsa Dr. William E.

Demingin työstä 1950-luvulla, jolloin hän julkaisi ehdotuksensa PDCA-syklistä. Fordin prosessit vietiin askeleen edemmälle tekemällä muutoksia valmistusprosessiin, jotka mahdollistivat työnvaihtelun variaation luomatta kestämättömiä tehottomuuksia. Suuri osa muutoksista sisälsi teknisiä muutoksia muun muassa koneiden toimintatapoihin, työnkulun etenemiseen ja kommunikointiin prosessitiimien välillä (Sixsigmadaily, 2017). Näin Lean-ajattelu sai alkunsa. Lean termi sai alkunsa kuitenkin vasta 1990-luvulla, jolloin James P.Womack ja Daniel T. Jones esittelivät sen ensimmäisen kerran kirjassaan ”The Machine that Changed the World” (Sharma, 2017, s. 38).

Vuonna 2001 julkaistiin Lean-periaatteisiin pohjaava nopea ohjelmistojen toimitusprosessi Agile (Ismail, 2018). Agile-toimintamalli on edeltäjä DevOps-toimintamallille, ja sen yleistyminen ohjelmistokehityksessä loivat tarpeen DevOps periaatteille. Ketterä kehitys eli Agile on termi erilaisille iteratiivisille ja inkrementaalisille ohjelmistokehitysmenetelmille. Ketterän kehityksen metodeilla on oma lähestymistapansa, joilla ongelmaa lähestytään. Jokainen metodi voidaan ajatella jakavan yhteisen vision ja perusarvot. Ketteriä menetelmiä soveltaessa otetaan huomioon jatkuvan suunnittelun, testauksen, integroinnin ja muut projektin ohjelmistojen kehityksen muodot. Yleisesti ketterät metodit kannustavat asioiden omaksumista ja johtamistapaa, joka edistää ryhmätyötä, järjestäytymistä ja vastuullisuutta kaikessa tekemisessä (Ismail, 2018). Ketterien metodien ryhmäkokonpanot koostuivat aikanaan pienistä ydinryhmistä, joissa henkilöstö oli pääasiassa kehittäjiä. Ketterien menetelmien kehittyessä ohjelmistojen tuotantoa varten tarvittiin suurempia ryhmäkokoonpanoja, joissa laadunvarmistus nousi keskeisimmäksi osaksi erillisiä tuotannon elinkaaren toimintoja. Ketterän metodologian yhtenä merkittävimmistä eduista on sen mukautuvuus muuttuviin kehitysympäristöihin, vaatimuksiin ja kehittämiseen. Ennen kuin Agile syntyi, kehitysryhmien tehtäviin kuului tuotannossa ilmenneiden ongelmien tunnistaminen ja tuotannossa vaadittujen ratkaisujen suunnittelu (planview, n.d). Kehitysryhmät käyttivät työssään vesiputousmallia ratkaisuja kehitettäessä ja tuotteiden saattamisessa markkinoille. Vesiputousmalli on aikaisin tuotannon elinkaaren lähestymistapa, jota käytettiin ja käytetään edelleen ohjelmistojen kehittämiseen. Vesiputousmalli kuvaa ohjelmistokehitysprosesseja lineaarisena peräkkäisenä virtauksena, mikä tarkoittaa sitä, että mikä tahansa kehitysprosessin vaihe toteutetaan vasta, kun edellinen vaihe on valmis (tutorialspoint, n.d). Tämä lähestymistapa saattaa kuulostaa hyvältä, mutta vesiputousmalli vaati joukkueita noudattamaan hankkeen alussa asetettuja vaatimuksia ja työn laajuuden rajoituksia, eikä kehitysprosessin aikana ilmenneitä muutoksia tai lisäyksiä ylensä tehty valmistavan hankkeen aikana (planview, n.d). Tämä tarkoitti usein sitä, että hankkeet saattoivat pahimmassa tapauksessa viivästyä vuosilla ennen

tuotteiden toimitusta markkinoille. Asiakkaan näkökulmasta viivästyminen tarkoitti sitä, että monet tuotannossa ilmenneet ongelmat jäivät ratkaisematta ja monissa tapauksissa hankkeet johtivat keskeneräisten tuotteiden hautausmaiksi (planview, n.d). Tämän vuoksi nykyään monet IT-toimialan yritykset suosivat ketteriä menetelmiä vesiputousmallin sijasta. Vesiputousmalli on kuitenkin edelleen toimiva ratkaisu ja sitä käytetään projekteissa, joissa ei haluta oppia virheistä iteratiivisesti. Tarkkaan määriteltyjen vaatimusjoukkojen ansiosta ketterä kehitysryhmä on aina tietoinen suoritettavista tehtävistä ja siitä, missä järjestyksessä kyseiset tehtävät tulisi suorittaa. Tuotannossa ilmenneisiin ongelmiin vastataan välittömästi asiakkaiden tarpeiden mukaan, eikä muutoksia jätetä tuotannon elinkaaren loppupuolelle.

Vaikka ketterien metodien edut ovat suorastaan pakottavia nykyajan ohjelmistokehityksessä, on niillä myös useita heikkouksia. Agilen heikkouksia ovat muun muassa työnkulun koordinointi, suunnittelun ongelmat projektien alkuvaiheessa, sekä pitkän aikavälin suunnittelun ja dokumentoinnin puute. Dokumentoinnin puute johtuu siitä, että ketterissä menetelmissä ne on tapana sivuuttaa. Dokumentoinnin puutteen vuoksi ohjelmistovaatimukset selkeytetään juuri kehitystyön aikana. Tämä haittaa etenkin uusien tiimin jäsenien toimintaa, jotka eivät tiedä yksityiskohtia tietyistä sovellusten ominaisuuksista tai kuinka heidän on suoritettava kyseiset ominaisuudet. Ketterät metodit vaativat myös enemmän aikaa ja energiaa, koska kehittäjien ja asiakkaiden on oltava jatkuvasti vuorovaikutuksessa keskenään. Jos asiakkaan palaute tai viestintä eivät ole selkeitä, kehittäjä voi keskittyä väärään kehitysalueeseen, mikä johtaa tuotannossa turhautumiseen ja huonojen päätösten tekoon. (itewiki, n.d)

DevOpsin tavoitteena on laajentaa edellä mainittuja ketterän metodin ideoita ja arvoja toiminnassa ja kehityksessä. Devopsin ajattelun kulttuuri voidaan katsoa alkaneen 2008-luvulla Patric Deboisin ja Andrew Clay Shaferin välisellä keskustelulla ketterän infrastruktuurin käsitteestä (Singh, 2019). Idea syntyi heidän välisestä vuoropuhelusta, jossa tavoitteena oli lisätä ohjelmistokehityksen tehokkuutta. Devops-menetelmien käyttöönotto suunniteltiin mahdollistamaan virtaviivaisemman tavan ohjelmistokehitykselle ja sovellusten käyttöönotolle. Devopsin päällimmäisenä tavoitteena on yhdenmukaistaa kehittäjien ja operaatiotiimien toiminta. Kehittäjien ja operaatiotiimien yhdenmukaistamisessa ihmiset, jotka tekevät sovelluksen elinkaaren aikana tiivistä yhteistyötä varmistavat sen, että sovellukset jatkavat toimintaansa myös koodin julkaisun jälkeen. Tähän kuuluu sovelluksen jatkuva kehitys ja sovelluksen jatkuvan ylläpidon menetelmät. DevOps

kannustaa näin nopeampaan, parempaan ja turvallisempaan tapaan toimittaa sovelluksen ja sen ominaisuudet liiketoimintamallin loppukäyttäjille.

2.2 DevOps-mallin periaate

DevOpsin tavoitteena on viestintä, yhteistyö ja toimintojen integrointi kehittäjien ja operaatioiden välille (NewRelic, n.d). Devops kuvaa yrityskulttuurin, sen käytäntöjen ja työkalujen välistä yhteisvaikutusta. Toimintamallilla pyritään yhdistämään kaikki projektin alle kuuluvat kehitys (Dev) ja operaatiotiimit (Ops), joiden tarkoituksena on pysyä aktiivisena koko sovelluksen suunnitellun elinkaaren ajan (NewRelic, n.d). Operaatiotiimit ovat pääasiassa vastuussa tuotteiden infastruktuurista ja seurannasta, kun taas kehittäjät vastaavat koodien kirjoittamisesta, testitapausten suorittamisesta ja virheiden korjaamisesta. Tiimit pyrkivät DevOpsin toimintamallissa tehokkaaseen ja riippumattomaan työhön. DevOpsissa hyödynnetään tekniikoita ja työkaluja, jotka mahdollistavat tavan minimoida manuaalisista tehtävistä koituvia virheitä ja jätteitä. Toimintamallissa käytetään etenkin automatisoinnin, jatkuvan ohjelmistojen integroinnin, jatkuvan toimituksen, jatkuvan testauksen, jatkuvan seurannan ja nopean korjaamisen menetelmiä (NewRelic, n.d).

DevOpsin ohjelmien tuotannon elinkaareen kuuluvat jatkuva suunnittelu-, kehitys-, toimitus-ja käyttöönoton vaiheet. Tuotantokehityksessä DevOps-tiimit aloittavat toimintamallin elinkaaren suunnitteluvaiheella, jossa tiimit määrittelevät ja kuvaavat käynnissä olevien sovellusten vaatimat ominaisuudet. Suunnitteluvaiheessa pyritään seuraamaan sovelluksen edistymistä kaikilla kehityksen tasoilla. Tähän kuuluu muun muassa virheiden seuranta, ketterän ohjelmistokehityksen hallinta Scrumilla, Kanban-taulujen käyttö ja projektin edistymisen seuranta (Microsoft Azure, n.d). Scrum määrittelee kuinka ryhmien jäsenien tulisi työskennellä yhdessä nopeuttamaan kehitystä ja muita laadunvarmistusprojekteja. Koko DevOpsin elinkaaren aikana tiimit pitävät päivittäisiä scrum kokouksia, aikatauluttavat projektiin liittyviä toimintoja käytössä olevien resurssien mukaan ja nimeävät Scrum Masterin, joka vastaa projektin etenemisestä. Projektille osoitetut käyttäjätarinat kuvataan Kanban-nimisillä tauluilla, joista tiimien on helppo katselmoida projektin etenemistä ja määrätä ohjelmistoprojektityöt nimetyille tiimien jäsenille. Kehitysvaiheessa DevOps-tiimit pyrkivät nopeaan innovatiiviseen kehitykseen käyttämällä tuottavuutta edistäviä työkaluja. Työkaluilla tiimit automatisoivat manuaalisia vaiheita ja iteroivat ne automattisen testauksen ja jatkuvan integroinnin avulla. Ratkaisut, joihin

kehitysvaiheessa päädytään, ei saa vaarantaa sovelluksen laatua ja vakautta. Tämän vuoksi kaikki muutokset katselmoidaan ja testataan ennen kuin muutokset julkaistaan kehitysympäristöön. Toimitusvaiheessa otetaan käyttöön kaikki kehitysvaiheessa käsitellyt muutokset tuotantoympäristöihin johdonmukaisesti ja luotettavalla tavalla. Toimitusvaiheessa määritetään etenkin sovelluksen elinkaaren aikana syntyneiden julkaisujen hallintaprosessi mahdollisimman selkein hyväksyntävaihein (Microsoft Azure, n.d). Operaatiovaiheeseen kuuluu tuotannon ohjelmien hallinnointi, jossa sovelluksen toimintaa ylläpidetään koko ohjelmiston tuotannon ajan. Kehittäjät monitoroivat palvelussa tapahtuvia muutoksia virheiden ja ongelmien varalta, sekä keräävät tietoa ohjelmistojulkaisujen ajoista. Lopuksi edetään takaisin suunnitteluvaiheeseen, jossa käydään läpi jatkokehityksen kannalta oleellinen palvelun suunnittelu ja aikataulutus, josta edetään syklin omaisesti muihin ohjelmiston kehityksen vaiheisiin toistaen prosessia koko ohjelman elinkaaren ajan.

2.3 Syyt käyttää DevOps-toimintamallia

DevOps on kulttuurimuutos, jonka avulla pyritään murtamaan esteitä kehitysryhmien ja operatiivisten tiimien välillä. Tiimien välisten vuorottelun sijaan DevOpsissa pyritään tuomaan tiimit yhteen yhdeksi joukkueeksi koko tuotannon elinkaaren ajaksi. Kun yksi joukkue omistautuu elinkaareen ja sen vaiheisiin, on koko tiimi vastuussa kaikista sovellukseen kohdistuvista muutoksista. Tämä parantaa ohjelmistojen laatua, kun koko tiimi ymmärtää sovelluksen toiminnan, siihen kohdistuvat ongelmat ja syyt kehitystarpeelle. Yksi DevOpsin luontaisista eduista on, että se kiihdyttää ohjelmistokehityksen toimintojen nopeutta. Devops mahdollistaa etenkin ohjelmistojen ominaisuuksien ja muutosten nopean toimituksen hyödyntämällä automatisoidun testauksen ja integroinnin menetelmiä. Tällä tavalla vähennetään virheiden löytöön ja korjaamiseen kuluvaa aikaa, kun kehittäjät pitävät tuotetta silmällä koko sovelluksen elinkaaren ajan. DevOps tarjoaa myös liiketoiminnallisia ja teknisiä etuja ohjelmistokehityksessä. DevOpsin etuihin kuuluu nopean toimituksen ja ongelmanratkaisun lisäksi parempi skaalautuvuus, vakaammat toimintaympäristöt, parempi resurssien käyttö, automaatio ja innovatiivinen kehitys.

3 DOCKER-KONTTIPALVELU

3.1 Yleiskatsaus

Docker on avoin alusta sovellusten kehittämiseen, lähettämiseen ja ajamiseen (Docker Hub, n.d). Docker-menetelmiä hyväksikäyttäen voidaan vähentää ohjelmien viivettä ohjelmien tuotannossa ja koodauksessa. Docker tarjoaa tavan pakata ja ajaa sovelluksia eristetyssä ympäristöissä, joita kutsutaan konteiksi. Dockerin mahdollistama eristetty ympäristö mahdollistaa sen, että samalla isäntäkoneella voidaan käyttää useita ohjelmia samanaikaisesti eri konteissa. Perinteisesti nykypäivän pilvipalvelutarjoajat käyttivät Dockerin tarjoaman Docker-konttien sijaan virtuaalikoneita eristämään käynnissä olevat palvelut toisistaan (Docker Hub, n.d). Virtualisointiin on kuitenkin aina liittynyt useita ongelmia. Yhtenä ongelmana voidaan pitää virtuaalikoneen hidasta resurssien latausaikaa. Virtuaalikoneen oma levytila itsestään on niin suuri, että sen käyttöönotto voi isäntäkoneella viedä useita minuutteja. Lisäksi vieraat OS-käyttöjärjestelmät ottavat tietyn verran levymuistia haltuunsa oli käytettävien sovellusten levytila mikä tahansa (Seshachala, 2014). Docker virtaviivaistaa virtuaalisoinnin ongelmia esittäen kaikki virtuaaliset ilmentymät erillisinä kontteina yksittäisessä Docker-ympäristössä. Ympäristössä kontit yhdistyvät palvelupinoiksi, kuten opinnäytetyössä käytettäväksi LAMP-ohjelmistopinoksi, joista kehittäjien on helppo seurata konttien lokitapahtumia ja ylläpitää sovelluksen toimintoja. Dockerin lopputuloksena on se, että kehittäjät ja muut Docker intoilijat kohtaavat vähemmän ongelmia kehityksessä, testauksessa ja uudelleentestauksen vaiheissa.

3.2 Docker-Image

Docker-image on Docker-konttien rakennuskomponentti. Docker-konttikuvat ovat kevyitä, itsenäisesti suoritettavia ohjelmistopaketteja, jotka sisältävät kaiken sovelluksen suorittamiseen tarvittavan lähdekoodin, järjestelmätyökalut, järjestelmäkirjastot ja niiden asetukset (docker, n.d). Docker-konttikuva voi esimerkiksi sisältää MySQL-relaatiotietokannan, Apache-http-palvelimen, Debian-Ubuntu-käyttöjärjestelmän tai jonkin muun halutun sovelluksen tai palvelimen. Konttikuva itsessään on vain lukutyyppinen tiedosto Docker-kontin luomiseksi (Docker Hub, n.d). Konttikuvan tarkoituksena on ohjeistaa valmiin ja suoritettavan sovelluksen version rakennus halutulle käyttöjärjestelmälle. Konttikuva vaaditaan, jotta haluttu koodi voidaan suorittaa Docker-ympäristössä.

Suurin osa avoimen lähdekoodin Docker-kuvista sisältävät pohjakuvan (basic image). Pohjakuva sisältää halutun ajettavan koodin, josta muodostetaan käytettävä sovellus tai palvelu. Pohjakuvaan voidaan käyttäjäkohtaisesti räätälöidä halutut työkalut ja kirjastot dockerfile-nimisen tiedoston kanssa. Pohjakuva on siis kuva, johon nykyinen käytettävä kuva on rakennettu päälle. Docker-konttikuvaa voidaan näin ajatella useista kerroksista koostuvana tiedostona, jota käytetään suorittamaan haluttu koodi Docker-kontissa (Rouse, 2020).

3.3 Docker-kontit

Docker-kontti voidaan ajatella eräänlaisena yksikkönä, jossa ohjelman koodi ja sen riippuvuudet voidaan pakata luotettavasti ja nopeasti tietokoneympäristöstä toiseen säiliökuvina (Docker Hub, n.d). Kontit ovat verrattavissa eräänlaisiin hakemistoihin, joissa hakemisto koostuu tarvittavista sovelluksista ja palveluista. Docker käyttää konttikuvatiedostoja konttien instantiointia varten. Konttikuvista muodostuu kontteja, kun kuvat ajetaan Docker Enginen kanssa.

Kontit eristävät sovelluksen ja sen riippuvuudet, jotta kaikki tarvittavat kirjastot, joita sovellus tarvitsee, voidaan ajaa. Kontit ovat itsenäisiä kokonaisuuksia, eivätkä ne häiritse käynnissä olevia kontteja tai niitä tukevaa palvelinta. Toiminnaltaan kontit ovat käyttöjärjestelmäriippumattomia, eikä niiden toiminta muutu oli käyttöjärjestelmä mikä tahansa. Kontit eivät vaadi omia resursseja etukäteen, vaan resurssien käyttö tapahtuu varsinaisen ajon aikana. Kontit itsessään ovat pieniä ja tarjoavat omat sisäänrakennetut mekanismit ohjelmistojen versiointiin ja komponenttien uudelleenkäyttöön (Yegulalp, 2018). Jokainen kontti voi suorittaa kokonaisen verkkosovelluksen tai palvelun, mutta luotettavuuden kannalta halutaan yleensä, että jokainen verkkosovelluksen ilmentymä toimisi eri isäntäpalvelimilla. Kontit ovat täten yleisessä ohjelmistokehityksessä toivottavia ratkaisuja, koska ne mahdollistavat tasaisen suorituskyvyn sovelluksen suorittamiseen käytettävistä järjestelmistä riippumatta. Lisäksi tämä mahdollista tietotekniikan ammattilaisille välineet, joissa ohjelmat ja niiden rajapinnat voidaan helposti ottaa käyttöön pienin muutoksin.

3.4 Dockerfile-tiedosto

Dockerfile on tekstitiedosto, joka määrittelee Docker-kuvan (Docker Hub). Dockerfile sisältää listan komennoista, jotka ovat tarkoitettu levykuvien muodostamiseen. Dockerfile on täten pohjimmiltaan rakennusohje, jolla määritetään sovelluksen lähdekoodi, palvelimen binaarit, kokoonpano ja muut riippuvuudet. Normaalisti Docker-kuvaa käyttämällä voidaan helposti asentaa yksittäisiä kontteja toisensa perään ilman juuri minkäänlaisia ongelmia käyttäjälle (Wallen, 2019). Dockerfilen hyödyt ilmenevätkin vasta, kun käyttäjältä vaaditaan useamman kontin hallinta Docker-ympäristössä. Käyttäjä saattaa esimerkiksi haluta uusimman version MySQL relaatiotietokannasta käyttöönsä lataamalla riippuvuudet (dependencies), jotka halutaan toteuttaa levykuvan kanssa. Tätä varten käyttäjän täytyy muokata kuvaa joko manuaalisesti käyttäjäkohtaisten tarpeidensa mukaan, tai luomalla Dockerfile-tiedosto jokaista saman kuvan muunnelmaa varten (Wallen, 2019). Dockerfile-tiedoston avulla voidaan rakentaa sama käytetty kuva uudelleen ilman, että käyttäjää vaaditaan manuaalisesti muokkaamaan levykuva tarpeidensa mukaan.

Dockerfile alkaa FROM-komennolla, mikä kertoo mihin sovellukseen tai palvelimeen kuva pohjaa (Butler, 2015). Aloittava peruskuva vaaditaan FROM-komennon kanssa, että voidaan aloittaa kuvan rakentaminen. Yleensä peruskuva ei riitä sellaisenaan, vaan kuvan kanssa voidaan haluta sovelluksen kirjastoja tietyn palvelimen tai sovelluksen toimintoja varten. Tämä tapahtuu Dockerfile tiedostossa RUN-nimisellä komennolla, joka nimestä päätellen rakentaa kirjastot ja niiden riippuvuudet Docker-ympäristön kontteihin ajon aikana. Docker RUN-komentoa käyttäessä on hyvä muistaa, että Docker käyttää välimuistia jokaisen vaiheen toteutuksessa (Linode, 2018). Useamman RUN-komennon käyttö Dockerfile-tiedostossa voi johtaa siihen, että vaadittujen riippuvuuksien komennot eivät joko toimi oikein tai ne eivät toimi ollenkaan. Tämän vuoksi komennot on hyvä yhdistää yksittäisiksi RUN-komennoiksi. Muita hyödyllisiä komentoja ovat EXPOSE, COPY, VOLUME, MAINTAINER, ADD, ENV ja WORKDIR.

3.5 Docker-compose

Docker-compose on monisäiliöisten Docker-sovellusten määrittelemiseen ja ajamiseen tarkoitettu työkalu, jossa kaikki sovellukset ja palvelut voidaan määrittää yhden YAML-tiedoston avulla (Docker Hub, n.d). Docker-compose YAML-tiedosto ”säveltää” sovelluksen kaikki vaadittavat tasot helposti luettavassa muodossa. Opinnäytetyössä esimerkiksi

pyritään tuottamaan toimiva LAMP-ohjelmistoympäristö, jossa vaadittavat tasot ovat Apache http-verkkopalvelin, Mariadb tai Mysql tietokanta ja php skriptauskieli. Jokainen Docker Compose YAML-tiedostossa kuvattu palvelu käsitellään erillisinä kontteina Docker ympäristössä (Docker Hub, n.d). Jos esimerkiksi suunniteltavaan sovellukseen tai palveluun vaaditaan useampi kuin yksi säiliö, niin docker-composen tarjoama selkeä kokonaisuus on varsin kätevä ratkaisu. Docker-compose säilöö kontit yhdelle isäntäkoneelle, jonka vuoksi se ei sovi ratkaisuksi, mikäli tavoitteena on saada haluttu palvelu skaalautumaan useiden Docker-isäntien kesken. Muita ratkaisuja ovat docker-swarm ja kubernetes.

Docker-compose tarjoaa testauksen ja automatisoinnin kannalta hyödyllisiä alikomentoja. Konttien sisällä toimivien sovellusten ja palvelimien lokitietoja voi esimerkiksi tarkastella helposti YAML-tiedoston juurikansiosta *sudo docker-compose logs* komennolla. Ympäristö ajetaan projektin juurikansiosta sudo docker-compose build komennolla, jossa docker-compose.yml tiedosto sijaitsee. Komento rakentaa YAML-tiedostossa määritetyt palvelut, jonka jälkeen palvelut voidaan ajaa *sudo docker-compose up -d* komennolla. Kun kontteja halutaan päivittää, esimerkiksi vanhan pohjakuvan vuoksi, saatetaan haluta poistaa kokonaisuudessaan käytetyt kontit ja rakentaa kontit uudestaan päivityksen jälkeeen. Poisto onnistuu *sudo docker-compose down* komennolla, joka poistaa kontit ja luodut verkot. Lisäämällä -v lipun liittenä komentosarjan loppuun käyttäjän onnistuu myös halutessaan poistaa kaikki yaml tiedostossa määritellyt voluumit. Useimmiten ei kuitenkaan haluta poistaa käytössä olevaa palvelinympäristöä, vaan halutaan yksinkertaisesti pysäyttää käynnissä olevat kontit, joissa palvelut pyörivät. Tällöin käytetään docker-compose stop ja docker-compose start komentoja. Kuten nimistä voi päätellä docker-compose stop pysäyttää käynnissä olevat kontit, kun taas docker-compose start käynnistää ne uudelleen. Lisäksi docker-compose ps listaa kaikki käynnissä olevat kontit, statuksen ja IP-osoitteen, jossa palvelu pyörii.

3.6 Docker-swarm

Docker-swarm on ohjelma, joka luo parvia. Parvet ovat yhden tai useamman Dockeria käyttävän tietokoneen klustereita. Docker-swarm antaa käyttäjälle tavan hallita Docker moottorien klustereita alkuperäiseltä Docker-alustan tasolta (Docker Hub, n.d). Docker-composen tapaan swarm on konttiorkesterimoottori, jossa palvelut kuvataan

ohjelmistopinoina. Palvelut voivat olla mitä tahansa sovellusinstansseja, kuten tietokantoja ja apuohjelmia.

Swarm koostuu useammasta Docker-isännästä, jotka toimivat parvitilassa solmuiksi kutsutuissa kokonaisuuksissa. Docker-swarm luo klustereissaan kahdenlaisia solmuja, jotka ylläpitävät klusterin tilaa. Näitä kahta solmua kutsutaan johtaviksi solmuiksi ja työtekijä solmuiksi. Johtaja-solmujen tarkoituksena on ylläpitää parven toimintaa, ajoittaa palveluita ja palvella swarm-tilan päätepisteitä. Palvelut ovat johtaville ja työntekijäsolmuille määritettävät tehtävät, jossa määritetään esimerkiksi mitä säiliökuvaa solmuissa käytetään, ja mitkä alikomennot suoritetaan kontien sisällä. Kun swarm-tilassa otetaan palvelut käyttöön, tilaa kuvaavien parvien johtajat hyväksyvät palvelumääritelmät palvelun halutuksi tilaksi (Docker Hub, n.d). Tällä tavalla palveluiden toiminta voidaan helposti ajoittaa solmuissa yhdeksi tai useammaksi replikaatiotehtäväksi, riippumatta siitä missä swarm-tilan solmuissa tehtävät ajetaan. Työntekijä-solmut ovat johtavien solmujen tapaan Docker moottorilla pyöriviä tapahtumia, joiden tarkoituksena on pyörittää sovelluksen käytössä olevia Docker-kontteja. Oletuksena kaikki johtaja-solmut ovat myös tyontekijä-solmuja, jotta käyttäjällä on mahdollisuus suorittaa komentoja Docker-palveluiden toimintojen luonnissa ja ajoittaa kaikki työntekijäsolmuissa asetetut tehtävät paikallisille Docker moottoreille. Docker pyrkii työntekijäsolmuilla ylläpitämään Docker-isännän määrittelemää tilaa (Docker Hub, n.d). Tällä tarkoitetaan sitä, että mikäli joku työntekijäsolmuista ei jostain syystä ole ajon aikana käytettävissä, suoritetaan solmun tehtävät swarm tilan muissa solmuissa. Swarm palveluiden tärkeimpänä etuna verrattuna itsenäisiin kontteihin on se, että käyttäjän onnistuu muokata palvelun kokoonpanoa, mukaan lukien verkot ja volyymit, joihin ne on kytketty, ilman vaatimusta käynnistää palveluita manuaalisesti komentoriviltä käsin (Docker Hub, n.d). Dockerilta onnistuu päivittää ja lopettaa solmujen palvelutehtävät vanhentuneille määrityksille ja luodaa uusia määrityksiä, jotka ovat lähimpänä haluttua kokoonpanoa.

4 LAMP-OHJELMISTOPINO

LAMP-ohjelmistopino on avoimen lähdekoodin www-ohjelmointialusta. LAMP-ohjelmistopinon lyhenne tulee sanoista Linux, Apache, MYSQL ja PHP. Muita LAMP-ohjelmistopinon kaltaisia ohjelmistopinoja ovat WAMP, XAMP ja MAMP. Mutta mitä ovat ohjelmistopinot? Tässä luvussa pyritään kuvaamaan LAMP-palvelinympäristön ja ohjelmistopinojen yleiset piirteet, edut ohjelmistokehityksessä ja toiminta palvelinympäristönä.

4.1 Yleistä

LAMP on yksi suosituimmista avoimen lähdekoodin ohjelmistopaketeista ja sitä käytetään laajasti ympäri maailmaa (Emelianov, 2017). Ohjelmistopinot ovat kokoelmia ohjelmistoista, jotka on koottu kerroksittain yhtenäisiksi ympäristöiksi. Ohjelmistopinojen päälimmäisenä tarkoituksena on toteuttaa tehokas alusta, jossa ohjelmistopinossa määritettyjä ohjelmia voidaan suorittaa yhdessä jonkin tehtävän toteutumiseen. Esimerkiksi LAMP-ohjelmistopino on suunniteltu www-verkkosovellusten kehittämiseen, ja se sopii erityisen hyvin dynaamisten verkkosivujen suunnitteluun. LAMP-ohjelmistopino määrittelee Linux-käyttöjärjestelmässä Apache HTTP-palvelimen, MySQL tai MARIADB-relaatiotietokannan ja PHP-ohjelmistokielen toiminnan. PHP-ohjelmistokieli voidaan halutessa korvata joko Pearl-tai Python-ohjelmistokielellä.

4.2 Toimintaperiaate

LAMP-ohjelmistopinossa tärkeimmät komponentit ovat Apache, MySQL tai MariaDB-relaatiotietokanta ja PHP-ohjelmistokieli. Apache toimii LAMP-ohjelmistopinossa HTTP-palvelimena ja sen toimintaan kuuluu pyyntöihin vastaaminen, joita lähetetään käyttäjien pyynnöstä selaimelta palvelimelle. Jos Apachen pyyntö on PHP-ohjelmointikielen skriptiä varten, välitetään pyyntö tiedostoon, jossa PHP-skripti sijaitsee. Kun pyyntö on välitetty tiedostolle, sen sisältämä PHP-skripti suoritetaan. Jos skriptin tarkoituksena on hakea tietoa relaatiotietokannasta, niin PHP kommunikoi Apachen lisäksi myös tietokannan kanssa. Lopuksi PHP lähettää tiedot Apache-verkkopalvelimelle, jotta se voi vastaavasti lähettää tulokset selaimelle.

4.3 Palvelinympäristön tarjoamat edut

Tärkein ominaisuus, minkä LAMP tarjoaa, on sen täysi avoimen lähdekoodin saatavuus (Emelianov, 2017). Kaikki tarvittavat komponenti ovat vapaasti käytettävissä kenelle tahansa. Tämä mahdollistaa sen, että esimerkiksi kehittäjien on mahdollista kehittää www-sovelluksia ilman, että heitä vaaditaan sijoittamaan osaa projektiin vaaditusta budjetista haluttua kehitystyötä varten. Tämän lisäksi LAMP on erittäin modulaarinen, joten kaikki sen komponentit voidaan helposti vaihtaa tarpeiden mukaan mihin tahansa avoimen lähdekoodin ratkaisuihin (Emelianov, 2017). Tähän kuuluu muun muassa edellä mainitut WAMP-, XAMP-ja MAMP-ohjelmistorajapinnat. LAMP auttaa myös vähentämään kehityksessä vaadittua kehitysaikaa. Koska LAMP on avoimen lähdekoodin ohjelmistopino, voidaan sillä rakentaa se mitä muut avoimen lähdekoodin julkaisijat ovat tehneet ja toteuttaa omia ratkaisuja. Tämän vuoksi LAMP-ohjelmistopinon muodostama kokonaisuus on tehokkain tapa kehittää yksinkertainen yritystason palvelinympäristö, koska se sisältää räätälöinnin, joustavuuden, kustannustehokkuuden ja tehokkaiden suojausominaisuuksien ratkaisut.

5 LINUX-KÄYTTÖJÄRJESTELMÄ

Aina kun kytket tietokoneen päälle, saat käyttöösi valikon, jolla voit suorittaa erilaisia toimintoja, joilla voit esimerkiksi selata internetiä ja katsella haluamiasi videoita. Tämän mahdollistaa tietokoneeseen sulautettu käyttöjärjestelmä tai ydin. Käyttöjärjestelmä on ohjelma, joka toimii jokaisen tietokoneen ytimenä huolehtien sen keskeisimmistä toiminnoista, kuten laitteiston ja ohjelmistojen välisestä vuoropuhelusta. Käyttöjärjestelmät ovat tietokoneen ohjelmistoja, jotka hallitsevat kaikkia tietokoneen laitteistoresursseja. Tässä luvussa opitaan mikä on linux ja mitä etuja se tarjoaa verrattuna muihin käyttöjärjestelmiin.

5.1 Yleistä

Linux on avoimen lähdekoodin käyttöjärjestelmä (Linux wiki, n.d). Linux on yksi yleisimmistä käyttöjärjestelmistä maailmassa, ja se tukee kaikkea tietokoneista palvelimiin. Linux on suomalaisen ohjelmistosuunnittelija Linus Torvaldsin ja Free Software Foundationin 1990-luvulla luoma käyttöjärjestelmä. Linux perustuu löysästi Unix pohjaiseen käyttöjärjestelmään (Linux wiki, n.d). Unix ja Linux ovat monella tapaa samanlaisia ja molemmilla on pääasiassa samankaltaiset ohjelmointityökalut, hakemistorakenne ja järjestelmätyökalut, sekä muut moduulit ja avainkomponentit. Suurin ero kahden järjestelmän välillä on se, että Unix ei ole ilmainen, toisin kuin Linux. Toinen, kenties merkittävin ero on se, että Unix on kokonainen käyttöjärjestelmä, kun taas Linux on vain ydin. Kuten useimmat Unixin kaltaiset käyttöjärjestelmät, myös Linux koostuu useista järjestelmän määrittävistä komponenteista. Näistä tärkeimmät ovat kernel-ydin, käynnistyslataaja (bootloader), Daemon-taustapalvelut ja Shell (Linux wiki, n.d).

5.2 Jakelu

Linuxia käyttäessä tulee väkisinkin kuulemaan Linux-jakeluista, joka lyhennetään sanalla ”Linux distro”. Distrot ovat avoimen lähdekoodin Linux-käyttöjärjestelmän versioita, joihin on pakattu yhteen Linuxissa käytettävät asennusohjelmat, hallintatyökalut ja lisäohjelmat (Linux wiki, n.d). Linux ei ole kuin muut käyttöjärjestelmät, vaan kaikki järjestelmäpalvelut, graafiset ohjelmat ja apuohjelmat ovat kehitetty itsenäisesti erillä toisistaan omina

avoimen lähdekoodin ohjelmistoina. Jakelut tekevät työn käyttäjän puolesta lataamalla kaiken avoimen lähdekoodin ohjelmat projektille ja kääntää ne siten, että ne voidaan sulauttaa yhtenäiseksi käyttöjärjestelmäksi. Jakeluilla voidaan asentaa uusia ohjelmia tai päivittää tietoturvapäivityksiä ohjelmistoversioihin esikäännetyissä, pakatussa muodossa. Jakelut tarjoavat siis valmiiksi esikäännetyt, nopeat ja helpot tavat ladata ohjelmistot pakatussa muodossa.

5.3 Linux-järjestelmän rakenne

Linux käyttää käyttöjärjestelmässään kolmea komponenttia. Tärkein on Kernel, joka toimii käyttöjärjestelmän ytimenä. Ydin hallitsee laitteistoresursseja ja joukon ohjelmistopaketteja, jotka muodostavat muut osat Linux-käyttöjärjestelmästä. Ydin toimii vuorovaikutteisessa suhteessa sen alapuolelle kuuluviin ohjelmiin ja laitteistoihin tarjoten tarvittavan abstraktin rajapinnan laitteistojen turvaamiseksi järjestelmätasolla. Yhtenä käyttäjärajapintana toimii Shell-niminen ohjelma. Shell kommunikoi käyttäjän asettamien komentojen perusteella ja suorittaa tarvittavat ytimen toiminnot. Linuxin sisältämät ohjelmat koostuvat pääosin oheislaitteista ja apuohjelmista, jossa aputoiminnot tarjoavat ytimen käyttöjärjestelmän toiminnoille menetelmiä yksilöllisten toimintojen suorittamiseksi. Linux sisältää myös järjestelmäkirjastoja, joiden toimintoja apuohjelmat ja muut sovellusohjelmat käyttävät Kernelin ominaisuuksien suorittamiseksi. (Linux wiki, n.d)

5.4 Linux hakemistorakenne

Linux-hakemiston rakenne perustuu pääasiassa tiedostojärjestelmän FHS-hierarkiastandardiin, joka määrittelee hakemistot ja niiden käyttötarkoitukset (refspecs.linuxfoundation.org, n.d). Linux-hakemistot sijaitsevat juurihakemiston alla. Juurihakemisto merkitään /-merkillä. Juurihakemiston alla on /bin-hakemisto, joka sisältää käyttäjälle ja käyttöjärjestelmälle välttämättömät binaarit. Hakemisto on välttämätön, koska se sisältää kaikki komentorivin komennot, järjestelmäohjelmat ja apuohjelmat. Hakemistossa /lib on kaikki välttämättömät kirjastot, jotka /bin kansio vaatii luodakseen tarvittavat binaarit (Linux wiki, n.d). /Sbin-hakemisto on samanlainen kuin /bin-hakemisto, mutta sisältää vain välttämättömät järjestelmähallinnan binaarit ja järjestelmän pääkäyttäjän komennot. Hakemistossa /home on kaikki kotikansiot käyttäjälle. Kansiossa /var on kirjoitettava vastine /usr-hakemistosta ja se sisältää ainoastaan ohjelmien lukutyyppiset

lokitiedostot ja muut muuttuvat kirjoitettavat tiedostot, jotka luodaan normaalin käytön aikana. Kotikansio sisältää käyttäjän datatiedostot ja henkilökohtaiset määritystiedostot. Kotikansiossa käyttäjän onnistuu vapaasti muokata tiedostoja, mutta Linux järjestelmän muut kansiot vaativat aina korkeammat pääkäyttäjän oikeudet.

5.5 Linux-järjestelmien edut

Linux on ilmainen, avoimen lähdekoodin käyttöjärjestelmä, joka on julkaistu GNU General Public License-lisenssillä (Linux wiki, n.d). Lisenssin etuna on se, että kuka tahansa voi suorittaa, muokata ja levittää lähdekoodia, kunhan tämä tapahtuu saman lisenssin nojalla. Lisenssi mahdollistaa käyttäjälle tavan räätälöidä käyttöjärjestelmän asennus halutulla tavalla mille tahansa laitteistolle. Etenkin ohjelmistokehittäjät hyötyvät tästä, koska lisenssin mahdollistama avoimen lähdekoodin tarkastelu ei ole mahdollista muissa omistusohjelmissa. Asennusprosessin joustavuuden vuoksi käyttäjän onnistuu myös valita latausprosessiin omat asennettavat moduulit ja kirjastot. Linuxin etuna on sen saatavuus monissa laitteistoalustoissa ja -laitteissa. Tämä on näkyvästi esillä etenkin vanhemmissa tietokoneissa. Lataamalla Linux-jakelun tarjoamia ytimiä tarvitsemiisi laitteisiin on oiva tapa elvyttää vanhempia tietokoneita, jotka eivät pysty käsittelemään suuria nykyaikaisia Windows-versioita (Linux wiki, n.d). Opinnäytetyössä käytetty Raspberry Pi on Linux ohjattu, joka itsessään on mahdollistanut laajan IOT-laitteiden variaation, joissa kaikissa on käytössä Linux-käyttöjärjestelmä. Esimerkiksi yksi kaikkien aikojen käytetyimmistä ohjelmistoalustoista, Android, saa virtansa Linux-käyttöjärjestelmästä. Lisäksi monet pilvipalveluntarjoajat tarjoavat virtuaalipalvelimia, jotka käyttävät Linuxia.

Unix-rakenteen ja avoimen lähdekoodin ohjelmien runsauden ansiosta virukset ja muut vakoiluohjelmat eivät juurikaan vaivaa Linuxia (Linux wiki, n.d). Tämä ei kuitenkaan tarkoita, että Linux olisi haittaohjelmille immuuni. Mikään käyttöjärjestelmä ei ole turvallisempi kuin jokin toinen käyttöjärjestelmä. Ero on lähinnä hyökkäysten lukumäärässä ja hyökkäysten laajuudessa. Olennainen osa Linux-tietoturvaa on mahdollisuus rajoittaa käyttäjien oikeuksia tiedostojärjestelmään (Linux wiki, n.d). Pääkäyttäjällä on kaikki vaadittavat oikeudet käyttöjärjestelmään, kun taas tavallisen käyttäjän oikeudet ovat rajoitettuja. Linuxin hakemistorakenteeseen on myös määritetty tarkkaan hakemistojen käyttöoikeudet niiden tarkoitusten mukaan.

6 APACHE WWW-PALVELIN

6.1 Yleistä

Internet voidaan ajatella olemassa olevana kokoelmana www-palvelimia, jotka isännöivät kaikkia verkkosivuja, joihin käyttäjän tulee päästä tietokoneen välityksellä. Palvelimien tarkoituksena on suorittaa palveluohjelmisto, joka käsittelee käyttäjien pyynnöt tapauskohtaisesti. Yksi tällainen ohjelmistopaketti on nimeltään Apache HTTP-palvelin, joka tarjoaa kehittäjille ilmaisen ratkaisun isännöidä verkkosivuja (Hernandez, 2019). Apache on suosittu avoimen lähdekoodin verkkoalusta, jonka tavoitteena on luoda tukeva kaupallisen tason toteutus, joka on vapaasti saatavilla kenen tahansa käyttöön. Apache on tarkoitettu alustana yksilöille ja instituutioille tarjoamaan työkalut luotettavien järjestelmien luontiin sekä kokeellisiin että operatiivisesti kriittisiin tarkoituksiin (httpd.apache.org, n.d).

6.2 Toimintaperiaate

Vaikka Apachea kutsutaan web-palvelimeksi, on se kuitenkin enemmänkin vain fyysisellä palvelimella toimiva ohjelmisto. Apachen päärooli on verkon kautta tapahtuvan viestinnän hallinta. Apachen tehtävänä on luoda yhteys palvelimen ja verkkosivuston selaimen kävijöiden välille toimittamalla pyyntöjä palveluilta toiselle (Hernandez, 2019). Esimerkiksi kun vieraileva käyttäjä haluaa avata sivun verkkosivulle, lähettää selain pyynnön sivustoa pyörittävälle palvelimelle. Aina kun Apache vastaanottaa pyynnön, se analysoi palvelimelle saapuvat pyynnöt otsikoittain soveltaen sille Config-nimisessä tiedostossa määritetyillä säännöillä sen, miten sen tulee toimia ja aloittaa toimintansa. Pyynnön onnistuessa palvelin lähettää tilakoodin pyydettyjen tietojen kanssa merkiksi siitä, että pyyntö on käsitelty palvelimella. Kun pyyntö on käsitelty Apache palvelin ei odota uusia viestejä, vaan asettuu lepotilaan. Lepotilassa Apache on odotustilassa, jossa se odottaa uusia pyyntöjä, joita se voisi käsitellä.

Apache käyttää viestinnässä HTTP/S nimistä protokollaa (httpd.apache.org, n.d). Palvelin ja asiakas kommunikoivat keskenään HTTP-protokollan välityksellä. HTTP/S eli Hyper Text Transfer Protocol (Secure) on protokolla viestinnän turvaamiseksi selaimen ja www-palvelimen välillä. HTTP/S-protokolla määrittelee, kuinka palveluiden välinen

viestintä muotoillaan ja siirretään selaimelta palvelimelle vastaamalla asianmukaisesti palvelimelle saapuviin pyyntöihin ja komentoihin (Hernandez, 2019).

6.3 Konfigurointi

Apachen oletuskokoonpano vaihtelee huomattavasti Linuxin eri jakelujen välillä. Useimmissa Linuxin jakeluissa Apachen määritystiedostot sijaitsevat /etc/apache2/-hakemistossa. Tähän kuuluvat muun muassa Debian-ja Ubuntu-jakelut. Muissa jakeluissa Apachen määritystiedostot sijaitsevat /etc/httpd/-hakemistossa. Oletusasennustiedosto on nimeltään http.conf, josta käsin löytyy kaikki oleellinen tieto Apachen konfigurointiin. Päähakemistosta löytyy useita tavallisia tekstitiedostoja ja alihakemistoja Apacheen liittyen. Hakemistoista tärkein on *apache2.conf-tiedosto*, joka on Apachen pääkonfiguraatiotiedosto. Tiedostolla määritetään Apachen oletusasetukset ja palvelimen määritystiedot. Palvelimen porttien määrittämiseen käytetään etc/apache2-hakemistossa sijaitsevaa *ports.conf*-tiedostoa, joita virtuaalisten isäntien tulisi kuunnella Apachen ajon aikana. Virtuaaliset isäntätiedostot määritetään taas *sites-available*-nimisessä hakemistossa, jossa määritellään palvelimella pyörivät verkkosivut. Hakemistoa käytetään pääasiassa kertomaan, mitä pyyntöihin liittyvää sisältöä näytetään verkkosivulla. Lisäksi mods-hakemistosta löytyy kaikki Apachen toimintaan liittyvät moduulit. (httpd.apache.org, n.d)

7 MYSQL/MARIADB-RELAATIOTIETOKANNAT

7.1 Yleistä MySQL-ja MariaDB-tietokannoista

MySQL on yksi yleisimmistä avoimen lähdekoodin relaatiotietokantojen hallintajärjestelmistä maailmassa, jonka loi ruotsalainen MySQL AB-yritys vuonna 1995 (Rieuf, 2016). MySQL on suunniteltu C/C++-ohjelmointikielillä. MySQL-relaatiotietokannan alkuperäisiä kehittäjiä ovat Michael Widenius, Allan Larson ja David Axmark, joiden tavoitteena oli tarjota tehokkaiksi ja luotettavaksi todettuja tiedonhallintavaihtoehtoja koti-ja ammattikäyttöön (Rieuf, 2016). MySQL-relaatiotietokantaa ylläpitää nykyään Oracle. MySQL-tietokannan omisti aikanaan Sun Microsystems, kunnes Oracle osti sen vuonna 2010. Nykyään MySQL-tietokannalle on olemassa lukuisia eri versioita, jotka käyttävät samoja relaatiotietokantojen syntakseja ja perustoimintoja. Näistä merkittävin on MariaDB-tietokanta, joka on MySQL-tietokannan alkuperäisten kehittäjien luoma MySQL-haarukka. Se syntyi kehittäjien huolenaiheista, jotka liittyivät MySQL-tietokannan siirtymisestä Oraclelle. MySQL-ja MariaDB-tietokanta perustuvat samaan relaatiotietokantojen hallintajärjestelmään ja molemmilla on samat ohjelmistoratkaisut. MariaDB on MySQL:n tapaan avoimen lähdekoodin relaatiotietokanta, joka on julkaistu GNU-lisenssin nojalla. MariaDB on DBMS-järjestelmä, joka suunniteltiin MySQL-tietokantaa laajemmalla arkkitehtuurilla tukemaan useita erilaisia käyttötoimintoja. Monet lisätyistä ominaisuuksista ovat lisänneet MariaDB-tietokannan suosiota relaatiotietokantajärjestelmänä ja nykyään se on yksi tunnetuimmista relaatiotietokannoista maailmalla.

7.2 Relaatiotietokannat

Relaatiotietokanta eli RDMS on kokoelma asiakkaan jäsentelemistä tiedoista (techterms, n.d). Se on paikka, jossa tallennettu data tallennetaan ja jäsennetään tauluihin, joita kutsutaan relaatioiksi. Relaatiotietokannat ovat tiukasti sidoksissa asiakas-palvelin-arkkitehtuurin toimintamalliin. Asiakas-palvelin-arkkitehtuurissa loppukäyttäjät, joita kutsutaan asiakkaiksi, pyrkivät pääsemään käsiksi keskuspalvelimen resursseihin. Resursseja kutsutaan verkkopalveluista tulevien pyyntöjen avulla, jotka tehdään yleensä jonkin graafisen käyttöliittymän tai komentorivikehotteiden kautta. Komentorivikehotteet ovat yleensä SQL-kyselyitä, joilla tieto haetaan suoraan tietokannasta. Kun pyyntö vastaanotetaan tulostaa palvelin halutut resurssit palvelimelta asiakkaiden puolelle.

Relaatiotietokannoissa relaatiot ovat keskenään kytköksissä toisiinsa, minkä vuoksi tiedon hankinta onnistuu helposti yksittäisten SQL-kyselyiden avulla ilman, että kyselyt joudutaan kohdistamaan yksittäisiin relaatioihin. Tietokannat, joissa taulut eivät ole kytköksissä keskenään ovat DBMS-hallintajärjestelmiä.

7.3 Tietokantojen erot

MariaDB-tietokannalla on useita tapoja optimoida relaatiotietokannan ominaisuuksia verrattuna sen edeltäjiin. MariaDB on ensinnäkin huomattavasti skaalautuvampi ja kykenee käsittelemään kymmenien tuhansien taulujen verran dataa (mariadb). MariaDB-tietokanta on myös nopeampi ja se käsittelee pienempiä tietomääriä tehokkaammin ja sujuvammin kuin MySQL-tietokanta. MariaDB tarjoaa toimintoja ja komentoja, joita käyttämällä on optimoitu MySQL-tietokannan suorituskykyä haittaavia tekijöitä. Koska MariaDB perustuu täysin avoimeen lähdekoodiin, on sillä useita etuja verrattuna perinteiseen MySQL-tietokantaan, joka käyttää kaksoislisensointimallia. Esimerkiksi useimmille MySQL-tietokannan kaupallisille laajennuksille on luotu vastaavat avoimen lähdekoodin ratkaisut MariaDB-tietokannassa. Tämä tarkoittaa sitä, että kaikki MariaDB-tietokannan palvelut ovat vapaasti kaikkien käytettävissä. Näiden etujen lisäksi MariaDB jakaa saman tietokanta-ja hakemistorakenteen kuin MySQL. Hakemistorakenteen identtisyyden vuoksi käyttäjät voivat helposti vaihtaa tai päivittää tietokannan suoraan ilman, että relaatioihin liittyvää dataa, taulukkomääritelmiä tai rakenteita jouduttaisiin päivittämään.

8 PHP-OHJELMISTOKIELI

8.1 Yleistä

PHP on palvelinpuolen avoimen lähdekoodin komentosarjakieli. PHP lyhenne tulee sanoista Hypertext Preprocessor language (php.net, n.d). PHP-ohjelmistokieltä tulkitaan aina ajettavalla palvelimella, mikä eroaa suuresti esimerkiksi Javascript-ohjelmistokielestä, jossa kielen tulkitsee itse verkkosivun selain. Molemmat ohjelmistokielet on kuitenkin tarkoitus upottaa osaksi HTML-sivuja, josta käsin niitä voidaan tulkita. PHP-skriptejä käytetään kolmessa eri osa-alueessa:

- palvelinpuolen komentosarjakielenä
- työpöytäsovelluksissa
- komentorivin komentosarjakielenä

PHP voi esimerkiksi kerätä selaimelta kerättyjä lomaketietoja, luoda dynaamisia sivusisältöjä tai lähettää ja vastaanottaa evästeitä. PHP ei rajoitu vain HTML-tulostukseen, vaan sillä pystyy myös tuottamaan kuvia, pdf-tiedostoja ja flash-videoita (php.net, n.d). PHP:llä on tuki tärkeimmille käyttöjärjestelmille ja suurimmalle osalle nykypäivänä tunnetuimmista verkkopalvelimista. Tämä tarkoittaa sitä, että käyttäjällä itsellään on vapaus valita haluamansa käyttöjärjestelmä ja web-palvelin, jossa PHP-skriptejä halutaan ajaa. Yksi merkittävimmistä ominaisuuksista PHP:ssä on sen mahdollistama tuki kaikenlaisille tietokannoille (php.net, n.d). Sivut, jotka vaativat tietokantojen kanssa kommunikointia, voidaan ladata useita tietokantakohtaisia laajennuksia toteuttamaan haluttu kokoonpano. PHP tukee myös palveluiden välistä kommunikointia protokollien avulla.

8.2 Toimintaperiaate

PHP toimii siten, että käyttäjä kommunikoi www-palvelimen kanssa, joka puolestaan vastaanottaa pyynnön selaimelta ajaen valitun php-skriptin. Mikäli käyttäjän tarvitsee kerätä PHP-skriptillä tietoa tietokannasta tai itse tietokantaan, kommunikoi PHP-skripti myös tietokannan kanssa. Lopuksi skripti tulostaa tiedot, jotka siihen ollaan joko erikseen määritetty tai vastaanotettu kyselyllä tietokannasta käsin.

8.3 Asennus ja konfigurointi

PHP asennetaan joko suoraan sudo apt-get install php komennolla tai määrittämällä se dockerfile-nimisessä tiedostossa. Komento ajaa uusimman PHP7:n isäntäjärjestelmälle ja siihen liittyviä riippuvuuksia. Docker-ympäristössä PHP voidaan määrittää dockerfile-tiedoston avulla, jossa sille voidaan asentaa tärkeimmät palvelimeen liittyvät riippuvuudet. Helpoin ratkaisu on ladata php ja palvelin yhtenä docker kuvana, jotta voidaan helposti määrittää, että palvelut varmasti kommunikoivat keskenään. Tarkempi ohjeistus php:n asentamiseen tapahtuu pavelinympäristön toteutus kappaleessa.

PHP:n konfigurointi tapahtuu php.ini nimisessä tiedostossa, joka sijaitsee Raspberry pi:llä yleensä */etc/php7/apache2* nimisessä direktiivissä (php.net, n.d). Tiedosto voidaan määrittää paikallisesti projektin kansiosta ympäristö variaabelilla viittaamalla siihen suoraan joko dockerfile-tai docker-compose.yml-tiedoston kautta. Suorituskyvyn parantamiseksi on hyvä vaihtaa tiedostosta tietyt muuttujien arvot. Muuttuja *memory_limit* esimerkiksi asettaa enimmäismäärän muistille, jota skriptit eivät saa ylittää. Se auttaa estämään mahdollisia virheitä ja vähentää skriptien aiheuttamaa kuormitusta palvelimella. Jos taas skriptit vaativat huomattavasti enemmän muistia mitä on saatavilla, niin *memory_limit* muuttujalla sitä voidaan myös lisätä. Muuttuja *max_execution_time* taas asettaa enimmäisajan sekunteissa, joka on se aika, joka sallitaan skriptille käytettäväksi. Skriptit voivat toisinaan epäonnistua suoritushetkellä, jos asetettu aika ei riitä niiden suorittamiseen. Apachea varten on hyvä esimerkiksi asettaa yli 300 sekunnin viive ennen aikakatkaisua. Testausta varten voidaan asettaa lokitiedot käyttöön *log_errors* muutujan avulla. Lokitietojen avulla voidaan katselmoida skriptien historia, joita käytetään vianetsintäongelmien hallinnoinnissa. Muita hyödyllisiä muuttujia, joita on hyvä määrittää php.ini-tiedostoon, ovat *post_max_size, file_uploads* ja *upload_max_filesize*.

9 PALVELINYMPÄRISTÖN TOTEUTUS

9.1 Rasbian-käyttöjärjestelmä

Raspbian-käyttöjärjestelmä on ilmainen avoimen lähdekoodin käyttöjärjestelmä, joka on tarkoitettu optimoitavaksi Raspberry Pi-laitteistoa varten (Raspbian.org, n.d). Raspbian-käyttöjärjestelmä on johdettu Debian-nimisestä Linux-jakelun käyttöjärjestelmästä ja se perustuu ARM-nimiseen arkkitehtuuriin (Raspbian.org, n.d). Debianin tapaan Raspbian-käyttöjärjestelmä mahdollistaa perusohjelmien ja apuohjelmien lisäksi tuhansien muiden avoimen lähdekoodin ohjelmien käytön ilmaiseksi ja luotettavasti ilman mitään vaatimuksia tai takuita. Raspbian-käyttöjärjestelmän asennuksessa käytetään SD-kortia, jossa tulisi olla vähintään 8Gt tallennustila Raspbian-käyttöjärjestelmän kuvatiedostoa varten. Kuvatiedostoa käyttäessä on hyvä huomioida, että ajettavalla koneella vaaditaan SD-kortinlukijan paikka, jotta käyttöjärjestelmä voidaan polttaa SD-kortille.

Toisin kuin virtuaalikoneissa Raspbian-käyttöjärjestelmä ajetaan suoraan Rasperry Pi-isäntäkoneelta käsin. Virtuaalikoneet ovat vähemmän tehokkaampia kuin oikeat fyysiset koneet, koska virtuaalikoneet käyttävät järjestelmän laitteistoa ja niihin liittyviä resursseja epäsuorasti isäntäkoneesta käsin. Jokaisen ohjelmiston asennuksen tai sen toimintojen suorituksen hetkellä virtuaalikone vaatii isäntäkoneelta pääsyn laitteiston resursseihin. Tämä hidastaa laitteiden käytettävyyttä etenkin jos isäntäkone pyörittää useampaa virtuaalikonetta samanaikaisesti yhdestä koneesta. Koska fyysinen kone ja virtuaalikone jakavat samat laitteisto ja ohjelmistoresurssit sekä niiden ominaisuudet, eivät ne voi toimia samalla tasolla. Toiminnot, jotka vaativat laitteistoresurssien käyttöä täysimääräisesti on suositeltavampaa käyttää fyysistä palvelinta ohjelmistojen kehitykseen. Tämän vuoksi työssä päädyttiin käyttämään Raspberry Pi-isäntäkonetta LAMP-ohjelmistopinon toteutuksessa.

9.2 Docker Raspberry Pi

Raspberry Pi-arkkitehtuuria kutsutaan ARM:ksi ja se eroaa tavallisen tietokoneen tai monien pilvi-ilmentymien takana olevasta arkkitehtuurista. Monien ohjelmistojen binaarit eivät toimi ARM-käyttöjärjestelmällä, koska ne ovat suunniteltu toimiviksi muilla laitealustoilla. Esimerkiksi kaikki ARM7l-arkkitehtuurille suunnitellut sovellukset eivät ole

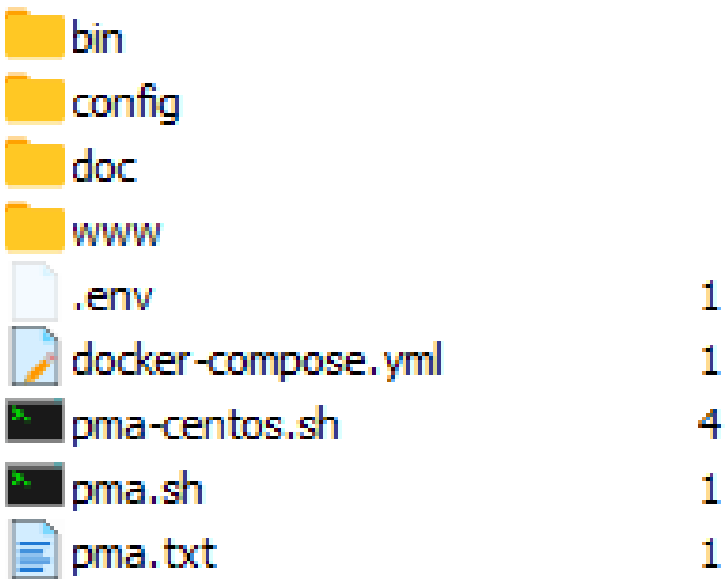
yhteensopivia muiden ARM-arkkitehtuurien kanssa. Tämä tulee ottaa huomioon esimerkiksi käyttäessä Dockerin sovelluskuvia. Docker-kuvien käyttö vaatii sen, että käyttäjä valitsee omaa laitealustaansa varten yhteensopivat sovelluksen versiot. Kuvien käyttöönotossa on otettava myös huomioon se kuinka luotettavasta lähteestä kuvat ovat ladattu, minkä vuoksi käyttäjän on suositeltavissa valita kuvat viralliselta Docker Hub-sivulta. Täysin virallisia kuvia ei ole olemassa, mutta monet Docker-ryhmät luovat jatkuvasti kuvia armhf-nimellä.

9.3 Käytetyt Docker-menetelmät

Työssä asennettiin Docker-ja Docker-compose-ohjelmat komennolla *sudo apt-get install docker docker-compose***.** Docker-ympäristöä hyödynnettiin toteuttamaan automatisoitu kokonaisuus, jossa LAMP-ohjelmistopinon sovellukset asennettiin erillisiin kontteihin. Docker-compose ratkaisua käytettiin työssä tarjoamaan helpon ratkaisun testaamiseen ja kehittämiseen yhden YAML-tiedoston välityksellä. Docker-composea käyttäessä huomioidaan, että toteutustapa mahdollistaa sovellusten ajon vain paikallisesti yhdeltä isäntäkoneelta, mutta se ei mahdollista www-sovelluksen skaalautumista yhden tai useampien palvelimien välillä. Tuotannossa useamman palvelimen skaalautuvuus useammasta isäntäkoneesta olisi parempi ratkaisu, kuin sovellusten säveltäminen docker-compose ohjelman kanssa. Tosin tuotannossa käytetyt kubernetes-klusterit ovat ideaalisempi ratkaisu konttien klusterointiin useampien isäntäkoneiden välillä, tarjoten samalla tuen julkiseen, yksityiseen ja hybridipilven isännöintiin. Tästä johtuen työtä ei toteutettu swarmin kanssa, vaikka se tarjoaakin korkeamman saatavuuden ja vikatietoisuuden kuin docker-compose.

Docker-compose tarjoaa myös sen, että sovellukset, jotka vaaditaan osaksi LAMP-ohjelmistopinoa, voidaan ladata käyttöympäristöön modulaarisemmin ja huolettavuudeltaan paremmin, kuin normaalissa Docker toteutuksessa. Tämä on eduksi kehittäjille, jotka voivat compose toteutusta hyväksikäyttäen määrittää ohjelmien binaarit paikallisesti yhdestä tai useammasta YAML-tiedostosta ympäristömuuttujia hyväksikäyttäen. Docker-compose mahdollistaa tällä tavalla tavan mukauttaa sovellukset erilaisiin ympäristöihin ja ajaa niitä omina yksittäisinä kokonaisuuksinaan.

9.4 Rakenne



Kuva 1. Docker-compose-kehitysympäristön hakemistorakenne.

Docker-compose LAMP-ympäristö rakennettiin kuvassa 1 esitetyllä arkkitehtuurilla. Kaikki sovellukset määritettiin docker-compose.yml nimisessä tiedostossa, joka sijaitsee projektikansion juuressa. Hakemiston juuresta docker-compose.yml tiedosto kutsuu projektissa määritettyjä direktiivejä. Kaikille direktiiveille on annettu DevOpsin kannalta kehittäjiä helpottavat nimikkeet, joista on helppo löytää muun muassa sovelluksille määritetyt binaarit ja konfiguraatiotiedostot. WWW-kansioon on määritetty .env-tiedoston ympäristömuuttujia hyväksikäyttäen polku Apachen oletusdokumentaation juurikansioon, joka sijaitsee Raspbian- ja monessa muussa Linuxin jakeluissa /var/www/html-hakemistossa. WWW-kansioon on tarkoitus määrittää kaikki sivuston kannalta oleellisimmat HTML-, PHP- ja CSS-tiedostot, jotka haluttiin ajaa Apache-palvelimella. Hakemistosta ./doc löytyy kaikki sovellusten kannalta kriittisimmät lokitiedot ja data, joita syntyy Docker-konttipalveluiden ajon aikana. Hakemisto ei ole välttämätön, koska lokitietoja onnistuu seuraamaan Docker-komentojen avulla. Tällä tavalla käyttäjän onnistuu kuitenkin katselmoida aikaisempien sessioiden ajon aikaisia tapahtumia. Lopuksi /config-hakemistosta löytyy kaikki sovellusten toimintaan liittyvät konfiguraatiotiedostot. Muut kuvassa 1 esitetyt tiedostot ovat PhpMyAdminin ajon aikana syntyneitä tiedostoja.

9.5 Docker-Apache ja PHP

Apachen konfigurointi aloitettiin docker-compose.yml-tiedostosta normaalilla pohjakuvan määrityksellä. Apachen pohjakuva sijoitettiin projektin ./bin/php-apache-hakemistoon dockerfile-nimisenä tiedostona, jota kutsutaan yaml-tiedoston kautta.

```
version: '3'

services:
 php-apache:
  build:
     context: ./bin/php-apache
  container_name: 'php-apache_container'
  restart: 'always'
  expose:
     - 80
  ports:
     - "80:80"
  volumes:
      - ${DOCUMENT_ROOT_DIR}:/var/www/html
      - ${APACHE_LOG_DIR}:/var/log/apache2
      - ${VHOSTS_DIR}:/etc/apache2/sites-enabled
  depends_on:
      - 'mariadb'
  networks:
      - container-php-network
```

Kuva 2. PHP-ja Apache-sovelluskontin määritystiedot.

Koska PHP-ohjelmistokielelle ja Apachelle on olemassa yhteinen pohjakuva php-apache, käytettiin työssä tätä takaamaan se, että PHP ja Apache kommunikoivat keskenään. PHP-ohjelmistokielen asentaminen onnistuu erillisen Docker-kuvan kanssa, mutta työtä helpottamiseksi PHP ja Apache on määritetty yhden pohjakuvan alle. Volume-käskyä käytettiin määrittämään projektin juurihakemiston, lokitietojen ja vhost direktiivin sidonta-asennus. Sidonta-asennuksen merkitys näkyy parhaiten suuremmissa palvelinympäristön projekteissa. Esimerkiksi tavanomaisessa sovelluskehityksessä koodi muuttuu nopeasti, minkä vuoksi ei ole mitään järkeä ladata koodeja suoraan konttien sisälle. Tämä johtuu siitä, että jokaisen kontin luonnin yhteydessä käyttäjä joutuisi rakentamaan uudelleen muutokset, jotka hän on jo aikaisemmin suorittanut Docker-ympäristössä. Sen sijaan docker-compose volume sidonta mahdollistaa sen, että Apachen ja PHP-ohjelmistokielen kannalta kriittisimmät tiedostot ja hakemistot johdetaan suoraan

projektin hakemistoista, josta käsin projektin kokoonpano voidaan helposti instansoida uudelleen jokaisen build-komennon yhteydessä. Tärkein näistä on kehitysympäristön toteutuksessa määritelty DOCUMENT_ROOT_DIR-ympäristömuuttuja, joka sitoutuu ajon aikana liittämään paikallisen WWW-kansion Apachen /var/www/html-hakemiston sisälle. Apachen isäntäporttina käytettiin normaalia HTTP-protokollan mukaista porttia 80, jotta Dockerin ajama kontti on suoraan tavoiteltavissa paikallisesta verkosta ja Raspberry Pi-isäntäkoneesta.

LAMP-ohjelmistopinossa Apache on riippuvainen MariaDB-relaatiotietokannasta, minkä vuoksi sen on hyvä olla käynnissä ennen Apache-palvelinta. Riippuvaisuus määriteltiin depends_on muuttujalla, joka ilmaisee sovellusten aloitusjärjestyksen. Docker-composen build-komennon ajan aikana Docker rakentaa siis sovellukset depends_on komennon mukaisessa järjestyksessä. Työn tapauksessa Apache instanssi luodaan vasta kun MariaDB on instantioitu. Tämän lisäksi Apache on määritetty container-php-network nimisellä aliaksella bridge-oletusverkkoyhteyteen. Bridge eli siltaverkkoyhteys luo konteille siltaverkoston, jonka avulla ohjelmistopinon kontit voivat olla yhteydessä toisiinsa. Tämä vaaditaan osaksi Apachea, jotta palvelin voi kommunikoida vapaasti muiden konttien kanssa. Muihin toimintoihin kuuluu restart-komento, joka käynnistää kontin uudestaan aina kun Apachen toiminta keskeytyy, ja container_name-komento, jolla konteille annetaan personalisoitu nimi.

9.6 Docker-MariaDB

LAMP-ohjelmistopinossa valitaan tavanomaisesti Apachen lisäksi joko MySQL tai MariaDB-relaatiotietokanta. Työssä sovellettiin MariaDB-tietokantaa, mutta MySQL sopii myös LAMP-ojelmistopinon toteutusta varten. MariaDB-kontin pohjakuva määritettiin Apachen tapaan testiympäristön /bin-hakemistossa, jota kutsuttiin docker-compose.yml-tiedostosta. Pohjakuvana käytettiin Raspberry Pi-isäntäkoneelle portitettua jsurf/rpi-mariadb:latest-pohjakuvaa. Pohjakuvan ongelmana oli se, että kyseinen ohjelmistoversio poikkeaa muiden sovelluskonttien versioista. Työssä kyseistä pohjakuvaa käytettiin, koska se sopi kehitysympäristössä käytettyyn Raspberry Pi ARM-arkkitehtuuriin. Pohjakuvan ajo ei sinällään vaikuttanut juurikaan Lamp-palvelinympäristön toimintaan, mutta DevOpsissa ja ketterän kehityksen menetelmissä olisi hyvä käyttää tuoreinta versiota relaatiotietokannasta.

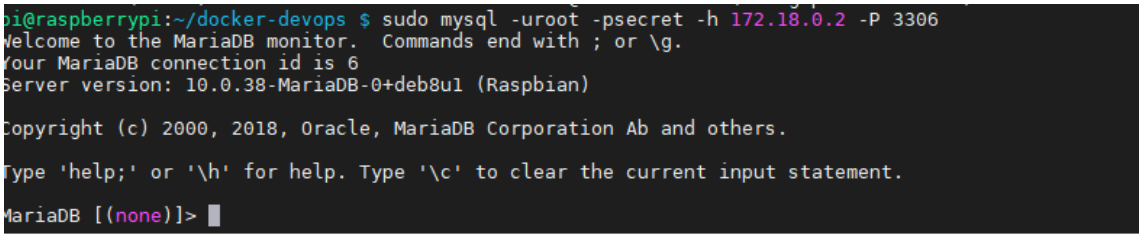
```
mariadb:
  build:
     context: ./bin/mariadb
  restart: 'always'
  container_name: 'mariadb_server_instance'
  command: --init-file /data/application/init.sql
  ports:
     - "3306:3306"
  volumes:
    - mb_server:/var/lib/mysql
    - ${MYSQL_DATA_DIR}:/var/lib/mysql
    - ${MYSQL_LOG_DIR}:/var/log/mysql
    - ${MYSQL_CONFIG_DIR}:/data/application/init.sql
  environment:
     TZ: "Europe/Helsinki"
     MYSQL_ALLOW_EMPTY_PASSWORD: "no"
     MYSQL_ROOT_PASSWORD: "root"
     MYSQL_USER: "lampdev"
     MYSQL_PASSWORD: "initlamp_pass"
  networks:
    - container-php-network
```

Kuva 3. MariaDB-sovelluskontin määritystiedot.

Pohjakuvan lisäksi docker-compose.yml-tiedostoon asetettin tiedostohakemistojen voluumit ja portti, jota kautta tietokannan kanssa voidaan kommunikoida. MariaDB-tietokannalle on luotu voluumeihin erikseen ulkoinen *mb_server*-muuttuja, jota käytettiin sidonta-asennukseen. Sidonta-asennuksen avulla voitiin rakentaa väylä /var/lib/mysql-isäntähakemistoon, jossa on normaalisti rajoitettu pääsy Docker-ympäristön ulkopuolelta. MariaDB-kontin tapauksessa sidonta luotiin, jotta data joka tietokantaan on pakattu, voidaan pitää tallessa myös silloin kun docker-compose-ympäristö käynnistetään uudelleen. Nimettyjen voluumien elinkaari on riippumaton sitä käyttävästä kontista, ja voluumi ilmoitetaan docker-compose.yaml-tiedoston lopussa. Voluumien lisäksi MariaDB-konttia varten asetettiin joukko ympäristömuuttujia. Ympäristömuuttujilla määritettiin MariaDB-relaatiotietokannan aikavyöhyke, root-käyttäjän-ja normaalin käyttäjän tunnus ja salasana.

Tietokannan käyttäjien oikeuksia varten asennettiin mysql-client-paketti *sudo apt -y install default-mysql-client*-komennon avulla. Komento asentaa vanhentuneen version mysql-client-paketista, joten tätä prosessia suorittaessa on suositeltavampaa ladata samainen kirjasto *sudo apt-get install mariadb-client*-komennolla. Paketin avulla tietokannan käyttäjä voi muodostaa yhteyden MySQL-palvelimeen. Tämän avulla kehitysympäristön käyttäjillä on käytössään komentorivin ohjelma, josta käsin tarvittavat SQL-kyselyt

voidaan suorittaa. MySQL komentorivin työkalua käytettiin työssä luomaan tarvittavat käyttäjien oikeudet tietokantaan. Tällä tavalla taataan se, että kehitysympäristön tietokantaan voi tehdä muutoksia vain ne tahot, joilla on vaadittavat oikeudet tietokantaan. Mysql-komentokehotteeseen kirjaudutaan *sudo mysql -u<MariaDB käyttäjänimi> -p <MariaDB käyttäjän salasana> -h <kontin ip-osoite> -P 3306-komennolla* (kuva 4).



Kuva 4. MySQL-komentokehoteen toiminta esitettynä.

Kirjautumisen jälkeen MariaDB-tietokantaan luotiin docker-compose.yml-tiedostossa määritetylle root-käyttäjälle pääsy tietokantaa*n Grant all privileges ON *.* TO '<MariaDB käyttäjä>'@'%' identified by <MariaDB root käyttäjän salasana>* SQL- komennolla.

Tietokannan asennuksessa määritettiin myös setup.sql-tiedosto, joka rakentaa PhpMyAdmin-toteutuksessa vaadittavan PMA-tietokannan. Docker-compose yaml-tiedostossa käytetty command-komento varmistaa, että Docker rakentaa PMA-tietokannan MySQL-palvelimelle jokaisen run-komennon ajon yhteydessä.

9.7 Docker-phpmyadmin

LAMP-ohjelmistopinon lisäksi työssä määritettiin phpmyadmin-ohjelmistotyökalu MariaDB-relaatiotietokantaa varten. PhpMyAdmin on ilmainen avoimen lähdekoodin ohjelmistotyökalu, joka on tarkoitettu MySQL tai MariaDB-tietokantapalvelimen hallintaa varten. Työkalu mahdollistaa graafisen käyttöliittymän tietokannan toimintoja varten. Sillä voi suorittaa SQL-kyselyitä, optimoida, muokata ja korjata tietokannan relaatiotauluja sekä lajitella ja suorittaa muita tietokannan hallintakomentoja. Kontin toteutuksessa ei käytetty virallista PhpMyAdmin konttikuvaa *phpmyadmin/phpmyadmin*, koska kyseinen kuva ei sovi työssä käytettyyn Raspberry Pi ARM-arkkitehtuuriin. Sen sijaan työssä käytettiin *ebspace/armhf-phpmyadmin* nimistä Raspberry Pi portin Docker-konttikuvaa.

```php
<?php
$cfg['Servers'][$i]['pmadb'] = 'phpmyadmin';

$cfg['Servers'][$i]['bookmarktable'] = 'pma__bookmark';
$cfg['Servers'][$i]['relation'] = 'pma__relation';
$cfg['Servers'][$i]['table_info'] = 'pma__table_info';
$cfg['Servers'][$i]['table_coords'] = 'pma__table_coords';
$cfg['Servers'][$i]['pdf_pages'] = 'pma__pdf_pages';
$cfg['Servers'][$i]['column_info'] = 'pma__column_info';
$cfg['Servers'][$i]['history'] = 'pma__history';
$cfg['Servers'][$i]['designer_coords'] = 'pma__designer_coords';
$cfg['Servers'][$i]['tracking'] = 'pma__tracking';
$cfg['Servers'][$i]['central_columns'] = 'pma__central__columns';
$cfg['Servers'][$i]['designer_settings'] = 'pma__designer_settings';
$cfg['Servers'][$i]['export_templates'] = 'pma__export_templates';
$cfg['Servers'][$i]['favorite'] = 'pma__favorite';
$cfg['Servers'][$i]['navigationhiding'] = 'pma__navigationhiding';
$cfg['Servers'][$i]['recent'] = 'pma__recent';
$cfg['Servers'][$i]['savedsearches'] = 'pma__savedsearches';
$cfg['Servers'][$i]['table_uiprefs'] = 'pma__table_uiprefs';
$cfg['Servers'][$i]['userconfig'] = 'pma__userconfig';
$cfg['Servers'][$i]['usergroups'] = 'pma__usergroups';
$cfg['Servers'][$i]['users'] = 'pma__users';

$cfg['Servers'][$i]['host'] = 'mariadb';
$cfg['Servers'][$i]['socket'] = '/var/run/mysqld/mysqld.sock';
$cfg['Servers'][$i]['connect_type'] = 'socket';
$cfg['Servers'][$i]['controluser']   = 'root';
$cfg['Servers'][$i]['controlpass']   = 'secret';
?>
```

Kuva 5. config.inc.php-tiedosto.

Kaikki PhpMyAdminin konfiguroitavat tiedot asetettiin config.inc.php-tiedostoon PhpMyAdminin ylätason /etc/phpmyadmin-hakemistoon. Tiedoston sisältö määritettiin projektin hakemistosta käsin docker-compose volume-komentoa hyväksikäyttäen. Tiedoston tarkoituksena on yleensä sisältää vain ne parametrit, jotka halutaan muuttaa config.inc.php-tiedostosta. Työn tapauksessa tiedostoa ei kuitenkaan ollut olemassa, minkä vuoksi config.inc.php-tiedosto määritettiin projektin ./config-hakemistosta. Tiedoston parametreja käytettiin luomaan PhpMyAdminin PMA-tietokanta, yhteyden tyyppi ja pääkäyttäjä. Pääkäyttäjänä käytettiin ympäristömuuttujissa käytettyä root-käyttäjätunnusta. Ympäristömuuttujillla PMA_HOST ja PMA_PORT määritettiin MYSQL-palvelimen osoitteen isäntänimi ja MySQL-palvelimen portti 3306. Lisäksi PMA_HOST-ympäristömuuttujalla myönnettiin oikeudet MariaDB-palvelimeen.

9.8 Docker-compose-kehitysympäristössä käytetyt komennot

Docker-compose-ympäristön ajo aloitettiin *sudo docker-compose build*-komennolla. Komennolla instantiointiin docker-compose.yml-tiedostossa määritetyt ohjelmien binaarit ja niille määritellyt konfiguraatiot docker-ympäristön kuviksi. Kaikki Build-komennolla instansoidut Docker-kuvat löytyvät *sudo docker-compose images*-komennolla. Modulaarisuuden nimissä LAMP-ohjelmistopinon ohjelmien Docker-kuvat ja niiden aliohjelmat on

määritetty projektin /bin-hakemistossa dockerfile-tiedostoilla, joista niitä kutsutaan docker-compose.yml-tiedostosta.

```
pi@raspberrypi:~/docker-devops $ sudo docker-compose up -d
Creating network "docker-devops_container-php-network" with driver "bridge"
Creating mariadb_server_instance ... done
Creating php-apache_container    ... done
Creating phpmyadmin_container    ... done
pi@raspberrypi:~/docker-devops $
```

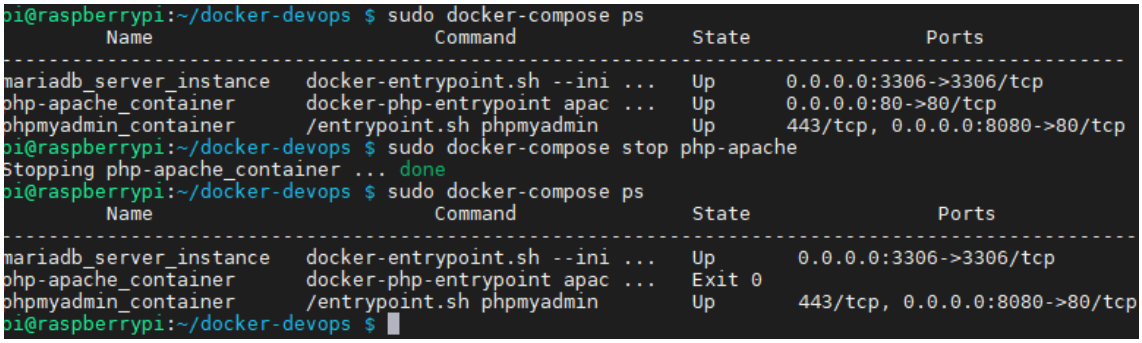
Kuva 6.Konttien ajo docker-compose-ohjelmalla.

Tärkeä osa docker-composen käyttöönottoa Raspbian-käyttöjärjestelmälle on automaattinen testaus. Docker-composea hyödynnettiin työssä määrittämään vaadittavat palvelut testiympäristönä, josta käsin voidaan helposti luoda ja poistaa tilanteen mukaan Dockerissa pyörivät kontit yksittäisillä komennoilla. Esimerkiksi kun palvelut ja niiden aliohjelmat on määritelty build-komennon kanssa, voidaan ne ajaa kontteina *sudo docker-compose up -d* komennolla (Kuva 6).

```
pi@raspberrypi:~/docker-devops $ sudo docker-compose down -v
Stopping php-apache_container    ... done
Stopping phpmyadmin_container    ... done
Stopping mariadb_server_instance ... done
Removing php-apache_container    ... done
Removing phpmyadmin_container    ... done
Removing mariadb_server_instance ... done
Removing network docker-devops_container-php-network
Volume mb_server is external, skipping
pi@raspberrypi:~/docker-devops $
```

Kuva 7. Docker-sovelluskonttien ja voluumien poisto.

Vastaavasti kun testiympäristön kontit halutaan poistaa kokonaan, käytetään sudo docker-compose down -v komentoa (Kuva 7). Komennon loppuun sisällytetty -v sitaatti poistaa kaikki palveluihin määritetyt voluumit, mikä on suositeltavaa, koska normaali down komento ei poista YAML-tiedostossa määritettyjä voluumeja tai network asetuksia. Kun konttien toiminta halutaan keskeyttää, esimerkiksi muutoksia varten, käytetään sudo docker-compose stop-komentoa, joka asettaa kaikki kontit exit-tilaan. Kun Docker-konttien toimintaa halutaan jatkaa, käytetään sudo docker-compose start-komentoa. Yleensä mikäli palvelinympäristöön tehdyt muutokset ovat koskevat ohjelmistojen toiminnan kannalta kriittisiä muutoksia tai ohjelmistojen versiopäivityksiä, on suositeltavampaa käyttää *sudo docker-compose down -v*-komentoa.



Kuva 8. Kuvakaappaus projektissa toteutetusta yksittäisten konttien ajon pysäyttämisestä.

Yksittäisten konttien ajon pysäytys onnistuu sudo docker-compose stop <docker kontti>-komennolla (Kuva 8). DevOpsin mukaisessa tuotantokehityksessä yksittäisten palveluiden toiminnan pysäyttämisen hyötynä on, että sovelluksen tai palvelun päivityksen aikana muut ohjelmat toimivat normaalisti vaikuttaen mahdollisimman vähän käyttäjien toimintaan. Kontin toiminta käynnistetään sudo docker-compose start <docker-kontti>-komennolla.

Docker-compose mahdollistaa myös lokitietojen, konttien tilan ja tietojen tulostamisen helposti komentorivin komentojen avulla. Komento *sudo docker-compose ps* listaa kaikki Docker-ympäristössä pyörivien konttien nimet, Docker-komennot, palveluiden tilat ja portin, josta käsin palvelu on tavoitettavissa (Kuva 8). Saman tekee *sudo docker inspect <container>*-komento, joka tulostaa komentoriviltä käsin kaikki konttiin liittyvät tiedot JSON-muodossa. Lokitietojen selaus onnistuu *sudo docker-compose logs* komennolla, jolla listataan kaikkien konttien ajon aikaiset lokitiedot.

9.9 Palvelinympäristön testauksen menetelmät

Ympäristöä testattiin paikallisesti kannettavalta tietokoneelta SSH-yhteyden avulla. SSH-yhteyden työkaluna käytettiin MobaXterm-nimistä etäverkkotyökalua graafisena käyttöliittymänä etäyhteyden muodostamiseen. SSH-yhteyttä käytettiin pääosin Raspberry Pi-isäntäkoneen kanssa kommunikointiin, mutta sen käytölle oli myös toinen syy. IT-alan organisaatiot, jotka ovat omaksuneet DevOps-toimintamallin käyttävät SSH-yhteyttä kehittäjien keskuudessa ongelmien vianmääritykseen ja ohjelmistojen laadun takaamiseen. Toisin kuin opinnäytetyössä, tuotantoympäristöjä ajetaan yleensä erilaisissa pilvipalveluissa tai virtuaalikoneissa paikallisen isäntäkoneen sijasta. Tämän

vuoksi työssä testattiin myös SSH-yhteyden toimivuutta, jotta LAMP-palvelinympäristöön voidaan muodostaa yhteys vastaavien tuotantosovellusten tapaan.

Työssä testattiin myös itse LAMP-palvelinympäristön toiminta. Selaimesta käsin haettiin paikallisesti kaikki Raspberry Pi-minitietokoneella pyörivät LAMP-ohjelmistopinon palvelut. Esimerkiksi Apachen toimivuus varmistettiin selaimella hakemalla sen oletussivua ja testaamalla staattisen sisällön toimivuutta sijoittamalla HTML-, PHP-ja CSS-tiedostoja Apachen HTML-hakemistoon. MySQL-tietokannan toiminta testattiin mysql-client-komentosarjatyökalun ja PhpMyAdminin kanssa. MySQL-tietokannassa testattiin erityisesti se, että tietokantaan voidaan muodostaa yhteys paikallisessa verkossa, ja että palvelin kykenee kommunikoimaan muiden LAMP-ympäristön ohjelmien kanssa. Docker-konttipalvelun toimintaa testattiin docker-ja docker-compose komennoilla, jotka esitettiin aiemmin edellisessä luvussa.

10 YHTEENVETO

Työn tarkoituksena oli toteuttaa toimiva Lamp-ohjelmistopinon palvelinympäristö Raspberry Pi-isäntäkoneelle seuraten DevOpsin toimintaperiaatteita. DevOps-toimintamallin hyödyt osoitettiin kehitysympäristön toteutuksessa käytetyillä docker-ja docker-compose konttipalveluilla. Docker-konttipalvelu on pitkään ollut ohjelmistokehittäjien suosiossa, koska se tarjoaa helpon kehitysympäristön verrattuna vastaavaan tuotannon virtuaalipalvelimeen. Docker-konttipalvelulla pyrin ensisijaisesti toteuttamaan modulaarisesti yksinkertaisen kehitysympäristön, jossa LAMP-ohjelmistopinon ohjelmistojen binaarit, lokitiedot ja konfigurointi tiedostot asetettiin omiin paikallisiin hakemistoihin. Hakemistorakenteen avulla kehittäjien on helpompi muokata palvelinympäristön toiminnallisuuksia ilman monimutkaisia komentokehotteen komentoja.

Työssä käytettiin docker-compose-ohjelmaa LAMP-ohjelmistopinon ohjelmien kontittamiseen. Ohjelma säveltää LAMP-ohjelmistopinon sovelluskontit yksittäisen docker-compose.yaml-tiedoston avulla. Menetelmä ei ole täydellinen sillä se mahdollisti vain yksittäisellä isäntäkoneella pyörivän palvelinympäristön. Docker-swarm ja Kubernetes klusterointi mahdollistaa palveluiden skaalautuvuuden useamman isäntäkoneen välille, minkä vuoksi nämä ympäristöt olisivat olleet kehitysympäristön toteutuksen kannalta ideaalisempia ratkaisuja kuin docker-compose-ympäristö. Kubernetes klusterointi on docker-swarmia tehokkaampi vaihtoehto, mutta koska kubernetes-ohjelma on tarkoitettu lähinnä suurempiin tuotantokehityksen palveluihin ja työn tarkoituksena oli LAMP-ohjelmistopinon asennus yhteen Raspberry Pi isäntäkoneeseen päädyin työssäni käyttämään docker-compose-ympäristön toteutusta. Devopsin-toimintamalliin täysin perustuvaa kehitysympäristöä oli vaikea toteuttaa yhden ohjelmistokehittäjän projektissa, minkä vuoksi keskityin docker-compose-ympäristössä LAMP-palvelinympäristön käytettävyyteen, muokattavuuteen ja tehokkuuteen liittyviin toimintoihin.

Työssä käytiin läpi perustiedot LAMP-ohjelmistopinon ohjelmista, johon lukeutuu Apache ja MySQL tai MariaDB-relaatiotietokanta. LAMP-ohjelmistopinon tapauksessa työssä käytettiin vielä PHP-ohjelmointikieltä. Lisäksi opinnäytetyössä käytiin läpi oleellisimmat perustiedot Linux pohjaisista järjestelmistä ja sen eroavista jakeluista, johon kuului työssä käytetty Raspian-käyttöjärjestelmä.

Lopputulokseksi saatiin paikallisesti Raspbery Pi-isäntäkoneella toimiva LAMP-palvelinympäristö. LAMP-ohjelmistopinon kehitysympäristössä oli muutama ongelma, joita

työssä olisi voinut parannella. Esimerkiksi työssä käytettiin osittain käyttäjien laatimia docker-kuvia Raspberry PI 4 ARM7l-arkkitehtuurille, koska virallisemmat Docker-pohja-kuvat eivät sopineet kyseiselle ARM-arkkitehtuurille. Työssä tämä ei haitannut, koska opinnäytetyön tavoitteena oli lähinnä antaa esimerkkiä vastaavaan palvelinympäristön toteutuksesta eikä toteuttaa täydellistä DevOpsin mukaista palvelinympäristöä. Työllä kuitenkin onnistuttiin toteuttamaan kotikäyttöön sopiva LAMP-palvelinympäristö Raspberry Pi-isäntäkoneelle.

LÄHTEET

Butler, T. (28. 9 2015). *CloudBees Rollout Blog*. Haettu 21. 3 2020 osoitteesta https://rollout.io/blog/what-is-a-dockerfile/

Docker Hub. (n.d). Haettu 8. 3 2020 osoitteesta https://hub.docker.com/

docker. (n.d). Haettu 14. 3 2020 osoitteesta https://www.docker.com/

Emelianov, M. (10. 7 2017). *opensourceforu*. Haettu 11. 4 2020 osoitteesta https://opensourceforu.com/2017/07/everything-need-know-lamp-stack-importance/

Hernandez, J. (5. 5 2019). *sumo logic*. Haettu 4. 4 2020 osoitteesta https://www.sumologic.com/blog/apache-web-server-introduction/

httpd.apache.org. (n.d). Haettu 19. 4 2020 osoitteesta https://httpd.apache.org/ABOUT_APACHE.html

Ismail, K. (19. 11 2018). *cmswire*. Haettu 29. 3 2020 osoitteesta https://www.cmswire.com/information-management/agile-vs-devops-whats-the-difference/

itewiki. (n.d). Haettu 15. 6 2020 osoitteesta https://www.itewiki.fi/opas/ketterat-menetelmat-agile-lean-ja-scrum/

Linode. (1. 6 2018). *linode*. Haettu 22. 3 2020 osoitteesta https://www.linode.com/docs/applications/containers/how-to-use-dockerfiles/

Linux wiki. (n.d). Haettu 18. 4 2020 osoitteesta https://www.linux.fi/wiki/Etusivu

mariadb. (n.d). Haettu 6. 6 2020 osoitteesta https://mariadb.com/kb/en/about-mariadb-software/

Microsoft Azure. (n.d). Haettu 25. 4 2020 osoitteesta https://azure.microsoft.com/en-us/overview/what-is-devops/#overview

NewRelic. (n.d). *NewRelic*. Haettu 13. 4 2020 osoitteesta https://newrelic.com/devops/what-is-devops

php.net. (n.d). Haettu 7. 4 2020 osoitteesta https://www.php.net/manual/en/intro-whatis.php

planview. (n.d). Haettu 15. 6 2020 osoitteesta https://www.planview.com/resources/guide/agile-methodologies-a-beginners-guide/history-of-agile/

Raspbian.org. (n.d). Haettu 8. 6 2020 osoitteesta https://www.raspbian.org/

refspecs.linuxfoundation.org. (n.d). Haettu 8. 6 2020 osoitteesta https://refspecs.linuxfoundation.org/FHS_3.0/fhs/index.html

Rieuf, E. (16. 12 2016). *datasciencecentral*. Haettu 7. 3 2020 osoitteesta https://www.datasciencecentral.com/profiles/blogs/history-of-mysql

Rouse, M. (6. 2 2020). *Techtarget Search ITOperations*. Haettu 15. 3 2020 osoitteesta https://searchitoperations.techtarget.com/definition/Docker-image

Seshachala, S. (24. 11 2014). *DevOps.com*. Haettu 23. 3 2020 osoitteesta https://devops.com/docker-vs-vms/

Sharma, S. (2017). *The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise.* Indianapolis, IN 46256: John Wiley & Sons, Inc.

Singh, H. (4. 10 2019). *appknox.* Haettu 8. 6 2020 osoitteesta https://www.appknox.com/blog/history-of-devops

Sixsigmadaily. (18. Joulukuu 2017). *Henry Ford and the Roots of Lean Manufacturing*. (Sixsigmadaily) Haettu 27. helmikuu 2020 osoitteesta https://www.sixsigmadaily.com/henry-ford-lean-manufacturing/

techterms. (n.d). Noudettu osoitteesta https://techterms.com/definition/rdbms

tutorialspoint. (n.d). Noudettu osoitteesta https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

Wallen, J. (19. 6 2019). *thenewstack*. Haettu 21. 3 2020 osoitteesta https://thenewstack.io/docker-basics-how-to-use-dockerfiles/

Yegulalp, S. (10. 10 2018). *InfoWorld*. Haettu 28. 3 2020 osoitteesta https://www.infoworld.com/article/3310941/why-you-should-use-docker-and-containers.html

LIITTEET

LIITE 1. docker-compose yaml

```
version: '3'

services:
 php-apache:
  build:
     context: ./bin/php-apache
  container_name: 'php-apache_container'
  restart: 'always'
  expose:
     - 80
  ports:
     - "80:80"
  volumes:
      - ${DOCUMENT_ROOT_DIR}:/var/www/html
      - ${APACHE_LOG_DIR}:/var/log/apache2
      - ${VHOSTS_DIR}:/etc/apache2/sites-enabled
  depends_on:
      - 'mariadb'
  networks:
      - container-php-network
```

```
mariadb:
  build:
     context: ./bin/mariadb
  restart: 'always'
  container_name: 'mariadb_server_instance'
  command: --init-file /data/application/init.sql
  ports:
     - "3306:3306"
  volumes:
    - mb_server:/var/lib/mysql
    - ${MYSQL_DATA_DIR}:/var/lib/mysql
    - ${MYSQL_LOG_DIR}:/var/log/mysql
    - ${MYSQL_CONFIG_DIR}:/data/application/init.sql
  environment:
     TZ: "Europe/Helsinki"
     MYSQL_ALLOW_EMPTY_PASSWORD: "no"
     MYSQL_ROOT_PASSWORD: "root"
     MYSQL_USER: "lampdev"
     MYSQL_PASSWORD: "initlamp_pass"
  networks:
    - container-php-network
```

```
phpmyadmin:
  image: ebspace/armhf-phpmyadmin:latest
    container_name: 'phpmyadmin_container'
    volumes:
      - ${PHPMYADMIN_CONFIG_DIR}:/etc/phpmyadmin/config.user.inc.php
    environment:
      PMA_ARBITRARY: 1
      PMA_HOST: mariadb
      PMA_PORT: 3306
      PMA_USER: "root"
      PMA_PASSWORD: "secret"
    ports:
     - "8080:80"
    depends_on:
    - mariadb
    networks:
    - container-php-network
networks:
   container-php-network:
     driver: bridge
volumes:
   mb_server:
     external: true
```

Liite 2. Dockerfile tiedostot

#MariaDB Dockerfile

```
FROM jsurf/rpi-mariadb:latest
```

Dockerfile php-apache

```
FROM php:7.4.2-apache

# Update
RUN apt-get -y update --fix-missing && \
    apt-get upgrade -y && \
    apt-get --no-install-recommends install -y apt-utils && \
    rm -rf /var/lib/apt/lists/*

# Install useful tools and install important libaries
RUN apt-get -y update && \
    apt-get -y --no-install-recommends install nano wget \
dialog \
libsqlite3-dev \
libsqlite3-0 && \
    apt-get -y --no-install-recommends install default-mysql-client \
zlib1g-dev \
libzip-dev \
libicu-dev && \
    apt-get -y --no-install-recommends install --fix-missing apt-utils
\
build-essential \
git \
curl \
libonig-dev && \
    apt-get -y --no-install-recommends install --fix-missing libcurl4 \
libcurl4-openssl-dev \
zip \
openssl && \
    rm -rf /var/lib/apt/lists/* && \
    curl -sS https://getcomposer.org/installer | php -- --install-
dir=/usr/local/bin --filename=composer


# Other PHP7 Extensions

RUN docker-php-ext-install pdo_mysql && \
    docker-php-ext-install pdo_sqlite && \
    docker-php-ext-install mysqli && \
    docker-php-ext-install curl && \
    docker-php-ext-install tokenizer && \
    docker-php-ext-install json && \
    docker-php-ext-install zip && \
    docker-php-ext-install -j$(nproc) intl && \
    docker-php-ext-install mbstring && \
    docker-php-ext-install gettext

#Environment variables to configure php
ENV PHP_UPLOAD_MAX_FILESIZE 10M
ENV PHP_POST_MAX_SIZE 10M
```

```
# Enable apache modules
RUN a2enmod rewrite headers

# Cleanup
RUN rm -rf /usr/src/*
```

Liite 3. ympäristömuuttujat

```
DOCUMENT_ROOT_DIR=./www

APACHE_LOG_DIR=./doc/log/apache

MYSQL_DATA_DIR=./doc/data/mysql

MYSQL_LOG_DIR=./doc/log/mysql

MYSQL_CONFIG_DIR=./config/mysql/setup.sql

PHPMYADMIN_CONFIG_DIR=./config/phpmyadmin/config.user.inc.php

VHOSTS_DIR=./config/vhost


PMA_CUSTOM_CONFIG=:/config/phpmyadmin/config.user.inc.php
```