TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Erki Aas 175960IDAR

VEEBIRAKENDUSTE SERVEERIMINE KONTEINERTEHNOLOOGIATE ABIL

Diplomitöö

Juhendaja: Tõnis Palts
Bakalaureus

Tallinn 2020

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Erki Aas

28.04.2020

Annotatsioon

Diplomitöö keskendub veebirakenduste serveerimisele konteinertehnoloogiate abil. Töö eesmärkideks on luua mitmest peremeesmasinast koosnev, konteinertehnoloogiatele põhinev majutuskeskkond ühe ettevõtte veebirakenduse kõrgkäideldavaks majutamiseks selle ettevõtte etteantud tingimustel. Teine eesmärk on anda ülevaade ja suunitlused autori töökoha Maagiline Vedur OÜ edasisteks võimalusteks veebirakenduste majutamiseks konteinertehnoloogiate abil, sealhulgas selgitada välja, kas ja kuidas peaks muutma rakendusi ning nende arendusprotsesse.

Diplomitöös antakse ülevaade konteinertehnoloogiatest üldiselt, teoreetilises osas võrreldakse kahte enimlevinud isemajutatavat konteinerite orkestratsiooniplatvormi Kubernetest ja Docker Swarmi, valitakse välja platvorm lähteülesande täitmiseks, vajadusel viiakse sisse muudatusi arendusprotsessides ja rakendustes, täidetakse eesmärk luua veebirakenduse majutuskeskkond konteinertehnoloogiate abil ning analüüsitakse Maagilise Veduri edasisi võimalusi veebirakenduste majutamiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 44 leheküljel, 4 peatükki, 2 joonist, 1 tabelit.

Abstract

Serving web applications using container technologies

This thesis will concentrate on serving web applications using container technologies. Thesis has two goals. The first goal is to create a multi host and highly availalble hosting environment for a web applications of a company using container technologies while complying with the terms given by this company. The second goal is to give an overview and instructions about possibilities and requirements to serve web applications using container technologies in author's workplace Maagiline Vedur OÜ, including finding out, whether applications and their development process should be changed.

The thesis will give an overview of container technologies in general, theoretical part will compare the two most common self-hosted container orchestration platforms - Kubernetes and Docker Swarm, of which one will be selected to fulfil the first goal. If needed, changes in applications and their development process will be introduced, after which the first goal will be fulfilled by creating a solution. To fulfill the second goal, an analysis will be given, which options and container technologies Maagiline Vedur can use to deploy upcoming projects.

The thesis is in Estonian and contains 44 pages of text, 4 chapters, 2 figures, 1 table.

Lühendite ja mõistete sõnastik

Konteinertehnoloogia	Tehnoloogia operatsioonisüsteemi tasemel virtualiseerimiseks, kus iga käideldav rakendus töötab iseseisvas ja isoleeritud keskkonnas ehk konteineris.
Orkestratsiooniplatvorm	Platvorm konteinerite käitlemiseks mitme peremehe peal samaaegselt ja hallatult.
Dockeri tõmmis	Kogumik faile, mis sisaldavad konteineri tööks kasutatavat tarkvara, ingl. *Docker image*.
PaaS	Teenus, kus teenusepakkuja haldab ja uuendab pakutavat (konteinertehnoloogia) platvormi kliendile kasutamiseks, ingl. P*latform as a Service.*
Serverless	Teenus, kus teenusepakkuja pakub teenust mingi rakenduse käitlemiseks, sealjuures teenuse skaleerimine on täiesti automaatne.
FaaS	Teenus, kus teenusepakkuja pakub võimalust mingi funktsiooni kasutamiseks, näiteks serverless teenuse käitamiseks, ingl. *Function as a Service*.
Tagarakendus	Veebirakenduse dünaamilisi ja loogilisi osasid teenindav komponent, ingl. *back-end*.
Eesrakendus	Veebirakenduse kasutajele nähtav kasutajaliides, mis võimaldab veebisaidi külastajal kahepoolselt suhelda saidi tagarakendusega, ingl. *front-end*.
Teenuse replikeerimine	Mõiste konteineriplatvormides, kus iga teenus hoitakse alati töös ettemääratud arvus eksemplarides, ingl. *replication*.
Automaatne sklaleeritavus	Konteinerplatvormide võimekus käivatada teenusest lisaeksemplare suure koormusega hakkama saamiseks.
Horisontaalne skaleeritavus	Teenuse koormustaluvuse tõstmine selle lisaeksemplaride lisamise teel, ingl. *horisontal scaling*.
Pod	Komplekt (Dockeri) konteineritest Kubernetese platvormil, mis on omavahel tihedalt seotud, saavad omavahel suhelda ning moodustavad koos mingi komponendi või teenuse.

CI/CD	Ingl. *Continuous Integration and Continuous Delivery*. Automatiseeritud protsess tarkvara ehitamiseks ja integreerimiseks olemasoleva koodibaasiga ning tarkvara automaatseks juurutamiseks.
Dockerfile	Fail, mis defineerib, kuidas luuakse Dockeri tõmmis.
Klaster	Kogum arvuteid või teenuseid mis töötavad tihedalt koos, võimaldades neid vaadelda ühe tervikuna, ingl. *cluster*.
Split-brain	*Split-brain* on olukord, kus mitmest eksemplarist koosnev süsteem ei suuda veaolukorras otsustada klastri õiget ning terviklikku seisukorda, võides põhjustada andmekadu ja anomaaliaid.

Sisukord

Jooniste loetelu

Tabelite loetelu

Sissejuhatus

Veebirakenduste kasvav kompleksus toob kaasa ka suurenenud nõuded selle rakenduse töökeskkonnale. Ühtlasi pole kuskile kadunud ka potentsiaalsed probleemid käideldavusega, mis võivad olla tingitud suurest koormusest, serveri tõrgetest või rakenduse mittekäideldavusest selle uuendamise ajal.

Neid probleeme aitab lahendada rakenduse majutamine konteinertehnoloogiate abil, mida kasutades luuakse iga rakenduse komponendi jaoks eraldiseisev ning isoleeritud keskkond ehk konteiner, kus on olemas just selle komponendi toimimiseks vajalikud tarkvarakomponendid ning mis võimaldavad liiasuse (ingl. redundancy) abil tagada kõrgkäideldavust, eeldusel, et rakendus on vastavalt arendatud.

Käesolev diplomitöö uurib võimalusi mitmest komponendist koosnevate veebirakenduste majutamiseks konteinerite ja nende orkestratsiooniplatvormide abil.

Põhirõhk on kahe suurima orkestratsiooniplatvormi Kubernetese ja Docker Swarm'i võrdlusel ning implementeerimisel. Analüüsitakse ja vajadusel muudetakse ka arendusprotsesse.

Diplomitöö eesmärk on luua toimiv lahendus ühe kindla teenuse majutamiseks kliendi poolt valitud keskkonnas. Protsessi käigus arvestatakse juba olemasolevaid tööriistu ning tulevase lahenduse potentsiaalseid murekohti, võimalikke eeliseid ja alternatiivseid lahendusi, seejuures luuakse suurem plaan toetamaks autori töökoha Maagiline Vedur OÜ edasisi valikuid rakenduste majutuse osas ning võimalusi tarkvara elutsükli täiendamiseks.

1 Konteinertehnoloogiatest üldiselt

Konteineritel põhinevate tehnoloogiate tuntuim platvorm on aastal 2013 alguse saanud Docker, mis on tänaseks kujunenud *de facto* valikuks rakenduste arendamisel ning majutamisel. [1]

Docker on tarkvara, millega saab luua, hallata ning uuendada konteinereid, mis on loodud kas teatud liiki tarkvara käitlemiseks (näiteks Apache konteinerid PHP rakenduste jaoks) või mõne tarkvarapaketi käitlemiseks tervikuna (näiteks Nextcloud'i rakenduse konteiner). [2]

Docker üksi annab tööriistad konteinerite loomiseks, tööshoidmiseks ning erinevate ühenduste (kataloogide haakimine, võrguühenduste loomine jne) loomiseks ühe peremehe (ingl. *host*) tasemel. Dockeri baasil on loodud hulk rakendusi, mis haldavad mitmeid Dockeri eksemplare (ingl. *instance*) samaaegselt selleks, et luua parem käideldavus ja skaleeritavus. Selliseid platvorme kustutakse orkestreerimisplatvormideks ning kaks levinumat neist on Docker Swarm ja Kubernetes. Lisaks kuuluvad Dockeri ökosüsteemi veel teised toetavad rakendused, näiteks Docker Registry, mis on privaatrepositoorium asutusesiseste Dockeri tõmmiste (ingl. *image*) hoidmiseks ja levitamiseks. Docker Registry baasil on loodud ka avalik ja vabalt kasutatav Docker Hub, mis sisaldab avalikke, üldlevinud ning avatud lähtekoodiga tarkvaradel põhinevaid tõmmiseid.

Docker on oma populaarsuse saavutanud, sest see teeb nii arendajatate kui administraatorite töö lihtsamaks. Pakendades rakenduse ja selle töötamiseks vajaliku eeldustarkvara (ingl. *dependency*) ühte paketti, muutub nii uute arenduskeskkondade kui toodangukeskkondade ülesseadmine palju lihtsamaks ja veakindlamaks. Ühtlasi on paljud Dockerit kasutavad platvormid saadaval nii pilveteenuste pakkujate poolt PaaSina ent samal ajal on ka täielikult isemajutatavad, pakkudes ühilduvust suurkorporatsioonide ning riigiasutuste kõrgete turvastandarditega. Näiteks on Dockeril ühilduvus USA riigisektoris nõutava FIPS 140-2 krüptograafia standardiga. [3]

1.1 Serverless tehnoloogiad

Konteinertehnoloogiate järglaseks ja konkurendiks on *serverless* tehnoloogia, mis pakub täielikult iseskaleeruvat ja platvormi mõttes haldusvaba keskkonda rakenduste käitlemiseks. [4] Üks näide sellisest teenusest on Amazon Web Services Lambda.

Serverless teenust liigitatakse FaaS (*Function as a Service*) teenuseks, mis on pilvetehnoloogiate arengus ühe sammu võrra kaugemal kui näiteks teenusepakkuja poolt hallatud Kubernetese keskkond, mida liigitatakse PaaS (*Platform as a Service*) teenuseks.

Kuigi serveless tehnoloogial on omad eelised suure ja muretu automaatse skaleeruvuse näol, ning seda on üritatud kasutada ka klassikaliste kliendipoolsete veebirakenduste implementeerimiseks, ei ole see mõeldud olema kogu aeg ootevalmis ning on pigem sobilik suuremahuliste taustaprotsesside käitlemiseks. [5] Ühtlasi on need platvormid täna üpris keskkonnapõhised, kuigi Knative platvorm lubab seda parandada, andes univeraalse platvormi *serverless* rakenduste käitlemiseks Kubernetese orkestratsiooniplatvormi peal. [6]

1.2 Konteinertehnoloogiate eelised

Konteinertehnoloogiate kasutamiseks on mitmeid eeliseid ja põhjuseid, ent ka neil on ka puudujääke, millega tasuks uue arhitektuuri planeerimisel ja kasutamisel arvestada.

1.2.1 Rakenduse keskkonna ühtsus

Docker pakendab rakenduse (komponendi) ja selle tööks vajaliku eeldustarkvara ühte tervikusse. Selline tervik, ehk Dockeri konteiner, töötab samamoodi kõikide arendajate arvutites kui ka hiljem toodanguserverites, vähendades ajakulu ja ressursse rakenduse töötamiskeskkonnast tingitud probleemide lahendamisel. [2]

1.2.2 Turvalisus jagatud keskkonnas

Kui iga rakenduse komponent töötab eraldi isoleeritud keskkonnas, on just selle komponendi turvaaukudest tingitud riskid madalamad. Kui pahatahtlik käituja peaks saama ligipääsu ühe komponendi kaudu kogu selle keskkonnale, on tal võimalik tegutseda ainult konteineri piires ning mõjutada teisi rakenduse komponente ainult niipalju, kui selleks rakendusel endal normaaltingimustel võimalusi on. Selline lähenemine on turvalisem, kui erinevate komponentide töötamine sama operatsioonisüsteemi sees ilma isolatsioonita. [7]

1.2.3 Jõudlus võrreldes mitme virtuaalserveriga

Isolatsiooni ühe füüsilise serveri tasemel saaks ka saavutada kasutades iga komponendi jaoks täiesti eraldi virtuaalset serverit, ent sellisel juhul oleks iga komponendi toimimiseks vaja palju rohkem ressursse ning taastusaeg ühe komponendi (täielikuks) taaskäivitamiseks suurem. [2]

Konteinerid jagavad sama peremeesmasina kernelit, ning seetõttu on nende isoleerimisest tingitud jõudluskadu ning (konteineri) käivitusaeg väiksed.

1.2.4 Suurem käideldavus

Dockerit kasutades on võimalik kasutada orkestratsiooniplatvorme, mis suurendavad käideldavust rakenduse komponendi tasemel. Selline lahendus on paindlikum kui kõrgkäideldavuse loomine ainult rakenduse tasemel ja pakub rohkem jõudlust, kui virtuaalmasinate kõrgkäideldavus hüperviisorite tasemel. Loomulikult peab olema kõrgkäideldavus implementeeritud kõikides infrastruktuuri kihtides, ent rakenduse jõudluse seisukohas on parim just mitme rakenduse eksemplari (ingl. *instance*) samaaegne töötamine, mille vahel on koormus ära jaotatud ning mida on võimalik ükshaaval uuendada. [8]

1.2.5 Hõlpsam tarkvara elutsükli haldus

Rakenduste juurutamine Dockeri abil on aluseks paljudele automatiseeritud protsessidele tarkvaraarenduse elutsükli (ingl. *Software Development Life Cycle*) juures,

nagu näiteks automaattestid ja juurutamine erinevatesse majutuskeskkondadesse. [8] Kuna ühe Dockeri konteineri uuendamine ja käivitamine töötab samamoodi igas keskkonnas, on võimalik neid ehitada, juurutada ja testida rakenduse eripäradest sõltumata.

1.3 Konteinertehnoloogiate puudused

1.3.1 Väiksem turvalisus

Kuigi Docker on arendatud turvalisust silmas pidades, on selle turvariskid siiski olemas, võrreldes näiteks eraldi virtuaalmasinate kasutamisega. Dockeri konteineri turvalisus sõltub Dockeri tarkvara ja selle kasutatava peremeesmasina kerneli turvauuendustest, samuti konteineri turvasätetest ja implementatsioonist. Tehniliselt on võimalik rakenduse kaudu konteinerile ligipääsedes saada omakorda turvaaukude kaudu ligipääs kogu peremeesmasina operatsioonisüsteemile, olgu selleks füüsilise või virtuaalserveri operatsioonisüsteem. Turvariskide maandamiseks tuleb hoida peremeesmasin pidevalt uuendatud ning Dockeri konteinereid luues pidada silmas turvalisuspraktikaid, näiteks tuleb rakenduse käitamiseks luua selleks eraldi loodud kasutaja, mitte kasutada juurkasutajat. [9]

1.3.2 Väiksem jõudlus sama operatsioonisüsteemi tasemel

Konteinerite sees peab alati töötama minimaalne komplekt tarkvara (süsteemiutiliidid, rakendustarkvara), mida erinevalt kernelist ei duplitseerita, vaid mis luuakse iga konteineri sees eraldi. Lisaks peab konteineris olev tarkvara suhtlemisel teiste komponentidega läbima Dockeri võrgukihti. Nendest ning teistest isoleerimisest tingitud põhjustest on rakenduste jõudlus madalam võrreldes ilma isolatsioonita töötamisest. [10]

1.3.3 Haldus ja treening

Nii nagu iga tehnoloogiaga, on Dockeri kasutamiseks vajalikud eelteadmised, seda nii arendajatel kui administraatoritel. Dockeri kasutamine loob vahele vajaliku, ent

harjumist vajava lisakihi võrreldes näiteks rakenduse paigaldamise jagatud majutuskeskkonda (shared hosting) või Linuxi virtuaalserverisse.

Üksikute Dockeri eksmplaride (ingl. *instance*) haldamine pole võibolla just kõige keerulisem ja aeganõudvam protsess, ent produktsioonivalmis, turvaliste ja kõrgkäideldavate keskkonade loomine ning haldamine on aga aega ja teadmisi nõudev protsess, mis võib vajada täiendavaid koolitusi ja inimressurssi, samuti vajab täiendavaid resursse arendusprotsessi täiendamine Dockeri ning CI / CD (ingl. c*ontinuous integration and continuous delivery*) lahenduste loomise näol. [11]

2 Lähtetingimused

2.1 Etteantud majutuskeskkond

Käesoleva diplomitöö esimene eesmärk on luua konteineritel põhinev ning kõrgkäideldav majutuskeskkond ühe ettevõtte veebirakenduse majutamiseks. Teenus koosneb tüüpilistest komponentidest, mida Maagiline Vedur OÜ erinevates rakendustes kasutab - Laravel (PHP raamistik) tagarakendus (ingl. *back-end*), Vue.js (Javascripti raamistik) eesrakendused (ingl. *front-end*) ning MySQL andmebaas. Antud projekti juurde kuulub ka üks Node.js mikroteenus PDF-vormingus arvete genereerimiseks. Lisaks kasutab teenus Nginx puhverserverit (ingl. *proxy*) päringute edastamiseks erinevatele komponentidele ning Matomo analüütikarakendust.

Klient soovib, et rakendust majutatakse Tartu Ülikooli serverikeskkonnas virtuaalserverites. Toodanguks on eraldatud 3 serverit. Lisaks pakub majutuse osutaja nende poolt hallatud kõrgkäideldavat Percona (MySQL/MariaDB-l põhinev kõrgkäideldav andmebaasimootor) andmebaasi, serverite vahel jagatud kettapinda ning HaProxy koormushaldurit. Lahendus luuakse just Tartu Ülikooli serverikeskkonda, sest kliendiks oleval ettevõttel on isiklik seos selle keskkonnaga ning seetõttu usaldab just seda keskkonda oma tundlike andmete hoidmiseks ja käitlemiseks. Maagilisel Veduril serverikeskkonna valikul kaasa rääkida polnud võimalik.

Seega on suur osa kõrgkäideldavusest juba olemas ning töö autori ülesanne oleks luua lahendus rakenduse enda kõrgkäideldavuse loomiseks kolme serveri abil ning arendusprotsesside läbiviimiseks sellel keskkonnas, sealhulgas leida viis uute versioonide automaatseks juurutamiseks.

Halduskoormus on jagatud Maagilise Veduri (konteineritel põhineva platvormi haldamine ning rakenduse juurutamine sellele platvormile) ning Tartu Ülikooli (kogu ülejäänud infrastruktuur) vahel.

2.2 Teiste rakenduste majutuskeskkonnad

Teiseks käesoleva diplomitöö eesmärgiks on leida üldiselt optimaalseim orkestratsiooniplatvorm, mida saaks Maagiline Vedur tulevaste ja olemasolevate projektide majutamisel kasutusse võtta.

Enamik Maaglise Veduri hallatavaid toodangukeskkondasid asuvad avalikus pilves asuvates virtuaalserverites. On ka projekte, mis on majutatud klientide privaatsetes pilvedes asuvates virtuaalserverites. Osa projekte on ka klassikalises veebimajutuses (ingl. *shared hosting*).

Kasutusele võetav orkestratsiooniplatvorm peaks olema ühilduv enamike avalikus pilves asuvate virtuaalserveritega ning nende juurde kuuluvate võimalustega, nagu näiteks virtuaalserverite vaheline privaatvõrk ning jagatud kettapind (ingl. *shared storage*).

Pilveteenuse pakkujate poolt pakutavate PaaSide (näiteks *Managed Kubernetes on DigitalOcean* ja *Amazon Elastic Kubernetes Service*) olemasolu valitava platvormi juures ei saa olema kindel nõue, ent nende valik antud platvormile on platvormide võrdluses sees.

2.3 Olemasolevad lahendused

Hetkel kasutatakse Maagilises Veduris Docker Compose'i, mis on Docker Inc. poolt loodud tööriist. Docker Compose võimaldab seadistusfailide abil luua ja seadistada Dockeri konteinerikeskkondasid ühe peremeesmasina raames. Docker Compose'i seadistusfaile hallatakse Giti versioonikontrolli abil, võimaldades kiiresti asuda arendama üksteise alustatud ja vahel keskkonna mõttes erivajadustega veebirakendust. Toodangukeskkonnad on praeguse seisuga enamasti siiski vastavalt vajadustele seadistatud virtuaalserverid ja veebimajutused.

Ideaalis võiks konteinerplatvormi kasutamine majutamisel kasutada võimalikult palju juba tehtud tööd Docker Compose projektide näol ning nõuda võimalikult vähe rakenduste kohaldamist.

2.4 Rakendusespetsiifilised tingimused

Enamik Maagilise veebilahendusi koosnevad kolmest komponendist:

- Eesrakendus ehk *front-end* (enamasti kasutatakse Javascripti raamistikku Vue.js)
- Tagarakendus ehk *back-end* (kasutatakse PHP raamistikku Laravel)
- Andmebaas (enamasti kasutatakse andmebaasimootorit MySQL)

Iseseadistatud keskkondades kasutatakse veebiserveri ja puhverserverina Nginx'i koos PHP-FPM mootoriga PHP rakenduste käitlemiseks.

Tüüpiliselt rakendus töötab järgmiselt:

1. Klient avab veebilehe. Eesrakenduse teenusest laetakse talle baasleht ning brauseris töötav Javascripti pakett lehele kasutajapoolse dünaamilisuse andmiseks. Eesrakendust ei laeta staatiliselt, sest osad lehed on otsingumootorite tarbeks tagarakendusest laetud andmetega juba eelvisualiseeritud (ingl. *pre-renderred*).

2. Brauseris töötav Javascripti pakett alustab vajadusel suhtlust tagarakendusega, kusjuures igale kliendile antakse tagarakenduse poolt unikaalne küpsis, mis tuleb kliendil järgmise päringuga kaasa anda, saades uue päringuga omakorda tagasi uue küpsise. Selline küpsiste vahetamine käib suhtluse lõpuni.

3. Sõltuvalt päringust, loetakse või kirjutatakse midagi andmebaasi, või loetakse serveri kettal olevat faili ning tagastatakse selle sisu (näiteks PDF failid).

4. Tagarakendus tagastab korrektse informatsiooni tagasi kliendile.

Selline töövoog loob loodavale mitmest eksemplarist koosnevale rakendusele järgmised tingimused:

1. Küpsised peavad olema erinevate tagarakenduste vahel ühilduvad.

2. Andmebaasi ajakohasus kõigis tagarakenduse eksemplarides on iga päringu ajal äärmiselt kriitiline.

3. Juhul kui kliendile tagastatakse mõni tagarakenduse kaudu laetav fail, peavad failid olema kõigi tagarakenduste jaoks võrdselt kättesaadavad.

Vaikimisi kasutavad Maagilise Veduri loodud tagarakendused sessioonide hoidmiseks failisüsteemi. Erinevate tagarakenduste vahel peaks seega olema seadistatud ühine kettapind või küpsiste hoidmine peaks toimuma andmebaasis.

MySQL vaikimisi ei võimalda mitme juhtserveriga toimivat andmebaasi, küll aga on olemas võimalus luua lisaeksemplare andmebaasist andmete lugemiseks. [12]

Selleks, et andmebaas oleks kõrgkäideldav, tuleks kasutada kas pilveteenuse pakutavat kõrgkäideldavat andmebaasi või andmebaas ise paigaldades valida andmebaasimootor selliselt, et see oleks ühtaegu ühilduv olemasolevates MySQL andmebaasides olevate andmetega ning toetaks sellist mitme eksemplariga, Dockeri baasil loodud platvormi.

Juhul kui tagarakendus tegeleb ka failide serveerimisega, peaksid failid asuma kõigi tagarakenduse eksemplaride poolt võrdselt kättesaadaval andmepinnal, näiteks jagatud võrgukettal.

3 Haldusplatvormide võrdlus

3.1 Sissejuhatus

3.1.1 Metoodika

Käesolevas peatükis võrreldakse kahte suuremat mitmest Dockeri eksemplarist koosnevate klastrite käitlemiseks mõeldud orkestratsiooniplatvormi: Docker Swarmi ja Kubernetest. Valikusse võeti just Kubernetes ja Docker Swarm, sest need on täielikult isemajutatavad, hästi dokumenteeritud, suure kommuuniga ning avatud lähtekoodiga. Mõlemad teenused on laialt tuntud, mistõttu autori arvates on nende haldamiseks vajalikud teadmised olemas suuremal hulgal potentsiaalsetel haldajatel võrreldes mõne vähemtuntud konteinerite orkestratsiooniplatvormiga.

Valitud platvorme võrreldakse järgmiste omaduste alusel:

- Kõrgkäideldavuse omadused
- Ülesseadmise keerukus
- Halduskoormus
- Nõudmised infrastruktuurile
- Pilveteenuste saadavus platvormile
- Ühilduvus Docker Compose’ga

Neid omadusi võrreldakse mõlema platvormi puhul ning koostatakse resolutsioon, kumba platvormi kliendi lahenduse loomiseks kasutatakse. Lisaks selgitatakse välja üldine suund rakenduste majutamisel, mille poole Maagiline Vedur uute lahenduste loomisel püüdlema hakkab ning millist orkestratsiooniplatvormi seejuures kasutatakse.

3.1.2 Docker Swarm

Docker Swarm on Docker Inc. poolt loodud konteinerite orkestratsiooniplatvorm. Docker Swarm alustas eraldiseisva tarkvarana, ent alates Docker versioon 1.12st on Docker Swarmi funktsionaalsus Dockeri põhimootorisse integreeritud [13], muutes selle kasutamise väga lihtsaks - Docker Swarmi ülesseadmiseks pole otseselt vaja ühtegi lisatarkvara peale Dockeri enda. Aastal 2019 omandas Mirantis Docker Enterprise'i - Docker Inci ärilise poole, mis pakub teenuseid ja kasutajatuge seda vajavatele firmadele ja asutustele. [14] Docker Inc jäi eraldiseisvaks firmaks, kes arendab platvorme Docker Hub ja Docker Desktop, ning kes koos Mirantise ja paljude teiste arendajatega panustavad edasi Dockeri baastarkvara ehk Docker Engine edasiarendamisesse. Kuigi Mirantise äri on muuhulgas suunatud hallatud Kubernetese teenuse pakkumisele, teatasid nad oma kavatsustest ka Docker Swarmi edasi arendada ning sellele kasutajatuge pakkuda. [15]

3.1.3 Kubernetes

Kubernetes on Google poolt loodud avatud lähtekoodiga orkestratsiooniplatvorm. Kubernetes sai alguse Google enda vajadustest, selle lähtekood muudeti avalikuks 2014 ning selle omandiõigused annetati 2015 aastal Cloud Native Computing Foundation organisatsioonile [16], kuhu kuuluvad mitmed Kubernetesega tegelevad ettevõtted ja tehnoloogiahiiud, sealhulgas Google, Amazon, Microsoft ja IBM. [17]

Kubernetes on orkestratsiooniplatvormide *de-facto* standard. See on hetkel kaugelt enim levinud platvorm konteineritele põhineva teenuse käitlemiseks. Kubernetese abil on ehitatud teisi, rohkem komplektseid platvorme (näiteks OpenShift) [18] ning paljudel suurtel pilveteenuse pakkujatel on olemas teenusepakkuja poolt hallatud Kubernetese platvormi teenus (PaaS). Kubernetes pakub mitmeid sisseehitatud võimalusi teenuse kvaliteedi tõstmiseks. Sellised võimalused on näiteks automaatne teenuse replikeerimine ühe selle eksemplari vea korral, automaatne skaleerimine ning sisseehitatud logihaldus. [11] Kuna Kubernetes on väga võimekas ja mitmest komponendist koosnev, on selle käsitsi ülesseadmine ning edasine haldus üldjuhul keerulised ning seetõttu nõuab arvestatavalt ressursse. Alati pole võimalik ka pilveteenuste pakkujate PaaSe kasutada. Tugi isemajutatud Kubernetesele või sellele loodud komplektplatvormidele võivad aga kulukaks osutuda.

3.2 Kõrgkäideldavuse omadused

Nii Kubernetes kui Docker Swarm on ehitatud kõrgkäideldavust silmas pidades. Mõlemad pakuvad automaatset teenuste taaskäivitamist selle vea korral, vajadusel teise peremeesmasina (ingl. *node*) peale kolimist, toetavad koormuse jaotamist (ingl. *load balancing*), hõlpsat horisontaalset skaleerimist ning mitme juhtserveriga topoloogiat.

3.2.1 Juhtserveri kõrgkäideldavus

Kubernetese juhtserverid koosnevad mitmest komponendist - loogikaga tegelevatest komponentidest *apiserver*, *controller-manager* ning *scheduler* ja andmebaasimootorist *etcd*. Kubernetese haldustööriista *kubeadm*-ga loodud klaster vaikimisi integreerib *etcd* koos teiste juhtserveri komponentidega ühte *pod'i* ehk rakenduskomplekti. Altenatiivselt on võimalik *etcd* komponent eraldada eraldi *pod'i* ning tekitada selle komponendi liiasus teisel tasemel, muutes süsteemi tervikuna töökindlamaks, ent keerulisemaks. [19]

Mõlemal juhul kasutatakse kõigi juhtserverite ees koormushaldurit, ka klastri töölisserverite (worker nodes) ja juhtserverite vahelises suhtluses. Koormushaldur tuleb seadistada eraldi. Selleks võib näiteks kasutada Nginxi või HaProxyt. [20]

Kubernetese kasutamiseks toodangus on äärmiselt soovitatav kasutada vähemalt kolme juhtserverit. Nii jääb ühe juhtserveri hävimisel alles ⅔ ehk enamus (üle 50%) juhtserveritest alles, mis võimaldab ülejäänud klastril otsutada tervikliku oleku ning seeläbi klastril tegevust jätkata. [19]

Docker Swarm kasutab samuti juhtservereid (ingl. *manager nodes*) ning töölisservereid (ingl. *worker nodes*). [21] Docker Swarmi puhul on kogu juhtloogika integreeritud Dockeri mootorisse endasse. Kõrgkäideldavuse tagamiseks on vaja mitut juhtserverit, soovitatavalt vähemalt kolme *split-brain* situatsiooni vältimiseks. Docker Swarmi puhul on ühteaegu aktiivne vaid üks juhtserver. Selle hävinemisel valib ülejäänud võrgustik ise, kasutades Raft protokolli, milline ootel olevatest varujuhtserveritest peamise juhtserveri rolli üle võtab, ning mille poole teised klastri liikmed pöörduma hakkavad. [22]

Nii Kubernetese kui Docker Swarmi puhul saab ning on toodangus vajalik luua mitu juhtserverit. Docker Swarmi puhul on juhtserverite lisamine lihtsam, sest see kasutab juhtserveri loomiseks vähem komponente ning seeläbi on ülesseadmine kiirem.

3.2.2 Teenuste kõrgkäideldavus

Kuberneteses kasutatakse teenuste käitlemiseks *pod'e* - komplekte (Dockeri) konteineritest, mis on omavahel tihedalt seotud, saavad omavahel suhelda ning moodustavad koos mingi komponendi. *Pod'idele* saab määrata, millises Kubernetese töölisserveris need töötavad ning *ReplicaSet* parameetri abil kui palju samasid pod'e üle klastri alati töös peab olema. Erinevad *podid* üle klastri seotakse kokku teenuse ehk *service* abil. Kubernetese *service'le* omistatakse IP aadress koos teenusele vastavate portidega, mille kaudu suunatakse päring edasi mõnda seda teenindavasse *pod'i. Service'i* IP aadress on ligipääsetav ainult klastri seest. Ühe *service* sees võib paikneda mitu *pod'i*, mida eristatakse *service* poole pöördudes portide ja päringu URL'i abil. Lisaks on vajalik ka *Ingress*, mis kasutades mõnda puhverserveri tarkvara seob avalikud URLid mõne kindla *service* poolt pakutava teenusega ning haldab koormust *service'i* eksemplaride vahel. [23]

Docker Swarmi puhul tagatakse teenuse kõrgkäideldavus seda teenindava konteineri tasemel *replicas* parameetri abil. Vastavalt *replica* parameetrile luuakse kogu klastri peale kokku just vastav number konteinereid, mis on ligipääsetavad iga klastri liikme vastavalt avalikult pordilt. [21] Docker Swarm'i *ingress* võrk kannab hoolt selle eest, et klastri liikme pordile tulnud ühendus suunatakse samas või teises peremeesmasinas asuva teenuse konteinerile. *Ingress* võrk haldab ka koormust nende konteinerite vahel. Väliste päringute teenindamiseks on võimalik kasutada mitmest avalikust IPst koosnevaid DNS kirjeid (ingl. *DNS Round Robin*) ent soovitatav on siiski kasutada eraldi koormushaldurit (ingl. *load balancer*), mis võib olla näiteks HaProxy, Nginx või mõni pilveteenuse pakkuja poolt hallatud koormushaldur. [24]

Etteantud arvu teenust pakkuvate konteinerite olemasolek on mõlema orkestreerimisplatvormi puhul tagatud, eeldusel, et on tagatud juhtserverite liiasus. See, kuidas kliendi päring teenust pakkuva konteinerini jõuab on aga Kubernetese ja Docker

Swarmi puhul küllaltki erinev - Kubernetesel on selleks rohkem vahekihte, mis annavad rohkem paindlikkust, ent vajavad rohkem planeerimist ning seadistamist.

3.3 Ülesseadmise keerukus

Kubernetese ülesseadmine on keeruline ja mitmetest sammudest koosnev protsess. Selleks on mitmeid variante, näiteks Kubernetese *kubeadm* haldustööriist, CloudStack provisioneerimistööriist või OpenShift platvorm. Lühidalt tuleb ühes serveris alustada klaster. Seejärel on vaja seadistada Kubernetesega tihedalt seotud võrguhaldur, mis kannab hoolt erinevate Kubernetese klasterit teenindavate eksemplaride omavahelise suhtluse ning klastrisiseste domeeninimede lahendamise eest. Seejärel on võimalik lisada masinad klastrisse ning alustada esimese teenuse juurutamist. Kogu protsessi juures on mitmeid võimalikke variante soovitud funktsionaalsuse saavutamiseks, mis võivad olla klastriti erinevad, sõltuvalt ülejäänud infrastruktuurist. Infrastruktuur võib üleseadmist mõjutada näiteks olenedes võrgust klastrit teenindavate serverite taga, kõrgkäideldavuse nõuetest (kuidas lahendatakse juhtserverite kõrgkäideldavus ning kui mitu juhtserverit saab klastris olema), klastri mahust (kas tegu on ühes majutuskeskkonnas asuva klastriga või klastri liikmed asuvad mitmes serverikeskkonnas) ning muudest teguritest. Kubernetese ülesseadmine kõige efektiivsemal viisil vajab põhjalikku analüüsi olemasolevatest vajadustest, planeeritavatest edasiarengutest ning ärilistest ja tehnilistest võimalustest.

Docker Swarmi ülesseadmiseks piisab ainult ühes Dockeri mootorit omavas serveris klastri algatamisest ning teistes, tulevastest varujuhtserverites või töölisserverites selle klastriga liitumisest. Klastriga saab liituda, teades klastri loonud juhtserveri IP aadressi ning salavõtit, mida juhtserveri Dockeri käsuliidese kaudu võimalik näha on. Dockeri *ingress* võrk luuakse ja seda hallatakse automaatselt - piisab kui klastri liikmed saavad omavahel teatud portide peal suhelda.

Autori hinnangul Docker Swarmi ülespanek koos HaProxyga võib võtta alla tunni. Kubernetese seadistamine koos kõige sinna juurde vajalikuga aga palju rohkem.

3.4 Halduskoormus

Kubernetesel on mitmeid komponente, seejuures allaoleva infrastruktuuriga tihedamalt seotud võrguhaldur. Lisaks on Kubernetese edukaks käitlemiseks suurel skaalal soovitatav kasutada mõnda haldustööriista, näiteks Consulit. Lisanduv halduskoormus võib tekkida ka kõikvõimalike integratsioonide implementeerimisel arenduskeskkondadega (GitLab ja teised) või ühtse sisselogimise (ingl. *SSO*) seadistamisel.

Samas, on loodud mitmeid (vabavaralisi) tööriistu, mis erinevate komponentide koos haldamist lihtsustavad, nii ühe Kubernetese klastri raames, kui ka mitmetest klastritest ja arenduskeskkondadest koosnevate süsteemide puhul. Sellisteks tööriistadeks on näiteks Oneinfra või Spectro Cloud. Lisaks ei saa kõrvale jätta Kubernetese hallatud platvorme pakkuvaid pilveteenuseid (PaaSe), mis jätavad klastrite halduse täielikult teenusepakkujate hooleks, võimaldades arendajatel tegeleda vaid rakenduse juurutamise ja haldamisega.

Docker Swarmi haldamine nõuab samuti teatud oskusi, ent selle kogumaht on tunduvalt väiksem ning autori hinnangul on kogu haldamine teostatav tuginedes ainult Dockeri enda dokumentatsioonile. Komponentidest tuleb tähelepanu pöörata Dockeri eksemplare majutatavate serverite operatsioonisüsteemide ning baastarkava, sh Docker Engine, värkendamisele, jooksvate teenuste käitlemisele (sh ressursikasutusele) ning mõnedele funktsionaalsusega seonduvatele näitajatele, näiteks *ingress* võrgu täituvusele.

Docker Swarmi oma kompaktuse tõttu pilveteenuste nimistust tavaliselt ei leia. Küll aga on olemas mõned avatud lähtekoodiga haldus hõlbustavad tööriistad, näiteks Portainer. Lisaks saab tellida kasutajatuge ja haldustuge Docker Swarmi haldamiseks Mirantiselt ning teistelt ettevõtetelt.

Autori hinnangul on Kubernetese isehaldamine keskmise kuni kõrge raskusastmega ning Docker Swarmi isehaldamine madala kuni keskmise raskusastmega.

3.5 Nõudmised infrastruktuurile

Nii Kubernetes kui Docker Swarm eeldavad toimimiseks töökindlat infrastruktuuri selle all, ent on ehitatud tagamaks kõrgkäideldavust ka kõikvõimalike infrastruktuuri tõrgete korral. Mõlema platvormi puhul on esmatähtis töökindel ning võimekas arvutivõrk, mis võimaldaks erinevatel klastri peremeesmasinatel omavahel muretult suhelda. Sellise võrgu ehitamiseks on võimalusi palju. Kubernetese puhul on võimalik teha Kubernetes võrgukihist teadlikuks ning sellega integreerida, suurendades effektiivsust ning töökindlust, ent nõudes võrgukihi ühilduvust Kubernetese tööriistadega. [25] Docker Swarm on allolevast võrgust teadmatu ning selle eksemplaride muretu suhtlemise tagamine peab toimuma võrgukihis endas.

Nii Kubernetese kui Docker Swarmi edukaks ning turvaliseks kasutamiseks toodangus on vaja vähemalt kolme eksemplari, olgu tegu virtuaalsete või füüsiliste serveritega. Ekstra turvalise keskkonna loomiseks on mõlema orkestratsiooniplatvormi puhul soovitatav hoida juhtservereid ja töölisservereid käitlevad eksemplarid lahus. Seda saab teha nii füüsilise või virtuaalserveri tasemel kui ka operatsioonisüsteemi tasemel eraldi konteinerimootoreid kasutades. Valikuid on mitmeid ning otsustama peaks organisatsiooni võimalustele, turvalisusnõuetele kui klastri(te) suurusele tuginedes.

Autori hinnangul on baastasemel klastri loomisel nõuded infrastruktuurile Kubernetese ja Docker Swarmi puhul võrreldavad, ent Kubernetes nõuab serveritelt pisut rohkem resursse kui Docker Swarmi kasutamisel ning Kubernetese kasutamiseks tuleb valida ja seadistada ka võrguhaldur.

3.6 Ühilduvus Docker Composega

Docker Compose on Docker Inc poolt loodud tööriist, mis võimaldab hõlpsalt luua mtimetest konteineritest koosnevaid keskkondasid, määrata konteineritele ressursse, haakida konteinerite külge püsivat kettapinda (ingl. *persistent storage*), defineerida konteinerite vahelisi võrke ning määrata muid parameetreid. Docker Compose töötab YAML vormingus seadistusfailide abil. Docker Compose on väga levinud tööriist kohalike arenduskeskkondade ülesseadmisel ning nende hõlpsal jagamisel teiste arendajatega. Docker Compose ühilduvus on oluline, sest selle abil saaks

arenduskeskkonnas loodud teenustevahelise loogika, näiteks konteineritevaheliste võrkude defineerimise, hõlpsalt toodangukeskkonda üle viia.

Docker Compose võimaldab luua ka teenuseid Docker Swarmi klastrites. [26] Paljud asjad on samad võrreldes kohalikus keskkonnas Dockeri keskkonna ülesseadmisega, näiteks teenuste definitsioonid, ressurssipiirangud ning konteineritevaheliste võrkude defineerimine. Ent arenduskeskkonade loomisel levinud lähenemine koodikaustade haakimisel konteinerisse Docker Swarmi puhul ei tööta, mistõttu tuleb tarkvara pakendada tervikuna tõmmistesse ning defineerida nende tõmmiste kasutamine Docker Compose's. Lisaks tuleb Docker Swarmi teenuste defineerimisel Docker Compose abil silmas pidada teatud Docker Swarmi spetsiifilisi nõudeid ja potentsiaalseid parameetreid, näiteks seda, mitu eksemplari teenusest klastrisse luuakse. Docker Compose kasutamine ning Swarmiga integreerimine on autori hinnangul hästi dokumenteeritud Dockeri ametlikus dokumentatsioonis.

Kubernetes Docker Compose seadistusfaile ei toeta, kuna sel on erinev arhitektuur (konteinerid on omakorda jagatud *pod*'idesse ja muud erinevused). Osad põhimõtted on aga Docker Compose ja Kubernetese vahel ühilduvad ning Docker Compose failid on tõlgendatavad Kubernetese definitsioonideks Kubernetese ametliku *kompose* tööriista abil. [27] Kompose kasutamine on autori hinnangul hästi dokumenteeritud Kubernetese ametlikus dokumentatsioonis.

Nii Kubernetes kui Docker Swarm on autori hinnangul osaliselt ühilduvad Docker Composega. Teenused ning nendevaheline suhtlus on tõlgendatavad orkestratsiooniplatvormide teenusteks, ent põhiprintsiip rakenduse (koodi) enda käitlemiseks on erinev - kohalikus masinas Docker Compose kasutades on võimalik koodikaust konteineri sisse haakida, ent orkestratsiooniplatvorme kasutades tuleb hakata rakendust tarnima konteineritõmmiste kaudu.

3.7 Lõppresolutsioon

Eelnevates punktides tehtud võrdluste alusel loodi võrdlustabel, kus mõlemale orkestratsiooniplatvormile anti 0, 0,5 või 1 punkti iga omaduse kohta.

Tabel 1. Orkestratsiooniplatvormide võrdlustabel

Omadus	Kubernetes	Docker Swarm
Juhtserveri kõrgkäideldavus	1	1
Teenuste kõrgkäideldavus	1	1
Ülesseadmise keerukus	0,5	1
Nõudmised infrastruktuurile	0,5	1
Ühilduvus Docker Compose'ga	0,5	0,5
Kokku	**3,5**	**4,5**

Docker Swarm on minimalistlik lahendus, mis ei paku nii suurt paindlikkust, kui Kubernetes, ent seda on autori hinnangul väga lihtne üles seada ja hallata. Põhiline funktsionaalsus kõrgkäideldava keskkonna ülesseadmiseks on olemas ning sellel käideldavat teenust on lihtne horisontaalselt skaleerida. Docker Swarm ei ole saavutanud nii suurt populaarsust, kui Kubernetes, võib-olla vähematest võimalustest tingituna, ent Docker Swarm sobib teatud oludesse väga hästi, mistõttu seda leiab ka toodangukeskkondadest ning Dockeri ärilist poolt omav Mirantis on lubanud ka edaspidi jätkata Docker Swarmi hooldamist ning edasiarendamist. [15] Seetõttu leiab autor, et Docker Swarm on hea orkestreerimisplatvorm enamike vajaduste täitmiseks veebirakenduste majutamisel.

Kubernetes on võimekas ning toodangus kasutamiseks valmis orkestratsiooniplatvorm ja selle kohta leiab palju informatsiooni. Kubernetes lahendab mitmeid halduspoliitikatega seotud probleeme, võimaldades ehitada ühe suure ning ressursiefektiivse platvormi paljude erinevate rakenduste käitlemiseks. Hästiehitatud Kubernetese klaster pakub töökindlust ja paindlikkust ning seda on võimalik ehitada ka mitme serverikeskkonna peale, näiteks jagades klastri osad ära asutuse enda serverikeskkonna ning avaliku pilve vahel. Ülesseatud Kubernetes on ka arendajasõbralik ning seetõttu on see leidnud suure populaarsuse. Kubernetese

ülesseadmine on aga mahukas töö, mis nõuab põhjalikku analüüsi ning läbimõtlemist. See võib olla õigustatud suuremate firmade puhul. Hallatud Kubernetese keskkonda pakkuvaid pilveteenuseid kasutades on võimalik ka väiksematel (arendus)firmadel hõlpsalt Kubernetest kasutada.

Autori hinnangul oleks Kubernetese ülesseadmine lähtetingimustes välja toodud projekti majutamiseks etteantud tingimustes liiga suur ettevõtmine, millel ei ole antud olukorras väga palju kasulikke eeliseid võrreldes Docker Swarmiga, mistõttu see veebiprojekt majutatakse Docker Swarmi abil. Analüüsi käigus leiti ka siiski, et Kubernetest võib kaaluda tulevaste projektide majutamisel, kus oleks võimalik kasutada hallatud Kubernetest pakkuvaid pilveteenuseid või klientide endi loodud Kubernetese keskkondasid. Lisaks leiti, et praegused arendusprotsessid tarkvara juurutamisel vajavad konteineritel põhinevale majutamisele üle minemisel igal juhul muutmist. Lisaks Docker Compose kasutamisel toodangus tuleks teha muutusi võrreldes selle kasutamisega arenduskeskkondade ülesseadmisel.

4 Lahenduse ülesseadmine

4.1 Arendusprotsesside muudatused

4.1.1 Vajalikud muudatused

Konteineritel põhinevat platvormi kasutades on vajalik tarkvara juurutades pakendada tarkvara ise konteineri sisse, luues iga tarkvarakomponendi jaoks oma konteineritõmmise ehk *Docker Image*. Tarkvara sisaldavate tõmmiste abil on võimalik kiirelt konteinereid tööle panna, neid enne toodangusse juurutamist testida ning hõlpsalt jälgida juurutatud tarkvara versioone ning nende iseärasusi vigade ilmnemisel.

Lõpplahenduse moodustavad komponendid omavad praegusel hetkel ainult automaatteste, mida töödeldakse GitLabi keskkonna tööliste (ingl. *runners*) abil CI/CD (*Continuous Integration and Continuous Delivery)* protsessi raames. Sama protsessi on võimalik kasutada ka konteinerite tõmmiste loomiseks, nende üleslaadimiseks tõmmiste repositooriumisse ning tõmmise juurutamiseks konteinerplatvormile.

GitLabi keskkond pakub nii isemajutatud kui avalikus variandis tasuta tõmmiste repositooriumit (ingl. *container registry*), mida on võimalik kasutada tõmmiste hoidmiseks ja juurutamiseks. [28] Sama funktsionaalsust pakub ka Docker Hub, ent kuna praegused arendusprotsessid juba GitLabis toimuvad, on autori arvates parem kõik ühes kohas hoida.

Seega tuleb iga veebirakenduse komponendi juurde luua tõmmise loomiseks vajalik Dockerfile, kus defineeritakse, kuidas rakenduse käitlemiseks vajalik konteiner koos tarkvarapakettidega luuakse ning kuidas tarkvara selles eelseadistatakse (näiteks vajalike PHP lisapakettide paigaldamine *Composer* tööriistaga).

Lisaks tuleb defineerida, millise loogika alusel tarkvara juurutama hakatakse, nii testkeskonda, eeltoodangu (ingl. *staging environment*) kui ka toodangukeskkonda, ning kuidas määratakse tarkvara versioonide numeratsioon.

4.1.2 Tõmmisefailide loomine

Iga veebirakenduses kasutatava komponendi jaoks loodi tõmmise loomiseks vajalik tõmmisefail ehk Dockerfile. Kõik tõmmisefailid loodi arenduskeskkonna tõmmisefailide alusel, ent toodanguks optimeerituna - kaotati ära enamus tõmmise ehitamise ajal kasutatud muutujaid (näiteks muutujad baastarkvara erinevate versioonide määramiseks, need on nüüd staatilised), lisati lisamoodulite paigaldamine ning tehti turvalisust tõstvaid muudatusi, sealhulgas rakendused käivitatakse uutes tõmmistes tavakasutajana, mitte juurkasutajana.

Tagarakenduse tõmmis põhineb nüüd David Négrieri (TheCodingMachine) poolt loodud *thecodingmachine/php:7.2-v3-slim-apache* tõmmisel. Ametliku tõmmise asemel sai valitud see kommuuni poolt loodud tõmmis, sest see pakub mitmeid juba seadistatud võimalusi, sealhulgas konteineri käivitamisel skriptide jooksutamist enne Apache veebiserveri käivitamist. Varasemalt käideldi tagarakendust PHP-FPM mootori abil, kasutades rakenduse üldist Nginxi puhverserverit. Selliseks seadistuseks oli vajalik koodikaust haakida nii PHP-FPM konteineri kui ka Nginx'i konteineri külge. Kuna aga rakenduse koodi Nginx'i konteinerisse enam haakida ei saa, oli vaja teistsugust lähenemist, kus back-endi käitlev konteiner oleks eraldiseisev täisväärtuslik veebiserver. Apache variant tõmmisest valiti seetõttu, et seda oli kõige lihtsam seadistada, mistõttu see oli kõige veakindlam lahendus. Alternatiivsed lahendused oleks olnud paigaldada eraldi Nginx puhverserver tagarakenduse konteinerisse ning kasutada seal PHP-FPM'i, teine lahendus oleks olnud seadistada Apache koos PHP-FPMga (mil oleks potentsiaalselt suurem jõudlus, mida hetkel korvatakse mitme tagarakenduse eksemplari samaaegse töötamisega) ning kolmas lahendus oleks olnud kasutada Laravel'i sisseehitatud arendusserverit (mida ei ole soovitatav toodangus kasutada). Valminud Dockerfile on lisas 1.

Lisaks kuulub veebirakendusse mitu Javascripti põhist projekti, millest kaks on Vue.js-i raamistikule ehitatud eesrakendused ning üks on mikroteenus PDF vormingus arvete

genereerimiseks. Mõlema eesrakenduse Dockerfile tuli sisuliselt identne, kuna mõlemad projektid põhinevad samal Javascripti raamistikul ning Node.js-i lisapakettide (*node_modules*) osas on samuti palju sarnasusi. Mõlemad tõmmisefailid loodi Node.js-i ametliku Dockeri tõmmise alusel, sinna lisati koodi käitlemine eraldi tavakasutaja õigustes ning Node.js-i lisapakettide paigaldamine tõmmise loomise ajal. Nendesse tõmmisefailidesse tuli siiski lisada ehitusaegsete muutujate lugemine, sest Nuxt.js raamistik kompileerib kasutajale saadetava sisu (HTML, CSS, Javascripti pakett ehk *bundle*) sisse ka kliendipoolsetes päringutes kasutatava tagarakenduse veebiaadressi, mis ei ole relatiivselt tuletatav veebilehe külastamise veebiaadressist (tagarakendusega suhtleb klient eraldi alamdomeeni kaudu). Ehituse ajal erinevate tagarakenduste aadressite määramiseks luuakse CI/CD protsessi käigus erinevad tõmmised erinevate keskkondade jaoks - tõmmised produktsiooni, eelproduktsiooni (*staging*) ning vajadusel teiste keskkondade jaoks. Alternatiiv oleks olnud kompileerida kasutajale saadetav sisu konteineri tööleminemise ajal, mis aga teeb konteineri käivitumise väga aeglaseks. Eesrakendusi ei saa ka staatiliselt serveerida, sest üks nende omadus on serveris visualiseeritud (ingl. *SSR* ehk *server-side renderred*) sisu kuvamine - dünaamiselt muutuv avalik sisu (näiteks müüdavad tooted) laetakse eesrakenduse serveri poolt alla ning luuakse vastavad eelvisualiseeritud lehed. See tehnoloogia tagab, et lehed on väga kiiresti laetavad ning seeläbi on need otsingumootorisõbralikud. Lisaks võimaldab see tehnoloogia näha lehe sisu kasutajatel, kel on brauseris Javascript välja lülitatud.

4.1.3 Tõmmiste loomine

Tõmmiste loomine saab olema tihedalt seotud kogu arendusprotsessiga, mis toimub GitLabi keskkonnas. Tõmmiseid luuakse GitLabi CI/CD protsessi käigus, seejuures arvestades Giti versioonikontrolli loogikat.

Tõmmisefaili loomine defineeritakse *gitlab-ci.yml* seadistusfailis, mis üheskoos rakenduse koodi, Dockerfile ja teiste rakendust toetavate failidega asub vastavas repositooriumis. Seadistusfailis defineeriti, mis tingimustel tõmmist looma asutakse, kuidas tõmmist luuakse ning milliseid teised tegevused sellele eelnevad või järgnevad.

Protsessi käigus otsustasime, et uus tõmmisefail luuakse iga peaharusse (ingl. *master branch*) ning iga väljalaskeharusse (ingl. *release branch*) lisandunud uuenduse (ingl.

commit) puhul. Vastavad tõmmisefailid nimetatakse eraldi ning nendele omistatakse uuenduse märkme (ingl. *commit tag*) kaudu versiooninumber. Peaharu tõmmised juurutatakse automaatselt eeltoodangu keskkonda ning väljalaskeharu tõmmised toodangukeskkonda.

4.2 Rakendusespetsiifilised muudatused

4.2.1 Keskkonnamuutujate haldus

Vaikimisi loetakse rakenduse tööks vajalikud muutujad, nagu näiteks andmebaasiühenduse parameetrid, rakenduse koodiga samas kaustas asutavast keskkonnamuutujate (*.env*) failist. Seda faili aga tõmmise sisse erinevatel põhjust pole mõistlik ega kohati võimalik selle ehitamise ajal panna. Keskkonnamuutujate kasutamist konteineritel põhinevas keskkonnas lahendatakse orkestratsiooniplatvormi ja konteinerimootori, antud juhul Docker Swarmi ja Docker Engine kaudu. Docker Swarmi teenuse loomise ajal on võimalik defineerida, millised keskkonnamuutujad on konteineri tööks vajalikud, ning kuidas need konteineri sisse laetakse. Keskkonnamuutujad on võimalik kirjutada teenuse definitsiooni endasse (Docker Compose seadistusfaili) või laadida need eraldi seadistusfailist. See annab palju paindlikkust definitsiooni enda haldamisel. Need keskkonnamuutujad teeb Docker konteineri töötamise ajal rakendusprotsessile kättesaadavaks. Nii Laravel kui ka Node.js suudavad neid muutujaid protsessi kaudu edukalt lugeda ning kasutada. Lisaks pakub Docker Swarm võimalust kasutada turvalisemaid muutujaid ehk *secreteid*. [29] *Secreteid* hoitakse peremeessüsteemis turvaliselt, nende väärtust Dockeri käsurea ega rakendusliidese kaudu lugeda ei saa ning neid liigutatakse klastri liikmete vahel krüpteeritud kujul. Secreteid saab omistada teenustele, mispeale haagitakse need töötava konteineri sisse. Konteineri sees on need kättesaadavad spetsiaalses kataloogis. Rakendused aga ei võimalda neid vaikimisi sellisel kujul kasutada. Selleks lisasime tagarakenduse tõmmise käivituskäsu ette skripti, mis need konteineri töölemineku ajal spetsiaalsest kataloogist välja loeb ning rakenduse jaoks keskkonnamuutujate kaudu kasutatavaks teeb.

Ka Kuberneteses on olemas *secretid* ning see võimaldab neid automaatselt keskkonnamuutujateks teisendada. [30]

4.2.2 Sessioonide hoidmine andmebaasis

Selleks, et mingist rakendusest mitu eksemplari samaaegselt koos töötada saaks, peaksid neis töötavad rakendused olema olekuta (ingl. *stateless*). Selleks, et tagarakendused saaks koos töötada, peaksid nad saama omavahel jagada andmeid, mida nad muidu haldaks iseseisvalt. Põhiline vajadus meie loodud rakendusel on see, et kasutajate sessioonid oleksid erinevate tagarakenduse eksemplaride vahel ühtsed. Vaikimisi hoiab Laravel sessioone failides, ent väga lihtsalt on võimalik viia sessionide hoidmine andmebaasi. See lähenemine ka valiti. Tagarakenduste ühisele andmebaasile loodi eraldi tabel sessioonide hoidmiseks ning tagarakendused seadistati seda kasutama. See võimekus on Laravelis sisse ehitatud ning lisaarendusi see ei vajanud. Potentsiaalse edasiarenguna saaks luua sessioonide hoidmiseks eraldi No-SQL tüüpi mitterelatsioonilise ja võib-olla muutmälupõhise andmebaasi, kasutades selleks näiteks tarkvara Redis. Selline andmebaas oleks palju kiirem ning parandaks seeläbi oluliselt klientide päringute latentsust. Redist saab seadistada ka andmeid püsivalt kettale salvestama [31], mis oleks kasulik klientide sisselogimiste stabiilsemaks hoidmiseks.

4.3 Keskkonna loomine

4.3.1 Serverite ülesseadmine

Ühe juhtmasina kaotamist tolereeriva Docker Swarmi klastri ehitamiseks on vaja vähemalt kolme juhtserverit, mistõttu on lahenduse loomiseks kasutatakse ka kolme serverit.

Kõigil virtuaalserveritel on 4 virtuaalset protsessorituuma (vCPU), 8GB muutmälu ning 20GB kohalikku kettapinda. Lisaks on iga masina külge teenusepakkuja poolt haagitud ühine võrguketas. Serveritel jookseb CentOS 7.6 operatsioonisüsteem. Võrguliiklus serverite ja ülejäänud võrgu vahel on piiratud. Täiendavate ühenduste loomine serverite vahel tuleb teenusepakkujal lubada. Lisaks on võimalik tekitada privaatvõrke ainult valitud serverite vahel, mida Docker Swarmi võrgu seadistamiseks ka kasutati.

Ülesseadmist alustati igale serverile Docker Engine paigaldamisega. CentOSi pakutud Dockeri asemel otsustati kasutada värskemate uuenduste saamiseks Dockeri enda ametlikku repositooriumit. Esialgu võeti kasutusele Docker Community Edition (Docker CE).

Dockeri paigaldamise järel loodi uus Docker Swarmi klaster esimesel toodanguserveril, seejuures määrates toodanguserverite privaatvõrgu kasutamine. Seejärel oli võimalik teised toodanguserverid klastrisse liita. Esimese asjana katsetati Docker Swarmi toimimist Docker *Hello Worldi* testkonteineri käivitamisega mitmes eksemplaris. Katse oli edukas. Seejärel muudeti teised toodanguserverid varujuhtserveriteks.

4.3.2 Klastri halduse lihtsustamine

Eraldiseisva Dockeri mootori kui ka Docker Swarmi hõlpaks haldamiseks on loodud vabavaraline tööriist Portainer. [32] Portainer loob lihtsalt kasutatava veebiliidese, mille kaudu on võimalik teostada mitmesuguseid Dockeri haldusega seotud protsesse, nagu näiteks uute teenuste lisamine, olemasolevate teenuste jälgimine, võrkude seadistamine, teenuste logide vaatamine ning konteineri sees käsurea kasutamine. Portaineri kasutuselevõtt võimaldab hõlpsamalt jälgida teenuse kui terviku toimimist ning vajadusel lihtsamini teostada analüüse erinevate teenuse komponentide töötamise osas.

Portainer paigaldati selle ametliku dokumentatsiooni järgi, tuginedes nende poolt loodud Docker Compose näidisseadistusfailidele. See seadistusfail loob teenuse, mis koosneb Portaineri rakendusest endast ühel juhtserveril ning Portaineri agentidest teistel serveritel. Portaineri rakendus ise vajab püsivat kettapinda selle seadistuse, sealhulgas kasutajakontode, hoidmiseks. Selle jaoks loodi ja haagiti Portaineri konteineri külge jagatud kettapinnal uus kataloog. Portaineri veebiliides töötab pordil 9000. Sellele ligipääsemiseks tuli paluda teenusepakkujal lubada ühenduste loomine sellele pordile serverikeskkonna VPNilt. Üldiselt Portaineri ülesseadmine oli autori arvates lihtne ning see töötab hästi.

4.4 Klastri halduse lihtsustamine

Teenuse majutamiseks orkestratsiooniplatvormil tuleb vastav seadistus luua. Selleks otsustati luua eraldi Docker Compose seadistus. Toodanguseadistus jagab ühiseid elemente arenduses kasutatava Docker Compose seadistusega, näiteks on samad teenuste nimed ning nende omavahelised võrgud, ent on samal ajal väga erinev.

Antud hetkel koosneb toodangus kasutatav teenusekomplekt järgmistest rakendustest:

- Nginx puhverserver päringute edastamiseks teistele teenustele
- Tagarakendus
- Eesrakendus avaliku veebi jaoks
- Teine eesrakendus siseveebi jaoks
- Mikroteenus arvete genereerimiseks
- Matomo analüütikarakendus

Kõikide rakenduste jaoks kasutatakse vastavaid tõmmiseid. Tõmmiste hoidmiseks kasuttatakse GitLabi privaatrepositooriumit, mistõttu tuli serverites Dockeri mootorid GitLabi sisse logida, kasutades selleks spetsiaalselt loodud teenuskasutajat. Konteinerite külge haagitakse siiski ka katalooge, ilma milleta pole teenuse töötamine võimalik. Näiteks tagarakendused hoiavad kliendile serveeritavaid PDF arveid püsivalt kettal, ning nende hoidmiseks kasutatav kataloog haagitakse tagarakenduste konteinerite külge. Selleks on igasse toodanguserverisse haagitud kataloog võrgukettalt. Teenusekomplektis kasutatavad valmisteenused, näiteks Nginx puhverserver, käivitatakse toodangus selle ametliku tõmmise kaudu, mitte iseehitatud tõmmist kasutades, nagu arenduses. Valmisteenuste seadistamine toimub muutujate ja seadistusfailide kaudu.

Kõiki muutujaid ei olnud mõistlik hoida *secretitena*. Vähemdelikaatsed muutujad defineeritakse vastava teenuse seadistusfailis ning need failid loetakse iga konteineri sisse keskkonnamuutujateks või haagitakse konteineri sisse soovitud kohta.

Lisaks *secretite* ja keskkonnamuutujate defineerimisele lisati seadistusfaili teenuse käitlemisega seotud parameetrid. Esiteks seadistati, et kõik rakendused taaskäivituks automaatselt igal juhul. Teiseks seadistati, et iga rakendust jookseks 2 eksemplari ning lisaks seadistati teenuse uuendamise ajastused - teenuste uuendamise ajal uuendatakse tõmmised ning taaskäivitatakse konteinerid ükshaaval, koos kerge viivitusega. Ideaalis peaks sellise töövooga rakenduse uuendamine kasutajale nähtamatult toimuma.

Esialgne testimine näitas, et rakendused suudavad edukalt mitmes eksemplaris töötada ning anomaaliaid ei esinenud.

Joonis 1. Töötav klaster Portaineri kasutajaliidesest vaadatuna

4.5 Teenuse ülalhoidmine

Pärast rakenduse töölesaamist jätkati lahenduse optimeerimise, täiustamise ning üldise haldamisega.

Esiteks seati üles rakenduse automaatne uuendamine GitLabi CI/CA protsessi kaudu. Selleks lisati CI/CA protsessi samm, mis väljalaskeharusse lisandunud uuenduse korral peale tõmmise valmistamist saadab HTTPS päringu Portaineri rakendusliidesele, mispeale Portainer vastava teenuse uuendamise algatab. Portainer võimaldab selle jaoks kasutada spetsiaalseid rakendusliidese otspunkte (ingl. *webhook*), mis tuleb iga teenuse jaoks Portaineri kasutajaliidesest sisse lülitada.

Järgmiste sammudena on plaanis täiendada kogu teenuse logimist. Selleks on plaanis kasutada Graylog logimistarkvara. Logima peaks nii rakenduste loogikas tekkivaid vigu kui ka majutamisplatvormil toimunud muudatusi (teenuste uuendused, taaskäivitused ja teised stabiilsust ning turvalisust puudutavad näitajad). Lisaks on plaanis välja töötada monitooringulahendus teenuse töö jälgimiseks.

Teenusekomplekti võib ka lisanduda uusi teenuseid. Kaalutakse välise andmebaasi asemel luua andmebaasiteenus konteinerkeskkonda. Sellega suureneks Maagilise Veduri halduskoormus teenuse käitlemisel, ent see annaks suuremat paindlikkust andmebaasi tarkvara versioonide valikul ning rohkem võimalusi andmebaasi parameetrite seadistamisel, kuna andmebaasi kasutaksid ainult teenusekomplekti teenused, mida on võimalik igatpidi seadistada.

5 Kokkuvõte

Diplomitöö täitis oma eesmärki ning lähteülesandeks toodud veebirakenduse kõrgkäideldav majutamine konteinertehnoloogiate abil sai teostatud. Analüüsi käigus selgus, et alati ei pruugi olla parim valik kõige populaarsem tehnoloogia turul - ülesanne teostati etteantud lähtetingimusi arvestades kasutades orkestreerimisplatvormi Docker Swarm, mitte *de facto* orkestreerimisplatvormide standardit Kubernetest. Docker Swarmi kasutamine sai põhjendatud poolhallatud majutuskeskkonnast tingituna ning eesmärgist anda autori töökohale võimalikult vähe halduskoormust teenuse ülevalhoidmiseks. Võimalikke probleeme ei leitud ning kindlust andis Dockeri ärilist poolt omava Mirantise kavatsus jätkata Docker Swarmi uuenduste ja kasutajatoe pakkumist. Diplomitöö tõestas, et ka Docker Swarmil on olemas kõrgkäideldavuse tagamiseks vajalik funktsionaalsus ning et Docker Swarmi on võimalik edukalt toodangus kasutada.

Autor leiab, et konteinertehnoloogiad on pidevas arengus, ning valikuid on palju. Autori arvates tuleks igal ettevõttel leida oma vajadustest ning võimalustest lähtuvalt just enda jaoks sobivaim tehnoloogia.

Lisaks täienesid autori töökoha Maagilise Veduri praktikad veebirakenduste arendusprotsesside loomisel ning teadmised rakenduste juurutamiseks konteineritel põhinevatesse keskkondadesse. Saadud teadmised võimaldavad Maagilisel Veduril optimaalsemalt rakendusi tarnida ning neid töökindlamalt ning parema koormustaluvusega toodangus käidelda, kasutades selleks erinevaid konteineritel põhinevaid tehnoloogiaid. Maagiline Vedur ei välista edasiste projektide majutamist ka Kubernetese või teiste orkestreerimisplatvormidde peal, kuna nüüd on olemas teadmised, kuidas rakendusi ja nendega seotud teenuseid selleks ette valmistada.

Kasutatud kirjandus

[1] Docker Inc., “Why Docker?”. [Võrgumaterjal]. https://www.docker.com/why-docker [Kasutatud 31 märts 2020].

[2] Docker Inc., ”What is a Container?”. [Võrgumaterjal]. https://www.docker.com/resources/what-container [Kasutatud 31 märts 2020].

[3] Docker Inc., ”The Industry-Leading Container Runtime”. [Võrgumaterjal]. https://www.docker.com/products/container-runtime [Kasutatud 31 märts 2020].

[4] Gartner Inc., “Gartner Identifies the Top 10 Trends Impacting Infrastructure and Operations for 2019”, detsember 2019. [Võrgumaterjal]. https://www.gartner.com/en/newsroom/press-releases/2018-12-04-gartner-identifies-the-top-10-trends-impacting-infras [Kasutatud 1 aprill 2020].

[5] TechMagic, “Serverless vs Docker Containers— what to choose in 2019?”, aprill 2019. [Võrgumaterjal]. https://medium.com/@TechMagic/serverless-vs-docker-what-to-choose-in-2019-80cb80f4b680 [Kasutatud 1 aprill 2020].

[6] The Knative Authors, “Knative koduleht”. [Võrgumaterjal]. https://knative.dev/ [Kasutatud 4 aprill 2020].

[7] Nathan McCauley, Docker Inc, ”Your Software is Safer in Docker Containers”, august 2016. [Võrgumaterjal]. https://www.docker.com/blog/software-security-docker-containers/ [Kasutatud 6 aprill 2020].

[8] The Kubernetes Authors, ”What is Kubernetes?”, märts 2020. [Võrgumaterjal]. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/ [Kasutatud 7 aprill 2020].

[9] Adrian Mouat, O’Reilly, “5 security concerns when using Docker”, veebruar 2016. [Võrgumaterjal]. https://www.oreilly.com/content/five-security-concerns-when-using-docker/ [Kasutatud 7 aprill 2020].

[10] Linode, ”When and Why to Use Docker”, jaanuar 2020. [Võrgumaterjal]. https://www.linode.com/docs/applications/containers/when-and-why-to-use-docker/ [Kasutatud 7 aprill 2020].

[11] Docker Inc, ”Kubernetes”. [Võrgumaterjal]. https://www.docker.com/products/kubernetes [Kasutatud 7 aprill 2020].

[12] Oracle Corporation, ”MySQL 8.0 Reference Manual | Chapter 17 Replication”. [Võrgumaterjal]. https://dev.mysql.com/doc/refman/8.0/en/replication.html [Kasutatud 9 aprill 2020].

[13] Docker Inc., ”Docker 1.12: Now with Built-in Orchestration!”, juuni 2016. [Võrgumaterjal]. https://www.docker.com/blog/docker-1-12-built-in-orchestration/ [Kasutatud 10 aprill 2020].

[14] Mirantis Inc., ”What We Announced Today and Why it Matters”, november 2019. [Võrgumaterjal]. https://www.mirantis.com/blog/mirantis-acquires-docker-enterprise-platform-business/ [Kasutatud 10 aprill 2020].

[15] Mirantis Inc., ”Mirantis will continue to support and develop Docker Swarm”, veeburar 2020. [Võrgumaterjal]. https://www.mirantis.com/blog/mirantis-will-continue-to-support-and-develop-docker-swarm/ [Kasutatud 10 aprill 2020].

[16] Frederic Lardinois, TechCrunch, ”As Kubernetes Hits 1.0, Google Donates Technology To Newly Formed Cloud Native Computing Foundation”, juuli 2015. [Võrgumaterjal]. https://techcrunch.com/2015/07/21/as-kubernetes-hits-1-0-google-donates-technology-to-newly-formed-cloud-native-computing-foundation-with-ibm-intel-twitter-and-others/ [Kasutatud 10 aprill 2020].

[17] Cloud Native Computing Fondation, ”Members”. [Võrgumaterjal]. https://www.cncf.io/about/members/ [Kasutatud 10 aprill 2020].

[18] Red Hat Inc, „Open Shift koduleht”. [Võrgumaterjal]. https://www.openshift.com/products/features [Kasutatud 10 aprill 2020].

[19] The Kubernetes Authors, ”Options for Highly Available topology”, juuni 2019. [Võrgumaterjal]. https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/ha-topology/ [Kasutatud 13 aprill 2020].

[20] The Kubernetes Authors, ”Creating Highly Available clusters with kubeadm”, november 2019. [Võrgumaterjal]. https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/ [Kasutatud 13 aprill 2020].

[21] Docker Inc., ”Swarm mode key concepts”. [Võrgumaterjal]. https://docs.docker.com/engine/swarm/key-concepts/ [Kasutatud 13 aprill 2020].

[22] Docker Inc., ”Raft consensus in swarm mode”. [Võrgumaterjal]. https://docs.docker.com/engine/swarm/raft/ [Kasutatud 13 aprill 2020].

[23] The Kubernetes Authors, ”Service”. [Võrgumaterjal]. https://kubernetes.io/docs/concepts/services-networking/service/ [Kasutatud 13 aprill 2020].

[24] Docker Inc., ”Use swarm mode routing mesh”. [Võrgumaterjal]. https://docs.docker.com/engine/swarm/ingress/ [Kasutatud 13 aprill 2020].

[25] The Kubernetes Authors, ”Cluster Networking”. [Võrgumaterjal]. https://kubernetes.io/docs/concepts/cluster-administration/networking/#the-kubernetes-network-model [Kasutatud 14 aprill 2020].

[26] Docker Inc., ”Compose file version 3 reference”. [Võrgumaterjal]. https://docs.docker.com/compose/compose-file/ [Kasutatud 14 aprill 2020].

[27] The Kubernetes Authors, ”Translate a Docker Compose File to Kubernetes Resources”. [Võrgumaterjal]. https://kubernetes.io/docs/tasks/configure-pod-container/translate-compose-kubernetes/ [Kasutatud 14 aprill 2020].

[28] GitLab, ”GitLab Container Registry”. [Võrgumaterjal]. https://docs.gitlab.com/ee/user/packages/container_registry/index.html [Kasutatud 16 aprill 2020].

[29] Docker Inc., ”Manage sensitive data with Docker secrets”. [Võrgumaterjal]. https://docs.docker.com/engine/swarm/secrets/ [Kasutatud 18 aprill 2020].

[30] The Kubernetes Authors, ”Secrets”. [Võrgumaterjal]. https://kubernetes.io/docs/concepts/configuration/secret/ [Kasutatud 18 aprill 2020].

[31] Redis Labs, ”Redis Persistence”. [Võrgumaterjal]. https://redis.io/topics/persistence [Kasutatud 20 aprill 2020].

[32] Portainer, ”Portaineri koduleht”. [Võrgumaterjal]. https://www.portainer.io/overview/ [Kasutatud 21 aprill 2020].

Lisa 1 – Tagarakenduse Dockerfile

```
ARG PHP_EXTENSIONS="apcu mysqli pdo_mysql soap gd"
FROM thecodingmachine/php:7.2-v3-slim-apache

MAINTAINER Maagiline

COPY --chown=docker:docker . .
RUN sudo chown -R www-data:www-data storage

RUN composer install

ENV APP_ENV=prod \
    APACHE_DOCUMENT_ROOT=public/ \
    APACHE_RUN_USER=www-data \
    APACHE_RUN_GROUP=www-data \
    STARTUP_COMMAND_1="for i in /run/secrets/*; do export $(cat $i)
> /dev/null 2>&1; done;"
```

Joonis 2. Tagarakenduse toodangutõmmise loomiseks kasutatav Dockerfile