TAMPEREEN
AMMATTIKORKEAKOULU

Verkkokauppojen Docker-julkaisu

PHP-, SQL-pohjaisten verkkokauppojen julkaisu Docker-ympäristössä

Esko Takku

Opinnäytetyö
Joulukuu 2017
Tietotekniikka
Ohjelmisto- ja Tietoliikennetekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Ohjelmisto- ja Tietoliikennetekniikka

TAKKU ESKO:

Verkkokauppojen Docker Julkaisu
PHP, SQL pohjaisten verkkokauppojen julkaisu Docker ympäristössä

Opinnäytetyö 48 sivua, joista liitteitä 0 sivua
Joulukuu 2017

Verkkokauppojen Docker Julkaisu
Asiasanat: docker docker-swarm traefik mariadb php sql docker-compose glusterFS

Tämä opinnäytetyö käsittelee yhtä mahdollisuutta web-sivustojen julkaisualustaksi. Sitä ei ole suunniteltu toimimaan tehokkaampana mahdollisena alustana, vaan automatisoituna ympäristönä, joka vaatii mahdollisimman vähän ylläpitoa ja on kuitenkin kykenevä vastaanottamaan suurta käyttäjäkuormaa. Aluksi valitaan paras mahdollinen tekniikka, jotta saavutettaisiin haluttu lopputulos. Kun nämä vaihtoehdot on käyty läpi, siirrytään käsittelemään lopputulosta ja järjestelmän rakennusta.

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Bachelor of Engineering
Software and Network Engineering

TAKKU ESKO:
Publishing web-stores on Docker
PHP, SQL based web-store publishing in Docker environment

Bachelor's thesis 48 pages, appendices 0 pages
December 2017

Publishing web-stores on Docker
Key words: docker docker-swarm traefik mariadb php sql docker-compose glusterFS

This work contains instructions in making and deploying publishcation service for websites. It is not designed to be fastest system, but to be completely automatic and failsafe with minimum upkeep and cost. In the beginning there is a comparison of different possible techniques. In the end there is an explanation and instructions for deployment of desired system.

SISÄLLYS

ERITYISSANASTO tai LYHENTEET JA TERMIT (valitse jompikumpi)

kontti	Paketoitu kevyt virtuaalipalvelin
reverse proxy	Välityspalvelin, joka ohjaa pyynnöt toiselle palvelimelle
traefik	Lopullisessa tuotoksessa käytettävä reverse proxy
docker	Suosittu kontitusalusta rakentamiseen ja julkaisemiseen
docker-compose	docker-pakettien julkaisua helpottava järjestelmä
docker swarm	docker-palvelinten muodostama ryhmä
mount	tiedoston tai sijainnin linkitys uuteen sijaintiin
api	sovellusrajapinta

1 JOHDANTO

Opinnäytetyön toisessa kappaleessa käsitellään Oscar Software Oy:n julkaisujärjestelmän lähtöpistettä. Kolmas kappale keskittyy tavoitteisiin, jotka uudistetun järjestelmän avulla halutaan saavuttaa. Neljäs kappale tuo esille joitakin vaihtoehtoja eri julkaisuprosessin osiin. Viides kappale käsittelee eri Docker-kontit, joita lopullisessa työssä tarvitaan. Kuudes kappale esittelee palvelun julkaisuun käytettävät docker-compose. Kappaleessa seitsemän käsitellään itse järjestelmän rakennusta perusasennuksesta lähtien. Viimeinen kappale keskittyy pohtimaan koko työn tarkoituksenmukaisuutta ja mahdollisia parannuskohteita.

2 LÄHTÖPISTE

Oscar Software Oy:n verkkokauppojen julkaisujärjestelmä työn aloitushetkellä tukeutui perinteiseen “web hotel” -malliin. Tätä mallia voidaan tällä hetkellä pitää vanhanaikaisena ja kankeana.

2.1 Malli

Lähtökohtainen julkaisumalli käyttää pohjanaan ubuntu 14.04 -käyttöjärjestelmässä pyörivää Apache-palvelinta. Yhdellä palvelimella voi olla useita kauppoja, joiden liikenteen Apache-http-palvelin jakaa domain-osoitteiden avulla, käyttäen sites-enabled-konfiguraatiota. Tietokantapalvelimena käytössä on erillinen mysql -palvelin, joka ylläpitää kauppojen tietokantoja samassa mysql-instanssissa.

2.2 Heikkoudet

Koska yksi palvelin pitää sisällään useita kauppoja, on erittäin hankalaa valmistautua nopealla reaktioajalla erilaisiin tiedossa oleviin rasitustilanteisiin front-endin, back-endin tai tietokannan osalta. Koska yhdellä palvelimella on monta sivustoa, ei ole realistista tuplata niitä perinteisellä palvelintuplauksella, jolloin paras tapai vastata erilaisiin rasitustilanteisiin olisi palvelimen tehon lisäämisen. Tämä ei kuitenkaan takaa toimintaa ainoastaan kyseiselle sivustolle ja ongelmatilanteessa se vie kaatuessaan alas muutkin palvelimen sivustot.

Koska sivusto tukeutuu vain yhteen palvelimeen, on kyseessä single point of failure -rakenne. Tämän takia,, jos mihin tahansa järjestelmän palvelimeen tulee odottamaton downtime, seurauksena on useiden muiden sivustojen kaatuminen. Koska järjestelmät

pyörivät itse käyttöjärjestelmän päällä, ei järjestelmän päivittäminen ole helppoa ja se saattaa aiheuttaa ongelmia jo käytössä olevissa ohjelmistoissa.

2.3 Vahvuudet

Koska käytettävät tekniikat ovat vanhoja ja testattuja, on hyvin epätodennäköistä, että ajaudutaan tilanteeseen, jossa kukaan ei osaa käyttää tai hahmottaa kyseistä arkkitehtuuria. Kyseinen tekniikka on myös hyvin yleinen, mistä syystä siihen on helppo löytää apua internetin ihmeellisestä maailmasta.

Arkkitehtuuri mahdollistaa useiden verkkokauppojen julkaisun samalla palvelimella, joten pienillä kustannuksilla on mahdollista pystyttää useita sivustoja, vaikka käytössä olisi vain yksi palvelin.

2.4 Järjestelmä

Vanhaa järjestelmää on ylläpidetty palveluntarjoajalta ostetuilla virtuaalipalvelimilla, joissa on asennettuna ubuntu 14.04. Näitä palvelimia on tarpeen mukaan lisätty, jotta yksittäiset palvelimet eivät ole täyttyneet turhista sivustoista. Palvelimet ovat toistensa kanssa lähiverkkohteydessä, jolla mahdollistetaan verkkojen sisäinen yhteys.

3 TAVOITTEET

Tässä luvussa käsitellään lopputuotoksen eri tavoitteita.

3.1 Redundancy

Halutun järjestelmän tulisi pystyä ongelmatilanteissakin jatkamaan toimintaansa automaattisesti. Jos esimerkiksi yksi järjestelmää ylläpitävistä palvelimista kaatuu, tulee järjestelmän jatkaa automaattisesti toimintaansa välittömästi tai hyvin pienellä aikajaksolla. Järjestelmä voidaan tarvittaessa laittaa toimimaan jopa eri palvelinsaleissa, jolloin järjestelmä pysyy toimintakunnossa, vaikka yksittäinen palvelinsali tippuisi verkosta. Ongelmatilanne ei myöskään saa vaikuttaa sivustojen dataan vaan kaiken datan tulee olla saatavilla jatkuvasti, vaikka yksittäinen palvelin sammuisikin. Järjestelmää ei ole suunniteltu kestämään useamman kuin yhden kriittisen palvelimen sammumista.

3.2 Down-Time-minimointi

Järjestelmän tulee kyetä päivittymään ja pysymään käytössä mahdollisimman pienellä down-timella. Esimerkiksi sivuston koodin päivittämisen ja testaamisen ei tarvitse näkyä asiakkaalle. Parhaassa tapauksessa myös databasen päivitys voitaisiin tehdä ilman, että sivusto on poissa käytöstä.

3.3 Reaktiokyky

Joissain tapauksissa asiakkaalla on etukäteen tieto sivustoon kohdistuvasta normaalia suuremmasta liikenteestä. Näissä tilanteissa järjestelmässä pitää kyetä skaalaamaan

sivuston kapasiteettia ylöspäin. Myös tietokannan tulisi kyetä reagoimaan kasvaneeseen kuormaan ilman katkoksia tai muita ongelmia.

3.4 Hinta

Järjestelmän tulee toimia hinnaltaan noin samanlaisessa ympäristössä, kuin käytössä oleva järjestelmä. Tämä tarkoittaa, että samaa palvelinkapasiteettia pyritään käyttämään tehokkaammin, ja näin saadaan kaikki mahdollinen hyöty ostetusta ympäristöstä.

4 MAHDOLLISUUDET

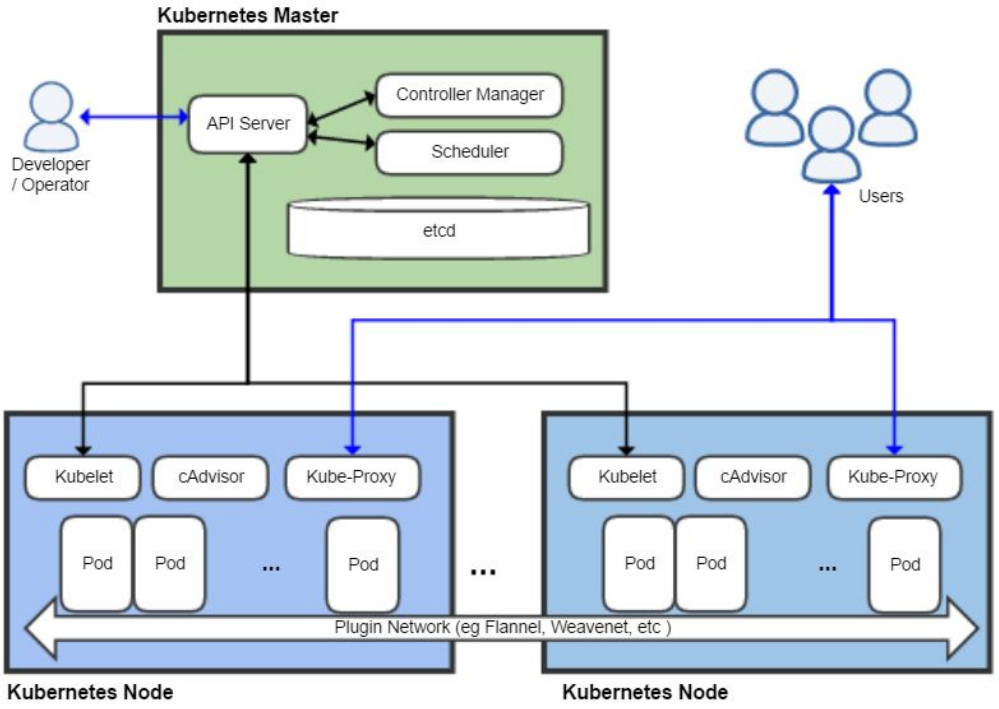
4.1 Vaihtoehdot

Tässä osiossa kerrotaan teknologiaratkaisujen eri mahdollisuuksista, varsinkin niiden eduista ja heikkouksista. Kaikki vaihtoehdot tukeutuvat Docker-järjestelmään, jonka pohjalle projekti on päätetty rakentaa. Docker valittiin, koska se on luotettava ja suosittu, tällä hetkellä nousussa oleva järjestelmä.

4.1.1 Kubernetes

Kubernetes on Googlen kehittämä kontitus-järjestelmä, joka on suunniteltu suurille käyttäjämäärille ja helppoa kehitystä silmälläpitäen. Se tukee useiden eri kontitustyökalujen kontteja, myös Docker-kontteja. Kubernetes sisältää miltei kaiken tarvittavan pakettien julkaisuun. Siihen on myös tarjolla laaja määrä hallintaan tarvittavia ohjelmistoja.

Kubernetes hoitaa käytännössä kaiken tehokkaasti ja ilman ongelmia. Se on myös käsitellyistä teknologioista toimintavarmin ja tehokkain. Suuret palveluntarjoajat tarjoavat valmiita klustereita, joihin on helppoa ja tehokasta julkaista omia konttejaan.
Sen ylläpito ja itse perustaminen vaatii kuitenkin tarkkaa ja melko laajaa perehtymistä. Se vaatii toimiakseen klusterin, jonka saa aikaan esimerkiksi etcd:llä. Varsinkin bare-metal-järjestelmässä sen perustaminen vaatii paljon konfigurointia. Tämän lisäksi järjestelmäosaajien kouluttaminen nollapisteestä on kalliimpaa, ja se vaatisi ulkopuolisen osaamisen ostamista. Kuvassa 1 on esitetty Kubernetesin toimintaperiaate.



KUVA1.Kuberneteksen toiminta malli
(https://upload.wikimedia.org/wikipedia/commons/b/be/Kubernetes.png)

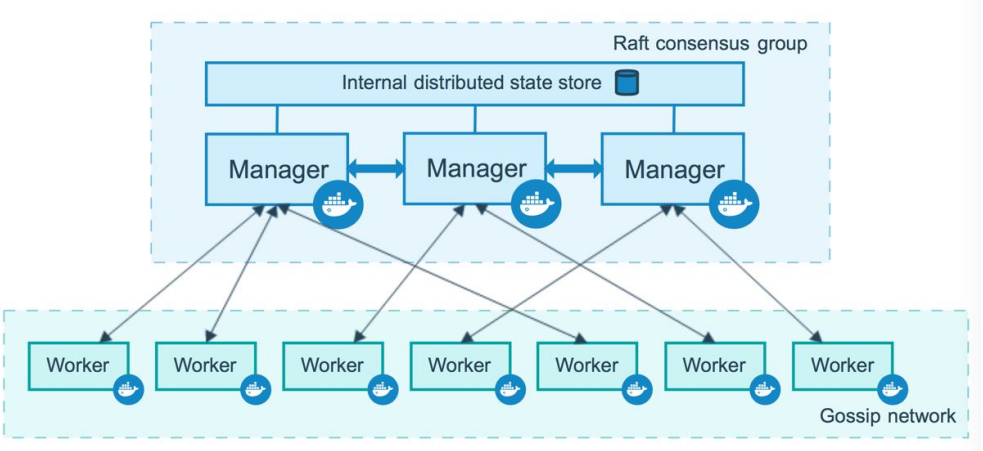
Kubernestes-ympäristön toimintaperiaate on melko yksinkertainen. Kaikki toiminta tukeutuu Kubernetes master -palvelimeen. Tämä palvelin tai palvelimet hoitavat konttien hallinnan ja ajoittamisen. Tapauksissa, joissa on käytössä useampia master -palvelimia, ne on yhdistetty etcd-klusterilla, jolloin niiden toiminta on indenttistä. Kun masterille on käsketty miten toimia, julkaisee se paketit nodeilleen, jotka hoitavat loadbalancer tehtävät. Eri kontit, eli podit, jakautuvat tasaisesti palvelimille, joihin käyttäjät ohjataan. Sekä konttien jakautuminen, loadbalancaaminen ja proxyäminen hoituu kuberneteksen tarjoamilla palveluilla. Tämä mahdollistaa erittäin tuotantovalmiin ja varman toiminnan järjestelmässä julkaistuille konteille.

4.1.2 Kontena

Kontena on Kuberneteksen tapainen Docker-konttien hallintajärjestelmä. Kontena ei ole yhtä suosittu kuin Kubernetes, mutta vaikuttaa käyttäjäystävällisemmältä. Suunnittelusta johtuen se on tarkoitettu paljon pienemmälle markkinalle, eli pelkästään docker-konttien hallintaan. Se on kuitenkin vielä melko nuori tekniikka ja sen pienuudesta johtuen sen hallintaan ei löydy hirvittävästi materiaalia. Tästä johtuen myös ongelmatilanteissa sen tuki täytyy hakea miltei suoraan valmistajalta. Tämä teknologia olisi ehkä ollut käyttötarkoitusta varten erittäinkin hyvä, mutta se ei ollut järkevä vaihtoehto, koska se ei ole vielä saanut osakseen tarvittavaa huomiota.

4.1.3 Docker-Swarm

Docker-Swarm on Dockerin itsensä kehittämä järjestelmä konttien julkaisuun. Se julkaistiin tuotantovalmiiksi vuoden 2015 lopulla, josta lähtien se on ollut kasvavassa suosiossa muiden konttienhallintajärjestelmien kanssa. Sen suurimpia etuja on sen täysin natiivi toimivuus dockerin omien komponenttien kanssa. Lisäksi se on erittäin helppokäyttöinen. Toisin kuin muut käsitellyt teknologiat Docker-Dwarm ei vaadi nodejen klusterointi,a vaan käyttää omaa apiaan sisäiseen kommunikointiin. Tämä mahdollistaa uusien koneiden lisäämisen erittäin helposti. Kuvassa 2 on esiteltynä suunniteltua arkkitehtuuria vastaava toimintamalli.



KUVA 2. Suunniteltua arkkitehtuuria vastaava docker-swarm toimintamalli (https://docs.docker.com/engine/swarm/images/swarm-diagram.png)

Toiminnaltaan Docker-Swarm vastaa suuresti kubernetestä, mutta itse toimintamalli on hieman erilainen. Docker-Swarm Manager -palvelimet hoitavat konttien hallinnan ja julkaisevat niitä haluttuihin sijainteihin. Docker-Swarm ei sisällä proxyjä eikä turhia hienouksia, vaan kaikki tällaiset asiat tulee hoitaa verkkoon julkaistavien palvelujen ja konttien avulla. Master-palvelimilla olisi hyvä olla pääsy samaan data storageen, jotta niiden toiminta pystyttäisiin pitämään mahdollisimman identtisenä. Tässä tapauksessa ei kuitenkaan ole tarvetta syvällisemmälle klusteroinnille, sillä Docker-Swarm hoitaa sisäisellä api:llaan kaikki siihen liittyvät yhteydet. Tämä mahdollistaa Docker-Swarmissa huomattavasti nopeamman järjestelmän rakentamisen kuin muissa vaihtoehdoissa.

4.2 Tietokanta

Sivuston tämänhetkinen tietokanta on Mysql 5.7, uuden tietokantaratkaisun on siis vähintäänkin tuettava tätä. Tästä johtuen järjestelmä on suunniteltu toimimaan MariaDB -tietokannassa. Näin ollen voidaan siirtyä Oraclen pikkuhiljaa alasajettavasta Mysql -kannasta avoimempaan ratkaisuun ilman kantamuokkauksia.

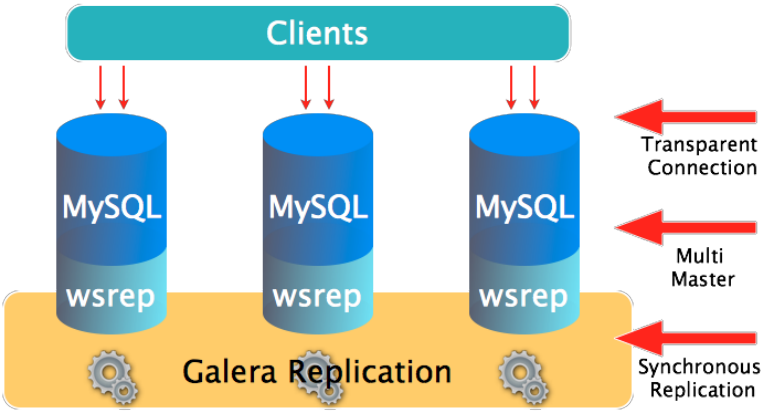
4.2.1 Perinteinen tietokanta

Perinteisellä tietokannalla tarkoitetaan yksittäistä palvelinta, johon on asennettu Mysql- tai MariaDB-tietokanta. Tällaisia tietokantoja on erittäin helppo pystyttää ja ylläpitää. Se on myös halpa, ja sen varmentaminen ja ongelmatilanteessa uudelleenpystytys on helppoa. Se ei kuitenkaan omaa minkäänlaista omaa kykyä toimia vikatilanteessa, jos jokin ohjelmiston osio, käyttöjärjestelmä, verkko tai palvelinrauta aiheuttaa vikatilanteen. Tällainen tilanne aiheuttaa käyttökatkoksen kaikissa sivustoissa, jotka tukeutuvat kyseisellä palvelimella oleviin tietokantoihin. Tälläinen tietokanta ei myöskään omaa minkäänlaista skaalautumiskykyä rasitustilanteissa, joten sillä täytyy olla varattuna paljon “turhaa” tehoa, jotta sen teho riittää kaikkina käyttöaikoina.

Näistä syistä perinteinen tietokantapalvelin ei tässä tilanteessa ole järkevä vaihtoehto, koska halutaan saavuttaa järjestelmä ilman single point of failurea.

4.2.2 Galera Cluster

Datan hallintaa ja sen saatavuutta voi myös parantaa klusteroimalla kyseisen tietokannan. Perinteisin valinta olisi asettaa vähintään kolme palvelinta galera-klusteriin ja hallita kantoja niiden välillä. Yleisin asetus tällaisessa tilanteessa on Master-Master, jolloin mihin tahansa kantaan voidaan tehdä lisäyksiä jotka vaikuttavat välittömästi kaikkiin kantoihin. Toimintavarmuus tällaisessa järjestelmässä on suuri ja sitä käytetäänkin erittäin raskaisiin ja käytettyihin tietokantoihin. Sen asennus on melko suoraviivaista ja galera, mariadb sekä mysql tarjoavat yksinkertaiset ja yksityiskohtaiset ohjeet klusterin luomiseen. Heikkoutena tässä tilanteessa on kuitenkin investoinnin suuruus. Jos tarvitaan kolme palvelinta pelkästään tietokantojen hallintaan, nousee aloituspalvelinten määrä huomattavasti. Ongelmia saattaa aiheuttaa myös palvelun jäykkyys. Vaikka se onkin helposti laajennettavissa, ei sen siirtäminen esimerkiksi toiselle palveluntarjoajalle ole täysin suoraviivaista. Se voi mahdollisessa datansiirtotilanteessa pakottaa tekemään siirron käsityönä tietokantadumppi kerrallaan. Lisäksi se vaatisi melko hyvää tietokantaosaamista sitä siirtäviltä henkilöiltä. Kuvassa 3 havainnollistetaan Galeran master-master -mallia.



KUVA 3. MySql Galera gluster 3 master malli (https://severalnines.com/sites/default/files/blog/node_5100/image1.png)

Galera-kluster mahdollistaa monenlaisia erilaisia tietokantarakenteita, joista yleisimpiä ovat MASTER-MASTER- sekä MASTER-SLAVE -mallit. Yllä kuvatussa MASTER-MASTER-mallissa vähintään kolme palvelinta asetetaan toimimaan yhdessä, jolloin muokkauksia voi tehdä mille tahansa palvelimelle. MASTER-SLAVE -mallissa sen sijaan muokkauksia voi tehdä vain yhdelle palvelimelle, mutta lukuja voi suorittaa miltä tahansa slave-palvelimelta. Kyseisiä palvelimia voi olla ohjelmallisesti rajoittamaton määrä. Tämän ratkaisun ongelma kuitenkin on yksittäinen master -palvelin, jonka kaatuessa mikään data ei ole käytettävissä.

4.2.3 Kontitettu tietokanta

Perinteinen tietokanta ei kykene haluttuun toimintaan. Galera Gluster on liian kallis vaihtoehto MASTER-MASTER-mallilla ja MASTER-SLAVE-mallissa on perinteisen kannan ongelmat. Tästä johtuen on etsittävä toinen ratkaisu, joka on kontitettu tietokanta.

Kontitetulla tietokannalla on mahdollista käyttää hyväkseen Galera-klusterin etuja ja paikata sen heikkoudet. Koska MASTER-MASTER-mallin käyttäminen vaatii tarkkoja osoitteita ja on hieman turhan epävarma kontittetussa ympäristössä, on järkevin vaihtoehto MASTER-SLAVE-mallin kontitettu tietokanta. Sen toimintavarmuus on samaa luokkaa perinteisen kontitetun tietokannan kanssa, mutta koska se voi ohjata liikennettä slaveilleen, se mahdollistaa rasitustilanteessa suuremmat lukunopeudet. Vaikka tässä tilanteessa kaikki toiminta tukeutuu yksittäiseen master-konttiin, se on helppo asettaa käynnistymään ongelmatilanteissa automaattisesti uudestaan. Näin ollen saavutetaan mahdollisimman pieni downtime. Tämä tekniikka ei mahdollista 100-prosenttista uptimeä, mutta ei myöskään vaadi käyttäjätoimenpiteitä yksittäisen palvelimen kaatuessa.

Kontitettu tietokanta helpottaa myös datan hallintaa. Se asetetaan kontteihin erilaisilla mounteilla, jolloin sen ansiosta on erittäin helppoa kerätä databasen data talteen siirtoa

tai varmuuskopiointia varten. Suurimpana ongelmana kontitetussa tietokannassa on kuitenkin sen varmistaminen, että data löytyy useasta palvelimesta samaan aikaan.

4.3 Datanhallinta

Jotta docker-kontit voivat julkaista ja tallentaa dataa käytön aikana, on palvelimilla oltava yhteinen data storage. Tämä mahdollistaa esimerkiksi konttien huomattavan pienentämisen tilanteissa, joissa ei ole tarpeellista, että jokainen kontti kantaa omaa kopiotaan käytettävistä tuotekuvista. Tämä kuitenkin samalla estää niiden julkaisun data storage -palvelimien ulkopuolella. Tietokannat eivät myöskään tukeudu tämän jälkeen vain yhteen palvelimeen, eikä järjestelmässä ole single point of failurea.

4.3.1 Unison

Unison on tiedoston synkronointityökalu, joka mahdollistaa tiedostojen synkronoinnin kahden eri palvelimen välillä. Tämä mahdollistaa tiedostojen ylläpidon useammassa eri sijainnissa. Kyseessä on kuitenkin database-tiedostojen ylläpito, mikä toisi ongelmia Unisonin toiminnan luonteesta johtuen. Koska Unison ei ole automaattisesti toimiva järjestelmä, on sen toiminta ajastettava. Jos tämä ajastus suoritettaisiin esimerkiksi minuutin välein, voisi se ongelmatilanteissa aiheuttaa tietokantadatan puuttumista useiden minuuttien edestä. Jotkin tapaukset voisivat myös aiheuttaa poikkeavuutta eri kansioiden välillä, mikä voisi estää datan synkronoinnin ja koko järjestelmän toiminnan. Tästä johtuen Unison ei ole haluttu vaihtoehto.

4.3.2 Järjestelmän ulkoinen data storage

Järjestelmän ulkoinen datastorage olisi toisilla palvelimilla oleva data storage, johon docker-kontit olisivat yhteydessä. Suosittu tekniikka tällaisessa tilanteessa on

ClusterHQ:n Flocker, joka mahdollistaa docker-volumeiden yhdistämisen useisiin erilaisiin data storage -järjestelmiin. Tämän tuotteen virallinen kehitys on kuitenkin jo hiipumassa, joten sen käyttö ei ole toivotuin vaihtoehto. Vaihtoehtoja ulkoiselle data storagelle olisi useita, mutta sellaisen perustaminen ja hallitseminen ei olisi kustannustehokasta. Se mahdollistaisi datan hallinnan helposti erillisessä järjestelmässä, jolloin sen varmuuskopiointi ja käsittely olisi paljon helpompaa. Estäviksi tekijöiksi järjestelmässä tulisi hinta sekä sitoutuminen kolmannen osapuolen palveluihin, mikätoisi datan hallintaan lisää rikkoutuvia osia sekä kohtia, joissa täytyisi luottaa kolmannelta osapuolelta saatavan tuen riittävyyteen.

4.3.3 GlusterFS

Hyvä kompromissi Unisonin ja ulkoisen data storagen välillä on käyttää palvelimien sisäistä levytilaa ja luoda niistä klusteroitu tallennustila. Tämä mahdollistaa yhden palvelimen poistumisen ilman että dataan on vaikutusta sekä datan käytön kaikilta klusterin palvelimilta. Tähän käytettävä palvelu on GlusterFS, jonka avulla voi muodostaa palvelinten välille oman swarmin. Tämän avulla jokainen jäsen voi päästä käsiksi yhteiseen data-kirjastoon milloin tahansa. Palvelu on jälleen kolmannen osapuolen tarjoama, mikä aiheuttaa lisää ongelmia, mutta koska sen toiminta on jo niin varmalla tasolla ja yhteisön tarjoama tuki niin kattava voi sitä käyttää kriittisissä tuotantoympäristöissä.

4.4 Liikenteen ohjaus

Koska kontitus-järjestelmä julkaisee samalle palvelimelle useita sivustoja, on niiden liikenne kyettävä ohjaamaan ulkoverkosta oikeaan kohteeseen. Tätä toiminnallisuutta varten on pystytettävä reverse proxy.

4.4.1 Nginx

Nginx on yksi tunnetuimmista ja käytetyimmistä proxyistä. Se on testauksissa osoittautunut aina nopeutensa puolesta kärkijoukkoon ja ssiinä on paljon hyödyllisiä ominaisuuksia, jotka antavat suuren joustavuuden eri toimintojen käytössä. Joidenkin pakettien ja automatisoinnin avulla olisi täysin mahdollista tehdä nginx-järjestelmä, joka toimisi tässä tilanteessa täydellisesti. Tämä kuitenkin vaatisi paljon tuotekehitystä, jotta saavutettaisiin tarpeeksi automatisoitu järjestelmä. Olisi melko helppoa luoda nginx:sän avulla järjestelmä, johon automatisoidusti lisättäisiin halutut muutokset. Tällä saataisiin aikaan tarvittavat toiminnallisuudet, jotta proxy osaisi suorittaa uudelleenohjauksen, mutta jos jostain syystä luotaisiin virheellinen konfiguraatiotiedosto, koko palvelun proxy olisi pois käytöstä. Tämän takia nginx järjestelmä ei ollut sopiva vaihtoehto.

4.4.2 apache

Apache reverse proxy vastaa teholtaan nginx-järjestelmää. Se ei kuitenkaan ole ominaisuuksiltaan aivan yhtä kykenevä. Tässä käyttötarkoituksessa myös apache proxyn ominaisuudet olisivat kuitenkin olleet riittävät. Apachen etuna nginx järjestelmään voisi myös pitää helpompaa konfiguraatiota, johtuen sen suppeammista ominaisuuksista. Samasta syystä kuin nginx proxy, eli liian suuresta riskistä konfiguraation vioittuessa, ei tätäkään koettu oikeaksi vaihtoehdoksi.

4.4.3 traefik

Traefik proxy on paljon tuntemattomampi ja pienempi tekijä kuin nginx tai apache. Se kuitenkin tarjoaa palvelua, joka on sopiva tällaiseen tarkoitukseen. Traefik proxy on norjalaisen yrityksen tekemä Open-Source ohjelmisto. Ohjelmisto tarjoaa automatisoitua real-time reverse proxya, joka on suunniteltu toimimaan docker ympäristössä. Tämä mahdollistaa uusien entryjen lisäämisen konfiguraatioon ilman

downtimeä, konfiguraatio tiedostojen muokkaamista tai uudelleen käynnistämistä. Traefikin teho ei ole samalla tasolla kuin nginx proxyn. Se on testien mukaan noin 85% nginx proxyn tehosta, joka on kuitenkin hyväksyttävä vaihtokauppa täysin automatisoidusta ja toimintavarmasta järjestelmästä.

5 DOCKER-KONTIT

Docker kontti on ikäänkuin kevennetty virtuaalipalvelin. Se sisältää vain pakollisen toiminnallisuuden tavoitteensa saavuttamiseen. Järkevin tapa kontittamisessa on konttien rakentaminen mahdollisimman pienissä osissa. Tämä mahdollistaa konttien mahdollisen tuplaamisen helposti ja tehokkaasti ilman että turhat osuudet järjestelmästä tuplaantuvat. Järjestelmän tässä versiossa ei vielä käytetä hyväksi mikrokontitusta vaan olemassa oleva verkkokauppa siirretään miltei kokonaisena konttiin. Tämä johtuu sivuston aikaisemmasta rakenteesta, joka on rakennettu php-pohjaiseksi ja yhdistää näkymää sekä backed toiminnallisuutta. Näin ollen nopein ja helpoin ratkaisu on koko verkkokaupan kontitus.

Alla esitetyissä kappaleissa perehdytään eri käytettyjen konttien DOCKERFILE tiedostoihin ja selitetään tarkemmin miksi jotkin valinnoista on tehty. Nämä valinnat on esitetty seuraavasti.

```
#!/usr/bin/env python
# coding=utf-8
```

Tiedoston alussa ilmoitetaan että kyseessä on python tiedosto. Tämä mahdollistaa tiedoston ajamisen suoraan unix-käyttöjärjestelmissä. Siinä ilmoitetaan myös tiedoston encodingin ovan utf-8, mikä mahdollistaa skandien ja muiden utf-8 merkkien käytön tiedostossa lähes kaikilla compilereillä.

```
import sys
```

Importoidaan sys-tiedosto pythonin oletuskirjastoista. Kirjastoa käytetään ohjelman lopussa sen ennenaikaiseen sulkemiseen.

```
print u"Tämä on pala python 2.7 koodia joka tulostaa tämän
lauseen."
```

Tehdään tuloste käyttäen pythonin tulostusfunktiota. Tuloste asetetaan myös “u”-merkinnällä unicode-muotoon, jotta voidaan tulostaa unicode-merkkejä.

```
sys.exit()
```

Ohjelmasta poistutaan. Tästä ei ole hyötyä, koska ohjelma loppuisi kuitenkin. Tämä on käytössä vain demonstraatiotarkoituksissa.

5.1 Verkkokauppa

```
FROM debian:stretch-slim
```

Dockerfilen alussa valitsemme mitä pohjaa käytämme. Tässä tilanteessa käytämme debian:stretch-slim versiota, joka on perus kooltaan vain parinkymmenen megan kokoinen. Debian on valittu sen pienen koon, tutun ympäristön ja laajan tuen takia.

```
# Locale
ENV LOCALE es_ES.UTF-8
# PHP Timezone
ENV TZ=Europe/Helsinki
```

Valitaan joitain ympäristömuuttujia, jotta käytettävän php:n toiminnot toimivat kyseisessä ympäristössä. Oikea locaali varmistaa, että toiminnassa ei ilmene kummallisia merkistövirheitä. Oikean aikavyöhykkeen valinta vaikuttaa esimerkiksi joidenkin rest-toiminnallisuuksien ajoittamiseen.

```
# Set repositories
RUN \
  echo "deb http://ftp.fi.debian.org/debian/ stretch main
non-free contrib\n" > /etc/apt/sources.list && \
  apt-get -qq update && apt-get -qqy upgrade
```

Tarvittavien pakettien asentamiseen vaaditaan jokin repository. Tässä asetetaan käyttöön suomen debian stretch -repositoryt. Tämän lisäksi rekisterit päivitetään ja tarkistetaan ohjelmistopäivitykset jotta epätodennäköiset mutta mahdolliset päivitykset saadaan mukaan asennukseen.

```
# Install some basic tools needed for deployment
RUN apt-get -yqq install \
  apt-utils \
```

```
  mysql-client \
  locales \
  wget
```

Käytetään debianin paketinhallintajärjestelmää apt-get ja tämän avulla asennetaan joitain toiminnan kannalta tärkeitä paketteja:Aplikaatio utilitieetit, mysql client tietokantaan yhdistämistä varten, Locales jotta järjestelmä tietää sijainnin ja ajan sekä osaa ajoittaa toimintansa, Wget paketti, jotta php voi käyttää wget toiminnallisuutta tiedostojen lataamiseen.

```
# Install locale
RUN \
  sed -i -e "s/# $LOCALE/$LOCALE/" /etc/locale.gen && \
  echo "LANG=$LOCALE">/etc/default/locale && \
  dpkg-reconfigure --frontend=noninteractive locales && \
  update-locale LANG=$LOCALE
```

Asetetaan localelle arvot aluksi asettamastamme muuttujasta.

```
# Configure Sury sources
RUN \
  apt-get -yqq install apt-transport-https lsb-release
ca-certificates && \
  wget -O /etc/apt/trusted.gpg.d/php.gpg
https://packages.sury.org/php/apt.gpg && \
  echo "deb https://packages.sury.org/php/ $(lsb_release -sc)
main" > /etc/apt/sources.list.d/php.list && \
```

Haetaan oikeat sertifikaatit, jotta voidaan varmistaa että ladatut php paketit ovat oikeat eivätkä kopioita. Tästä on hyötyä esimerkiksi tilanteessa, jossa käytettävä pakettipalvelin kaapataan.

```
  # Update repositories cache and distribution
  apt-get -qq update && apt-get -qqy upgrade
```

Päivitetään jo asennetut paketit. Tällä varmistetaan että sertifikaatit tulevat voimaan.

```
# Install PHP7 with Xdebug (dev environment)
RUN apt-get -yqq install \
```

```
  php7.0                \
  php7.0-curl           \
  php7.0-gd             \
  php7.0-mbstring       \
  php7.0-mcrypt         \
  php7.0-mysql          \
  php7.0-xml            \
  php7.0-zip            \
  php7.0-apcu           \
  php7.0-opcache        \
  php7.0-memcache       \
  php7.0-memcached      \
  php7.0-redis          \
  libapache2-mod-php7.0
```

Asennetaan tarvittavat php paketit, jotta verkkokauppa sivusto toimii halutusti. Tässä tapauksessa käytetään php 7.0 -pohjaisia paketteja.

```
# PHP Timezone
RUN \
  echo $TZ | tee /etc/timezone && \
  dpkg-reconfigure --frontend noninteractive tzdata && \
  echo "date.timezone = \"$TZ\";" >
/etc/php/7.0/apache2/conf.d/timezone.ini && \
  echo "date.timezone = \"$TZ\";" >
/etc/php/7.0/cli/conf.d/timezone.ini
```

Asetetaan aikavyöhykkeet php:lle.

```
# Install Apache web server.
RUN apt-get -yqq install apache2
```

Asennetaan apache web -server. Tämä toimii itse frontin toiminnan pohjana ja hostaa verkkosivun. Tässä käytetään apache2 serveriä, joka on suosituin apache serveri.

```
RUN phpdismod apcu opcache xdebug xhprof
```

Asetetaan php asetuksia, jotka vähentävät debuggausasetuksia. Tämä vähentää erilaisia debuggausilmoituksia, joka taas parantaa tietoturvaa erilaisissa ongelmatilanteissa.

```
RUN a2enmod     \
  access_compat         \
  actions               \
  alias                 \
  auth_basic            \
  authn_core            \
  authn_file            \
  authz_core            \
  authz_groupfile       \
  authz_host            \
  authz_user            \
  autoindex             \
  dir                   \
  env                   \
  expires               \
  filter                \
  headers               \
  mime                  \
  negotiation           \
  php7.0                \
  mpm_prefork           \
  reqtimeout            \
  rewrite               \
  setenvif              \
  status                \
  ssl
```

Asetetaan apache2 -ominaisuuksia käyttöön. Osa näistä on jo oletuksena käytössä, mutta valittu joukko kattaa kaikki halutut ominaisuudet mikäli oletusasetukset paketissa muuttuisivat. Näillä annetaan esimerkiksi .htaccess -tiedostolle oikeus uudelleenohjata sivustoja ja php:lle lupa tehdä haluttu sivustorakenne.

```
# # Cleanup some things.
RUN apt-get -q autoclean && \
  rm -rf /var/lib/apt/lists/*
```

Siivotaan paketeista turhaa dataa ja poistetaan turhia tiedostoja. Kaikki poistettu data muuttaa lopullisen paketin kokoa pienemmäksi.

```
RUN rm /etc/apache2/sites-enabled/000-default.conf
RUN touch /etc/apache2/sites-enabled/000-default.conf && \
echo "<VirtualHost *:80>" >>
/etc/apache2/sites-enabled/000-default.conf && \
echo '<Directory "/var/www/html">' >>
/etc/apache2/sites-enabled/000-default.conf && \
echo "AllowOverride All" >>
/etc/apache2/sites-enabled/000-default.conf && \
echo "</Directory>" >>
/etc/apache2/sites-enabled/000-default.conf && \
echo "ServerAdmin webmaster@prospercart.fi" >>
/etc/apache2/sites-enabled/000-default.conf && \
echo "DocumentRoot /var/www/html" >>
/etc/apache2/sites-enabled/000-default.conf && \
echo "ErrorLog ${APACHE_LOG_DIR}/error.log" >>
/etc/apache2/sites-enabled/000-default.conf && \
echo "CustomLog ${APACHE_LOG_DIR}/access.log combined" >>
/etc/apache2/sites-enabled/000-default.conf && \
echo "</VirtualHost>" >>
/etc/apache2/sites-enabled/000-default.conf
RUN echo ServerName localhost >> /etc/apache2/apache2.conf
```

Poistetaan apachen default-sites konfiguraatio ja asetetaan uusi oletuskonfiguraatio. Portiksi valitaan 80, koska salaus suoritetaan järjestelmässä jo palveluntarjoajan tasolla. Lisäksi varmistetaan, että apache tietää ajavansa itseään localhost- ympäristössä. Tämä mahdollistaa tarvittaessa localhostin kutsumisen.

```
RUN rm -rf /var/www/html
RUN mkdir /var/www/html
COPY prospercart/ /var/www/html/
RUN rm -rf /var/www/html/admin
RUN rm /var/www/html/general/dummy.php
RUN rm -rf /var/www/html/images
RUN mkdir /var/www/html/images
RUN chmod 777 -R /var/www/html/images
```

Poistetaan oletuksena oleva web-root -kansion sisältö. Kansioon kopioidaan tarpeellinen data eli verkkosivuston koodi. Tämän jälkeen poistetaan dummy.php tiedosto, joka yliajaa tietokannassa olevia tietoja verkkosivuston domainiin liittyen. Myös kuvat

poistetaan, koska joissain tapauksissa kuvia voi olla kymmeniä gigoja. Tämä pitää itse kaupan koon pienempänä, vaikkakin rajoittaa sen julkaisua muille kuin tiedosto clusterissa oleville palvelimille. Tämän ongelman voi myöhemmin korjata tekemällä erillisen kontin kuvien tarjoamiseen, jolloin kaikkien käytettävien palvelimien ei tarvitse olla datastoragessa. Kansio luodaan uudestaan ja koska käyttäjällä on kyky muokata siellä olevaa materiaalia on sen oikeudet myös asetettava siten, että muokkaus on mahdollista.

```
EXPOSE 80
CMD /usr/sbin/apache2ctl -D FOREGROUND
```

Lopuksi vielä käskemme portin 80 julkiseksi muille sisäisille konteille. Tämä mahdollistaa siihen sisäisen liikenteen kyseisessä portissa. Tämän jälkeen käynnistetään apachen palvelu etualalla, jolloin kontti jää käyntiin. Jos apachen käynnistys suoritettaisiin normaalisti, kontti sammuisi välittömästi sen käynnistyttyä ja olettaisi saaneensa ajon päätökseen.

Tämä rakenne valittiin verkkosivusto kontin julkaisuun. Samaa konttirakennetta voi käyttää sekä hallinta, kehitys että julkaisuosioon, näin ollen mahdollisimman moni osa on kehityksen alusta asti samaa kuin julkaisuversiossa.

5.2 Tietokanta

Tietokantana päädyttiin käyttämään mariadb- tietokantaa johtuen sen aktiivisemmasta ja avoimemmasta kehityksestä, kuin tällähetkellä käytössä ollut mysql. Käytössä päädyttiin bitnami:n tarjoamaan mariadb konttiin. Kyseinen kontti ei ole kovin monimutkainen ja suurin osa sen ominaisuuksista tukeutuu mariadb:n omiin ominaisuuksiin.

```
FROM bitnami/minideb-extras:jessie-r22
```

```
LABEL maintainer "Bitnami <containers@bitnami.com>"

ENV BITNAMI_PKG_CHMOD="-R g+rwX" \
    BITNAMI_PKG_EXTRA_DIRS="/bitnami/mariadb/conf" \
    HOME="/"

# Install required system packages and dependencies
RUN install_packages libaio1 libc6 libgcc1 libjemalloc1
libncurses5 libssl1.0.0 libstdc++6 libtinfo5 zlib1g
RUN bitnami-pkg unpack mariadb-10.1.29-0 --checksum
56736369e551e55eace7cb82c98b7956698fb015158a22b709498757f53e35
bb

COPY rootfs /

ENV ALLOW_EMPTY_PASSWORD="no" \
    BITNAMI_APP_NAME="mariadb" \
    BITNAMI_IMAGE_VERSION="10.1.29-r0" \
    MARIADB_DATABASE="" \
    MARIADB_MASTER_HOST="" \
    MARIADB_MASTER_PORT_NUMBER="" \
    MARIADB_MASTER_ROOT_PASSWORD="" \
    MARIADB_MASTER_ROOT_USER="" \
    MARIADB_PASSWORD="" \
    MARIADB_PORT_NUMBER="3306" \
    MARIADB_REPLICATION_MODE="" \
    MARIADB_REPLICATION_PASSWORD="" \
    MARIADB_REPLICATION_USER="" \
    MARIADB_ROOT_PASSWORD="" \
    MARIADB_ROOT_USER="root" \
    MARIADB_USER="" \

PATH="/opt/bitnami/mariadb/bin:/opt/bitnami/mariadb/sbin:$PATH
"

EXPOSE 3306

USER 1001
ENTRYPOINT ["/app-entrypoint.sh"]
CMD ["/run.sh"]
```

Kontti käyttää toimintaansa bitnamin omaa debian-konttia ja asettaa joitain mariadb-koukkuja joita voimme myöhemmin kutsua nostaesamme kontin pystyyn ja asettaessamme sille asetuksia. Bitnamin kontin etu itse tehtyyn konttiin on yhteisön tekemä testaus ja verkosta löytyvä tuki.

5.3 Proxy

Reverse proxynä käytetään Traefik proxyä. Se on suunniteltu toimimaan automaattisesti docker swarmin kanssa. Se ei nopeudeltaan vastaa nginx proxyä, mutta sen ei pitäisi suunnitellussa ympäristössä osoittautua ongelmaksi. Traefik valittiin, koska sen automatisointi halutussa ympäristössä onnistuu tehokkaasti. Tämä mahdollistaa, että uuden sivuston julkaisutilanteessa ei tarvita minkäänlaista lisäkonfiguraatiota, vaan kaikki tapahtuu automaattisesti.

6 DOCKER-COMPOSE

Docker-compose on useiden docker-konttien hallintaan käytetty järjestelmä. Se tukee yaml-muodossa olevia konfiguraatio-tiedostoja. Se toimii sekä itsenäisesti olevassa docker asennuksessa että docker swarm -ympäristössä. Sen avulla voi julkaista kontteja erilaisin rajoittein eri palvelimille ja asettaa niille jo käynnistyksen yhteydessä erilaisia kapasiteetti rajoituksia.

Kontit, jotka määrätään docker-compose tiedostossa, pystyttyvät palveluina (service) kun ne käynnistetään swarm ympäristössä. Jos docker-composea käytetään käynnistämisessä tavallisessa docker ympäristössä, se käynnistää ne kontteina.
Docker-compose tiedostojen esittelyyn käytetään samaa tekniikkaa kuin Dockerfilejen esittelyyn kappaleessa viisi.

6.1 Kauppa

```
version: '3'
```

Docker-compose tarvitsee tiedoston alussa tiedon minkä version pohjalta tiedosto on tehty. Tekohetkellä versio 3 on uusin, josta johtuen järjestelmä käyttää sitä.

```
services:
```

Aloitamme service listan, jossa kerrotaan eri käynnistettävän containerit.

```
  SIISTITEMP:
   image: 192.168.61.126:5000/SIISTITEMP
   build: ./store
   restart: always
```

Aloitamme ensimmäisen servicen. Aluksi kerrotaan missä imagea säilytetään, mutta kuitenkin rakennamme sen kansiosta. Koska imagen sijainti julistetaan, voidaan samalla

tiedostolla suorittaa julkaisu repositorioon ja jakaminen nodejen välillä. Tiedostossa asetetaan myös restar valinta, joka aiheuttaa kontin uudelleen käynnistymisen tilanteissa, joissa se jostain syystä kaatuu. Osoitteessa ja palvelun nimenä on tällä hetkellä SIISTITEMP. Tätä käytetään python-skriptissä ja sen tilalle korjataan käytetty domain muokattuna muotoon, jota voi käyttää nimenä repositoriossa. Tämä mahdollistaa kyseisen compose-tiedoston kopioimisen kaikkiin julkaistaviin kauppoihin.

```
   networks:
    - traefik_testnet
    - SIISTITEMP
   volumes:
    -
/gluster/WEBTEMPLATE/store/prospercart/images:/var/www/html/im
ages
```

Jotta kontti voi toimia muiden julkaistavien osien kanssa yhdessä, on sille asetettava joitain liitoksia. Networks kohta määrää mihin verkkoihin kontti yhdistyy. Tässä määrätään myös volume mountit, joita kontit käyttävät. Tässä tapauksessa tarvitsemme vain kuvakansiot jaetusta dataglusterista. Tämä mahdollistaa kuvien jättämisen pois itse kontista, jolloin niiden koko jää huomattavasti pienemmäksi. Tämä kuitenkin estää konttien pystyttämisen glusterfs-palvelinten ulkopuolella.

```
   deploy:
    replicas: 2
    labels:
     - "traefik.port=80"
     - "traefik.docker.network=traefik_testnet"
     - "traefik.frontend.rule=Host:TEMPLATE"
     - "traefik.backend.loadbalancer.sticky=true"
```

Jokaiselle kontille voi määrätä myös julkaisu asetukset deploy-asetuksella. Tässä tapausessa määrätään replikoita 2 kappaletta. Kauppa siis julkaistaan kahteen konttiin. Labels kohta antaa kontille arvoja, jotta traefikin automatisoitu julkaisu osaa käsitellä ohjauksen oikein. Julkaisu portti on 80 ja ohjaukseen käytettävä verkko traefik_testnet

on sama kuin verkko, joka annettiin kontille käyttöön. Host arvoon tulee kaupan domain, joka asetetaan pythonjulkaisu -skriptillä. Viimeinen arvo määrää loadbalancerin stickyksi, jolloin php osaa pitää arvonsa sivunvaihtojen yli. Näin voidaan välttyä esimerkiksi asiakkaan ostoskoritietojen katoamiselta sekä muilta odottamattomilta käytöksiltä.

```
admin_SIISTITEMP:
  image: 192.168.61.126:5000/SIISTITEMP
  build: ./store
  restart: always
  networks:
   - traefik_testnet
   - SIISTITEMP
  volumes:
   - /gluster/WEBTEMPLATE/store/prospercart:/var/www/html
   - /gluster/WEBTEMPLATE/store/log:/var/log/apache2:rw
  deploy:
   labels:
     - "traefik.port=80"
     - "traefik.docker.network=traefik_testnet"
     - "traefik.frontend.rule=Host:admin.TEMPLATE"
     - "traefik.backend.loadbalancer.sticky=true"
```

Admin-kontin toiminta on miltei identtinen kauppakontin kanssa. Se haetaan samasta osoitteesta ja se käyttää samoja verkkoja. Volumeissa se kuitenkin hakee koko koodin pelkkien kuvien sijaan. Sen lisäksi admin kopioi apachen logit debuggausta varten. Kontille asetetaan osoite admin subdoimainilla, jolloin on selkeää missä tilassa käyttäjä on.

```
mariadb-SIISTITEMP:
  image: bitnami/mariadb:10.1.28-r1
  restart: always
  volumes:
   - '/gluster/DBTEMPLATE:/bitnami'
  networks:
   - SIISTITEMP
```

```
    environment:
      - MARIADB_REPLICATION_MODE=master
      - MARIADB_REPLICATION_USER=repl_user
      - MARIADB_USER=my_user
      - MARIADB_PASSWORD=local
      - MARIADB_DATABASE=prospercart
```

Tietokanta-kontti hoitaa tietokannan ylläpidon. Tämä image haetaan bitnamin repositoirosta ja se tallentaa datansa glusteriin. Sille myös määrätään joitain arvoja enviorment variableiksi. Näiden avulla voidaan säätää kannan salasanat ja muut tarpeelliset tiedot ensikäynnistyksen yhteydessä.

```
  mariadb-slave-SIISTITEMP:
    image: 'bitnami/mariadb:10.1.28-r1'
    networks:
      - SIISTITEMP
    depends_on:
      - mariadb-SIISTITEMP
    environment:
      - MARIADB_REPLICATION_MODE=slave
      - MARIADB_REPLICATION_USER=repl_user
      - MARIADB_USER=my_user
      - MARIADB_PASSWORD=local
      - MARIADB_DATABASE=prospercart
      - MARIADB_MASTER_HOST=mysql
      - MARIADB_MASTER_PORT_NUMBER=3306
```

Tämä kontti tulee mariadb master -kontin slaveksi. Se kopioi master kontin datat itseensä ja tarjoaa niitä verkkosivuille. Tätä konttia voidaan monistaa mielin määrin, jolloin niitä voidaan tarvittaessa lisätä kuorman kasvaessa. Slave konttia ei kuitenkaan ole kytketty kansioon, joten se ei ole kykenevä tekemään muokkauksia kantaan. Tässä tapauksessa arvoiksi annetaan slave- ja replikaatio-käyttäjän tunnukset sekä kannan portti.

```
networks:
  traefik_testnet:
    external : true
  SIISTITEMP:
```

Tiedoston lopussa määritellään vielä käytettävät verkot ja mountit. Tässä tapauksessa käytämme pelkästään volume mountteja suoraan sijaintiin, joten on tarvetta määritellä vain verkot. Traefik_testnet on etukäteen luotu verkko, joten se on lisäasetuksella external. SIISTITEMP on vain tätä palvelua varten.

6.2 Proxy

Traefik proxy käynnistetään myös omasta compose-filestään. Niitä käynnistetään vain yksi stack ja monistetaan tarpeellinen määrä kontteja tarvittaville palvelimille.

```
version: '3'

services:
  proxy:
    image: traefik
    command: --docker --docker.swarmmode --docker.watch --web
```

Trafeik proxy haetaan traefikin repositorioista ja otetaan käyttöön. Tällä kertaa annetaan myös käynnistyksen yhteydessä tulevat komennot. Käskemme docker, koska toimimme docker-ympäristössä. Käytämme swarm modea, sillä haluamme traefikin valvovan jatkuvasti uusia kontteja ja haluamme käyttöön web-valvonnan.

```
    networks:
      - traefik-net
    ports:
      - "80:80"
      - "8080:8080"
```

Asetamme saman ulkoisen verkon, jota käytämme verkkokaupoissa jotta traefik saa niihin yhteyden. Tässä tapauksessa tulee myös julkaista portteja, jotta dockerin ulkopuoliset palvelut saavat yhteyden kontteihin.

```
    deploy:
      update_config:
```

```
        parallelism: 1
      mode: global
      placement:
        constraints:
          - node.labels.for == master
```

Tällä kertaa halutaan tietää millä palvelimilla reverse proxy -palvelut ovat. Tästä syystä asetamme deploy-asetuksissa parallelismin päälle ja placement-asetuksen laitteisiin joihin on asetettu label master. Näin voimme olla varmoja siitä, millä palvelimilla palvelu on.

```
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /dev/null:/traefik.toml

networks:
  traefik-net:
```

Lopuksi tulee vielä asettaa volumet. Tälle kontille tulee antaa käyttöön docker.sock -hakemisto, mikä ei kuitenkaan ole tietoturvallista, sillä se tarkoittaa että kontti saa käyttöönsä kaikki tiedot mitä docker tekee. Tästä johtuen on hyvin tärkeää että kontti, joka käyttää tätä ominaisuutta, on täysin luotettava. Tämän lisäksi luodaan traefik-net -verkko johon muut kontit yhdistävät.

7 YMPÄRISTÖN PYSTYTYS

Tässä kappaleessa kerrotaan järjestelmän rakennuksesta. Alkupisteenä on kolme kappaletta Ubuntu 16.04 -palvelimia. Parhaiden tulosten saamiseksi asennuksien tulisi olla fyysisesti eri palvelimilla, mutta kuitenkin samassa palvelinsalissa. Mikäli palvelimet eivät ole samassa salissa, voi se aiheuttaa ongelmia datan klusteroinnissa. Järjestelmän saa helposti toimimaan myös palvelinsalien yli, jolloin saavutetaan palvelinsali failsafe. Tämä kuitenkin vaatisi suurempia investointeja palvelimien kokoon ja tehoon konttien koon kasvaessa huomattavasti. Lisäksi tämä toisi suurempia riskejä, koska yksittäinen järjestelmä johtaisi useampien sivustojen ylläpitoa kuin tarpeellista. Shell skriptin esittelyyn käytetään samaa tekniikkaa kuin Dockerfilejen esittelyyn kappaleessa viisi.

7.1 Docker-Swarm

Asennus on suunniteltu alkavaksi tyhjältä pohjalta, eli asennuksessa on vain ubuntu ja sen perusjärjestelmäpaketit sekä ssh-paketit. Asennuksen soveltaminen esimerkiksi debian-ympäristöön ei vaadi muuta kuin oikea sertifikaatin ja repositorion valinnan.

```
sudo apt-get update && \
sudo apt-get -y install apt-transport-https ca-certificates
curl gnupg2 software-properties-common && \
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add - &&\
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu  $(lsb_release -cs)
stable" &&\
sudo apt-get update &&\
sudo apt-get -y install docker-ce &&\
sudo apt-get -y install python-pip &&\
sudo pip install docker-compose
```

Käytetään ubuntun apt-get paketinhallintajärjestelmää. Ensin haetaan päivitykset, jonka jälkeen lisätään oikeat sertifikaatit. Tämän jälkeen lisätään dockerin repositorio ja päivitetään rekisterit uudestaan. Tämän jälkeen asennetaan Dockerin CE versio sekä python pip. Docker-asennus ei vaadi muita toimenpiteitä, se on nopea ja yksinkertainen eikä kestä kauaa.

```
docker swarm init
```

Kun Docker on asennettu ja aloitetaan ensimmäisen palvelimen konfigurointi on aloitettava uusi swarm. Tämä komento aloittaa swarmin ja mahdollistaa muiden nodejen siihen liittymisen.

```
docker swarm join-token manager
```

Kun swarm on aloitettu, pyydetään palvelimelta sen swarm join token. Saadun komennon avulla muut palvelimet voivat liittyä kyseisen palvelimen swarmiin. Tässä tapauksessa on kyseessä master join -token, joten palvelin liittyy swarmiin masterina. Tämä mahdollistaa palvelujen hallitsemisen kaikilta palvelimilta. Tässä ympäristössä ei olisi välttämätöntä asettaa kaikkia palvelimia mastereiksi, mutta se ei kyseisessä ympäristössä tuo erikoista haittaakaan.

```
docker swarm join --token TOKENHASH IP-ADDRESS:PORT
```

Tämän jälkeen kahteen muuhun palvelimeen, jotka ovat saaneet saman docker perusasennuksen, voi antaa seuraavan komennon. Tällöin ne liittyvät swarmiin ja ovat valmiita vastaanottamaan komentoja.

```
docker node update --label-add for=master Nodename
```

Docker swarm-mode antaa mahdollisuuden antaa nodeilleen erilaisia tietoja. Tässä tapauksessa lisätään label master. Koska kaikki kolme nodea ovat master modessa, ei swarm osaa niitä helposti erotella. Tämä kuitenkin mahdollistaa docker-composen julkaisevan paketteja, jotka käynnistyvät vain tietyillä nodeilla.

```
sudo echo '{"insecure-registries":["192.168.61.126:5000"]}' |
sudo tee  /etc/docker/daemon.json
```

Jokaiselle palvelimelle tulee myös asettaa insecure-registry. Tämä komento sallii paikalliseen rekisteriin yhdistämisen, vaikka yhteys olisi salaamaton. Tässä tapauksessa tulee kuitenkin olla varma, että yhdistää oikeaan palvelimeen, eikä sitä missään tapauksessa kannata käyttää internetin yli.

```
docker run -d -p 5000:5000 --restart=always --name registry
registry:2
```

Tämän jälkeen tulee pystyttää registry. Tähän rekitseriin tallennetaan kaikkien kauppojen kontit, jotta palvelimet osaavat julkaista niitä.

Tämän jälkeen käynnistetään Traefik proxy palvelu, jonka jälkeen palvelinympäristön Docker Swarm -osio on valmis. Jos kaikki konteissa käytettävä data olisi laitettu suoraan registeryyn, eli kaupan dockerilessä ei olisi poistettu kuvatiedostoja, olisi järjestelmä nyt riittävä niiden julkaisuun. Se toimisi suurella määrällä worker nodeja ja pystyisi julkaisemaan swarmiin repositoriossaan olevia kontteja. Node-asennus tapahtuisi samalla tavalla master-asennukseen nähden, poislukien swarm init -tokenin master moden sijaan olevan worker modessa.

7.2 GlusterFS

GlusterFS on datanhallintajärjestelmä, jota käytetään tässä ratkaisussa. Se pystyy muodostamaan oman klusterin, jossa se jakaa datan. Gluster fs on helppo ja tehokas ratkaisu, jonka pystyttäminen on nopeaa. Tässä tapauksessa asennetaan GlusterFS asennuksen kaikille master koneille. Tämä mahdollistaa konttien julkaisun kaikissa klusterin palvelimissa ja mahdollistaa kuvien käyttämisen konteissa, joka pienentää niiden kokoa. Myös tietokanta asennetaan klusteriin, jolloin yksittäisen palvelimen kaatuessa tietokanta ja kauppa nousee automaattisesti takaisin käyttöön.

Seuraavat toiminnot tulee suorittaa kaikille käytössä oleville palvelimille.

```
sudo apt-get install -y software-properties-common
sudo add-apt-repository ppa:gluster/glusterfs-3.8
```

Jotta GlusterFS -asennus voidaan aloittaa, on jälleen haettava tarvittavat repositoriot. Tässä tapauksessa valittiin versio 3.8. Valinta suoritettiin johtuen kyseisen version tuoreista ominaisuuksista, mutta riittävästä vakaudesta sekä kehittäjän suosituksista valintahetkellä.

```
sudo apt-get update
sudo apt-get install -y glusterfs-server
sudo service glusterfs-server start
```

Otetaan uudet repositoriot käyttöön päivittämällä rekisterit. Tämän jälkeen asennetaan glusterfs-server. Kun asennus on suoritettu glusterfs ei tässä vaiheessa tarvitse enempää konfiguraatiota ja se voidaan käynnistää. Gluster fs -volumejen lisääminen on järkevintä mountatuille erillisille osioille. Tällä varmistetaan, että lisätty data ei täytä koko palvelinta.

```
sudo gluster peer probe HOSTNAME
```

Kaikille palvelimille tulee antaa toistensa hostnamet etc/hosts -tiedostoon. Kun ne on lisätty ja tarvittaessa netowrking service käynnistetty uudelleen voidaan palvelimet lisätä klusteriin. Tämä tulee tehdä vain kerran yhdellä palvelimella, jolloin muut palvelimet osaavat etsiä loput klusterin jäsenet automaattisesti.

```
gluster volume create gstorage disperse-data 4 redundancy 2
DockerMaster1:/glusterdata/bricka
DockerMaster2:/glusterdata/bricka
DockerMaster3:/glusterdata/bricka
DockerMaster1:/glusterdata/brickb
DockerMaster2:/glusterdata/brickb
DockerMaster3:/glusterdata/brickb
```

Tämä komento on yhdellä rivillä.

Tällä komennolla luodaan jaettu data storage kuudessa osassa. Jokaiselle palvelimelle asetetaan sekä brickA, että birckB. Neljä brickeistä on dataa ja kaksi jää varalle. Näin ollen mikä tahansa palvelimista voi tippua pois ja siitä huolimatta on käytössä kaikki tallennettu data. Tämä toimii ikään kuin raid 5 järjestelmänä, mutta verkon yli.

```
sudo mkdir /gluster
mount.glusterfs HOSTNAME:/gstorage /gluster
```

Kun gluster on luotu on se hyvä vielä mountata haluttuun sijaintiin. Tämä ei ole välttämätöntä, mutta se helpottaa datan hallintaa ja kansiorakenteissa.

/etc/fstab

```
HOSTNAME:/gstorage /storage glusterfs defaults,_netdev 0 0
```

/etc/rc.local

```
mount /storage
```

Jotta mountit pysyvät käynnistyksen jälkeen on seuraaviin tiedostoihin vielä lisättävä seuraavat rivit.

7.3 Julkaisujärjestelmä

Toimivan järjestelmän lisäksi tulee järjestelmän käytön olla yksinkertaista ja helppoa kenelle tahansa. Tästä syystä julkaisutoiminta tulee automatisoida. Tavoitteena on tehdä järjestelmä, joka julkaisee uuden kaupan automaattisesti nappia painamalla. Tämän jälkeen korvaamalla ja muokkaamalla luotuja tiedostoja voidaan muokata kauppa haluttuun muotoon.

Kaupan pystytyskoodi esitellään samalla tavalla kuin aikaisemmat koodit. Tarkemmat tiedot kappaleessa on esitetty kappaleessa viisi.

```
#!/usr/bin/env python
# coding=utf-8
import sys
```

```
import subprocess
import os

domain = sys.argv[1]
if "." not in domain:
    print "piste puuttuu? onko ihan oikee domain"
    sys.exit(0)
if len(domain) < 3:
    print "hanki parempi domain"
    sys.exit(0)
```

Skriptin alussa importoidaan tarvittavat kirjastot ja tarkastetaan, että annetut parametrit ovat järkeviä.

```
if not os.path.exists('/gluster/WEB%s' % domain) or not
os.path.exists('/gluster/DB%s' % domain):
    subprocess.check_output(['cp', '-R',
'/asiakkaat/TEMPLATE', '/asiakkaat/%s' % domain])
    subprocess.check_output(['cp', '-R',
'/asiakkaat/TEMPLATE', '/gluster/WEB%s' % domain])
    subprocess.check_output(['cp', '-R', '/gluster/TEMPLATE',
'/gluster/DB%s' % domain])
    else:
    print "Domain folder already in use or other stuff"
    sys.exit(0)
```

Tarkastetaan, että kaupan nimellä ei ole jo luotu kansioita. Mikäli ei ole, niin kopioidaan tarvittavat tiedostot oikeisiin sijainteihin.

```
filu = ""
with open('/gluster/WEB%s/store/prospercart/general/dummy.php'
% domain, "rb") as file:
    filu = file.read()
with open('/gluster/WEB%s/store/prospercart/general/dummy.php'
% domain, "wb") as file:
    file.write(filu.replace("TEMPLATE", domain))
```

Kun tarvittavat tiedostot on kopioitu, asetetaan tiedostosta löytyvä template kohta oikeaan arvoon. Tämän avulla Docker-Compose löytää oikeat tiedostot. Tässä kohdassa

muokataan kaupan käyttämää dummy- tiedostoa. Näin tietokannassa käytettävä data voi viitata julkaistavaan sivuun, mutta admin- sivusto osaa ohjata itsensä oikein.

```
with
open('/gluster/WEB%s/store/prospercart/general/config.php' %
domain, "rb") as file:
    filu = file.read()
with
open('/gluster/WEB%s/store/prospercart/general/config.php' %
domain, "wb") as file:
    file.write(filu.replace("TEMPLATE",
domain.lower().replace(".", "").replace("ä", "a").replace("ö",
"o")))

# docker-compose TEMPLATE KAUPANIMI
with open('/gluster/WEB%s/docker-compose.yml' % domain, "rb")
as file:
    filu = file.read()
with open('/gluster/WEB%s/docker-compose.yml' % domain, "wb")
as file:
    file.write(filu.replace("TEMPLATE",
domain).replace("SIISTITEMP", domain.lower().replace(".",
"").replace("ä", "a").replace("ö", "o")))
```

Samalla tavalla kuin dummy.php -tiedostoon tehdyt muokkaukset, tulee nämä asiat tehdä myös muihin käytettäviin tiedostoihin. config.php Sisältää tietokantayhteyteen tarvittavat tiedot, ja docker-compose hoitaa itse kaupan pystytyksen.

```
subprocess.check_output(['docker-compose', 'build'],
cwd='/asiakkaat/%s' % domain)
subprocess.check_output(['docker-compose', 'push'],
cwd='/asiakkaat/%s' % domain)
subprocess.check_output(['docker', 'stack', 'deploy',
'--compose-file', 'docker-compose.yml',
domain.lower().replace(".", "").replace("ä", "a").replace("ö",
"o")], cwd='/asiakkaat/%s' % domain)
```

Kun kaikki tarvittavat muutokset on tehty, tulee kontit vielä julkistaa repositiossa. Tämä mahdollistaa niiden käytön kaikissa repositorioon yhteydessä olevissa palvelimissa. Tämän lisäksi käyttäjällä tulee olle helppo keino työntää koodiin tehdyt muutokset suoraan julkaistaviin kauppoihin. Tätä varten on tehty yksinkertaistettu get pyyntöön perustuva järjestelmä, viimeistelyssä versiossa tulee siihen vielä lisätä salaus sekä käyttäjän tunnistus.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import subprocess
from flask import Flask

from flask_cors import CORS


app = Flask(__name__)
CORS(app)
```

Tehdään yksinkertainen flask rajapinta, jotta käyttäjä voi vain avaamalla sivun aloittaa sivuston päivityksen. Tiedoston alussa tuttuun tapaan haetaan tarvittavat kirjastot ja aloitetaan flask- applikaatio. Tässä käytetään myös CORS -kirjastoa, jotta voidaan tehdä cross site requestejä ilman ongelmia.

```
@app.route('/updatestore', methods=['GET'])
def hello():
    p = subprocess.Popen(["docker-compose", "build"],
cwd="/asiakkaat/kauppa1")
    p.wait()
    p = subprocess.Popen(["docker-compose", "push"],
cwd="/asiakkaat/kauppa1")
    p.wait()
    p = subprocess.Popen(["docker", "stack", "deploy",
"--compose-file", "docker-compose.yml", "kauppa1"],
cwd="/asiakkaat/kauppa1")
    p.wait()
    return "Kaupan päivitys on suoritettu odota noin minuutti"
```

Luodaan kaupan päivitykselle rajapinta, joka päivittää kaupan rakentamalla sen uudestaan. Tämä ei ole viimeistelty tapa, koska tämä vaatisi kaupan julkaisuun pääsyn sisäverkossa olevaan rajapintaan. Rajapintaa ei voi dockeroida, koska sen tulee käyttää komentoja suoraan host-palvelimelta ja tästä johtuen myös rajapinnan päivittäminen on hankalaa. Todennäköisesti lopullinen ratkaisu on rest rajapinta, johon postataan oikeat parametrit ja tunnukset, jotta oikea kauppa päivittyy. Tätä on helppo ylläpitää käyttäen jo järjestelmässä olemassa olevia asiakkaan tunnuksia.

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=88)
```

Lopuksi rajapinta julkaistaan portissa 88.

Jotta kaikki nykyisen järjestelmän toiminnallisuudetsaadaan myös uuteen järjestelmään, tulee vielä lisätä ssh yhteys. Tämä saavutetaan avaamalla admin-kontteihin open ssh ja avaamalla niiden käyttämät portit. Kontin asennuksen yhteydessä luotaisiin uusi ssh-käyttäjä ja traefik asetettaisiin ohjaamaan kyseistä liikennettä. Sen kautta olisi mahdollista antaa asiakkaalle mahdollisuus muokata koodia, sekä yhdistää mysql-kantaan ilman minkäänlaista kosketusta itse julkaisupalvelimille.

8 POHDINTA

Kontitus järjestelmän käyttö parantaa suuresti palvelimista saatavaa hyötyä pienellä vaivalla. Tämä ominaisuus, joka vastaa hieman nykypäivänä jo perinteistä palvelimen virtualisointia, on siihen verrattuna erittäin helppo ja joustava. Se pystyy nopeasti ja tehokkaasti luomaan ympäristöjä ja palveluita, jotka ovat laajennettavissa ja julkaistavissa lähes mille tahansa alustalle. Kontitus-järjestelmät ovat tällä hetkellä suosiossa johtuen internetin kasvamisesta ja jatkuvasta tarpeesta luotettavammille palveluille ja suuremmille käyttäjämäärille. Viimeisten vuosien aikana järjestelmät ovat myös tulleet tasolle, että niitä voidaan käyttää luotettavasti tavallisten yritysten toiminnassa ja ne eivät ole enää pelkästään suurten yritysten testiympäristöissä. Tämän on mahdollistanut erityisesti Docker, joka on viime vuosina tullut erittäin suosituksi ohjelmistotuottajien keskuudessa.

Julkaisujärjestelmissä on yrityksen tarjoama tuki hyvin tärkeää ja tästä johtuen joissain ympäristöissä ei Docker ole vielä oikea vaihtoehto. Jotta tällaisen järjestelmän julkaiseminen on järkevää, on yrityksellä oltava riittävä osaaminen ja tietotaito sen ylläpitämiseen. Lisäksi lisätyt toiminnallisuudet tuovat toiminnan varmistamiseen aina uutta riskiä. Mitä enemmän rikkoutuvia osia on, sitä todennäköisempää on rikkoutuminen, joka kuitenkin tässä tapauksessa mitätöityy lisättyjen failsafe ominaisuuksien takia. Tämä ei kuitenkaan ole ilmaista, vaikka samasta palvelin määrästä saadaankin enemmän tehoa rasitustilanteessa. Pienellä käytöllä olevat sivustot ovat todennäköisesti paljon halvempaa pitää vanhanaikaisella apache- palvelimella ilman ylimääräistä load balancia tai muuta vastaavaa järjestelmää.

Järjestelmän virtualisointi, tässä tapauksessa kontittaminen, on ollut käytössä julkaisujärjestelmissä erilaisissa muodoissa jo pitkän ajan. Nykypäivänä miltei kaikki palvelimet on jollain tavalla virtualisoitu. On hyvin harvinaista löytää palvelimia, jotka eivät ole joko suorita tai ole jollain tavalla virtualisoituja. Tämä mahdollistaa parhaan hyödyn saamisen ja tutkimusten mukaan ainakin yli 75% käytetyistä palvelimista on

virtualisoitu. Virtualisoidun palvelimen luominen ja muokkaaminen ei kuitenkaan ole yhtä dynaamista ja nopeaa kun docker-kontin. Tästä johtuen on järkevämpää ja halvempaa asettaa kaupat kontteihin ja julkaista niissä kuin asettaa jokainen kauppa omalle virtualisoidulle palvelimelleen. Koska käytössä ei ole omaa palvelinrautaa ei ole realistista olettaa nopeaa ja dynaamista virtuaalikoneiden hallintaa, jonka docker mahdollistaa. Tällä hetkellä markkinat näyttäisivät olevan siirtymässä kohti kontitus-järjestelmiä, johtuen niiden nopeammasta pystytyksestä ja riippumattomuudesta valmistaja kohtaisiin ominaisuuksiin. Tämän takia on hyvin kannattavaa pyrkiä sijoittamaan tässä nousuvaiheessa palvelujen kontittameseen ja hyödyntämään sen tuomia mahdollisuuksia, jotta käytettävät tekniikat pysyisivät kehityksen mukana.

Tässä tapauksessa tehokkain tapa olisi näiden tapojen jonkinlainen yhdistäminen:luoda järjestelmä, joka siirtää ongelmia tuottavat sivustot helposti ja nopeasti toisenlaiselle palvelimelle, jolloin sivusto saadaan tarvittaessa kestämään suurta rasitusta. Jokainen kontti vie hieman enemmän tehoa palvelimelta kuin pelkkä apachen hostaama kansio. Se on kuitenkin kykenevä paljon raskaampaan käyttöön, jolloin sen käyttö joissain kaupoissa on lähes pakollista tehokkaan ja vakaan toimintaympäristön takaamiseksi. Tämän voisi siis toteuttaa jonkinlaisena asiakkaalle myytävänä lisäpakettina, jolloin tuotteen hieman suurempi julkaisukustannus olisi oikeutettu.

Järjestelmä itsessään on jo julkaisukunnossa. Se käyttää tutkittuja ja luotettavia osia ja on tehty vastaamaan suosituksia ja standardeja, mutta vaatisi testausta ja benchmarkkausta, jotta saataisiin selville kuinka paljon rasitusta se realistisesti kestää. Olisi hyvä myös testata erilaisia onglematilanteita, joissa yksittäinen palvelinjärjestelmästä kaatuu tai vioittuu. Näin voitaisiin varmistaa, että järjestelmä on kokonaisuutena valmis tuotantoympäristöksi.

LÄHTEET

Kuberneteksen oma dokumentaatio (1.12.2017)
https://kubernetes.io

Traefikin github repository (1.12.2017)
https://github.com/containous/traefik

Bitnamin Docker-kontti (1.12.2017)
https://github.com/bitnami/bitnami-docker-mariadb

MariaDB:n oma dokumentaatio (1.12.2017)
https://mariadb.org/

Dockerin tarjoama dokumentaatio (1.12.2017)
https://docs.docker.com/

GlusterFS ja Docker Swarm blogi teksti (1.12.2017)
http://embaby.com/blog/using-glusterfs-docker-swarm-cluster/

Tutkimus palvelinten virtualisoinnista (1.12.2017)
http://techgenix.com/server-virtualization-trends-2017

Unisonin oma dokumentaatio (1.12.2017)
https://www.cis.upenn.edu/~bcpierce/unison

Nginx dokumentaatio (1.12.2017)

https://www.nginx.com/resources/admin-guide/reverse-proxy