jamk.fi

Kubernetes ja klusteroitava verkkosovellus

Jyri Sakkara

Opinnäytetyö
Marraskuu 2016
Tekniikan ja liikenteen ala
Ohjelmistotekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Sakkara, Jyri	Opinnäytetyö, AMK	11 2016
	Sivumäärä 48	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Kubernetes ja klusteroitava verkkosovellus

Tutkinto-ohjelma

Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)

Lappalainen-Kajan Tarja, Huotari Jouni

Toimeksiantaja(t)

Aitio Finland Oy

Tiivistelmä

Opinnäytetyössä tutustuttiin käyttöjärjestelmätasoiseen konttiperustaiseen virtualisointiin käyttäen hyväksi Docker ja Kubernetes ohjelmistoja. Docker on toteutus konttiperustaisesta virtualisoinnista, jonka suosio on lisääntynyt räjähdysmäisesti viime vuosien aikana. Kubernetes laajentaa Dockerin konttivirtualisointia tuomalla lisäksi monipalvelinympäristö asennukset ja helpot työkalut sovelluksien hallintaan klusteroidussa monipalvelinympäristössä.

Työssä rakennettiin kolmen palvelimen testiympäristö omalle tietokoneelle käyttäen hyväksi VirtualBox-virtuaalipalvelinalustaa. Virtuaalipalvelimiin asennettiin kolmen palvelimen Kubernetes-klusteri. Klusterin ominaisuuksiin tutustumista ja sen arvioimista varten muokattiin olemassa olevasta verkkosovelluksesta klusterointikelpoinen. Sovelluksesta luotiin asennus Kubernetes-klusteriin, jossa kolme kopioita sovelluksesta asentui hajautettuna kolmelle eri virtuaalipalvelimelle ja sovellus näkyi ulospäin vain yhtenä tavallisena verkkosovelluksena.

Työn tarkoituksena oli selvittää mikä on Kubernetes, mitä sillä voi tehdä ja miten sitä käytetään. Tämän lisäksi tarkoitus oli selvittää, mitä vaaditaan verkkosovellukselta, että sen voi asentaa Kubernetes-klusteriin. Toimeksiantajaa kiinnosti, onko Kubernetes toimiva ratkaisu ja onko siitä hyötyä toimeksiantajalle.

Docker sekä Kubernetes osoittautuivat toimiviksi ratkaisuiksi, joiden avulla konttiperustainen virtualisointi oli helppoa. Työn aikana saatujen käyttökokemusten perusteella arvioitiin, että toimeksiantajalle on hyötyä lisätä Kubernetes-klusteri palvelininfrastruktuuriinsa.

Avainsanat (asiasanat)

Kubernetes, docker, klusterointi, pilvi, java, virtualisointi, kontti

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Sakkara, Jyri	Bachelor's thesis	11 2016
		Language of publication: Finnish
	Number of pages 48	Permission for web publication: x

Title of publication
Kubernetes and clustered web application

Degree programme
Software Engineering

Supervisor(s)
Lappalainen-Kajan Tarja, Huotari Jouni

Assigned by
Aitio Finland Oy

Abstract

The purpose of this bachelor's thesis was to learn about and test container based virtualization by using Docker and Kubernetes platforms. Docker is an implementation of container based virtualization the popularity of which has sky rocketed in the last few years. Kubernetes extends the virtualization offered by Docker by adding support to distribute containers across multiple servers and easy to use tools for maintaining containers in a clustered environment.

For the thesis, three virtual servers were created by using VirtualBox platform and Kubernetes cluster was installed to these three virtual servers. For test and evaluating purposes an existing software was modified so that it could be run in the clustered environment. The software was deployed to Kubernetes so that it was replicated and clustered to three different servers and from outside it still behaved like a single server application.

The object of the thesis was to find out what Kubernetes is, what can be done with it and how can it be used. An additional objective was to figure out what is required from a web application to be able to be deployed to clustered environment. The assigner of the bachelor's thesis was interested in Kubernetes and wanted to know if it is a production ready solution and if it could provide any benefits to the assigner's business.

Docker and Kubernetes both proved out to be great solutions which made use of container based virtualization easy and from the usage experiences and test results gained from the thesis it was concluded that the assigner would benefit by adding Kubernetes cluster to existing server infrastructure.

Keywords/tags (subjects)

Docker, Kubernetes, container, virtualization, cluster, clustering, web, java

Miscellaneous

Sisältö

Kuviot

1 Työn lähtökohdat

1.1 Taustaa

Käyttöjärjestelmätason virtualisointi on yleistymässä tietotekniikan maailmassa. Se lupaa tehokuutta ja säästöjä palvelinympäristöjen infrastruktuuriin tavalliseen palvelinperustaiseen virtualisointiin verrattuna. Käyttöjärjestelmätason virtualisointialustoja on ilmestynyt jo useita, mutta vuonna 2014 julkaistu konttiperustainen (container) virtualisointialusta Docker on parissa vuodessa saavuttanut lähes monopoliaseman kilpailijoihinsa nähden. Klusteroiduissa monipalvelinympäristöissä Docker ei itsessään vielä riitä korvaamaan palvelinperustaisen virtualisoinnin klusterointiratkaisuja, mutta Dockerin rinnalle on kehittynyt monia Dockeria hyväksikäyttäviä järjestelmiä, joilla Docker voidaan hajauttaa monipalvelinympäristöön. Yksi näistä ratkaisuista on nimeltään Kubernetes, jonka taustalla on teknologiajätti Google. Dockerin ja Kuberneteksen avulla voidaan luoda hajautettu monipalvelinympäristö käyttöjärjestelmätason virtualisoinnille.

1.2 Toimeksiantaja

Aitio Finland Oy on ohjelmistoalan yritys, jonka toimintaan kuuluu omaa tuotekehitystä, erilaisia asiakasprojekteja, sovelluskehittäjien vuokraamista asiakkaiden omiin projekteihin ja koulutuksien järjestämistä. Yritys perustettiin vuonna 2003 ja se on kasvanut reilun kymmenen vuoden aikana yhden miehen yrityksestä 12 henkilöä työllistäväksi ohjelmistoyritykseksi, jonka liikevaihto on jo lähellä miljoonaa euroa.

Yrityksen toiminta on keskittynyt asiakkaiden toiminnan tehostamiseen tuomalla uusia ideoita ja toimittamalla tietojärjestelmiä asiakkaiden tarpeisiin.

1.3 Työn tavoitteet

Toimeksiantajalla on palvelinperustainen monipalvelinvirtualisointijärjestelmä, jossa pyöritetään lukuisia eri palveluja eri asiakkaille. Pääsääntöisesti yhden asiakkuuden yksi palvelu vaatii yhden virtualisoidun palvelimen, jonka päälle asiakkaan ratkaisu on rakennettu. Tällaisia virtuaalipalvelimia kertyy paljon ja niissä on paljon samankaltaisuutta, mikä vie turhia resursseja.

Opinnäytetyön tavoitteena oli tutustua konttipohjaiseen virtualisointimalliin ja tarkastella, voisiko nykyisiä infrastruktuuriratkaisuja täydentää konttiperustaisella virtualisoinnilla käyttäen hyväksi Kubernetes virtualisointijärjestelmää ja Docker-kontteja. Lisäksi selvitettiin, mitä vaaditaan verkkosovelluksilta, jotta niitä voi hallita Kubernetes-ympäristössä ja miten sovelluksien hallinta käytännössä tapahtuu. Toimeksiantajaa kiinnostaa, onko Kubernetes käyttökelpoinen valinta oikeaan tuotantokäyttöön ja voisiko sitä soveltaa nykyisten tai uusien palvelujen pyörittämiseen.

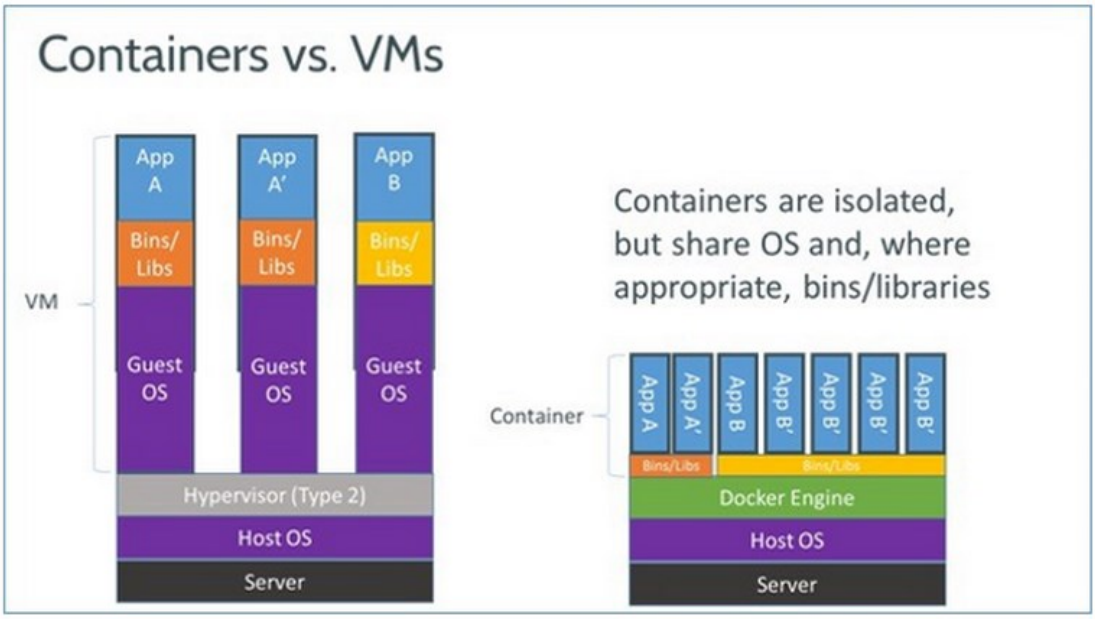
Kubernetes-infrastuktuuri toteutettiin omalle työasemalle virtuaalisena VirtualBox-työkalun avulla, mutta kuitenkin niin, että opinnäytetyön tuloksena syntyivät ohjeet, joilla se voitaisiin asentaan oikeaan tuotantoympäristöön.

2 Teoria

2.1 Konttipohjainen virtualisointi

Tavalliset virtualisointiratkaisut perustuvat pohjimmiltaan laitteiston ja sen päällä pyörivän käyttöjärjestelmän virtualisointiin. Jokaisessa virtuaalikoneessa on oma käyttöjärjestelmä, omat kirjastot sekä oma virtualisoitu laitteisto, jotka vaativat kaikki omat resurssinsa. Konttipohjainen virtualisointi eroaa siinä, että se pyörii yhden laitteiston ja yhden käyttöjärjestelmän päällä luoden virtualisoituja kontteja, jotka jakavat käyttöjärjestelmän resursseja. Virtuaalikonteilla ei ole omia laitteita eikä omaa kokonaista käyttöjärjestelmää, vaan ne jakavat isäntäkoneen käyttöjärjestel-

män ydintä sekä yleensä myös isäntäkoneen kirjastoja. Sen ansioista kontit ovat resurssivaatimuksiltaan kevyitä verrattuna tavallisiin virtuaalikoneisiin ja konttipohjaisella virtualisoinnilla pystyy ajamaan noin 2-3 kertaa enemmän sovelluksia kuin virtuaalipalvelimilla samanlaisessa ympäristössä. Virtuaaliratkaisuiden eroja on havainnollistettu kuviossa 1. (Vaughan-Nichols 2015.)



Kuvio 1, Virtuaalikone vs konttiarkkitehtuuri

Kontti on eristetty taso, joka sisältää yleensä vain yhden sovelluksen ja sen tarvitseman käyttöjärjestelmäympäristön minimaalisena toteutuksena. Kontti on siis pieni paketti, joka sisältää sovelluksen koko ympäristön, ja sitä voidaan helposti asentaa, replikoida sekä siirrellä eri palvelimien välillä ilman, että tarvitsee säätää jokaista asennusta erikseen eri ympäristöissä. Koska kontti on eristetty kokonaisuus ja sisältää kaiken tarvitsemansa, voidaan olla varmoja, että sovellus pyörii samalla tavalla eri ympäristöissä. Konttivirtualisoinnin käyttötarkoitus on pääasiassa sovellusten asennusympäristöjen virtualisointi siinä, missä tavallisessa virtualisoinnissa virtualisoidaan kokonaisia palvelimia. Konttivirtualisointi on parhaimmillaan, kun samasta sovelluksesta halutaan tehdä useita asennusympäristöjä, esimerkiksi testi- ja tuotantoympäristöt. (Vaughan-Nichols 2014.)

Ajatus käyttöjärjestelmätasolla suoritettavasta konttiperustaisesta virtualisoinnista ei ole uusi. Sen juuret juontavat aina 1970-luvulle saakka, jolloin chroot-

järjestelmäkutsu tuotiin osaksi Unix-käyttöjärjestelmiä. Chroot-järjestelmäkutsulla voitiin vaihtaa prosessin tiedostojärjestelmän juurihakemisto haluttuun polkuun ja näyttää prosessille vain haluttu osa tiedostojärjestelmästä. Tämä oli ensimmäinen edistysaskel käyttöjärjestelmätason virtualisointiin, ja se on osa nykypäivänkin virtualisointiratkaisuja. Seuraavat edistysaskeleet tulivat vasta 2000-luvulla, kun Linux-käyttöjärjestelmiin tuli tiedostojärjestelmien lisäksi mahdolliseksi eriyttää suoritettaville prosesseille muitakin käyttöjärjestelmän resursseja, kuten muistia ja verkko-osoitteita. Ensimmäisiä kokonaan kapseloituja konttipohjaisia ratkaisuja sovellusten suorittamiseen tuli 2000-luvun loppupuolella, mutta todellinen läpimurto tuli Dockerin myötä vuonna 2013, joka helppokäyttöisyydellään ja valmiilla ekosysteemillään toi konttiperustaisen virtualisoinnin suuren yleisön suosioon. (Rani 2016.)

2.2 Klusteri

Klusteri on joukko resursseja, jotka tekevät työtä saman yhteisen päämäärän eteen. Klusterissa resursseille jaetaan työtehtäviä ja resurssit jakavat tilatietoa keskenään. Klusterointia käytetään, kun halutaan enemmän suoritustehoa, kuin mitä yksi resurssi pystyy tarjoamaan, tai kun tilallinen sovellus halutaan monistaa useampaan osaan paremman saatavuuden takia. Esimerkiksi käyttökatkoa vaativat muutokset voidaan tehdä palveluihin resurssi kerrallaan, jolloin yksi resurssi on aina käytettävissä.

2.2.1 Kuormanjako

Jotta klusteri osaa jakaa kuormaa resurssien kesken, on sillä oltava kuormanjakaja. Kuormanjakaja on yhteyspiste sovellukseen, joka on tietoinen kaikista resursseista ja osaa jakaa pyyntöjä resurssien kesken. Kuormaa jaetaan erilaisten, eri käyttötarkoituksiin soveltuvien algoritmien perusteella. Resurssi voidaan valita esimerkiksi satunnaisesti, järjestyksessä tai jollakin painostuksella.

2.2.2 Klusteroiva verkkosovellus

Jotta verkkosovelluksia voidaan suorittaa klusteroituna niin, että kuormanjakaja jakaa yhteyksiä eri resurssien välillä, on sovellusten itsessään oltava myös klusteroitavia. Tämä tarkoittaa sitä, että sovelluksen on osattava toimia osana sovellusjoukkoa, aiheuttamatta ongelmia useiden asennusten yhdenaikaisuuden kanssa. Jos sovellus

käyttää tietokantaa, levyjärjestelmää tai muuta pysyvän tiedon varastointitapaa, on tietovaraston oltava yhteinen ja kaikkien sovellusresurssien käytettävissä. Jos sovelluksessa pidetään muistissa käyttäjien istuntoja, on pidettävä huoli, että käyttäjä ohjataan aina sellaiselle sovellusresurssille, josta käyttäjän istunto löytyy.

Käyttäjän istunnon löytymiseksi on kaksi ratkaisutapaa. Kuormanjakaja voidaan säätää niin, että se ohjaa aina samasta lähteestä tulevat pyynnöt samalle resurssille, jolloin resurssilla on aina tarjota käyttäjälle käyttäjän istunto. Tätä asetelmaa kutsutaan tahmeiden istuntojen (Sticky sessions) asetelmaksi. Tahmeiden istuntojen huono puoli on se, että jos se resurssi, jolle käyttäjää ohjattiin, menee vikatilaan, on käyttäjän istunto mennyttä ja käyttäjä joutuu aloittamaan istuntonsa alusta jollakin toisella resurssilla. Toinen ratkaisu ja saatavuuden kannalta parempi vaihtoehto on tehdä istuntojenhallinnasta yhteinen kaikille resursseille. Se voidaan saavuttaa jakamalla käyttäjien sessiot jokaisen sovellusresurssin kesken tai ulkoistamalla istuntojenhallinta klusterin ulkopuolelle, jolloin istunnot ovat kaikkien resurssien saatavilla. Tällöin yhden resurssin poistuminen ei vaikuta istuntojen saatavuuteen ja kuormanjakaja voi keskittyä vain jakamaan kuormaa saatavilla olevien resurssien välillä. Yhteisen istuntojenhallinnan kanssa ongelmaksi voi muodostua sovellusten päivittäminen. Klusteroidussa sovellusjoukossa ideaalinen tapa suorittaa sovellusten päivitys on päivittää vain osa sovellusresursseista kerrallaan ja pitää osa resursseista toiminnassa. Jos sovelluksilla on yhteinen istuntojenhallinta ja istunnonhallintaan tulee muutoksia päivityksien yhteydessä, voi sovellus ajautua virhetilanteeseen kun ohjataan käyttäjä vanhan version sovellusresurssilta päivitettyyn versioon resurssista.

3 Tekniikat

3.1 Docker

3.1.1 Yleistä

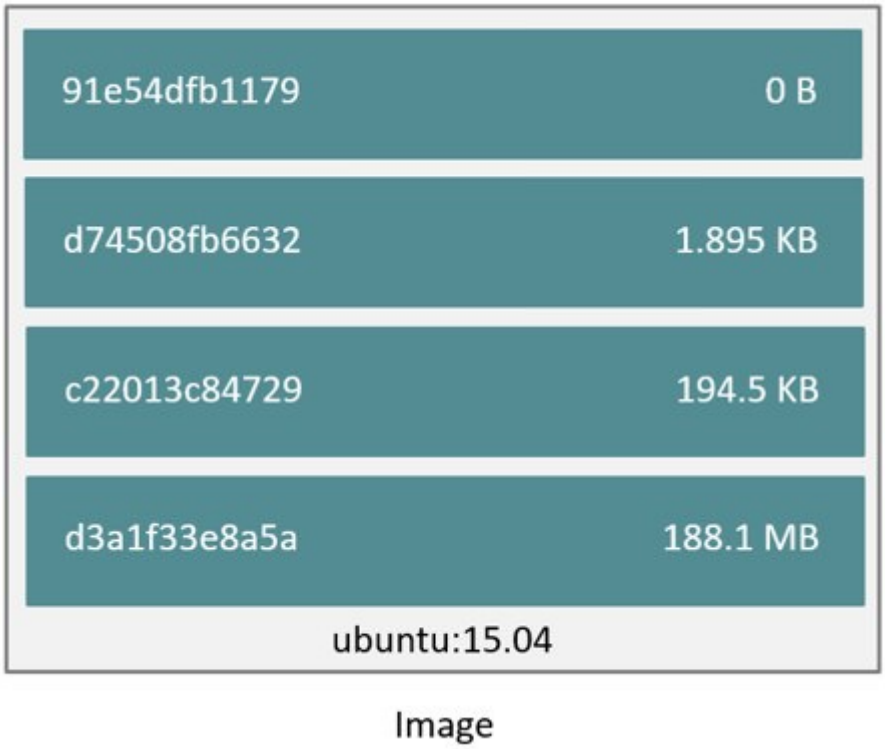
Docker on toteutus konttipohjaisesta virtualisoinnista, joka tarjoaa helppokäyttöiset työkalut konttien luomiseen, asentamiseen, ajamiseen, versiontiin ja jakamiseen. Docker käyttää konttien pyörittämiseen Linux-käyttöjärjestelmäytimen nimiavaruus (namespace) ja kontrolliryhmä (cgroups) -ominaisuuksia, joilla voidaan eristää Linuxin

prosesseja toisistaan ja sallia prosesseille tietty kapseloitu näkymä käyttöjärjestelmän resursseista. Docker pyörii Linux-käyttöjärjestelmän taustalla ja tarjoaa kevyen ja nopean konttiympäristön sovellusten pyörittämiseen. (Wilson 2014.)

Docker julkaistiin vuonna 2013 avoimen lähdekoodin sovelluksena Github verkkopalveluun dotCloud-yrityksen toimesta, ja se saavutti nopeasti suuren suosion kehittäjien keskuudessa. Vuoden sisällä julkaisusta isot yritykset kuten Red Hat ja Amazon alkoivat tukemaan Dockeria omissa järjestelmissään, ja tähän päivään mennessä Dockeria on ladattu yli sata miljoonaa kertaa. (Brief history of Docker containers n.d.) Alkuvuonna 2016 Dockerin koodikannasta tehdyn analyysin mukaan Dockerin kehityksen taustalla Dockerin omien työntekijöiden lisäksi vaikuttaa monia isoja teknologia-alan yrityksiä kuten Cisco, Google, Huawei, IBM, Microsoft ja Red Hat. (Porterie 2016.)

3.1.2 Docker-levykuva

Dockerissa kontin sisältö määritellään Docker-levykuvana (Docker image). Levykuvasta voidaan käynnistää kontteja suoritukseen ja levykuvia voidaan julkaista muille ladattavaksi. Levykuva on kerrosmainen rakenne, joista jokainen kerros on oma levykuvansa (ks. Kuvio 2). Levykuvan alimmaisena kerroksena on käyttöjärjestelmäkuva, jonka päälle kasataan muita kerroksia, kunnes kontti sisältää kaiken tarvittavan. Alempien kerrosten levykuviin viitataan kirjoitussuojatussa tilassa ja niitä ei voi muokata muissa kerroksissa. Sen sijaan muutokset ja päivitykset aiempiin kerroksiin määritellään uuteen kerrokseen. Uusi kerros sisältää vain tiedon muuttuneista asioista, ja vanhat kerrokset pysyvät uudelleen käytettävinä, mikä on tärkeää, sillä alempia kerroksia jaetaan usein monen levykuvan kesken. Tämän ansiosta Docker-kuvat pysyvät pieninä ja niiden käsittely nopeana. Myöskin Docker-kuvien määrittelytiedostot pysyvät pieninä ja selkeinä, koska niiden määrittely alkaa aina edellisestä kerroksesta, eikä samaa tarvitse toistaa uudestaan.



Kuvio 2, Esimerkki Docker-levykuvan kerroksista

Kuviossa 2 on esimerkki Ubuntun levykuvasta, joka koostuu 4 kerroksesta. Alimmainen pohjakerros on tilantarpeeltaan suurin ja muut kerrokset verrattain pieniä, koska ne sisältävät vain tiedon muuttuneista asioista.

Oman sovelluksen levykuvan kerrokset voisivat olla seuraavanlaisia:

4	Sovellus	Laajentaa Tomcat-kuvaa asentamalla sovelluksen.
3	Tomcat	Laajentaa Java-kuvaa asentamalla Tomcat-sovelluksen.
2	Java	Laajentaa pohjalevykuvaa asentamalla javan.
1	Ubuntu	Pohjalevykuva

Konttia suorittaessa Docker tallentaa jokaisen kerroksen omana levykuvanaan paikalliseen tiedostojärjestelmään, ja niitä voidaan jakaa eri konttien kesken. Jos samalla Docker-palvelimella haluttaisiin suorittaa konteissa kahta eri Java-sovellusta Tomcat-sovelluspalvelimen avulla, voisivat molempien levykuvat rakentua Tomcat-levykuvan päälle. Ensimmäistä konteista käynnistäessä Docker lataa kaikki levykuvat paikallisel-le levylle ja käynnistää niiden avulla kontin. Kun toinen konteista lähtee suoritukseen, on edellisestä kontista tallennettu jo kaikki Tomcat-levykuvan sisältämät kerrokset

tiedostojärjestelmään. Toista konttia varten ei tarvitse enää tallentaa kuin uuden sovelluksen kerros. Näillä ratkaisuilla levytilan tarve pysyy pienenä ja kerrokset uudelleen käytettävinä. (Understand images, containers and storage drivers n.d.)

3.1.3 Docker-levykuvien määrittelytiedosto

Docker-levykuvien määrittelytiedosto on nimeltään Dockerfile. Dockerfile on tiedosto, jossa rakennetaan halutun kaltainen levykuva komento kerrallaan käyttäen omaa levykuvien määrittelysyntaksia. Yksinkertaisimmillaan määrittelytiedostoon kirjataan käskyjä, joilla kopioidaan tiedostoja tietokoneen levyjärjestelmän ja kontin välillä, muokataan tiedostoja kontin sisällä, ajetaan komentoja tai säädellään kontin omia asetuksia. Näitä toistetaan, kunnes levykuva on halutun kaltainen. (Ks. kuvio 3)

```
FROM ubuntu
MAINTAINER Kimbro Staken

RUN apt-get install -y software-properties-common python
RUN add-apt-repository ppa:chris-lea/node.js
RUN echo "deb http://us.archive.ubuntu.com/ubuntu/ precise universe" >> /etc/apt/sources.list
RUN apt-get update
RUN apt-get install -y nodejs
#RUN apt-get install -y nodejs=0.6.12~dfsg1-1ubuntu1
RUN mkdir /var/www

ADD app.js /var/www/app.js

CMD ["/usr/bin/node", "/var/www/app.js"]
```

Kuvio 3, Dockerfile esimerkki nodejs-sovelluskontista

3.1.4 Docker data volume

Dockerilla suoritettavat kontit ovat muistinvaraisia, ja kaikki tieto, jota kontin sisälle tallennetaan, häviää, kun kontti sammutetaan. Jotta konteilla voidaan suorittaa sovelluksia joilla on tarve tallentaa asioita tiedostojärjestelmään, pitää konttiin liittää linkki pysyvään tiedostojärjestelmään. Tällaista kutsutaan Docker data-volumeksi. (Understand images, containers and storage drivers n.d.)

Docker data volume on Docker-isäntäkoneen tiedostojärjestelmässä kansio tai tiedosto, joka linkitetään suoraan kontin tiedostojärjestelmään. Yhteen konttiin voi liit-

tää useita data volumeita, ja useat kontit voivat linkittyä samaan data volumeen. (Understand images, containers and storage drivers n.d.)

3.1.5 Docker ja verkot

Docker tarjoaa konteille valmiiksi asennettuna 3 eri tietoverkkoa, johon ne voidaan liittää. Nämä verkot ovat nimeltään bridge, none ja host. (Understand Docker container networks n.d.)

Bridge-verkko on Docker-isäntäkoneen sisällä oleva virtuaaliverkko, johon kaikki kontit liitetään, jollei toisin määritetä. Tämä mahdollistaa kommunikaation konttien välillä, mutta eristää kontit ulkopuolelta tuleville yhteyksille. None-asetus jättää kontilta verkon tyhjäksi ja eristää kontin kaikista muista konteista. Host-asetus jakaa isäntäkoneen tietoverkot kontin kanssa. (Understand Docker container networks n.d.)

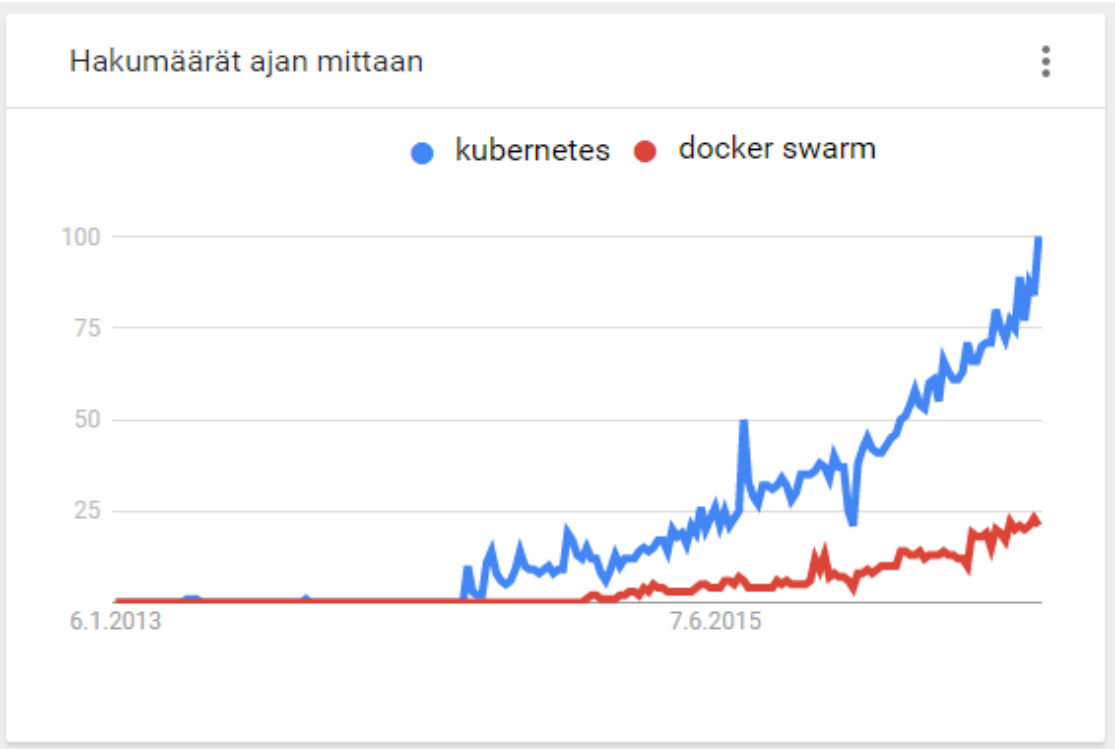
Kolmen vakioverkon lisäksi Dockeriin voidaan luoda omia verkkoja ja liittää niihin kontteja, jos halutaan eristää tietyt kontit muista konteista, mutta mahdollistaa niille kuitenkin kommunikaatio keskenään. Huolimatta siitä, mihin verkkoon kontti kuuluu, voidaan kontille määrittää portteja, joita se julkaisee ulospäin, joihin Docker isäntäkoneelta pääsee käsiksi. (Understand Docker container networks n.d.)

3.1.6 Docker-rekisteri

Koska levykuvien rakentaminen perustuu olemassa olevien kuvien päälle rakentamiseen ja kuvien jakamiseen sekä uudelleenkäyttöön, on ollut tärkeää, että levykuvia saa noudettua ja lisättyä helposti Docker-asennuksiin. Tätä varten on kehitetty Docker-rekisteri, joka toimii varastona valmiille levykuville. Dockerilla on julkinen rekisteri, Docker Hub, josta löytyy tuhansia valmiita kuvia, virallisia ja epävirallisia, joita voi käyttää osana omaa levykuvaa tai suorittaa Dockerilla sellaisenaan. Kontteja käynnistäessä Docker etsii levykuvia ensin lokaalista tiedostojärjestelmästä ja sen jälkeen yrittää ladata niitä Docker Hubista. Docker Hubin lisäksi Docker-rekistereitä voi asentaa omille palvelimilleen ja käyttää niitä osana Docker-asennusta. (Docker Overview n.d)

3.1.7 Docker-klusteri

Docker ei itsessään tarjoa mahdollisuutta suorittaa kontteja usean palvelimen välillä, vaan sen rooli on luoda yhdelle palvelimelle konttiympäristö. Tämä ei kuitenkaan ole ihanteellinen tilanne, ja tuotantokäytössä on selkeä tarve jakaa työtaakkaa ja saatavuutta pois yhden palvelimen varasta. Tätä varten on tullut Dockerista erillisiä kolmannen osapuolen järjestelmiä, jotka käyttävät Dockeria pohjalla, mutta laajentavat siihen monipalvelintuen ja siihen liittyviä työkaluja. Yleisimmät näistä ovat Docker Swarm ja Kubernetes, joista Kubernetes oli Googlen hakumäärien perusteella kirjoitushetkellä selkeästi suositumpi, kuten kuviosta 4 voidaan havaita.



Kuvio 4, Kubernetes vs Docker Swarm Google Trends haulla 10.10.2016

3.2 Kubernetes

3.2.1 Yleistä

Kubernetes on Googlen perustama avoimen lähdekoodin alusta, jolla automatisoidaan Docker-sovelluskonttien hallintaa klusteriympäristössä. Kubernetes tarjoaa helppokäyttöisen rajapinnan, jolla voidaan hallita sovelluskonttien asentamista, skaa-

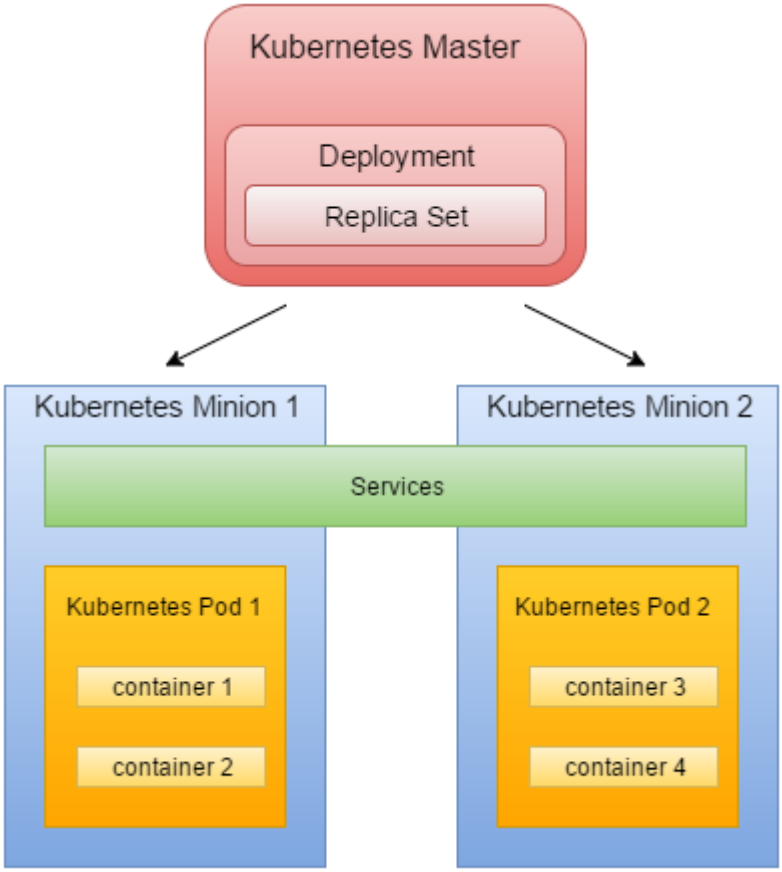
lausta, päivittämistä, monitorointia ja resurssien jakamista klusteriympäristössä. (Whatisk8s n.d.)

Kubernetes tarjoaa monta eri tapaa toteuttaa klusterin, ja sitä voidaan suorittaa lähes missä vain missä Dockeriakin. Käytännössä se vaatii vain palvelimen, jossa pyörii Linux-käyttöjärjestelmä, joten sen voi asentaa lähes minkälaiseen palvelinympäristöön tahansa. Sitä voidaan asentaa niin virtuaalipalvelimille, oikeisiin fyysisiin palvelimiin kuin useimpiin kaupallisiin pilviympäristöihin. Jotkin julkiset pilvipalvelut, kuten Amazon Web Services, Google Container Engine ja Microsoft Azure tarjoavat jopa valmiita Kubernetes-klustereita valmiiksi asennettuna. (Kubernetes getting started guide n.d.)

Kuberneteksessa pääajatus on Docker-konttien hallinta, mutta kontit ovat vain pieni osa kokonaisuutta. Kuberneteksen avulla voidaan skaalata ja hajauttaa kontteja usealle palvelimelle ja sen avulla voidaan hoitaa konttien päivitykset automaattisesti. Kubernetes valvoo suoritettavia kontteja ja korjailee tilannetta automaattisesti, jos kontteja joutuu vikatilaan. Se tarjoaa työkalut konttien hallintaa ja valvomiseen ja rapapinnat konttien ohjelmallisen käsittelyyn. (Kubernetes getting started guide n.d.)

3.2.2 Kubernetes-käsitteet

Kubernetes tuo Docker-käsitteiden lisäksi mukanaan oman ekosysteeminsä omine käsitteineen ja työkaluineen. Kuviossa 5 on kuvattu Kubernetes-käsitteiden hierarkiaa.



Kuvio 5, Kubernetes käsitekaavio

Kubernetes Pod (kapseli)

Kuberneteksessa ei suoraan hallita sovelluskontteja, vaan sovelluskontit kapseloidaan Kubernetes Podeihin eli kapseleihin. Kapseli on Kubernetes-klusterin perusyksikkö, joita Kuberneteksella hallitaan ja joiden ympärille toiminta on rakennettu. Kapselit voivat sisältää yhden tai useamman Docker-kontin sillä periaatteella, että yksi sovelluskokonaisuus on yhdessä kapselissa. (User Guide - Pods n.d.)

Replica Set (sovellusryhmä)

Sovellusryhmä on määritys, jolla määritellään, kuinka moneen kopioon kapseli replikoidaan. Sovelluksen kapselista muodostetaan yhtenäinen ryhmä kapseleita ja ryhmälle asetetaan säännöksi, kuinka monta kopiota sovelluksen kapselista pitää pysyä käynnissä Kubernetes-klusterissa. Kubernetes luo ja poistaa kapseleita dynaamisesti eri toimintojen yhteydessä ja sovellusryhmä pitää huolen siitä, että mitä tahansa tapahtuukin, pysyy kapseleita aina käynnissä haluttu määrä. Kapseleiden määrä voi-

daan asettaa ryhmälle minimi- ja maksimiarvoina tai tarkkana arvona. (User Guide - Replica Sets n.d)

Kubernetes Deployment (asennusmääritys)

Asennusmääritykseen kirjataan kaikki sovellusasennuksen tiedot, kuten sovellusryhmä ja sen koko, suoritettava kapseli ja sen sisältämien konttien levykuvat. Asennusmäärityksellä kuvataan koko sovelluskokonaisuus replikointineen ja versioineen, joka kuuluu olla suorituksessa. Se reagoi asetusten muutoksiin ja päivittää muutokset tarvittaessa suorituksessa olevaan sovelluskokonaisuuteen. Sillä mahdollistetaan kapseleiden ja sovellusryhmien muutoksienhallinta suorituksen aikana. Kun asennusmäärittely lisätään Kubernetekseen, luo se samalla sovellusryhmän sekä kapselit. Asennusmäärittelystä voi esimerkiksi vaihtaa kapselin sisällä olevaa kontin sovellusversiota toiseen ja silloin asennusmäärittely vaihtaa suorituksessa olevat kapselit uusiin versioihin ilman, että kapseleiden tarjoamaan palveluun tulee käyttökatkoa. (User Guide - Deployments n.d.)

Asennusmäärittely vaihtaa kapselit niin, että se luo muuttuneita kapseleita varten toisen uuden sovellusryhmän ja luo uuteen ryhmään uusia kapseleita samalla poistaen vanhoja kapseleita vanhasta ryhmästä. Asennusobjekti pitää huolen, että päivityksen aikana kapseleiden määrä noudattaa sitä, mitä sovellusryhmälle oli asetettu säännöksi. Kun kaikki uudet kapselit on luotu ja vanhat sovellusryhmät tyhjennetty vanhoista kapseleista, voidaan poistaa vanha sovellusryhmä. Kapselipäivitys voidaan ongelmatilanteissa vetää takaisin, jolloin asennusobjekti käy saman prosessin uudestaan läpi toisin päin, palauttaen vanhan version kapselit suoritukseen. (Kuo J. 2016.)

Kubernetes Service (palvelu)

Kapselit ovat häilyviä asennuksia, joita lisäillään ja poistellaan sekä sovellusryhmien että asennusmääritysten toimesta. Ryhmät muuttuvat, kapseleiden määrät vaihtelevat ja kapseleiden osoitteet vaihtelevat. Kaikkien kapselikopioiden hyödyntäminen sellaisenaan olisi työlästä ilman kerrosta, joka on tietoinen juuri sen hetkisistä kapseleista ja niiden osoitteista. Tätä varten on Kubernetes-palvelut. Palveluun määritellään mitä kapseleita palveluun kuuluu ja miten kapseleihin saa yhteyden. Palvelu pitää kirjaa dynaamisesti vaihtuvista kapseleista ja toimii niille samalla yhteyspisteenä ja kuormanjakajana yhdistäen ne yhdeksi kokonaisuudeksi, johon pääsee käsiksi

yhdestä osoitteesta. Kaikki yhteydet kapseleihin tulisi pääsääntöisesti hoitaa palvelumäärityksen kautta. (User Guide - Services n.d.)

Palvelu voi olla tyypiltään joko ClusterIP-, NodePort- tai LoadBalancer-tyyppinen. ClusterIP-tyyppinen palvelu on sellainen, että siihen saa yhteyden vain Kubernetes-klusterin sisältä. Se sopii hyvin taustajärjestelmiin, joihin ei ole tarkoituskaan päästä käsiksi klusterin ulkopuolelta. NodePort-tyyppinen palvelu taas julkaisee palvelun jokaiselle Kubernetes Node-palvelimelle tiettyyn porttiin näkyväksi ja palveluun pääsee silloin käsiksi ulkoverkosta suoraan Node-palvelimien IP-osoitteista ja valitusta portista. LoadBalancer tyyppinen palvelu tarvitsee klusterin ulkopuolella toimivan ja Kuberneteksen kanssa yhteensopivan kuormanjakosovelluksen. LoadBalancer-tyyppisiä palveluita käytetään yleensä pilviratkaisussa, joissa Kubernetes ympäristöä tarjoavat pilvitoimittajat tarjoavat myös erillisen kuormanjakopalvelun. (User Guide - Services n.d.)

Label (etiketti)

Häilyvät asennukset luovat Kubernetes-objekteille nimiä ja tunnisteita dynaamisesti ja niiden varassa linkittäminen tai komentojen ajaminen vaatisi jatkuvaa tarkkailua. Tämän helpotukseksi Kubernetekseen on lisätty etiketit. Etiketit ovat avainarvopareja, joita voidaan antaa eri Kubernetes-objekteille. Etikettien käyttötarkoitus on tunnistaa ja ryhmitellä eri objekteja ja kokonaisuuksia sekä mahdollistaa erillisen metadatan liittämisen objekteihin. Esimerkiksi palvelumäärityksen määritellään sen hallitsemat kapselit kapseleiden etikettitiedoilla, jolloin palvelu voi kysyä Kuberneteksel-ta kaikki kapselit, joilta löytyy joku tietty etiketti ja siten pysyä kartalla kapseleista, joita palveluun kuuluun. Myös ajettaville komennoille voidaan antaa etiketti, jolla kohdistaa komennon vaikutus haluttuihin objekteihin. (User Guide - Labels n.d.)

```
app: esimerkkisovellus
release: stable
tier: front
```

Figure 1 Esimerkkejä etiketeistä

Namespace (nimiavaruus)

Kubernetes objekteja voidaan eritellä eri nimiavaruuksien alle ja työkaluille voidaan määritellä mitä nimiavaruutta käytetään, jolloin laajojen objektimäärien hallinta on helpompaa, kun voi keskittyä pienempään joukkoon kerralla. Valmiiksi asennettuna Kubernetes klusterissa on kaksi nimiavaruutta, kube-system ja default. Default nimiavaruus on vakiona käytössä ja kaikki objektit luodaan sen alle, ellei toisin määritellä. Kube-system nimiavaruudessa näkyvät Kubernetes järjestelmän luomat objektien määrittelyt ja ne eivät näy sen takia suoraan käyttäjille, ellei käyttäjät erikseen tarkastele kube-system nimiavaruutta. (User Guide - Namespaces n.d.)

Secret (salaisuus)

Salaisuudet ovat kokoelma avainarvopareja, joiden avulla voidaan jakaa arkaluontoista tai muuttuvaa tietoa käytettäväksi sovelluskonttien sisällä. Ne sopeutuvat hyvin sellaisen tiedon jakamiseen, jota ei haluta tallentaa konttien levykuvamäärityksiin, mutta tarvitaan kuitenkin sovelluksen suorituksessa. Salaisuudet luodaan Kubernetes klusteriin, kuten muutkin objektit, mutta jotta niitä voidaan käyttää sovelluskonteissa, pitää sovelluskontin kapselimääritykseen erikseen lisätä, mitä salaisuuksia sovelluskontti voi nähdä. Kapseliin voidaan määritellä salaisuudet näkymään, joko käyttöjärjestelmän ympäristömuuttujana tai tiedostona halutussa polussa kontin sisällä. Kun Kuberneteksesta muutetaan jonkun salaisuuden arvoa, päivittyy muutos automaattisesti myös kapseleille. Yleisimpiä käyttökohteita salaisuuksille ovat käyttäjätunnukset ja salasanat sovelluskontin ulkopuolisiin järjestelmiin. (User guide – Secrets n.d.)

3.2.3 Työkalut

Kubernetesta voidaan hallita Kubectl-komentokehoteohjelmalla tai Kubernetes Dashboard -verkkosovelluksen graafisesta käyttöliittymästä. Kubectl on näistä tehokkaampi, ja sillä voidaan hallita kaikkea, mitä Kuberneteksella voidaan tehdä. Dashboard on taas kevyempi ja helppokäyttöisempi, mutta sen tarjoamat ominaisuudet eivät ole niin kattavat kuin Kubectl-ohjelmalla. Lisäksi Kubectl ja rajapinnat päivittyvät aina kun Kubernetes päivittyy, mutta Dashboard noudattaa omaa päivitysaikatauluaan. Näiden kahden lisäksi Kubernetes tarjoaa myös valmiit rajapinnat, joilla hallintaa voidaan tehdä ohjelmallisesti. (Introduction to Kubernetes 2016.)

3.2.4 Vaatimukset

Kubernetes pyörii lähes millä tahansa Linux-jakelulla, jolla voi pyörittää Dockeria, eikä se aseta sen suurempia vaatimuksia kirjastojen suhteen. Muutama palvelu pitää kuitenkin asentaa, että Kubernetes pystyy toimimaan usean eri palvelimen välillä, kun Docker itsessään pyörii vain yhden palvelimen sisällä.

Että kommunikaatio palvelimien välillä on mahdollista ja palvelimien tila pysyy kaikilla palvelimilla tiedossa, tarvitsee Kubernetes yhteisen tietovaraston, johon kaikki palvelimet voivat lukea ja kirjoittaa tilatietojaan. Tämän takia Kubernetesta varten pitää asentaa ETCD-avainarvoparitietovarastopalvelu.

Jotta Docker-sovelluskontit saadaan kommunikoimaan eri palvelimien välillä, tarvitaan joka palvelimelle jokin tietoverkkopalvelu, jolla voidaan luoda palvelimien välille virtuaalisia tietoverkkoja.

3.2.5 Kubernetes ja verkot

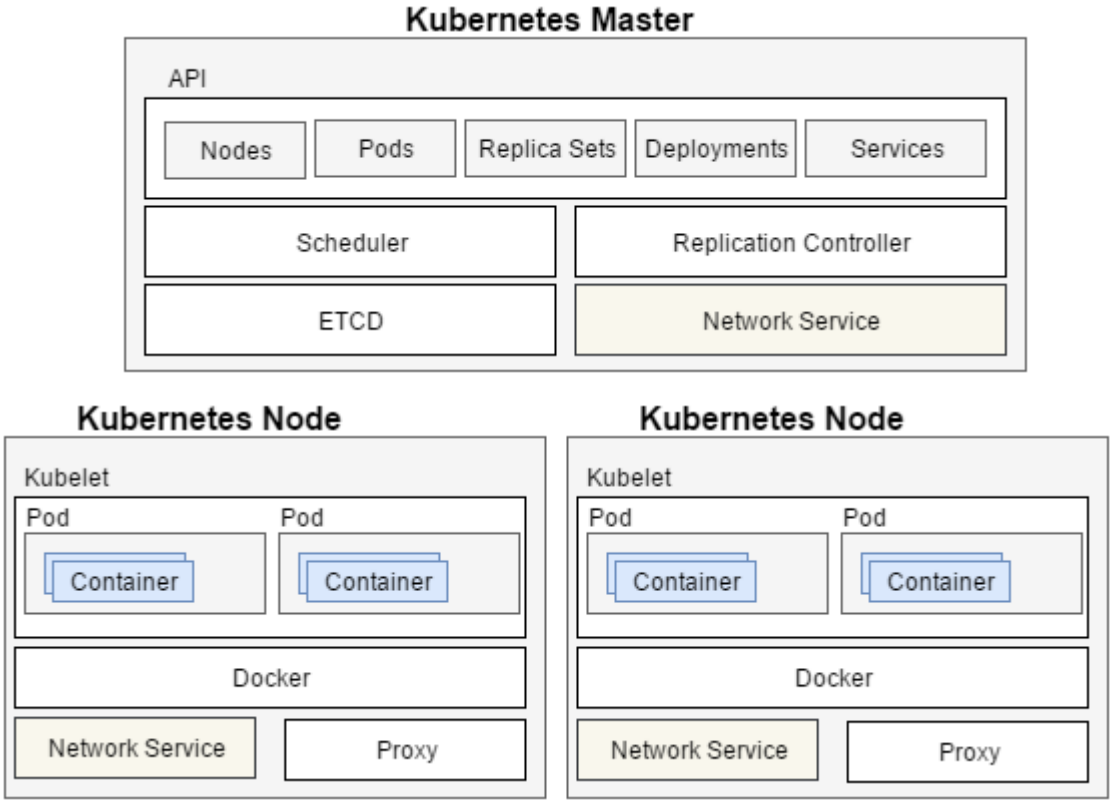
Toisin kuin Docker, jossa konttien välille pystyy määrittelemään virtuaalisia tietoverkkoja haluamallaan tavalla ja eriyttää eri kontteja eri verkkoihin, on Kuberneteksessa päädytty ratkaisuun, jossa kaikki kapselit ovat saman virtuaaliverkon sisällä ja pystyvät kommunikoimaan toisten kapseleiden kanssa riippumatta siitä, millä palvelimella kapseli pyörii. (Kubernetes networking 2016.)

Kuberneteksessa on tärkeää, että kapselit pystyvät itse ilmoittamaan oman IP-osoitteensa ja rekisteröimään itsensä Masterin tietoisuuteen ja että Kubernetes pystyy jakamaan vapaita IP-osoitteita kapseleille ja palveluille käytettäväksi. Jokainen kapseli saa oman virtuaalisen IP-osoitteen, joka on sama sekä kapselin sisällä että jokaiselta palvelimelta ulkoapäin tarkastellessa. (Kubernetes networking 2016.)

Sovelluskontit kapseleiden sisällä saavat itselleen saman IP-osoitteen kuin kapselilla on, ja saman kapselin sisällä olevat sovelluskontit käyttäytyvät verkon osalta, kuin ne olisivat samalla fyysisellä palvelimella. Palvelut saavat myös Kubernetekselta oman virtuaalisen IP-osoitteensa, josta Kubernetes osaa uudelleenohjata kapseleiden IP-osoitteisiin halutulla tavalla. (Kubernetes networking n.d.)

3.2.6 Kubernetes-klusterin arkkitehtuuri

Kubernetes-klusteri koostuu klusteria hallinnoivasta Master-komponentista sekä yhdestä tai useammasta työtä tekevästä Node-palvelimesta (ks. Kuvio 6).



Kuvio 6, Kubernetes arkkitehtuuri

Kubernetes Master

Kubernetes-klusteria kontrolloiva komponentti on nimeltään Kubernetes Master. Master toimii yhteyspisteenä koko klusterin käsittelyyn ja Masteriin luodaan kaikki Kubernetes-määrittelyt. Master ohjailee ja käskyttää Node-palvelimia, sekä jakaa niille kontteja suoritettavaksi. Master itsessään koostuu neljästä eri komponentista, jotka ovat API, Replication Controller, ETCD ja Scheduler. (Ellingwood 2016.)

API-komponentti on Masterin ydin. Se on sovellus, johon objektien, kuten kapselit, määrittelyt tallennetaan. Lisäksi API tarjoaa rajapinnat kaikkiin Kubernetes-klusterin toimintoihin. (Ellingwood 2016.)

Replication Controller on prosessi, joka tarkkailee API-komponenttiin tallennettujen sovellusryhmien asetuksia sekä Node-palvelimilla pyöriviä kapseleita. Se pitää huolen, että oikea määrä kapseleita on suorituksessa ja korjailee tilannetta tarvittaessa. (Ellingwood 2016.)

Scheduler on prosessi, joka huolehtii kapseleiden jakamisesta Node-palvelimien välillä. Se tarkkailee, millaisia resurssivaatimuksia kapseleille on asetettu sekä minkä verran milläkin Node-palvelimella on resursseja jäljellä. Näiden tietojen perusteella Scheduler pyrkii valitsemaan parhaan asennuspaikan kapseleille. (Ellingwood 2016.)

Etcd on avoimen lähdekoodin avainarvoparitietovarastopalvelu, jonka käyttötarkoitus on toimia yhteisenä konfiguraatiotietojen tallennuspaikkana hajautetuissa järjestelmissä. Kubernetes käyttää tätä palvelimien tilatietojen jakamiseen palvelimien kesken. Etcd yleensä asennetaan Master-palvelimelle, mutta se ei ole riippuvainen muista komponenteista ja se voidaan asentaa muuallekin.

Network Service on jokin virtuaaliverkkosovellus. Kuberneteksen toteutus ei itsessään sisällä komponenttia, joka osaa huolehtia virtuaaliverkkojen luomisesta palvelimien välillä. Verkkojen luomisen vastuu on jätetty Kuberneteksen ulkopuolelle, koska vaatimukset verkkojen luomiselle vaihtelevat asennusympäristöjen mukaan. Pilvipalveluilla on omat verkkoratkaisunsa, joita Kubernetes osaa hyödyntää, ja omiin Kubernetes-asennuksiin on muutamia eri vaihtoehtoja virtuaaliverkkojen toteutukseen.

Node-palvelin

Node-palvelimet ovat Kubernetes-klusterin työjuhdat. Node-palvelimille asennetaan kaikki suoritettavat kapselit ja niiden sisällä olevat sovelluskontit. Node-palvelimet koostuvat kolmesta eri komponentista, Dockerista, Kubelet-komponentista sekä välityspalvelin palvelusta. (Ellingwood 2016.)

Docker-sovelluskonttien suorittamista varten pitää olla Docker-asennus. Jokaisella Node-palvelimella on oma Docker-asennus, jossa sen vastuulla olevien kapseleiden sovelluskontit suoritetaan.

Kubelet-komponentin rooli Node-palvelimella on hallita palvelimelle asennettuja kapseleita ja niiden sisältämiä kontteja. Master palvelimen API-komponentti kom-

munikoi Kubelet-palvelun kanssa ja Kubelet-palvelu asentelee, muokkaa ja poistelee kapseleita API-komponentin käskyjen mukaisesti. (Ellingwood 2016.)

Proxy Service -komponenttia eli välityspalvelinta tarvitaan Kuberneteksen tietoverkkoratkaisuiden toteuttamiseksi. Jokaisella Node-palvelimella on oma välityspalvelin, joka osaa ohjata Node-palvelimelle tulevia yhteyksiä oikeisiin kapseleihin ja suorittaa yksinkertaista kuormanjakoa kapseleiden välillä. (Ellingwood 2016.)

3.2.7 Kubernetes klusterin asentaminen

Kuberneteksen voi asentaa monella tapaa ja asennustapa pitää valita ympäristön mukaan. Suurista pilvipalveluista Kuberneteksen saa valmiiksi asennettuna ja sen voi ottaa suoraan käyttöön. Rautapohjaisiin asennuksiin ja virtuaaliympäristöihin löytyy omat asennusohjeet eri komponentteja varten käyttötapauksista riippuen. Lisäksi koko klusterin asentamiseen löytyy automatisoituja ratkaisuja, jossa ei tarvitse huolehtia yksittäisistä komponenteista, vaan kaikki asennetaan automaattisesti. Tällaisia ratkaisuja ovat esimerkiksi Vagrant-perustainen asennus ja Docker-perustainen asennus.

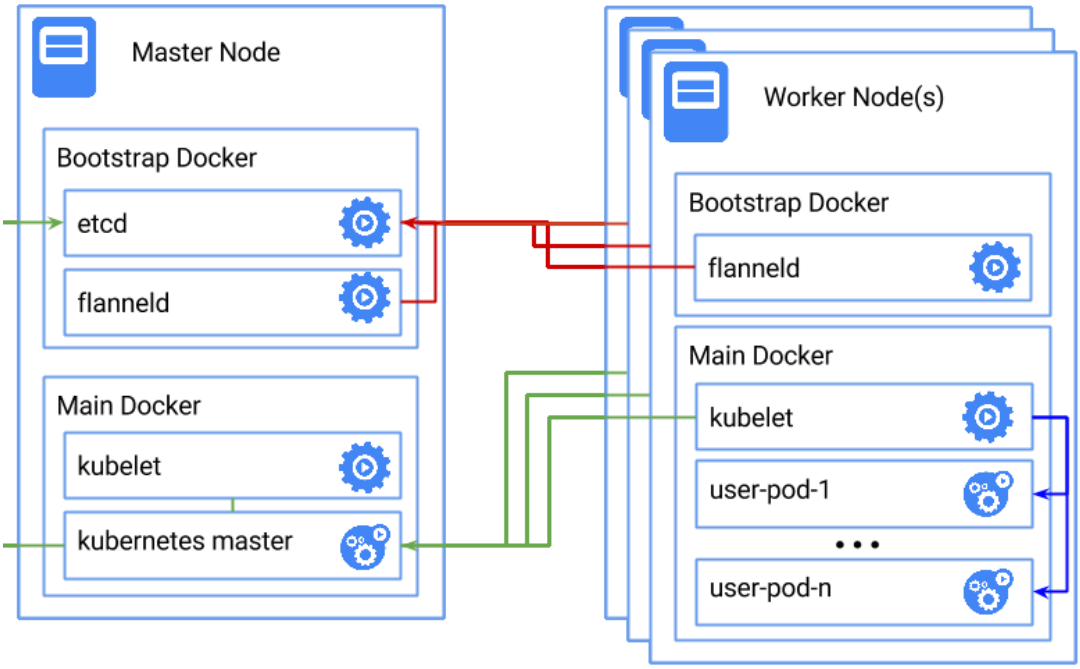
4 Toteutus

4.1 Ympäristön kuvaus

Kubernetes klusterin asennustavaksi valittiin Docker-perustainen asennus, jossa erillinen asennusohjelma asentaa sekä Master että Node-palvelimien komponentit automaattisesti, käyttäen hyväksi komponenteista valmiiksi muodostettuja Docker-levykuvia. Docker-perustaiseen asennusohjelmaan tietoverkkoratkaisuksi oli valittu Flannel niminen verkko-ohjelmisto, joka asentui myös automaattisesti.

Docker-perustaisessa asennuksessa kullekin palvelimelle asennettiin kaksi eri Docker instanssia, joilla Kuberneteksen levykuvia on tarkoitus suorittaa. Näitä Docker instansseja kutsuttiin Bootstrap Dockeriksi ja Main Dockeriksi. Bootstrap Dockerin rooli oli pyörittää Etcd ja Flannel sovelluksien sovelluskontteja, sillä niiden piti sijaita Main Dockerin ulkopuolella, että Main Docker pystyi niitä käyttämään omiin toimintoihinsa. Main Dockeriin asennettiin kaikki muut komponentit, joita Kubernetes-klusteriin

kuului. Main Dockerin rooli oli myös lopulta majoittaa ne sovelluskontit, joita kapseleiden avulla suoritetaan Kuberneteksessa (ks. Kuvio 7). (Running Multi-Node Kubernetes Using Docker N.d.)



Kuvio 7, Kubernetes asennus Dockerilla (Running Multi-Node Kubernetes Using Docker N.d.)

Tämä oli automatisoitu asennustapa, jolla voitiin asentaa Kubernetes nyt virtuaalikoneisiin ja samalla tavalla voidaan myöhemmin asentaa Kubernetes myös oikeaan hajautettuun ympäristöön.

Ympäristöä varten suunniteltiin 3 identtisen Linux-palvelimen Kubernetes-klusteri, jossa yksi palvelimista toimi sekä Master että Node roolissa ja kaksi muuta palvelinta toimi pelkästään Node roolissa. Palvelimet luotiin lokaalille työasemalle käyttäen VirtualBox-virtualisointialustaa.

4.2 Klusterin asennus

4.2.1 Virtuaalikoneiden luominen

Virtuaalikoneiden käyttöjärjestelmäksi valittiin Ubuntu Linux. Virtuaalikoneet asennettiin lataamalla Ubuntun asennusmedia sen virallisilta sivuilta ja luomalla Virtual-

Box-järjestelmään uusi virtuaalikone sen avulla. Ubuntu asennettiin vakio valinnoilla ja ilman lisäosia.

Ensimmäisenä palvelimelle asennettiin Docker-ohjelmisto Ubuntun pakettienhallintasovelluksesta apt-get install komennolla.

$ sudo apt-get install docker-io

Tämä asensi Dockerin polkuun /usr/bin/docker ja teki siitä automaattisesti käynnistyvän palvelun. Lisäksi asennus loi docker nimisen käyttäjäryhmän, jonka jäsenillä oli oikeudet käyttää Dockeria.

Dockerin asennuksen jälkeen käyttäjätunnus liitettiin docker käyttäjäryhmää.

$ sudo usermod -a -G docker käyttäjätunnus

Näillä säädöillä Ubuntu-palvelin oli valmis automaattiseen Docker-perustaiseen Kubernetes asennukseen. Virtuaalikoneesta kloonattiin kaksi kopiota ja ne nimettiin rooliensa mukaisesti KubeMaster, KubeMinion1 ja KubeMinion2.

4.2.2 Kubernetes Master asennus

Docker-perustaisen Kubernetes asennuksen asennusohjelman voi ladata Github-palvelusta. Ohjelma ladattiin palvelimille git clone komennon avulla.

```
$ git clone https://github.com/kubernetes/kube-deploy
```

Asennussovellus sisälsi omat asennustiedostot sekä Master että Node-palvelimien komponenttien asennusta varten. Master palvelimen komponentit asennettiin ajamalla master.sh asennustiedosto.

```
$ cd docker-multinode
$ ./master.sh
```

Asennustiedosto aloitti asentamalla ensin Bootstrap Dockerin ja sen sisälle Etcd ja sekä Flannel palvelujen Docker-kontit. Kun nämä oli käynnistynyt, asennus ohjelma asensi Bootstrap Dockerin rinnalle Main Dockerin ja sen sisälle Kubernetes-klusterin

komponentit. Kubernetes Master asennettiin KubeMaster nimiselle virtuaalipalvelimelle.

4.2.3 Kubernetes Node-palvelimen asennus

Ennen kuin Node-palvelimen komponentit voitiin asentaa asennustiedostolla, piti palvelimelle asettaa ympäristömuuttujaan Master palvelimen IP-osoite, jotta se pystyi kommunikoimaan jo asennetun Master palvelimen kanssa.

```
$ export MASTER_IP=${SOME_IP}
```

Tämän jälkeen voitiin ladata ja ajaa Node-palvelimen asennustiedosto samalla tavalla kuin Master palvelimenkin.

```
$ git clone https://github.com/kubernetes/kube-deploy
$ cd docker-multinode
$ ./worker.sh
```

Kuten Master palvelimenkin asennuksessakin, niin myös Node-palvelimen asennuksessa asentui ensin Bootstrap Docker ja siihen Flannel. Sen käynnistyttyä asentui Main Docker ja sinne Node-palvelimen komponentit. Node-palvelimen komponenttien asennus toistettiin kaikkiin kolmeen virtuaalipalvelimeen.

4.2.4 Kubectl

Klusterin hallintaa varten ladattiin KubeMaster virtuaalipalvelimelle Kubectl komentokehoteohjelma.

```
curl -sSL https://storage.googleapis.com/kubernetes-
release/release/v[KUBECTL_VERSION]/bin/linux/amd64/kubectl >
/usr/local/bin/kubectl

chmod +x /usr/local/bin/kubectl
```

Kubectl ohjelmalla voitiin nyt tarkistaa, että kaikkien Kubernetes-palvelimien asennus onnistui (ks. Kuvio 8).

```
jyri@jyri-VirtualBox:~/websoft$ kubectl get nodes
NAME            STATUS    AGE
192.168.0.15    Ready     30d
192.168.0.17    Ready     30d
192.168.0.18    Ready     30d
```

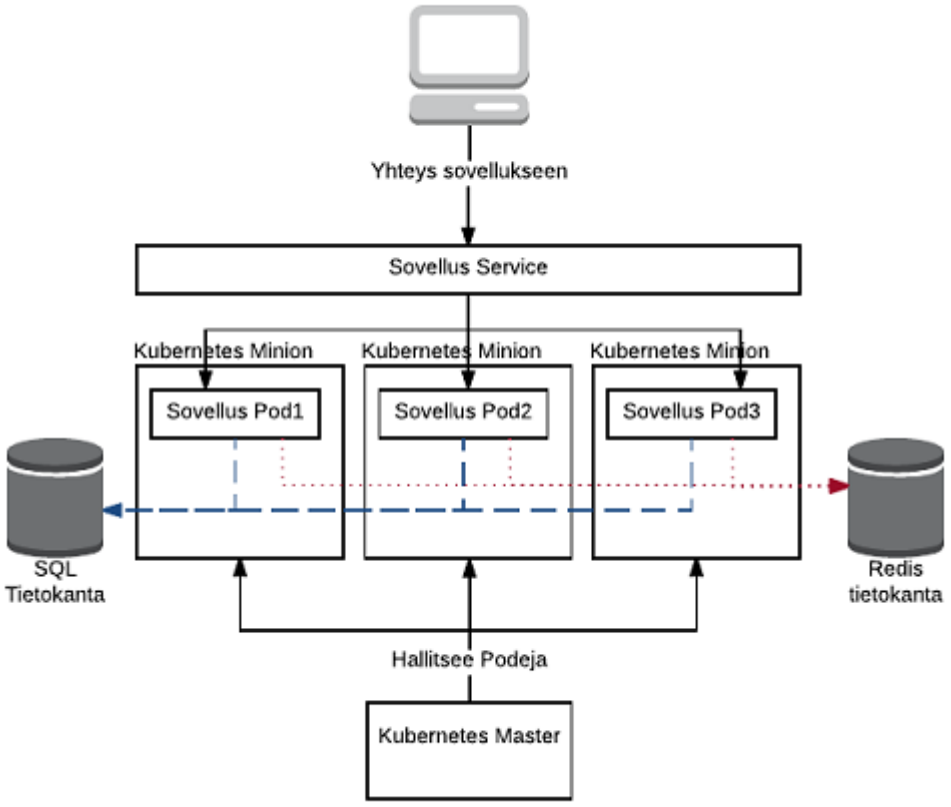
Kuvio 8, Kubectl get nodes komento

4.3 Evaluointiin käytettävä sovellus

Klusterin evaluointia varten käytettiin olemassa olevaa verkkosovellusta, jolla pystyttiin testaamaan, että klusteri toimii oikein. Lisäksi sen avulla oli tarkoitus saada selville, mitä klusterointi vaatii itse sovellukselta. Sovellus oli tilallinen sovellus, jossa oli arvioinnin kannalta oleellisia ominaisuuksina sisäänkirjautumissivu, sisältösivu, istuntojen hallinta ja uloskirjautumisen mahdollisuus. Sovellus käytti relaatiotietokantaa tietojen tallennukseen.

Jotta sovellusta voitiin klusteroida, piti siitä ulkoistaa kaikki sovelluksen tilaan liittyvät asiat, jotka pitää olla kaikkien replikoitujen kopioiden tiedossa. Testattavassa sovelluksesta tällaisiksi asioiksi tunnistettiin tietokanta, sovelluksen istuntojenhallinta ja istuntotiedot. Sovelluksesta ulkoistettiin istuntojen tallennus sovelluksen ulkopuolelle erilliseen Redis-sovellukseen kaikkien kopioiden käytettäväksi. Samoin tietokanta jätettiin Kuberneteksen ulkopuolelle ja se oli kaikkien kopioiden käytettävissä.

Sovellusta varten luotiin oma Docker-levykuva, jonka avulla sitä voitiin asentaa Dockeriin. Docker sovelluskonttia hyväksikäyttäen sovelluksesta muodostettiin Kubernetes asennusmäärittely, jossa määriteltiin kapseli suoritettavaksi kolmesti replikoituna. Kapselikopiot hajautettiin eri palvelimille. (ks. Kuvio 9.)



Kuvio 9, Sovellus Kuberneteksessa

4.3.1 Testatun sovelluksen kuvaus

Sovellus on Java kielellä ohjelmoitu verkkosovellus, joka käyttää Spring-kehyskirjastoja toimintoihinsa. Käyttöliittymä on toteutettu Spring MVC käyttöliittymäkehyksellä ja kirjautumisominaisuudet sekä käyttäjien todennus on toteutettu Spring Security kehyksellä. Sovellus käyttää tietokantaa ja tietokantaa käsitellään Spring Data JPA kehyksen avulla.

Sovelluksen riippuvuuksia ja kääntämistä hallitaan Maven kuvaustiedoston avulla ja sovelluksesta kääntyy sovelluspalvelimissa sellaisenaan suoritettava war (Web application Archive) verkkosovellustiedosto.

4.3.2 Istuntojen hallinnan ulkoistus sovelluksesta

Spring-sovelluksissa istuntojen ulkoistus voidaan tehdä käyttäen Spring Session kirjastoa. Spring Session kirjasto tarjoaa rajapinnan, jota vasten voi toteuttaa oman is-

tuntojenhallinnan sovellukseen ja valmiita toteutuksia eri ratkaisuun perustuvista istuntojenhallinnoista. Redis-sovellusta varten löytyi valmis Spring Session toteutus nimeltä Spring Session Redis. Jotta Spring Session Redis saatiin otettua käyttöön sovelluksessa, piti kirjasto lisätä sovellukseen ja sovellus määrätä käyttämään sitä.

Ensiksi sovelluksen Maven-kuvaustiedostoon lisättiin riippuvuus Spring Session Redis kirjastoon.

```
<dependency>
   <groupId>org.springframework.session</groupId>
   <artifactId>spring-session-data-redis</artifactId>
   <version>1.2.1.RELEASE</version>
   <type>pom</type>
</dependency>
```

Jonka jälkeen Maven latasi kirjaston, ja se oli valmis käytettäväksi sovelluksessa. Jotta sovellus osasi vaihtaa muistinvaraisen istuntojen hallinnan Redis-pohjaiseen istuntojen hallintaan, piti vielä muuttaa sovelluksen konfiguraatiotietoja.

Sovelluksen konfiguraatiot sijaitsivat erillisessä konfiguraatioluokassa. Konfiguraatioluokkaan lisättiin Java Bean, jonka avulla sovellus voi luoda yhteyksiä Redis-palvelimeen. Tämän lisäksi luokkaan lisättiin annotaatiomerkintä @EnableRedisHttpSession, joka ohjasi sovellusta käyttämään sessioiden tallentamiseen Redis-palvelua käyttäen hyväksi aiemmin lisättyä Java Beania.

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.data.redis.connection.jedis.JedisConnectionFactory;
import
org.springframework.session.data.redis.config.annotation.web.http.EnableRedis
HttpSession;

@Configuration
@EnableRedisHttpSession
public class ApplicationConfig {

   ...

   @Bean
   public JedisConnectionFactory connectionFactory() {
      JedisConnectionFactory jedisConnectionFactory = new JedisConnectionFac-
tory();
      jedisConnectionFactory.setHostName("192.168.0.11");
      return jedisConnectionFactory;
   }

   ...
}
```

Näiden lisäksi lisättiin Spring-sovelluksen application.properties tiedostoon määriteltiin Redis-palvelimen tiedot, joiden avulla siihen saatiin muodostettua yhteys.

```
spring.redis.host=192.168.0.11
spring.redis.password=secret
spring.redis.port=6379
```

Näillä muutoksilla sovelluksen istuntojenhallinta oli ulkoistettu erilliseen Redis-sovellukseen.

4.3.3 Sovelluksen kapselointi Dockerkonttiin

Sovelluksesta tehtiin sellaisenaan pyörivä Docker-sovelluskontti, jota voitiin ajaa missä vaan Docker-ympäristössä. Sovelluskonttiin kapseloitiin hyvin sovelluskontteihin sopeutuva Alpine Linux ja sen päälle Java kirjastot. Sovelluksen pyörittämistä varten asennettiin Tomcat-sovelluspalvelin ja sille suoritettavaksi itse sovellus. Kaikki asetukset säädettiin Docker-levykuvassa.

Docker levykuvan määrittelytiedosto eli Dockerfile aloitettiin komennoilla:

```
FROM tomcat:8-jre8
MAINTAINER "Jyri Sakkara"
```

Sovelluksen levykuvan pohjaksi valittiin Tomcat-levykuva. Tomcat-palvelimen levykuva koostui Alpine Linux käyttöjärjestelmän, Javan ja Tomcat-palvelimen levykuvakerroksista. Sovelluksen levykuvasta määrittyi oma kerros muiden päälle. Sovelluskonttiin lisättiin sovelluksen war (Web application Archive) asennustiedosto Tomcat sovelluksen webapps kansioon, josta Tomcat osaisi ottaa sen suoritukseen:

```
COPY ./sovellus.war /usr/local/tomcat/webapps/sovellus.war
```

Tämän jälkeen avattiin sovelluskontista Tomcat sovelluksen käyttämä portti, jotta siihen saatiin yhteys myös kontin ulkopuolelta:

```
EXPOSE 8080
```

Tomcat kerroksen päälle rakennettuna nämä riittivät yksinkertaisen verkkosovelluksen konttimäärittelyiksi. Dockerfile löytyy kokonaisuudessaan liitteestä 1.

4.3.4 Docker-levykuvan julkaisu

Jotta sovelluskonttia voitiin käyttää Kuberneteksen kanssa, piti siitä ensin julkaista Docker-levykuva Docker-rekisteriin, josta Kubernetes osaisi sitä ladata. Ensiksi luotiin Docker-määrittelytiedostosta Docker-levykuva build komennon avulla (ks. Kuvio 10.).

```
$ docker build -t jyri/websoft .
Sending build context to Docker daemon 31.46 MB
Step 1 : FROM tomcat:8-jre8
 ---> 8eb221fb7331
Step 2 : MAINTAINER "Jyri Sakkara"
 ---> Using cache
 ---> 120d5de29a41
Step 3 : RUN rm -rf /usr/local/tomcat/webapps/*
 ---> Using cache
 ---> 10fb876c47c7
Step 4 : COPY websoft.war /usr/local/tomcat/webapps/websoft.war
 ---> 5bf04d1eb6b8
Removing intermediate container 18ed078b6aa6
Step 5 : EXPOSE 8080
 ---> Running in 78832fb7d710
 ---> 712cbf02fe31
Removing intermediate container 78832fb7d710
Successfully built 712cbf02fe31
```

Kuvio 10, Docker build komento

Luodusta Docker levykuvasta lisättiin merkintä Docker-rekisteriin tag komennon avulla (ks. Kuvio 11.).

```
jyri@DESKTOP-VCE6N56 MINGW64 /i/workspace/FileMediator/docker
$ docker tag 712cbf02fe31 192.168.0.20:5000/websoft
```

Kuvio 11, Docker tag komento

Kun merkintä oli lisätty, voitiin levykuva julkaista Docker-rekisteriin push komennolla (ks. Kuvio 12).

```
$ docker push 192.168.0.20:5000/websoft
The push refers to a repository [192.168.0.20:5000/websoft]
407dec215d0d: Pushed
66d8e5ee400c: Pushed
2f71b45e4e25: Pushed
v2: digest: sha256:4ded2c08aa94bc86dcb162245caeb52e4b7663757a055e73cd30183f3cfbd658 size: 3461
```

Kuvio 12, Docker push komento

Näiden toimenpiteiden jälkeen sovelluksen levykuva oli ladattavissa Docker-rekisteristä.

4.3.5 Kubernetes määrittelyt

Kubernetes objekteja voi luoda suoraan Dashboard-käyttöliittymästä tai määrittelemällä kutakin varten oma määrittelytiedosto yaml-formaatilla.

Sovelluksesta tehtiin oma Kubernetes-asennusmäärittely, jonka hallitsema sovellusryhmän tehtävänä oli huolehtia, että kapseleita oli aina 3 suorituksessa. Asennusmäärittelyn rinnalle luotiin sovelluksesta myös oma Kubernetes Service eli palvelu.

4.3.5.1 Asennusmäärittely

Määrittelyt luotiin Dashboard-käyttöliittymästä luomisen sijasta yaml-tiedostoina, sillä ne antoivat enemmän säätömahdollisuuksia ja olivat helpommin ylläpidettäviä. Asennusmäärittely kirjattiin seuraavasti:

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: websoft-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: websoft
    spec:
      containers:
      - name: websoft
        image: 192.168.0.20:5000/websoft:V1.0
        ports:
        - containerPort: 8080
```

Määrittelytiedoston ensimmäisellä rivillä kerrottiin apiVersion tiedolla, minkä Kubernetes-version määrittelyä oli tarkoitus käyttää. Toisella rivillä kind-tiedolla kerrottiin, minkä Kubernetes-objektin määrittely oli kyseessä. Yllä on kuvattu asennusmääritys, joka tekee kolme kapselikopiota websoft nimisestä sovelluskontista ja lisää jokaiselle kapselille etiketin, josta kapselit voidaan tunnistaa. Kapseleista määriteltiin kapselin sisältämän kontin Docker-levykuvan nimi, latauspolku sekä portti, josta sovelluskontti vastaa kontin ulkopuolelta.

Tämä asennusmääritys lisättiin Kubernetekseen Kubectl-ohjelmalla käyttäen create komentoa (ks. Kuvio 13).

```
jyri@jyri-VirtualBox:~/websoft$ kubectl create -f deployment.yaml --record
deployment "websoft-deployment" created
```

Kuvio 13, Asennusmäärittelyn luominen Kubectl-ohjelmalla

Kubernetes loi määrittelyn perusteella asennusmäärityksen ja sovellusryhmän, joka piti huolen siitä, että ajossa on 3 replikoitua kopiota sovelluksesta. Kubectl-ohjelmalla voitiin nyt tarkistaa luotujen objektien tilat (ks. Kuviot 14 ja 15).

```
jyri@jyri-VirtualBox:~/websoft$ kubectl get deployments
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
websoft-deployment    3         3         3            3           6s
```

Kuvio 14, Asennus luomisen jälkeen

```
jyri@jyri-VirtualBox:~/websoft$ kubectl get rs
NAME                            DESIRED   CURRENT   AGE
websoft-deployment-2288171132   3         3         31s
jyri@jyri-VirtualBox:~/websoft$ kubectl get pods -o wide
NAME                                  READY   STATUS    RESTARTS   AGE   IP          NODE
websoft-deployment-2288171132-2s6du   1/1     Running   0          51s   10.1.62.2   192.168.0.17
websoft-deployment-2288171132-dkale   1/1     Running   0          51s   10.1.24.3   192.168.0.18
websoft-deployment-2288171132-u5l4e   1/1     Running   0          51s   10.1.62.3   192.168.0.17
```

Kuvio 15, Sovellusryhmä ja kapselit asennusmäärityksen luomisen jälkeen

4.3.5.2 Palvelu

Kubernetes Service kuvattiin yaml tiedostona seuraavasti:

```
apiVersion: v1
kind: Service
metadata:
  name: websoft-service
  labels:
    name: websoft-service
spec:
  type: NodePort
  ports:
    # the port that this service should serve on
    - port: 8888
      targetPort: 8080
      nodePort: 30000
  # label keys and values that must match in order to receive
traffic for this service
  selector:
    app: websoft
```

Alussa oli samankaltaiset apiVersion ja kind määrittely kuin asennusmäärityksenkin kuvauksessa. Palveluun määriteltiin palvelun tyyppi, palvelun käyttämät portit ja palveluun kuuluvien kapseleiden etiketit.

Yllä kuvattiin Kubernetes-palvelu, joka selector-tietoon määritetyn etiketin perusteella osasi yhdistyä aiemmin lisätyn asennusmäärittelyn luomiin kapseleihin. Palvelun tyypiksi valittiin NodePort-tyyppi, jotta siihen päästiin käsiksi ulkoverkosta ja kaikki portit määriteltiin manuaalisesti.

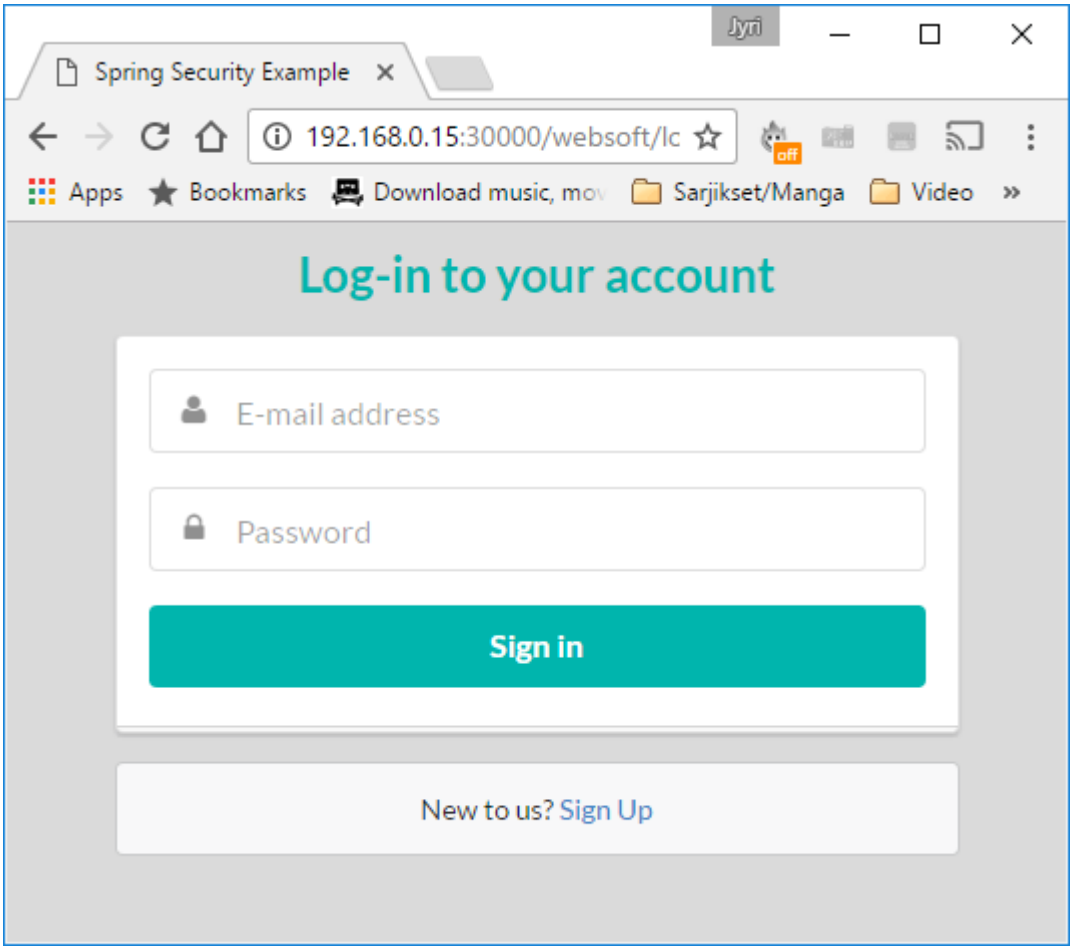
Palveluryhmittymä luotiin samalla tavalla Kubectl ohjelmalla kuten asennusobjektikin (ks. Kuvio 16).

```
jyri@jyri-VirtualBox:~/websoft$ kubectl create -f service.yaml
You have exposed your service on an external port on all nodes in your
cluster.  If you want to expose this service to the external internet, you may
need to set up firewall rules for the service port(s) (tcp:30000) to serve traff
ic.

See http://releases.k8s.io/release-1.3/docs/user-guide/services-firewalls.md for
 more details.
service "websoft-service" created
```

Kuvio 16, Palveluryhmittymän luonti Kubectl ohjelmalla

Nyt sovellukseen päästiin käsiksi navigoimalla selaimella jonkin Node-palvelimen IP-osoitteeseen ja palvelun määrittelyssä määriteltyyn 30000 porttiin. Kuviossa 17 on avattu sovelluksen kirjautumissivu Node-palvelimen IP-osoitteesta ja portista 30000. Node palvelin ohjaa selaimen sovelluksen palvelumäärityksen kautta jollekin kolmesta kapselista.



Kuvio 17, sovelluksen etusivu Node-palvelimen kautta avattuna

5 Ylläpito

5.1 Klusterin tila ja muutokset

5.1.1 Node-palvelimien tilojen tarkistaminen

Kubectl-ohjelmalla voitiin listauttaa kaikki Node-määrittelyt ja lyhyen yhteenvedon niiden tilasta (ks. Kuvio 18).

```
jyri@jyri-VirtualBox:~/kube-deploy/docker-multinode$ kubectl get nodes
NAME            STATUS     AGE
192.168.0.15    NotReady   11d
192.168.0.17    Ready      4m
192.168.0.18    Ready      11d
192.168.0.21    Ready      11d
```

Kuvio 18, Nodet ja tilat

Listaus kertoi Node-palvelimien IP-osoitteen, joka toimi samalla Node-palvelimen nimenä. Lisäksi listaus kertoi Node-palvelimen tilan ja sen elinajan. Tila voi saada arvot Ready, NotReady, OutOfDisk, DiskPressure ja MemoryPressure.

NotReady-tila kertoo, että Master ei pysty kommunikoimaan Node-palvelimen kanssa. DiskPressure ja MemoryPressure tilat ilmoittavat, että kyseiset resurssit ovat käymässä vähiin Node-palvelimella. OutOfDisk-tila kertoo, että levytila on jo loppunut.

Tarkempia tietoja yhdestä Node-palvelimesta voi kysellä Kubectl ohjelman describe komennolla (ks. Kuvio 19).

```
jyri@jyri-VirtualBox:~/kube-deploy/docker-multinode$ kubectl describe node 192.168.0.17
Name:                   192.168.0.17
Labels:                 beta.kubernetes.io/arch=amd64
                        beta.kubernetes.io/os=linux
                        kubernetes.io/hostname=192.168.0.17
Taints:                 <none>
CreationTimestamp:      Sat, 01 Oct 2016 12:32:34 +0300
Phase:
Conditions:
  Type                  Status  LastHeartbeatTime
  ----                  ------  -----------------
  OutOfDisk             False   Sat, 01 Oct 2016 13:18:50 +0300
  MemoryPressure        False   Sat, 01 Oct 2016 13:18:50 +0300
  DiskPressure          False   Sat, 01 Oct 2016 13:18:50 +0300
  Ready                 True    Sat, 01 Oct 2016 13:18:50 +0300
Addresses:              192.168.0.17,192.168.0.17
Capacity:
 alpha.kubernetes.io/nvidia-gpu:        0
 cpu:                                   2
 memory:                                4046644Ki
 pods:                                  110
Allocatable:
 alpha.kubernetes.io/nvidia-gpu:        0
 cpu:                                   2
 memory:                                4046644Ki
 pods:                                  110
System Info:
 Machine ID:                    b40fe06cb43d4f2f84210141bfd0be00
 System UUID:                   4ED0FAEE-98E1-425C-B2EA-3BAD691FEDF1
 Boot ID:                       0d16b72a-4e5f-452a-a03c-4324fb43cdd1
 Kernel Version:                4.4.0-36-generic
 OS Image:                      Debian GNU/Linux 8 (jessie)
 Operating System:              linux
 Architecture:                  amd64
 Container Runtime Version:     docker://1.11.2
 Kubelet Version:               v1.4.0
 Kube-Proxy Version:            v1.4.0
ExternalID:                     192.168.0.17
```

Kuvio 19, Noden kuvaus

Komento listasi hyvin yksityiskohtaisen listauksen Node-palvelimen tiedoista. Aiemmin läpikäytyjen tilatietojen lisäksi Node-palvelimesta oli nähtävissä sekä resurssien

kokonaismäärä, että vapaiden resurssien määrä. Niiden avulla voitiin arvioida Node-palvelimen tämänhetkistä käyttöastetta.

5.1.2 Uuden Node palvelimen lisääminen klusteriin

Kubernetes Node on ainoa objekti, jota Kubernetes Master ei pysty itse luomaan. Node määritys lisätään Kubernetes-klusteriin, kun uusi Node-palvelin rekisteröi itse itsenä Masterin tietoisuuteen. Uuden Node-palvelimen lisäys tapahtui lisäämällä uusi palvelin klusterin tietoverkkoon ja asentamalla sinne Node-palvelimen vaatimat komponentit. Kun palvelut konfiguroitiin ja käynnistettiin, yritti Node-palvelimen Kubelet-komponentti rekisteröidä Noden osaksi klusteria. Node-palvelin ilmoitti itsestään, paljonko sillä on prosessoreita, muistia ja levytilaa, sekä kaikki tarvittavat osoitteet kommunikoimiseen. Master-palvelin lisäsi näiden tietojen avulla Node määrityksen ja kun Master-palvelin oli hyväksynyt rekisteröinnin, oli Node-palvelin käytettävissä.

5.1.3 Noden poistaminen klusterista

Node-palvelimet voivat mennä vikatilaan tai jäädä tarpeettomaksi klusterin elinkaaren aikana, jolloin niitä on tarve poistella. Ennen kuin Node-palvelinta voitiin kuitenkaan poistaa, piti sieltä ensin poistaa kaikki suorituksessa olevat kapselit ja asentaa ne jollekin toiselle Node-palvelimelle suoritettavaksi. Kuberneteksessa voidaan merkata drain komennolla Node poistuvaksi, jolloin Master siirtää kaikki kapselit muille Node-palvelimille ja estää uusien kapseleiden luomisen kyseiselle Nodelle.

Node-palvelin merkattiin poistuvaksi kubetcl drain komennolla (ks. Kuvio 20)

```
jyri@jyri-VirtualBox:~/kube-deploy/docker-multinode$ kubectl get pods -o wide
NAME                                  READY     STATUS    RESTARTS   AGE       IP          NODE
websoft-deployment-2288171132-4geqa   1/1       Running   0          7m        10.1.7.2    192.168.0.18
websoft-deployment-2288171132-c9gxw   1/1       Running   0          3h        10.1.55.4   192.168.0.17
websoft-deployment-2288171132-sppcb   1/1       Running   0          3h        10.1.85.2   192.168.0.21
websoft-deployment-2288171132-u0zua   1/1       Running   0          3h        10.1.85.3   192.168.0.21
jyri@jyri-VirtualBox:~/kube-deploy/docker-multinode$ kubectl drain 192.168.0.18
WARNING: Ignoring DaemonSet-managed pods: k8s-proxy-v1-2hlfa
pod "websoft-deployment-2288171132-4geqa" deleted
node "192.168.0.18" drained
jyri@jyri-VirtualBox:~/kube-deploy/docker-multinode$ kubectl get pods -o wide
NAME                                  READY     STATUS    RESTARTS   AGE       IP          NODE
websoft-deployment-2288171132-1x2mu   1/1       Running   0          10s       10.1.55.5   192.168.0.17
websoft-deployment-2288171132-c9gxw   1/1       Running   0          3h        10.1.55.4   192.168.0.17
websoft-deployment-2288171132-sppcb   1/1       Running   0          3h        10.1.85.2   192.168.0.21
websoft-deployment-2288171132-u0zua   1/1       Running   0          3h        10.1.85.3   192.168.0.21
```

Kuvio 20, Kubectl drain komento

Drain komennon jälkeen voitiin tarkistaa, että Node palvelin oli merkitty poistuvaksi (ks. Kuvio 21).

```
jyri@jyri-VirtualBox:~/kube-deploy/docker-multinode$ kubectl get nodes
NAME            STATUS                      AGE
192.168.0.17    Ready                       3h
192.168.0.18    Ready,SchedulingDisabled    11d
192.168.0.21    Ready                       11d
```

Kuvio 21, Poistuva Node-palvelin

Lopulta Node määrityksen voi poistaa kokonaan Masterin tiedoista delete komennon avulla (ks. Kuvio 22).

```
jyri@jyri-VirtualBox:~/kube-deploy/docker-multinode$ kubectl delete node 192.168.0.18
node "192.168.0.18" deleted
jyri@jyri-VirtualBox:~/kube-deploy/docker-multinode$ kubectl get nodes
NAME            STATUS     AGE
192.168.0.17    Ready      3h
192.168.0.21    Ready      11d
jyri@jyri-VirtualBox:~/kube-deploy/docker-multinode$
```

Kuvio 22. Kubectl delete node komento

5.2 Sovelluksen ylläpito klusterissa

5.2.1 Sovelluksen tila

Sovelluksen Kubernetes-asennuksen tila

Kun sovellus on asennettu Kubernetes-klusteriin, koostuu se kolmesta eri käsitteestä, jotka ovat palvelu, asennusmääritys ja kapselit. Kun tutkitaan sovelluksen tilaa, on syytä tutkia kaikkia näitä sovelluksen osia. Kun sovelluksen asennus on kunnossa, on sillä toimiva palvelumääritys, asennusobjekti sisältää halutun määrän oikean version kapseleita ja kapselit ovat kaikki toimintakuntoisia. Näitä asioita tarkisteltiin Kubectl-ohjelman describe komennolla sekä palvelusta (ks. Kuvio 23) että asennusmäärityksestä (ks. Kuvio 24).

```
jyri@jyri-VirtualBox:~/websoft$ kubectl describe service websoft-service
Name:                   websoft-service
Namespace:              default
Labels:                 name=websoft-service
Selector:               app=websoft
Type:                   NodePort
IP:                     10.0.0.125
Port:                   <unset> 8888/TCP
NodePort:               <unset> 30000/TCP
Endpoints:              10.1.24.2:8080,10.1.57.4:8080,10.1.76.2:8080
Session Affinity:       None
No events.
```

Kuvio 23, Kubectl descibe service komento

```
jyri@jyri-VirtualBox:~/websoft$ kubectl describe deployment websoft-deployment
Name:                   websoft-deployment
Namespace:              default
CreationTimestamp:      Sat, 01 Oct 2016 19:11:08 +0300
Labels:                 app=websoft
Selector:               app=websoft
Replicas:               3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
OldReplicaSets:         <none>
NewReplicaSet:          websoft-deployment-2755706913 (3/3 replicas created)
Events:
  FirstSeen     LastSeen        Count   From
  ---------     --------        -----   ----
  3m            3m              1       {deployment-controller }
```

Kuvio 24, Kubectl describe deployment komento

Sovelluksen tila kapselin sisällä

Ulkoisesti kapseli voi vaikuttaa Kuberneteksen silmiin toimivalta, mutta joskus on syytä tarkkailla kapselia ja sen sisällä pyörivää sovellusta tarkemmin. Kubectl-ohjelmalla voitiin suoraan tulostaa kapselin sisällä pyörivän sovelluskontin logitustiedot ruudulle logs-komennon avulla (ks. Kuvio 25).

```
jyri@jyri-VirtualBox:~/websoft$ kubectl logs websoft-deployment-2755706913-q314x
```

Kuvio 25, Kubectl logs komento

Tämän lisäksi voitiin Kubectl ohjelman avulla ajaa komentoja suoraan kapselin sisällä olevaan sovelluskonttiin exec-komennolla. Sillä voitiin esimerkiksi käynnistää termi-

naaliyhteys suoraan konttiin ja tutkia sovelluksen tilaa kontin sisältä käsin (ks. Kuvio 26).

```
jyri@jyri-VirtualBox:~/websoft$ kubectl exec websoft-deployment-2755706913-q314x -i -t -- bash -il
root@websoft-deployment-2755706913-q314x:/usr/local/tomcat#
```

Kuvio 26, Kubectl exec komento

5.2.2 Skaalaus

Klusteroidun sovelluksen kapselien määrää voidaan muutella halutusti niin kauan, kun klusterissa riittää resursseja. Sovelluksen kapseleiden määrä määräytyy asennusmäärityksen hallitseman sovellusryhmän säännöissä. Helpoin tapa skaalata sovelluksen kapseleiden määrää on Kubectl-ohjelman scale-komento.

Kuviossa 27 nähdään, kuinka sovelluksen replikaatioiden määrää nostettiin kolmesta neljään scale-komennon avulla:

```
jyri@jyri-VirtualBox:~/websoft$ kubectl get deployments
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
websoft-deployment    3         3         3            3           1h
jyri@jyri-VirtualBox:~/websoft$ kubectl scale deployment/websoft-deployment --replicas=4
deployment "websoft-deployment" scaled
jyri@jyri-VirtualBox:~/websoft$ kubectl get deployments
NAME                  DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
websoft-deployment    4         4         4            4           1h
jyri@jyri-VirtualBox:~/websoft$ kubectl get pods -o wide
NAME                                   READY     STATUS    RESTARTS   AGE       IP          NODE
websoft-deployment-2755706913-cmvhx    1/1       Running   0          1h        10.1.76.2   192.168.0.18
websoft-deployment-2755706913-jo6r2    1/1       Running   0          16s       10.1.24.3   192.168.0.21
websoft-deployment-2755706913-q314x    1/1       Running   0          1h        10.1.24.2   192.168.0.21
websoft-deployment-2755706913-uva3s    1/1       Running   0          1h        10.1.57.4   192.168.0.17
```

Kuvio 27, Kubectl scale komento

Scale-komento päivitti asennuksen sisältämän sovellusryhmän määrittelyyn uuden arvon kapseleiden määräksi ja sovellusryhmä piti huolen, että annettu määrä kapseleita pysyi suorituksessa. Skaalaamalla voitiin sekä vähentää että lisätä kapseleiden määrää. Sovelluksen yhteyspisteenä toimiva palvelumääritys pysyi tietoisina kapseleiden määrästä ja niiden IP-osoitteista skaalauksesta huolimatta.

5.2.3 Sovelluksen päivitys

Päivityksen testaamista varten sovelluksen etusivulle lisättiin pieni tekstimuutos ja siitä käännettiin uusi versio. Jotta sovelluksen uusi version voitiin ottaa käyttöön Ku-

berneteksessa, piti siitä ensiksi julkaista uusi Docker-levykuva ja julkaista se Docker-rekisteriin, josta Kubernetes pääsi siihen käsiksi. Kuviosta 28 nähdään, kuinka sovelluksesta luotiin ja julkaistiin v2 versio samalla tavalla kuin ensimmäisestäkin versiosta.

```
$ docker build -t jyri/websoft:v2 .
$ docker tag 712cbf02fe31 192.168.0.20:5000/websoft:v2
$ docker push 192.168.0.20:5000/websoft:v2
```

Kuvio 28, Docker build, tag ja push komennot

Tämän jälkeen sovelluksen asennusmäärittelyä muutettiin Kubectl ohjelman edit komennon avulla, joka avasi objektin määrittelyn tekstieditoriin muokattavaksi (ks. Kuvio 29).

```
jyri@jyri-VirtualBox:~/websoft$ kubectl edit deployment/websoft-deployment
```

Kuvio 29. Kubectl edit komento

Määrittelystä etsittiin container osa ja muutettiin sen sisältämän image määritys osoittamaan uuteen levykuvaversioon.

```
      app: websoft
    spec:
      containers:
      - image: 192.168.0.20:5000/websoft:v2
```

Kuvio 30. Edit deployment

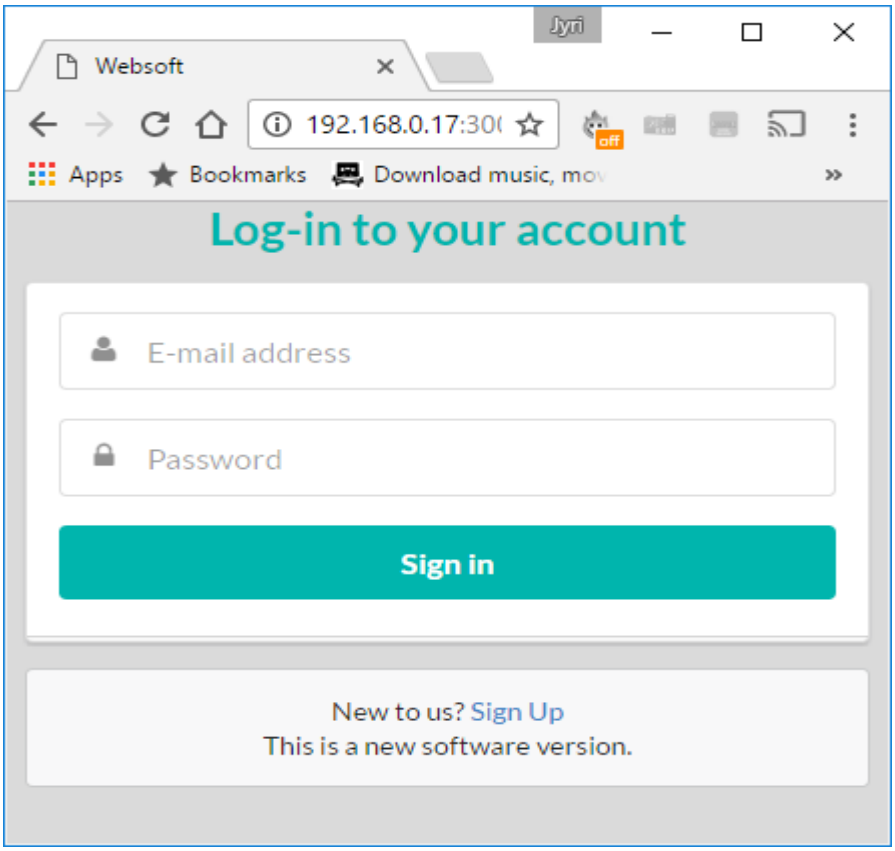
Kun määrityksen muutokset tallennettiin, Kubernetes reagoi muutoksiin ja asennusmäärittely loi uuden sovellusryhmän uutta sovellusversiota varten ja käynnisti sinne uusia versiota kapseleista. Samalla pitäen huolen, että suorituksessa olevien kapseleiden määrä pysyi säännöissä määriteltyjen vähimmäis- ja enimmäismäärän puitteissa.

```
deployment "websoft-deployment" edited
jyri@jyri-VirtualBox:~/websoft$ kubectl rollout status deployment/websoft-deployment
deployment websoft-deployment successfully rolled out
jyri@jyri-VirtualBox:~/websoft$ kubectl get deployment
NAME                 DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
websoft-deployment   4         4         4            4           2h
jyri@jyri-VirtualBox:~/websoft$ kubectl get rs
NAME                            DESIRED   CURRENT   AGE
websoft-deployment-2755706913   0         0         2h
websoft-deployment-4074291459   4         4         30s
```

Kuvio 31, Kapseliversioiden päivitys

Lopuksi kaikki 4 kapselia olivat uusia versioita, uudessa sovellusryhmässä. Vanha sovellusryhmä oli tyhjä ja valmis poistettavaksi.

Päivityksen onnistumisen varmistukseksi tarkistettiin suorituksessa olevan sovelluksen käyttöliittymästä, että pieni tekstimuutos oli ilmestynyt näkyviin. Kuviosta 32 nähdään sovelluksen etusivu Node-palvelimen IP-osoitteen kautta avattuna ja siinä pieni tekstimuutos, jonka päivitys toi tullessaan.



Kuvio 32, Sovelluksen etusivu päivittyneenä

6 Pohdinta

Opinnäytetyön tavoitteena oli tutustua Docker-virtualisointialustaan sekä sen päälle rakennettuun Kubernetes-monipalvelinympäristöön. Tarkoituksena oli selvittää, mitä Kubernetes-järjestelmällä voidaan tehdä, miten sitä käytetään, miten se otetaan käyttöön ja arvioida, onko Kuberneteksella potentiaalisia käyttökohteita toimeksiantajan yritystoiminnassa. Dockerista on jo kirjoitettu opinnäytetöitä, minkä takia tässä työssä keskityttiin enemmän Kubernetekseen ja sen arviointiin. Kubernetes oli aiheena vielä tuore, eikä siitä ollut juurikaan painettua kirjallisuutta, minkä takia lähteinä käytettiin ainoastaan Internetistä löytyviä lähteitä. Työssä ei verrattu Kubernetesta muihin Docker-monipalvelinympäristöratkaisuihin, vaan sitä arvioitiin itsenäisesti sen tuomien mahdollisuuksien ja käytettävyyden perusteella. Dockerin ja Kuberneteksen lisäksi työssä käsiteltiin aiheeseen vahvasti liittyvää verkkosovellusten klusterointia, sen tuomia hyötyjä ja kuinka sellainen voidaan saavuttaa. Työssä kokeiltiin, millaista on suorittaa omaa sovellusta Kubernetes-ympäristössä ja mitä se vaatii sovellukselta.

Työssä selostettiin Kuberneteksen peruskäsitteet ja käytiin läpi tavallisimmat Kuberneteksen käyttötapaukset. Työn avulla voidaan tutustua Kubernetekseen ja sitä voidaan käyttää ohjeena Kuberneteksen käyttöönotoissa. Työssä suunniteltiin pieni mutta käyttökelpoinen monipalvelinklusteri ja se saatiin rakennettua, kuten oli suunniteltu. Verkkosovelluksesta muodostettiin klusteroitava versio, ja se asennettiin Kubernetekseen ja hajautettiin usealle palvelimelle.

6.1 Kehitysideat

Työssä olisi vielä voinut tutkia tarkemmin, miten hajautus Kubernetes-klusteriin lisää suorituskykyä sovellukselle ja miten suorituskykyä voidaan valvoa, miten kapseleille olisi määritelty fyysisiä resurssirajoja tai kuinka isäntäkoneiden tietojärjestelmän levykansioita jaetaan kapseleiden välille silloin, kun käytössä on useita palvelimia. Lisäksi työssä rakennettu Kubernetes-klusteri oli muuten vikasietoinen, mutta koko klusteri oli riippuvainen yhden palvelimen varassa pyörivästä Master-komponentista. Sen hajauttaminen olisi ollut hyvä seuraava tutkinnan kohde, sillä jos se menisi vikatilaan, menisi koko klusteri vikatilaan ja koko klusterin yksi päätarkoituksista on pitää

palvelut toiminnassa yksittäistä vikatiloista huolimatta. Skaalatuissa sovelluksissa olisi myös ollut hyödyllistä tutustua, kuinka keskitettyjä logitusjärjestelmiä voidaan käyttää Kuberneteksen kanssa. Jos sovellusta aiotaan skaalata laajasti, silloin tietyn tapahtuman etsiminen sovelluksen logitustiedoista menee vaikeaksi ilman keskitettyä logienhallintaa.

6.2 Analyysi

Dockerin käsitteet oli helppo omaksua ja niiden avulla ymmärsi, mistä konttivirtualisoinnissa on kyse. Dokumentaatio oli selkeää ja Internetistä löytyi erinomaisia ohjeita eri käyttötarkoituksia varten. Dockerin käyttö oli helppoa ja vaikka suurin osa ajasta tulikin vietettyä Kubernetes-termien parissa, niin Docker-asioihin palatessa ei niitä tarvinnut kerrata niiden yksinkertaisuuden takia. Dockerin toiminnassa ei ole moitittavaa ja se sopii erinomaisesti pienten virtualisoitujen konttien luomiseen, julkaisuun ja suorittamiseen. Docker-rekisterit ovat hyvä tapa jakaa kontteja eri palvelimien kesken ja eri versioiden säilytystä varten.

Windows-ympäristössä Docker-alustan suorittaminen oli turhan monimutkaista sen vaatiman Docker Toolbox -työkalun ja sen oman virtuaalikonekerroksen takia. Docker-sovelluskontit pyörivätkin virtuaalipalvelimen Dockerissa, eikä isäntäkoneella, jonne olit asentavasi Dockerin. Päästääkseen käsiksi kontteihin isäntäkoneelta ei riittänyt, että avasi kontista portteja, vaan piti tietää avata portteja myös automaattisesti luoduista Docker-virtuaalikoneista. Lisäksi jotkin Dockerin ominaisuuksista eivät toimineet Dockerin Windows-versiossa. Linux-käyttöjärjestelmässä Dockerin käyttäminen oli mutkatonta.

Kubernetesta on kritisoitu siitä, että se tuo paljon omia käsitteitä Dockerin rinnalle ja hankaloittaa infrastruktuurin ymmärtämistä. Tämä piti paikkansa myös tämän työn kohdalla, ja Kuberneteksen omaksuminen vei paljon enemmän aikaa kuin Dockerin omaksuminen. Työn aikana tuli kevyesti tutustuttua myös Kuberneteksen kilpailijaan Docker Swarmiin, ja se vaikutti Dockerin jälkeen selkeältä ja yksinkertaiselta ottaa käyttöön Dockerin rinnalle. Kubernetes on hankalampi ja sen tapauksessa ei riitä, että kehittäjät hallitsevat Dockerin perusteet, vaan tarvitaan vähintään yksi Kubernetes-asiantuntija, joka selviää Kubernetes-haasteista. Vaikka Kuberneteksen asenta-

minen olikin erittäin yksinkertaista käyttäen Docker-perustaista asennustapaa, niin silti työn aikana piti tutustua yksittäisten komponenttien asennuksiin ja toimintoihin erinäisten vikatilojen takia.

Vikatilat johtuivat suurimmaksi osin tuoreesta asennussovelluksesta ja VirtualBox-virtuaalipalvelimien jumitiloista. Docker-perustaisesta asennussovelluksesta julkaistiin ensimmäinen versio noin puoli vuotta ennen työn aloittamista ja sitä kehitettiin jatkuvasti työn aikana. Muutamiin koodivirheisiin ja versioristiriitoihin tuli törmättyä, mutta ne ratkesivat aina päivittämällä asennusohjelma ja asentamalla Kubernetes uudelleen. Koska kaikki Kubernetes määrittelyt olivat erikseen yaml-tiedostoissa, oli klusterin uudelleenpystytys nopeasti tehty.

Kubernetes on monimutkainen kokonaisuus, koska ongelma, jota sillä ratkotaan, on monimutkainen ja Kubernetes tekee sen niin selkeästi kuin pystyy. Loppuvaikutelmaksi jäi, että Kubernetes on pätevä konttienhallintaohjelmisto ja tekee sen minkä lupaa. Sen avulla on helppoa luoda monistettuja hajautettuja konttikokonaisuuksia usean palvelimen välille, ja niiden hallinta on lähes automaattista. Sovellusten skaalaus useammalle kontille, käyttökatkottomat konttikokonaisuuksien päivitykset tai viallisten päivitysten takaisinveto onnistui yhdellä komennolla. Kubernetes tarjoaa hyvät työkalut ja käyttöliittymän klusterin tilan tarkisteluun sekä mahdollistaa rajapintojensa ansioista omatekoisen automatisoidun sovellusten asennustenhallinnan. Ylläpitäjälle on lähes yhdentekevää, onko sovellusta varten yksi vai tuhat replikoitua konttia suorituksessa, sillä sovellusta käsitellään kokonaisuutena kapseleiden määrästä riippumatta. Kapseleita eli ja kuoli satunnaisesti sovellusvirheiden takia mutta Kubernetes piti huolen siitä, että vaadittu määrä toimivia kapseleita pysyi suorituksessa ja palvelun tarjoaman kuormanjakajan takaa vastasi aina haluttu määrä sovelluksen instansseja.

Kuberneteksessa oli kuitenkin puutteensa. Esimerkiksi omien tietoverkkojen määritystä kapseleiden välille jäi kaipaamaan. Niiden avulla olisi pystynyt pitämään eri sovelluskokonaisuudet erossa toisista sovelluskokonaisuuksista ja vähentämään väärinkäytön mahdollisuuksia murtotilanteissa. Toinen puute oli palveluryhmittymille kiinteän irrallisen IP-osoitteen määritys sen sijaan, että se sidottaisiin jonkin Node-palvelimen IP-osoitteeseen. Kun hajautuksella on tarkoitus parantaa vikasietoisuutta, on epäkäytännöllistä joutua käyttämään jonkin Node-palvelimen IP-osoitetta, sillä

pitää varautua siihen, että Node-palvelin voi mennä vikatilaan. Käytännössä tämä tarkoittaa sitä, että jos haluaa oikeasti vikasietoisen Kubernetes-palvelun, joka kestää Node-palvelimen poistumisen, pitää asentaa Kuberneteksen ulkopuolelle oma kuormanjakajasovellus, johon pitää säätää jokaista palvelua varten kuormanjako kaikkien Node-palvelimien kesken.

Docker ja Kubernetes ovat toimiva yhdistelmä ja parempi vaihtoehto perinteiseen palvelinperustaiseen virtualisointiin joissakin tapauksissa. Palvelinperustainen virtualisointi ja konttivirtualisointi eivät kuitenkaan ole toisiaan poissulkevia ratkaisuja ja näiden kombinaatiolla saadaan aikaan tehokkaita infrastruktuureja. Kuberneteksen avulla sovellusten klusterointi ja hajautus on tehty niin helpoksi, että se yleistynee vielä lisää kehittäjien keskuudessa sekä yrityksien omissa palvelininfrastruktuureissa. Pilvipalveluissa Kubernetes on jo yleisesti käytössä. Konttiperustainen virtualisointi vaikutti tutkimuksen ja havaintojen perusteella niin tehokkaalta ratkaisulta, että se tulee yleistymään lähitulevaisuudessa paljon ja sen ymmärtäminen tulee olemaan tarpeellista lähes kaikille sovelluskehittäjille.

Toimeksiantajalla on joitakin palveluja, joissa samaa sovellusta ajetaan usealle asiakkaalle eriytettynä kukin omaksi virtuaalipalvelimeksi. Kaikilta palvelimilta löytyy sama käyttöjärjestelmä ja samat kirjastot, sama sovelluspalvelin ja sama sovellus eri konfiguraatiolla. Tällaisen infrastruktuurin voisi korvata muutamalla tehokkaalla virtuaalipalvelimella, joista rakentaisi Kubernetes-klusterin ja asentaisi asiakkaan sovellukset kontteina näille palvelimille. Asiakkaiden ympäristöt pysyisivät edelleen erillään toisistaan mutta levyjärjestelmän resursseja säästettäisiin paljon. Lisäksi etuna olisi myös helppo hajautus, joka parantaisi vikasietoisuutta sekä suorituskykyä. Tämä kuitenkin vaatisi muutoksia olemassa oleviin sovelluksiin, että niitä voitaisiin hajauttaa.

Lähteet

Brief history of Docker containers 2016) n.d. Viitattu 1.10.2016 http://searchservervirtualization.techtarget.com/feature/A-brief-history-of-Docker-Containers-overnight-success

Docker Overview N.d. Viitattu 30.09.2016 https://docs.docker.com/engine/understanding-docker/

Ellingwood J. 2016. Introduction to Kubernetes n.d. Artikkeli digitalocean.com verkosivustolla. Viitattu 30.09.2016 https://www.digitalocean.com/community/tutorials/an-introduction-to-kubernetes

Kubedeploy docker multinode n.d. Viitattu 30.09.2016 https://github.com/kubernetes/kube-deploy/blob/master/docker-multinode/README.md

Kubernetes getting started guide N.d. Kubernetes käyttöopas verkkosivustolla Kubernetes.io. Viitattu 30.09.2016 http://kubernetes.io/docs/getting-started-guides/

Kubernetes networking n.d. Viitattu 1.10.2016 https://github.com/kubernetes/kubernetes/blob/master/docs/design/networking.md

Kuo J. 2016. Using Deployment objects with Kubernetes. Blogikirjoitus Kubernetes.io verkkosivustolla. Viitattu 30.10.2016. http://blog.kubernetes.io/2016/04/using-deployment-objects-with.html

Porterie A. 2016 Docker – Updated project statistics. Github verkkosivustolle tallennettu tilastoraportti. Viitattu 1.10.2016 https://gist.github.com/icecrime/18d72202f4569a0cab1ee60f7583425f

Rani O. 2016. History of containers. Blogi-kirjoitus aquasec.com verkkosivustolla. Viitattu 30.09.2016 http://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016

Running Multi-Node Kubernetes Using Docker N.d. Viitattu 31.10.2016. Kube-deploy projektin dokumentaatio. https://github.com/kubernetes/kube-deploy/blob/master/docker-multinode/README.md

Understand Docker container networks N.d. Viitattu 30.09.2016 https://docs.docker.com/engine/userguide/networking/dockernetworks/

Understand images, containers, and storage drivers N.d. Viitattu 30.9.2016 https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/

User guide – Deployments. N.d. Kubernetes käyttöopas verkkosivustolla Kubernetes.io. Viitattu 30.10.2016. http://kubernetes.io/docs/user-guide/deployments/

User guide – Labels. N.d Kubernetes käyttöopas verkkosivustolla Kubernetes.io. Viitattu 30.10.2016. http://kubernetes.io/docs/user-guide/labels/

User guide – Namespaces. N.d Kubernetes käyttöopas verkkosivustolla Kubernetes.io. Viitattu 30.10.2016. http://kubernetes.io/docs/user-guide/namespaces/

User guide – Pods N.d. Kubernetes käyttöopas verkkosivustolla Kubernetes.io. Viitattu 30.10.2016. http://kubernetes.io/docs/user-guide/pods/

User guide – Replica Sets. N.d. Kubernetes käyttöopas verkkosivustolla Kubernetes.io. Viitattu 30.10.2016. http://kubernetes.io/docs/user-guide/replicasets/

User guide – Secrets. N.d. Viitattu 5.10.2016 http://kubernetes.io/docs/user-guide/secrets/

User guide – Services. N.d Kubernetes käyttöopas verkkosivustolla Kubernetes.io. Viitattu 30.10.2016. http://kubernetes.io/docs/user-guide/services/

Vaughan-Nichols S. 2014. What is Docker and why is it so darn popular? ZDNet verkkosivut. Viitattu 30.09.2016 http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular

Vaughan-Nichols S. 2015. Containers vs Virtual Machines. ITWorld verkkosivut. Viitattu 30.09.2016. http://www.itworld.com/article/2915530/virtualization/containers-vs-virtual-machines-how-to-tell-which-is-the-right-choice-for-your-enterprise.html

Whatisk8s N.d. Viitattu 30.09.2016 http://kubernetes.io/docs/whatisk8s/

Wilson B. 2014 What Is Docker? How Does It Work? Artikkeli devopscube.com verkkosivustolla. Viitattu 30.09.2016 http://devopscube.com/what-is-docker/

Liitteet

Liite 1. Sovelluksen Dockerfile ja Kubernetes-määrittelyt

MINGW64:/c/Users/jyri

```
MAINTAINER "Jyri Sakkara"
RUN rm -rf /usr/local/tomcat/webapps/*
COPY websoft.war /usr/local/tomcat/webapps/websoft.war
EXPOSE 8080
~
~
~
~
~
~
~
~
<ileMediator/docker/Dockerfile [dos] (22:44 08/08/2016)4,38 Bot
```

jyri@jyri-VirtualBox: ~/websoft

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: websoft-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: websoft
    spec:
      containers:
      - name: websoft
        image: 192.168.0.20:5000/websoft
        ports:
        - containerPort: 8080
```

jyri@jyri-VirtualBox: ~/websoft

```
apiVersion: v1
kind: Service
metadata:
  name: websoft-service
  labels:
    name: websoft-service
spec:
  type: NodePort
  ports:
    # the port that this service should serve on
    - port: 8888
      targetPort: 8080
      nodePort: 30000
  # label keys and values that must match in order to receive
traffic for this service
  selector:
    app: websoft
~
```