



Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Francesco de Lorenzo

Konttipohjaisen ohjelmiston siirtäminen Kubernetesiin

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintäteknikka

Insinöörityö

11.11.2020

Tekijä Otsikko	Francesco de Lorenzo Konttipohjaisen ohjelmiston siirtäminen Kuberneteseseen
Sivumäärä Aika	25 sivua 11.11.2020
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintäteknikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Matti Oosi
<p>Insinööri työ koostuu yrityksen konttimalliin perustuvan arkistointiohjelmiston siirtämisestä Kuberneteseseen sekä siihen liittyvästä tutkimustyöstä. Työn tavoitteena oli luoda Minikubella toimiva Kubernetes-ympäristö, jota voidaan käyttää muun muassa kehitysympäristönä.</p> <p>Insinööri työssä asennettiin ja konfiguroitiin Minikube. Minikuben asentamisen jälkeen yrityksen ohjelmiston komponentit asennettiin Kubernetes-podeihin, jotta ne saatiin toimimaan yhtenä kokonaisuutena Kubernetesessä. Podien lisäksi tietyille komponenteille suunniteltiin ja toteutettiin Kubernetes-palvelut (eng. Kubernetes service) jotka mahdollistavat tiedon siirron podista toiseen.</p> <p>Yrityksen ohjelmisto saatiin onnistuneesti siirrettyä Kuberneteseseen. Ohjelmiston komponentit saatiin toimimaan podeissa ja ne saatiin kommunikoidaan keskenään. Yritys sai työstä myös Kuberneteseseen liittyvää uutta tietoa ja kokemusta, jota aiotaan hyödyntää tulevaisuudessa.</p> <p>Kaikkia Kubernetesen tarjoamia ominaisuuksia ei ole kuitenkaan vielä tutkittu ja toteutettu, joten työssä on vielä varaa jatkokehitykseen. Tulevaisuudessa onkin tarkoitus tutkia ja kehittää työssä syntyneitä Kubernetes-ympäristöä.</p>	
Avainsanat	Kubernetes

Author Title	Francesco de Lorenzo Transfer of Containerized Software into Kubernetes
Number of Pages Date	25 pages 11 November 2020
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Software Engineering
Instructors	Matti Oosi, Lecturer
<p>The study consists of designing and implementing a Kubernetes environment from the company's already containerized archiving software. The aim was to use Minikube to implement a working Kubernetes environment, which can be later used as a development environment.</p> <p>First, Minikube was installed and configured. Next, the components of the company's software were installed into Kubernetes pods. Additionally, services were implemented for some of the pods to allow the transfer of data between pods.</p> <p>As a result, the company's software was successfully installed in a Kubernetes environment. Each of the software components can run inside Kubernetes pods and they are able to communicate with each other. The company has also obtained valuable knowledge and experience about Kubernetes.</p> <p>Not all features available in Kubernetes have been explored and implemented yet, however, and there is still room for further development. In fact, there are plans to continue the development of the project in the future.</p>	
Keywords	Kubernetes

Sisällys

Lyhenteet

1	Johdanto	1
2	Virtualisaatio	2
3	Käytetyt teknologiat	4
3.1	Kubernetes	4
3.2	Minikube	6
3.3	Pod	6
3.4	Palvelut	7
4	Minikuben asennus	8
5	Ohjelmiston siirtäminen Kubernetesiin	11
5.1	Suunnittelu	11
5.2	Podin mariadb-api2 määrittäminen	13
5.3	Palvelun mariadb-api2-nodeport määrittäminen	15
5.4	Podin api2-docker määrittäminen	17
5.5	Podin logcatalog-data-tool-määrittäminen	20
5.6	Muut ohjelmiston komponentit	23
6	Yhteenveto	24
	Lähteet	25

Liitteet

Liite 1. Liitteen nimi

Liite 2. Liitteen nimi

Lyhenteet

JSON	JavaScript Object Notation. Standardi tiedostomuotodatan tallentamiseen ja välittämiseen.
VM	Virtual Machine. Emuloitu tietokone.
VDI	Virtual Desktop Infrastructure. Järjestelmä, jossa etätyöasemia ylläpidetään keskitetyllä palvelimella.
IP	Internet Protocol. Verkkokerroksen protokolla, jota käytetään datapakettien välittämiseen Internet-verkossa.
API	Application Program Interface. Rajapinta, joka määrittelee, miten ohjelman käyttäjät voivat tehdä pyyntöjä.
SSH	Secure Shell. Yleisesti käytetty salatun tietoliikenteen protokolla.

1 Johdanto

Ohjelmistotuotanto on siirtymässä käyttämään virtualisointia entistä enemmän. Yhä useammin ohjelmistoja ei enää asenneta suoraan isäntälaitteelle, vaan ohjelmistoja kehitetään, testataan ja otetaan käyttöön erilaisten virtualisointiteknologioiden avulla.

Työ tehdään yritykselle Fail-Safe IT Solutions Oy, joka on alun perin vuonna 2010 perustettu IT-yritys. Fail-Safe tarjoaa muun muassa lokien keräys- ja hallintajärjestelmää ja monipuolisia IT-konsultointipalveluja.

Tässä insinööriyössä siirretään osa Fail-Safen arkistointiohjelmistoa Kubernetesiin. Kubernetes on virtualisointityökalu, joka mahdollistaa sovellusten loogisen hajauttamisen usealle laitteelle. Työn tavoitteena on luoda ympäristö, jota voidaan tulevaisuudessa käyttää tuotteen kehittämiseen sekä testaamiseen.

Yrityksen ohjelmisto, joka on tarkoitus siirtää Kubernetesiin, on jo valmiiksi konttimalliin pohjautuva. Kontit ovat ohjelmiston loogisia yksiköitä, joita voidaan ajaa helposti esimerkiksi Dockerilla. Ohjelmisto halutaan siirtää Kubernetesiin, koska ohjelmiston kehitystä ja käyttöönottoa halutaan automatisoida enemmän. Työ sisältää tutkimuksen Kubernetesistä ja siitä, miten ohjelmisto saadaan siirrettyä. Lisäksi insinööriyö kattaa sekä työn suunnittelun että toteutuksen. Työn lopullisena tavoitteena on siis saada ohjelmisto toimimaan Kubernetesissä paikallisesti. Työssä käytetään työkalua nimeltä Minikube, joka mahdollistaa Kubernetes-klusterin luomisen ja suorittamisen paikallisesti.

Tässä työssä käydään ensin läpi, miksi virtualisaatio on yleisesti käytetty ohjelmistotuotannossa. Lisäksi selitän yleisellä tasolla, miten se toimii. Sen jälkeen esittelen tarkemmin Kubernetesin ja siihen liittyvät käsitteet. Lopuksi kuvailen yksityiskohtaisesti työn kulun Minikuben asennuksesta lähtien.

2 Virtualisaatio

Alun perin yleisenä käytäntönä ohjelmistot asennettiin suoraan laitteelle. Yritysten sovellukset olivat käynnissä vain fyysisillä palvelimilla eikä mitään mahdollisuutta rajata niiden resurssien käyttöä ollut. Esimerkiksi jos yhdellä palvelimella oli useita sovelluksia käynnissä, oli mahdollista, että yksi sovellus käyttäisi suuren osan laitteen resursseista, jolloin toinen sovellus ei toimisi yhtä tehokkaasti. Ratkaisuna tähän yrityksillä oli oma fyysinen palvelin jokaista sovellusta kohti. Tämäkään ei kuitenkaan ollut tehokas ratkaisu, koska resurssit olivat tässä tilanteessa alikäytettyjä ja palvelimen resurssit olivat kalliita.

Virtualisaatio kehitettiin ratkaisuna tähän ongelmaan. Virtualisaatio on prosessi, jonka avulla voidaan simuloida useita virtuaalisia laitteita samalla fyysisellä laitteella. Virtualisaatio sallii muun muassa usean käyttöjärjestelmän suorittamisen samanaikaisesti yhden fyysisen laitteen prosessorilla. Virtualisaatio mahdollistaa myös sovellusten eristämisen toisistaan, jolloin data ei voi siirtyä helposti sovelluksesta toiseen, mikä puolestaan lisää tietoturvallisuutta. Virtualisaation mukana tulee paljon muitakin hyötyjä. Palvelimelle on helppo lisätä tai poistaa sovelluksia, jolloin resurssit käytetään tehokkaammin ja sovelluksia on helppo pilkkoa pienempiin loogisiin osiin, mikä tekee järjestelmästä tehokkaamman ja helpommin hallittavan. [1.]

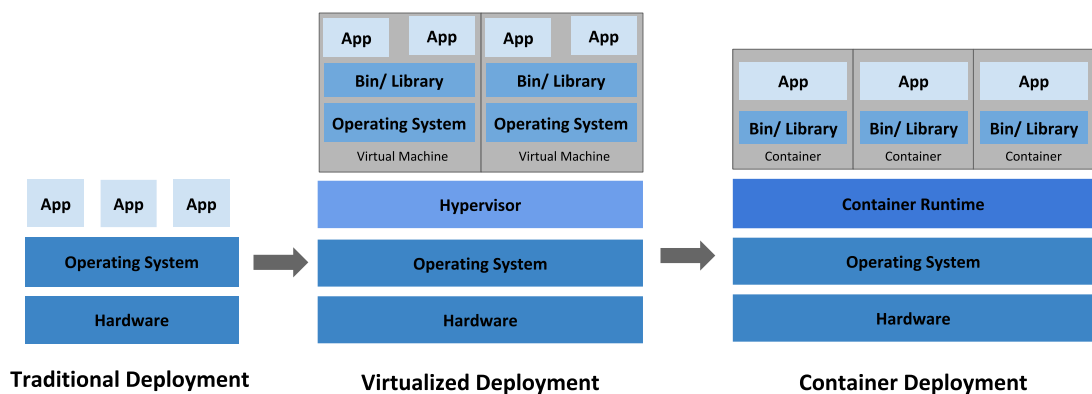
Yhä yleistyvät virtualisoinnin käyttökohteet ovat muun muassa VDI:t (Virtual Desktop Infrastructure). VDI:t mahdollistavat kehittäjille tarpeelliset resurssit ilman, että kehittäjällä täytyy olla fyysinen laite käytettävissä [2]. Myös tässä projektissa on käytetty VDI:tä, jonka virtuaalikoneessa oli käytettävissä CentOS 8 -käyttöjärjestelmä.

VM-mallissa käyttöjärjestelmässä on asennettuna hypervisor, joka suorittaa ja hallinnoi virtuaalikoneita. Jokaisessa virtuaalikoneessa on asennettu oma täysvaltainen käyttöjärjestelmä, joka suoritetaan virtualisoidun laitteiston päällä. VM-mallissa jokaisella virtuaalikoneella on siis määritelty erikseen virtuaalilaitteisto, jolloin virtuaalikoneessa suoritettava käyttöjärjestelmällä ei ole vapaata pääsyä virtuaalikoneen isäntäkäyttöjärjestelmään.

Virtualisoinnin käsite on syntynyt jo 60-luvulla, ja se on kehittynyt ja laajentunut siitä lähtien [3]. Myös virtualisointiin liittyvät teknologiat ovat kehittyneet ja nykyään on olemassa

monia tehokkaita työkaluja, jotka helpottavat virtualisointia. Yksi esimerkki tunnetuimmista ja käytetyimmistä virtualisoinnin mahdollistavista työkaluista on Docker. Virtualisointitekniologioiden suosio on huomattavassa kasvussa, ja Docker on kolmanneksi suosituin alusta kehittäjien keskuudessa heti Linuxin ja Windowsin jälkeen [4].

VM-mallin jälkeen alettiin hiljalleen käyttää kontteja. Kontit ovat samankaltaisia kuin virtuaalikoneet, mutta sen sijaan, että niillä on oma käyttöjärjestelmä, niin ne käyttävät fyysisen laitteen käyttöjärjestelmää. Jokaisessa kontissa on kuitenkin sovellukset sekä paketoitua binääritiedostot sekä kirjastot. Konttimallin ja VM-mallin erot näkyvät kuvassa 1, joka kuvaa mallien arkkitehtuurillisia eroja.



Kuva 1. Virtualisoinnin kehitys [5].

Vaikka konteilla on omat tiedostojärjestelmät ja allokoitua resurssit, niin ne käyttävät vähemmän laitteen resursseja kuin virtuaalikoneet. Keveyden lisäksi konteissa on se hyöty, että ne ovat irrallaan alla olevasta järjestelmästä, jolloin ne ovat paljon helpommin siirrettäviä laitteesta toiseen kuin virtuaalikoneet.

Konttimallin hyötyjä ovat muun muassa:

- Kontin image on helppokäyttöisempi ja tehokkaampi verrattuna VM-imaan, jolloin sovellusten kehitys ja käyttöönotto on ketterämpää.
- Konttien imaget rakennetaan usein, mikä parantaa sovellusten jatkuva kehitystä, integrointia ja käyttöönottoa.
- Konttien käyttö erottaa kehityksen ja käyttöönoton luomalla imaget kehitysvaiheessa käyttöönottovaiheen sijaan.

- Kontteja voi suorittaa lähes millä tahansa laitteella ja käyttöjärjestelmässä, jossa on asennettuna ohjelmisto konttien ajoa varten.
- Kontit eristävät sovelluksen muista sovelluksista.
- Konttien käyttö takaa sen, että sovelluksella on aina sama ympäristö.

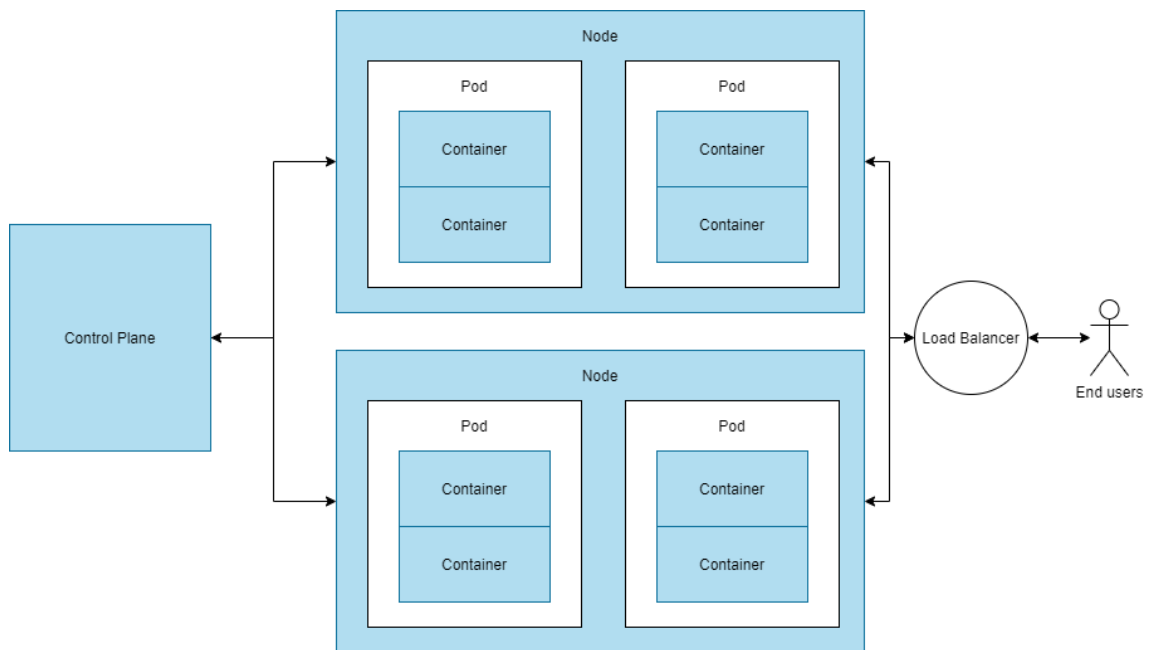
3 Käytetyt teknologiat

3.1 Kubernetes

Kubernetes on hyvin suosittu avoimeen lähdekoodiin perustuva järjestelmä, joka mahdollistaa konttien (eng. container) hallinnoimisen automatisoimisen [6]. Kubernetes helpottaa muun muassa sovellusten käyttöönottoa, hallintaa ja skaalausta. [5.]

Kubernetes on alun perin Googlen kehittämä alusta, jonka ensimmäinen versio julkaistiin vuoden 2015 heinäkuussa ja viimeisin versio (1.19.3) julkaistiin lokakuussa 2020. Kubernetesin kehitykseen vaikutti vahvasti Googlen toinen projekti nimeltä Borg. Borg on klusterinhallintatyökalu, joka on suunniteltu suurille klustereille. Nykyään Kubernetesin kehityksestä vastaa Googlen sijasta Cloud Native Computing Foundation (CNCF).

Kuvassa 2 näkyy Kubernetesin eri komponentit ja niiden suhteet toisiinsa. Kubernetes on suunniteltu skaalautumaan kaiken kokoisille sovelluksille. Kubernetes saavuttaa lähes rajattoman skaalautuvuuden käyttämällä *klusteria*. Kubernetes-klusterit koostuvat nodeista. *Node* on Kubernetes-klusterin käytössä olevia laitteita. Klusteria hallinnoi *control plane*, joka tekee päätöksiä koko klusterin puolesta. Control plane sisältää prosesseja, jotka esimerkiksi ajoittavat klusterin nodeja ja käynnistävät nodeja tarpeen mukaan. [7.]



Kuva 2. Yksinkertaistettu Kubernetes-arkkitehtuuri.

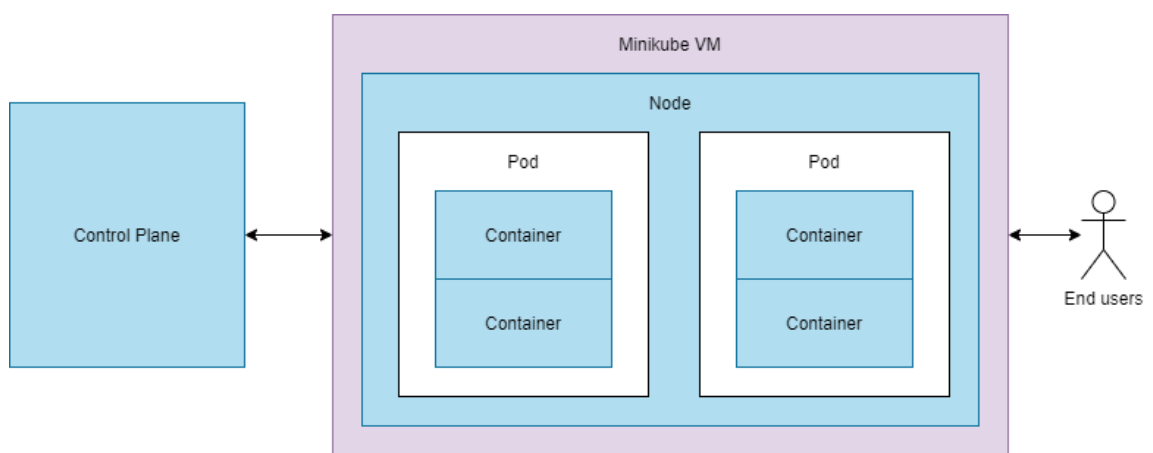
Kubernetes on otettu käyttöön monessa yrityksessä moniin eri projekteihin. Suuret ja tunnetut yritykset kuten Adidas, IBM, Nokia ja Spotify ovat kaikki ottaneet Kubernetesen käyttöön eri syistä [8]. Esimerkiksi Spotify otti Kubernetesen käyttöön, koska siinä on enemmän ominaisuuksia kuin heidän aikaisemmin käyttämässään alustassa. Lisäksi Kubernetesen ympärille on rakentunut suuri yhteisö, ja Spotify halusi ottaa käyttöön alustan, joka oli muuttumassa standardiksi konttijärjestelmien keskuudessa [9].

Kubernetesista usein vertaillaan Dockeriin ja muihin samankaltaisiin alustoihin. Kubernetes ei kuitenkaan ole tarkoitettu Dockerin korvaajaksi, vaan ne hyötyvät toistensa toiminnasta. Docker on ohjelmisto, joka mahdollistaa konttien ja niihin pakattujen sovellusten ajamisen. Kubernetes puolestaan on suunniteltu automatisoimaan esimerkiksi Dockerin kaltaista konttialustaa. Kubernetesistä nimittäin suositellaan käytettävän juuri Dockerin kanssa yhdessä [10].

3.2 Minikube

Kubernetes on kehitetty tukemaan monia laitteita ja suuria sovelluksia. Kubernetes on kuitenkin kehittänyt työkalun, joka mahdollistaa Kubernetesen käytön paikallisesti.

Minikube on työkalu, joka mahdollistaa Kubernetesen käytön paikallisesti yhdellä laitteella. Minikuben käyttöä suositellaan sovellusten kehittämiseen sekä Kubernetesen opettelemiseen. Minikube luo virtuaalikoneen, jonka sisälle se asentaa yhden noden klusterin (kuva 3). Node voi sisältää podeja ja palveluja samalla tavalla kuin normaalissa Kubernetesen käytössä.



Kuva 3. Yksinkertaistettu Minikube-arkkitehtuuri.

Minikube sisältää lähes kaikki Kubernetesen ominaisuudet, joten sitä voi käyttää ohjelmistokehityksessäkin eikä ainoastaan Kubernetesen opettelemiseen. Lisäksi Minikubea voi käyttää Windowsilla, Macilla sekä Linuxilla. Minikube on kevyt versio Kubernetesestä, ja siksi tämä projekti toteutetaan Minikubella.

3.3 Pod

Kubernetesessä kontteja ei ajeta suoraan, vaan ne on kääritty podeihin. Pod (suom. palko) on Kubernetesen pienin ajettava yksikkö, joka sisältää yhden tai useamman kontin. Podien on tarkoitus olla lyhytaikaisia ja kertakäyttöisiä yksiköitä, joita voidaan luoda lisää tarpeen mukaan ja poistaa, jos pod tai sitä ympäröivä node kaatuvat.

Kubernetesissa podeja voidaan konfiguroida JSON- tai YAML-tiedostoilla. Käytännössä YAML- ja JSON-tiedostotyyppien välillä ei ole mitään merkityksellisiä funktionaalisia eroja ja tätä työtä varten päätettiin käyttää ainoastaan JSON-tiedostoja. Podien konfiguraatitiedostoon voi määritellä erinäisiä ominaisuuksia kuten podin sisältämät kontit tai alustuskontteja, joita pod käyttää vain silloin, kun pod käynnistetään.

3.4 Palvelut

Kubernetes jakaa podeille automaattisesti IP-osoitteet, kun pod luodaan. Kubernetesin klusterimallin takia podeille jaetut IP-osoitteet vaihtelevat, eikä niitä pysty määrittelemään esimerkiksi podin konfiguraatitiedostoissa. Podiin ei siis saa luotettavasti suoraa yhteyttä, vaan on käytettävä palvelua (Kubernetes service).

Kubernetesissä palvelu on olio, joka määrittelee tietyn joukon podeja sekä säännöt podeihin pääsyä varten. Yleensä palvelut kohdistuvat podeihin, jolloin luodaan ylimääräinen abstraktiotaso podiin pääsyä varten. Podeihin kohdistuminen saadaan aikaiseksi valitsimilla (selector), joilla viitataan podin tunnisteisiin (label). Tunnisteet ovat Kubernetes-olioihin määritellyjä avain-arvo-pareja, joita voidaan käyttää podin tunnistamiseen. Palvelun voi myös kohdistaa esimerkiksi toiseen palveluun, jolloin kohdistuminen on asetettava manuaalisesti kohdepalvelun IP-osoitteen avulla.

Tässä dokumentissa termi palvelulla tarkoittaa nimenomaan Kubernetesin abstraktiota eikä geneeristä palvelua.

4 Minikuben asennus

Minikube asennettiin VDI:tä käyttäen etättyöasemalle CentOS 8 -käyttöjärjestelmälle. Etättyöasemalle oli mahdollisuus yhdistää joko VNC-järjestelmää tai SSH-protokollaa käyttäen. SSH on yleisesti käytetty protokolla etäyhteyksiä varten ja tässä työssä käytettiin vain sitä.

Minikube pystyy käyttämään useita eri ajureita eri käyttöjärjestelmille ja kontti- tai VM-pohjaisille virtualisointimalleille. Linux-pohjaisille käyttöjärjestelmille Minikube tukee Docker-, KVM2-, VirtualBox- ja PodMan-ajureita. Lisäksi käyttäjällä on mahdollisuus käyttää vaihtoehtoa None, joka mahdollistaa Minikuben käytön käyttäjän määrittelemässä VM-ympäristössä [11].

Tähän työhön kuitenkin valittiin Docker-ajuri. Tähän valintaan päädyttiin, koska ohjelmisto, joka Minikubeen on tarkoitus siirtää, on jo valmiiksi Dockeria käyttävä ohjelmisto. Lisäksi Docker-ajuri on helppokäyttöisin saatavilla olevista ajureista.

Docker-enginen asennukseen käytettiin dnf-paketinhallintajärjestelmää. Paketinhallintajärjestelmät automatisoivat ohjelmien asennusta ja hallintaa. Dnf on oletuspaketinhallintajärjestelmä CentOS 8 -käyttöjärjestelmässä. Paketinhallintajärjestelmät kuten dnf käyttävät .repo-loppuisia tiedostoja ohjelmien löytämiseen. Fail-Safella on omat pakettivarastot, jotka eivät ole paketinhallintajärjestelmille ennestään tuttuja. Monissa Linux-pohjaisissa käyttöjärjestelmissä on olemassa hakemisto nimeltä /etc/yum.repos.d, joka sisältää repo-tiedostoja. Tähän hakemistoon lisättiin tiedot yrityksen omasta pakettivarastosta tiedostoon docker-ce.repo, jonka jälkeen dnf osaisi löytää pakettivarastosta löytyvät Docker-paketit.

Docker-engine asennettiin dnf-komennolla. Komento koostuu neljästä osasta, joista ensimmäinen on itse dnf ja se kertoo Linuxille, mitä ohjelmaa halutaan käyttää. Toinen osa komennosta on install. Install on dnf-komennon alikommento, joka kertoo dnf:lle, että halutaan asentaa jokin ohjelma. Komennon kolmas osa eli docker-ce viittaa aiemmin luotuun repo-tiedostoon, joka puolestaan viittaa yrityksen omaan pakettivarastoon. Komennon viimeinen osa on esimerkki unix-tyylisissä käyttöjärjestelmissä käytetyistä komentojen optioista. Perinteisesti optiot alkavat yhdellä miinusmerkillä (ASCII 0x2D), mikäli käytetään lyhyttä optiota, ja kahdella miinusmerkillä, jos taas käytetään pitkää optiota. Tässä

tapauksessa optio "--nobest" ilmoittaa dnf-komennolle, että ei tarvitse yrittää löytää parasta mahdollista asennettavan ohjelman versiota, koska CentOS suosii oletuksena Podmania.

Minikubea hallitaan kubectl-ohjelmalla. Ohjelma asennetaan lataamalla sen binääritiedosto curl-komennolla ja antamalla ladatulle tiedostolle ajo-oikeudet (Esimerkkikoodi 1). Curl hakee kubectl:n googleapis-sivulta, jonka jälkeen haetulle tiedostolle annetaan ajo-oikeus chmod-komentoa käyttäen. Linux-järjestelmissä chmod on yleisesti käytetty komento, jonka avulla käyttäjä voi muuttaa tiedostojen oikeuksia. Komennossa "+x" tarkoittaa, että tiedostolle kubectl halutaan lisätä ajo-oikeus (executable).

```
curl -O https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl && chmod +x kubectl
```

Esimerkkikoodi 1. Kubectl-ohjelman asennus.

Lähes kaikissa käyttöjärjestelmissä on jonkinlainen ympäristömuuttuja, joka kertoo käyttöjärjestelmälle, missä kaikki komennot ja ohjelmat säilytetään. CentOS 7 -käyttöjärjestelmässä ja monissa muissa UNIX-tyylisissä käyttöjärjestelmissä tämä ympäristömuuttuja on PATH. Jos siis halutaan, että kubectl-komento löytyy mistä tahansa työhakemistosta, niin joko sen sijainti täytyy lisätä PATH-ympäristömuuttujaan tai sitten kubectl on siirrettävä hakemistoon, joka on jo PATH-muuttujassa. Tässä tapauksessa kubectl siirrettiin käyttäjän kotihakemiston bin-hakemistoon (~/.bin), joka sisältää käyttäjän omat binääritiedostot ja joka on oletuksena kirjattuna PATH:ssa.

Itse Minikube asennetaan samalla tavalla kuin kubectl. Minikube haetaan curl-komennolla, tiedostolle lisätään ajo-oikeudet chmod-komennolla ja se siirretään käyttäjän kotihakemiston bin-hakemistoon, jotta sen voi ajaa ilman täydellistä polkua mistä vain.

Minikube on tässä vaiheessa toimintakykyinen ja sen pystyy käynnistämään minikube start -komennolla. Komentoon lisättiin --driver=docker, joka pakottaa minikuben käyttämään Dockerin ajuria. Ajuria ei tarvitse erikseen määrittää, jos muita ajureita ei ole asennettuna, mutta kehitysympäristöistä saattaa löytyä muitakin ajureita.

Minikubeen asennettiin lisäksi SSL-sertifikaatti pem-tiedostoon sekä luotiin salaisuudet komennolla "kubectl create secret". Tämä sisältää muun muassa tunnukset yrityksen

Docker-rekisteriin, josta haetaan konttien imaget. Lopuksi lisättiin käyttäjä docker-ryhmään ja ajettiin vielä komennot `minikube stop` ja `minikube start`, jotta muutokset astuisivat voimaan. Käyttäjä täytyy lisätä docker-ryhmään, jotta pöytä pystyy ajamaan Minikubella (esimerkkikoodi 2).

```
gpasswd -a $USER docker
```

Esimerkkikoodi 2. Käyttäjän lisääminen docker-ryhmään.

Minikuben toimintaa voidaan testata luomalla sinne yksinkertaisen podin. Esimerkkikoodissa 3 määritellään yksinkertainen pod, joka sisältää yhden kontin. Kontissa on yksinkertainen image CentOS 7 -käyttöjärjestelmästä.

```
{
  "metadata": {
    "name": "test-pod",
    "namespace": "default"
  },
  "apiVersion": "v1",
  "kind": "Pod",
  "spec": {
    "containers": [
      {
        "image": "docker.fail-safe.net/net-fail-safe-tools/docker-
baseimages/centos7/base:119",
        "imagePullPolicy": "Always",
        "name": "test-container"
      }
    ],
    "imagePullSecrets": [
      {
        "name": "docker.fail-safe.net"
      }
    ],
    "restartPolicy": "Always"
  }
}
```

Esimerkkikoodi 3. Testipodin konfiguraatiodostoto test.json.

Kuvassa 4 ajetuilla komennoilla voidaan käynnistää pod, tarkastella sen tilaa sekä pysäyttää se. Tulostuksista näkyy, että pod saadaan käynnistettyä onnistuneesti.

```
$ kubectl apply -f test.json
pod/test-pod created
$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
test-pod     1/1     Running   0          7s
$ kubectl delete -f test.json
pod "test-pod" deleted
```

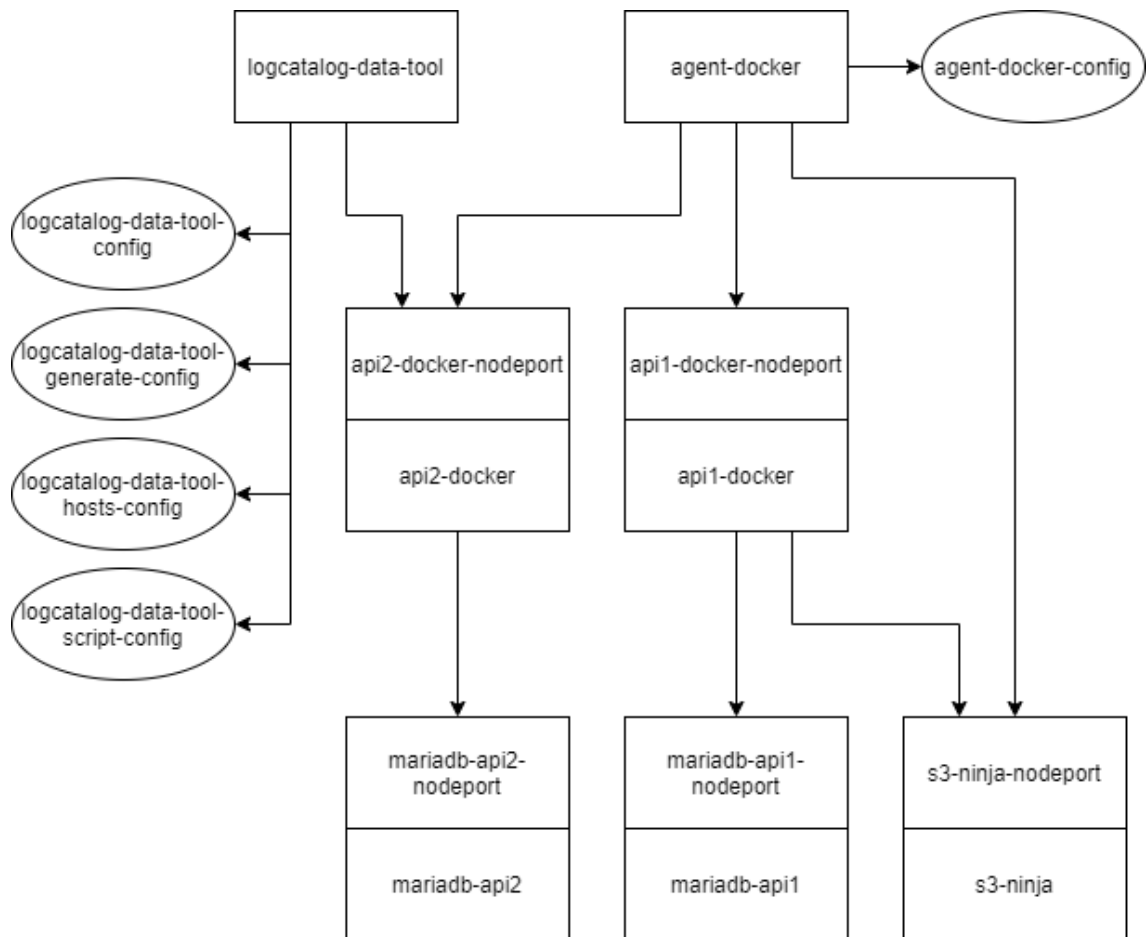
Kuva 4. Testipodin käynnistys ja pysäytys.

5 Ohjelmiston siirtäminen Kubernetesiin

5.1 Suunnittelu

Kubernetesin podit konfiguroitiin JSON-tiedostojen avulla. Konfiguraatitiedostoissa käyttäjä voi määritellä muun muassa podin sisällä olevat kontit sekä niiden imaget. Podien konfiguraatitiedostojen lisäksi luotiin configmappeja podien käytettäväksi. Configmapit ovat Kubernetesin olioita, joilla voidaan säilyttää ei-luottamuksellista dataa avain-arvo-pareina. Configmappeja luodaan työn aikana, jotta saadaan esimerkiksi applikaation konfiguraatitiedostoja podien sisälle ilman, että joudutaan muokkaamaan koko imagea.

Yrityksen ohjelmiston komponentit sekä niiden väliset riippuvuussuhteet määriteltiin etukäteen. Kuvassa 5 on eritelty ohjelmiston komponentit, joille halutaan luoda omat podit. Kuvassa näkyy myös palvelut (nodeport-loppuiset) sekä configmapit (soikiot), jotka on määriteltävä, jotta ohjelmisto saadaan siirrettyä Kubernetesiin onnistuneesti.



Kuva 5. Kaavio ohjelmiston komponenteista sekä niiden riippuvuussuhteista.

Kuva 2 kuvaa lopullista Kubernetesiin siirretyn ohjelmiston arkkitehtuuria. Kaavio sisältää sekä podit että palvelut, jotka halutaan Kubernetesiin. Yksi laatikko kuvaa joko podia tai podiin sidottua palvelua. Palvelut ovat kaikki tyyppiä "nodeport", ja ne tunnistetaan kaaviosta niiden nimestä. Lisäksi kaavioon on merkitty config-loppuiset configmapot, joilla injektoidaan dataa podeihin, kun ne käynnistetään. Nuolet taas kuvaavat komponenttien välisiä riippuvuuksia. Kaaviosta näkyy, että esimerkiksi api2-docker on riippuvainen mariadb-api2:sta, jolloin mariadb-api2:lle täytyy luoda palvelu, jota api2-docker pystyy käyttämään. Toisaalta esimerkiksi agent-dockerista ei riipu mikään, joten sille ei tässä vaiheessa tarvitse luoda omaa palvelua ollenkaan. Työssä luotiin podit sekä palvelut kaavion mukaan alhaalta ylöspäin riippumattomista riippuvaisiin.

5.2 Podin mariadb-api2 määrittäminen

Ensimmäisenä esimerkkinä podin konfiguraatiosta, joka luotiin on mariadb-api2.json (esimerkkikoodi 4). Tämä pod sisältää MariaDB tietokannan, joka on välttämätön muulle ohjelmistolle. Tästä podista tehtiin myöhemmin palvelu, jotta muut podit voivat yhdistyä siihen.

```
{
  "metadata": {
    "name": "mariadb-api2",
    "namespace": "default",
    "labels": {
      "run": "lms-stack",
      "app": "mariadb-api2",
      "env": "dev"
    }
  },
  "apiVersion": "v1",
  "kind": "Pod",
  "spec": {
    "dnsPolicy": "ClusterFirst",
    "terminationGracePeriodSeconds": 0,
    "containers": [
      {
        "image": "docker.fail-safe.net/net-fail-safe-archive/mari-
adb:33",
        "imagePullPolicy": "Always",
        "name": "mariadb",
        "ports": [
          {
            "protocol": "TCP",
            "name": "mariadb",
            "containerPort": 3306
          }
        ],
        "env": [
          {
            "name": "MYSQL_ROOT_PASSWORD",
            "value": "placeholder_password"
          },
          {
            "name": "MYSQL_DATABASE",
            "value": "placeholder_database"
          },
          {
            "name": "MYSQL_USER",
            "value": "placeholder_user"
          },
          {
            "name": "MYSQL_PASSWORD",
            "value": "placeholder_password"
          }
        ]
      }
    ],
    "imagePullSecrets": [
      {
        "name": "docker.fail-safe.net"
      }
    ]
  }
}
```

```

    ],
    "restartPolicy": "Always"
  }
}

```

Esimerkkikoodi 4. Podin konfiguraatiotiedosto mariadb-api2.json.

Metadatan kentät sisältävät muun muassa podin nimen ja nimitilan. Kubernetesissa fyysisen klusterin voi jakaa virtuaaliklustereiksi käyttäen nimitiloja. Klusterin jakaminen voi helpottaa koko järjestelmän hallitsemista, jos ympäristössä on paljon käyttäjiä useammassa ryhmässä. Jokaisessa nimitilassa olioiden nimien on oltava uniikkeja, eli yhdessä nimitilassa ei saa olla kahta samannimistä podia. Tämän takia ensimmäisen podin nimeksi asetettiin mariadb-api2 eikä pelkästään mariadb, koska ohjelmistoon määriteltiin toinenkin MariaDB-tietokannan sisältävä pod, joka palvelee ohjelmiston muuta komponenttia. Lopulta metadatatassa on vielä yksi kenttä, eli "label" (suom. tunniste). Näitä käytettiin myöhemmin palvelun luomisessa, mutta niitä käytetään valitsimina, joiden avulla voidaan vaikuttaa useampaan podiin samanaikaisesti.

Konfiguraatiotiedoston kenttä "apiVersion" määrittelee, mitä Kubernetes API:n versiota halutaan käyttää olion luomiseen. "Kind"-kenttä puolestaan kertoo, minkä tyyppinen olio on. "Spec"-kenttä on podin määrittelyn tärkein, koska se sisältää kaiken tiedon podin sisällöstä.

DNS-käytännöt voi määrittää jokaiselle podille erikseen. Ensimmäinen spec-kentän sisäinen kenttä on dnsPolicy, ja se määrittää podin DNS-käyttötavan.

Kubernetesin pod voi sisältää useamman kontin. Kontit määritetään spec.containers-kentässä. Vaikka podissa voi olla useita kontteja, kuitenkin suositellaan, että yksi pod sisältäisi vain yhden loogisen kokonaisuuden. Tässä projektissa riittää, että kaikki podit sisältävät vain yhden kontin.

Kontin image-kenttä kertoo Kubernetesille, mistä kyseisen kontin image täytyy hakea ja imagePullPolicy määrittää, haetaanko image aina määritetystä rekisteristä vai yritetäänkö käyttää jo haettua imagea. ImagePullPolicyn vaihtoehtoina ovat "Always" (aina), "IfNotPresent" (jos imagea ei löydy paikallisesti) ja "Never" (aina käytetään paikallista). Samoin kuin podin nimi, niin kontinkin nimi on oltava uniikki podin sisällä.

Kontin portteja voi määritellä "ports"-kentässä. Porttiin voi määritellä sen nimen sekä portin protokollan ja numeron. Portti 3306 on rekisteröity MySQL-tietokantajärjestelmille ja on sen takia määritelty nimenomaan tämän kontin portin numeroksi [12].

Kontteihin on myös määritelty ympäristömuuttujia, jotka niiden sensitiivisyytensä takia on muutettu tässä dokumentissa. Tähän määritellyt ympäristömuuttujat ovat kyseisessä kontissa saatavilla. Lopuksi imagePullSecrets määrittää secretin, josta löytyvät esimerkiksi tunnukset yrityksen rekisteriin. RestartPolicy määrittää podin uudelleenkäynnistykseen. Kun restartPolicy on "Always", pod käynnistetään aina uudelleen, jos se ei ole käynnissä eli myös niissä tapauksissa, kun pod sammuu ilman virheitä. "OnFailure"-vaihtoehdolla pod käynnistetään vain niissä tapauksissa, kun pod kaatuu jonkin virheen takia ja "Never"-vaihtoehdolla podia ei käynnistetä uudelleen, jos se kaatuu.

Pod käynnistettiin komennolla `kubectl apply -f mariadb-api2.json`, jossa `-f mariadb-api2.json` kertoo komennolle, että podin konfiguraatio löytyy kyseisestä tiedostosta. Podin tilaa voi tarkkailla komennolla `kubectl get pods` (kuva 6), josta näkyy, että pod on käynnistynyt ilman virheitä.

```
$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
mariadb-api2  1/1     Running   0           12s
```

Kuva 6. Komennon `kubectl get pods` tulostus.

5.3 Palvelun mariadb-api2-nodeport määrittäminen

Kubernetes jakaa IP-osoitteita podoille dynaamisesti. IP-osoitteet saa näkyville `kubectl get pods -o wide`-komennolla, kun sille antaa lisäksi option `-o wide` (kuva 7). Tällöin komento tulostaa ylimääräisiä tietoja kuten podien IP-osoitteet. Näitä osoitteita voi käyttää samoin kuin muitakin, mutta Kubernetes ei aina anna podoille samoja osoitteita, kun ne käynnistetään uudestaan. Tämän takia ne täytyy sijoittaa niin kutsuttuihin palveluihin. Palvelulle voidaan määrittää tiettyyn nimitilaan nimi ja sen voi sijoittaa podiin, jolloin palvelun nimeä käyttäen voidaan yhdistää podiin ilman, että sen IP-osoitetta täytyy tietää.

```
$ kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP           NODE       NOMINATED NODE   READINESS GATES
mariadb-api2  1/1     Running   0           32s   172.18.0.4   Mini-kube <none>   <none>
```

Kuva 7. Komennon `kubectl get pods -o wide` tulostus.

Palvelut määritellään samankaltaisella tavalla podien kanssa. Mariadb-api2:lle luotiin palvelu tiedostoon `mariadb-api2-nodeport.json` (esimerkkikoodi 5), joka sisältää palvelun konfiguraatiot.

```
{
  "metadata": {
    "name": "mariadb-api2-nodeport",
    "namespace": "default"
  },
  "apiVersion": "v1",
  "kind": "Service",
  "spec": {
    "selector": {
      "app": "mariadb-api2",
      "env": "dev"
    },
    "ports": [
      {
        "protocol": "TCP",
        "port": 3306,
        "name": "mariadb",
        "targetPort": 3306
      }
    ],
    "type": "NodePort"
  }
}
```

Esimerkkikoodi 5. Palvelun konfiguraatitiedosto `mariadb-api2-nodeport.json`.

Palvelun määrittämisessä `name` ja `namespace` määrittävät osoitteen, jonka kautta pääsee palvelun takana olevalle podille. Esimerkiksi `mariadb-api2`-podin sisällä olevaan tietokantaan voi yhdistää osoitteesta `mariadb-api2-nodeport.default`.

Palvelu sidotaan podiin valitsimilla. Kun palvelu ajetaan ylös, se yhdistetään podiin, jolta löytyvät tunnisteista avain-arvo-parit, jotka on määritelty palvelun `selector`-kenttään. `Mariadb-api2`:een asetettiin sille uniikki avain-arvo-pari `"app": "mariadb-api2"`, ja sama asetettiin sille tarkoitetun palvelun valitsimiksi.

Palvelun portteihin määritellään sekä portti, jota käytetään palveluun yhdistämiseen, että portti, joka on määritelty podin konfiguraatioon (`targetPort`). Palvelun tyyppiä määriteltiin `NodePort`, jolloin palvelu paljastetaan staattiselle portille.

Palvelu käynnistetään samalla komennolla kuin podit. Kun komento `kubectl apply -f mariadb-api2-nodeport.json` on ajettu, niin palvelu on täysin toimintakelpoinen. Ajossa olevat palvelut sekä niiden portit ja IP-osoitteet saadaan näkyville komennolla `kubectl get svc` (kuva 8).

```
$ kubectl get svc
NAME                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
mariadb-api2-nodeport NodePort      10.107.21.40  <none>         3306:30223/TCP  22s
```

Kuva 8. Komennon `kubectl get svc` tulostus.

Palvelun ja podin toiminta testattiin tässä vaiheessa. Ylimääräinen pod luotiin testaamista varten. Pod sisältää yksinkertaisen CentOS 7 imagen, johon asennettiin mysql-client. Podiin pääsee sisälle komennolla, jonka syntaksi on muotoa `kubectl exec -it <podin nimi> -- bash`. Sieltä testattiin yhteyttä mysql-komennolla. Jos yhteys toimii, niin vastauksena pitäisi tulla `access denied`. Mysql-komentoon on määritelty `mariadb-api2-nodeport`-palvelussa asetetut arvot hostin nimelle ja nimitilalle. Kuvan 9 tulostuksesta näkyy, että yhteys toimii toivotulla tavalla.

```
$ kubectl exec -it test -- bash
[root@test /]# mysql -h mariadb-api2-nodeport.default
ERROR 1045 (28000): Access denied for user 'root'@'172.18.0.3' (Using password: NO)
```

Kuva 9. Tietokannan ja palvelun testaaminen.

5.4 Podin `api2-docker` määrittäminen

`api2-docker` määritettiin samankaltaisesti kuin `mariadb-api2` muutamaa lisäystä lukuun ottamatta esimerkkikoodin 6 kuvaamalla tavalla. Tämä pod sisältää osan yrityksen ohjelmistosta, joka käyttää aikaisemmin luodun `mariadb-api2`-podin sisältämää tietokantaa.

```
{
  "metadata": {
    "name": "api2-docker",
    "namespace": "default",
    "labels": {
      "run": "lms-stack",
      "app": "api2-docker",
      "env": "dev"
    }
  },
  "apiVersion": "v1",
  "kind": "Pod",
```

```

"spec": {
  "dnsPolicy": "ClusterFirst",
  "terminationGracePeriodSeconds": 0,
  "containers": [
    {
      "image": "docker.fail-safe.net/net-fail-safe-archive/api2-
docker:117",
      "imagePullPolicy": "Always",
      "name": "api2-docker",
      "ports": [
        {
          "protocol": "TCP",
          "name": "api2",
          "containerPort": 8080
        }
      ],
      "env": [
        {
          "name": "FS_CLIENT_ARCHIVER_BATCH_SECRET",
          "value": "placeholder_secret"
        },
        {
          "name": "FS_CLIENT_API2_BATCH_SECRET",
          "value": "placeholder_secret"
        },
        {
          "name": "FS_CLIENT_API2_UI_SECRET",
          "value": "placeholder_secret"
        },
        {
          "name": "FS_SPRING_DATASOURCE_URL",
          "value": "jdbc:mysql://mariadb-api2-nodeport.de-
fault/placeholder_database"
        },
        {
          "name": "FS_SPRING_DATASOURCE_PASSWORD",
          "value": "placeholder_password"
        },
        {
          "name": "FS_SPRING_DATASOURCE_USERNAME",
          "value": "placeholder_user"
        }
      ]
    }
  ],
  "initContainers": [
    {
      "name": "init-api2-docker-1",
      "image": "docker.fail-safe.net/net-fail-safe-tools/docker-
baseimages/centos7/base:119",
      "env": [
        {
          "name": "FS_SPRING_DATASOURCE_HOSTNAME",
          "value": "mariadb-api2-nodeport.default"
        },
        {
          "name": "FS_SPRING_DATASOURCE_PORT",
          "value": "3306"
        },
        {
          "name": "FS_SPRING_DATASOURCE_DATABASE",
          "value": "placeholder_database"
        },
        {
          "name": "FS_SPRING_DATASOURCE_USERNAME",

```

```

        "value": "placeholder_user"
    },
    {
        "name": "FS_SPRING_DATASOURCE_PASSWORD",
        "value": "placeholder_password"
    }
],
"command": ["/bin/sh", "-c", "yum -y install mysql; for i in
{1..5}; do sleep 1; row=$(mysql -h$FS_SPRING_DATASOURCE_HOSTNAME -
P$FS_SPRING_DATASOURCE_PORT -D$FS_SPRING_DATASOURCE_DATABASE -
u$FS_SPRING_DATASOURCE_USERNAME -p$FS_SPRING_DATASOURCE_PASSWORD -Ns -e 'SE-
LECT 1'); echo \"Returned $row\"; if [ $row -eq 1 ]; then exit 0; fi; done;
exit 1;"]
}
],
"imagePullSecrets": [
    {
        "name": "docker.fail-safe.net"
    }
],
"restartPolicy": "Always"
}
}

```

Esimerkkikoodi 6. Podin määrittelytiedosto api2-docker.json.

Api2-docker-podin spec-kenttään lisättiin kuitenkin kenttä `initContainers`. Kubernetesissä on mahdollisuus määrittää kontteja, jotka ajetaan silloin, kun pod käynnistetään. Podia ja sen sisältämiä kontteja ei käynnistetä ennen kuin `initContainer`-kentässä määritellyt kontit ovat lopettaneet suorittamisen virheettömästi. Nämä alustuskontit voivat sisältää skriptejä ja apuohjelmia, joita ei ole sovelluksen konteissa ja joita tarvitaan vain konttien alustukseen tai esimerkiksi tarkastamaan, että palvelut, joista pod on riippuvainen, ovat toiminnassa.

Alustuskontin haluttiin tässä tapauksessa testaavan, että tietokanta on toiminnassa ennen kuin pod käynnistetään. Alustuskontin imagena käytettiin tyhjää CentOS 7 -käyttöjärjestelmän imagea. Alustuskontin konfiguraatiossa oleva `command`-kenttä sisältää komennon, joka ajetaan, kun alustuskontti käynnistetään. Tässä tapauksessa komento on `sh`, jolle annetaan parametreiksi `-c` ja shell-skripti, joka halutaan ajaa.

Käytössä ei ollut imagea, jossa olisi MySQL-client valmiina asennettuna, minkä takia skriptin ensimmäinen osa asentaa sen. Tämä ei ole optimaalinen tapa saada MySQL käyttöön kontissa, koska esimerkiksi tuotantoympäristöissä ei ole välttämättä yhteyttä paikallisen verkon ulkopuolelle turvallisuussyistä. Lisäksi tämä menetelmä on suhteellisen hidas, mutta tässä vaiheessa haluttiin vain konseptitodistus alustuskonteista, joten ratkaisu oli toistaiseksi riittävä. Konseptitodistuksella (eng. Proof of concept) on tarkoitus

todistaa, että jokin ajatus tai menetelmä on mahdollista toteuttaa. Tulevaisuudessa kuitenkin voisi olla olemassa imaget, jotka sisältävät valmiiksi testaamiseen tarvittavat ohjelmat ja skriptit, jotka ajettaisiin automaattisesti, kun kontti käynnistetään. Silloin koko komentokentän voisi jättää alustuskontin konfiguraatiosta pois, ja se tekisi podin käynnistyksestä yksinkertaisempaa.

Loput komennossa ajettavasta skriptistä testaa sekunnin välein toimiiko tietokantayhteys halutulla tavalla. Jos skriptin lopetuskoodi on mitään muuta kuin 0, niin kontti tulkitsee sen virheeksi eikä pod käynnisty. Koska restartPolicy-kentälle on asetettu arvoksi "Always", pod yritetään käynnistää uudelleen, jos skripti epäonnistuu, eli Kubernetes yrittää käynnistää podia niin kauan, että alustuskontti onnistuu eli tietokantayhteys toimii.

Api2-docker sisältää API:n, jota käytetään muissa podeissa. Siitä siis luotiin myös palvelu samalla tavalla kuin mariadb-api2:lle. Api2-docker-podin palvelu määritettiin siten, että se löytyy osoitteesta api2-docker-nodeport.default:8080.

5.5 Podin logcatalog-data-tool-määrittäminen

Pod logcatalog-data-tool sisältää muun muassa testidataa sekä tiedostoja, joita voidaan muuttaa tarpeen mukaan. Näitä tiedostoja ei ole käytännöllistä muuttaa imagessa, koska joka kerta kun niiden sisältöä halutaan muuttaa, niin koko image pitäisi luoda uudestaan. Tämän takia käytetään Kubernetesin configmap-olioita. Configmapit mahdollistavat ei-luottamuksellisen datan siirtämisen podeihin niiden luontivaiheessa.

Configmapit täytyy ensin luoda isäntälaitteen tiedostoista tai hakemistoista. Yksi configmap voi sisältää useamman tiedoston, mutta jokaiselle configmapille määritellään yksi asennuspiste podin konfiguraatitiedostossa, joten kaikki yhden configmapin sisältämät tiedostot menevät samaan paikkaan. Tällöin, jos haluaa käyttää configmappeja asentamaan dataa eri paikkoihin, niin täytyy luoda configmapit jokaiselle sijainnille.

Tähän podiin halutaan asentaa tiedostoja neljään eri hakemistoon, joten luomme neljä hakemistoa, joista luodaan configmapit. Hakemistojen nimet ovat:

- logcatalog-data-tool-configmap

- logcatalog-data-tool-generate-configmap
- logcatalog-data-tool-hosts-configmap
- logcatalog-data-tool-script-configmap.

Configmapit eivät voi sisältää alihakemistoja, joten kaikki hakemistot, joista luodaan configmapit sisältävät vain ja ainoastaan tiedostoja. Jokaisesta hakemistosta luotiin configmapit käyttäen jälleen kubectl-komentoa (esimerkkikoodi 7).

```
kubectl create configmap logcatalog-data-tool-config --from-file=logcatalog-
data-tool-configmap/
```

Esimerkkikoodi 7. logcatalog-data-tool-config-nimisen configmapin luontikomento.

Configmapit näkyvät kubectl get configmap -komennolla. Olemassa olevia configmappeja voi lisätä podeihin ja kontteihin määrittämällä niille polut kontin sisällä. Ensin configmapit on määriteltävä volumeiksi. Jokaiselle volumelle määritellään nimi sekä configmap, jonka dataa halutaan käyttää. Podiin konfiguraatioon määriteltyjä volumeja voi käyttää podin konteissa määrittämällä volumeMounts-kenttään halutut volumet. Näin on tehty logcatalog-data-tool-podin konfiguraatitiedostossa, joka on esitelty esimerkkikoodissa 8. VolumeMounts-kenttään määritellään, minne volume halutaan asentaa sekä mikä pod-tasolla määritellyistä volumeista sinne halutaan asentaa.

```
{
  "metadata": {
    "name": "logcatalog-data-tool",
    "namespace": "default"
  },
  "apiVersion": "v1",
  "kind": "Pod",
  "spec": {
    "dnsPolicy": "ClusterFirst",
    "terminationGracePeriodSeconds": 0,
    "volumes": [
      {
        "name": "config-volume",
        "configMap": {
          "name": "logcatalog-data-tool-config"
        }
      },
      {
        "name": "hosts-config-volume",
        "configMap": {
          "name": "logcatalog-data-tool-hosts-config"
        }
      },
      {
        "name": "generate-config-volume",
        "configMap": {
          "name": "logcatalog-data-tool-generate-config"
        }
      }
    ]
  }
}
```

```

    }
  },
  {
    "name": "script-config-volume",
    "configMap": {
      "name": "logcatalog-data-tool-script-config"
    }
  }
],
"containers": [
  {
    "image": "docker.fail-safe.net/net-fail-safe-lms/rest-02/log-
catalog-data-tool:146",
    "imagePullPolicy": "Always",
    "name": "logcatalog-data-tool",
    "ports": [
      {
        "protocol": "TCP",
        "name": "data-tool",
        "containerPort": 8080
      }
    ],
    "volumeMounts": [
      {
        "mountPath": "/opt/Fail-Safe/rest-02-data-tool/etc",
        "name": "config-volume"
      },
      {
        "mountPath": "/config/hosts",
        "name": "hosts-config-volume"
      },
      {
        "mountPath": "/data-tool/logs",
        "name": "generate-config-volume"
      },
      {
        "mountPath": "/data-tool/script",
        "name": "script-config-volume"
      }
    ],
    "env": [
      {
        "name": "SPRING_MAIN_WEB_ENVIRONMENT",
        "value": "false"
      },
      {
        "name": "SPRING_MAIN_BANNER_MODE",
        "value": "off"
      },
      {
        "name": "REST_LOG_CATALOG_TOKEN_URL",
        "value": "http://api2-docker-nodeport.de-
fault:8080/oauth/token"
      },
      {
        "name": "REST_LOG_CATALOG_CLIENT_SECRET",
        "value": "placeholder_secret"
      },
      {
        "name": "REST_LOG_CATALOG_ENDPOINT",
        "value": "http://api2-docker-nodeport.default:8080/api"
      }
    ],
    "command": [
      "/bin/sh"
    ]
  }
]

```

```

    ],
    "args": [
      "-c", "tar -xzf /config/hosts/hosts.tar.gz -C /data-tool;
/bin/bash /data-tool/script/data-tool.sh; sleep inf;"
    ]
  },
  "imagePullSecrets": [
    {
      "name": "docker.fail-safe.net"
    }
  ],
  "restartPolicy": "Always"
}

```

Esimerkkikoodi 8. Podin määrittelytiedosto logcatalog-data-tool.json

Jokainen aikaisemmin luotu configmap määriteltiin podiin sekä podin ainoaan konttiin. Podiin määritelty volume hosts-config-volume sisältää testidataa pakatussa hosts.tar.gz-tiedostossa. Podin alustuksessa paketointi halutaan avata toiseen hakemistoon. Kontin määrittelyssä oleva command-kentän komento ajetaan parametreilla, jotka on määritelty args-kenttään. Komennossa ajetaan skripti, jonka tar-komento purkaa hosts.tar.gz-tiedoston hakemistoon /data-tool. Lopuksi ajetaan volumessa script-config-volume sijaitseva skripti, jonka seurauksena testidata tallennetaan tietokantaan.

5.6 Muut ohjelmiston komponentit

Ohjelmistoon kuului vielä muita komponentteja, jotka lisättiin samalla tavalla kuin aikaisemmat podit. Configmappeja luotiin vain podeille logcatalog-data-tool ja agent-docker. Lopulliseen ympäristöön kuuluvat podit ovat:

- mariadb-api2
- mariadb-api1
- api2-docker
- api1-docker
- s3-ninja
- logcatalog-data-tool
- agent-docker.

Podeille luotiin myös palvelut, mikäli niihin tarvitsee saada yhteys podin ulkopuolelta kuten toisesta podista. Ympäristöön tehtiin seuraavat palvelut:

- mariadb-api2-nodeport
- mariadb-api1-nodeport
- api2-docker-nodeport
- api1-docker-nodeport
- s3-ninja-nodeport.

6 Yhteenveto

Työn lopullisena tavoitteena oli luoda yrityksen ohjelmistosta Kubernetes-pohjainen kehitysympäristö. Ympäristö oli tarkoitus luoda Minikubea käyttämällä, jonka avulla voidaan ajaa Kubernetes-klusteri yhdellä laitteella.

Työssä siirrettiin osa yrityksen ohjelmistosta Kuberneteseseen onnistuneesti. Yritys sai samalla paljon tietoa Kubernetesesta, kun aikaisempaa kokemusta kyseiseltä alustalta ei ollut kovinkaan paljon. Työssä kehitettyä ympäristöä on tarkoitus käyttää yrityksen tuotteen kehitykseen ja testaamiseen.

Työssä syntynyt ympäristö ei kuitenkaan ole lopullinen ja siinä on varaa jatkokehitykselle. Esimerkkinä ovat alustuskontit, jotka tarkastavat, että järjestelmät ja palvelut, joista pod on riippuvainen, ovat toiminnassa. Tässä työssä alustuskontti luotiin vain yhdelle podille ikään kuin konseptitodistuksena. Tarkoituksena on tulevaisuudessa kehittää alustuskonttien käyttöä sekä määritellä sellaiset kaikille podeille tarpeen mukaan.

Lähteet

- 1 What is virtualization? Verkkoaineisto. <<https://opensource.com/resources/virtualization#:~:text=Virtualization%20is%20the%20process%20of,on%20a%20computer%20system%20simultaneously.>> Luettu 30.10.2020.
- 2 What is VDI (virtual desktop infrastructure)? Verkkoaineisto. <<https://www.citrix.com/glossary/vdi.html>> Luettu 30.10.2020.
- 3 Graziano, Charles David. 2011. A performance analysis of Xen and KVM hypervisors for hosting the Xen Worlds Project. Verkkoaineisto. <<https://lib.dr.iastate.edu/cgi/viewcontent.cgi?referer=https://en.wikipedia.org/&httpsredir=1&article=3243&context=etd>> Luettu 2.11.2020.
- 4 Developer Survey Results. 2019. Verkkoaineisto. <https://insights.stackoverflow.com/survey/2019#technology-_platforms> Luettu 2.11.2020.
- 5 What is Kubernetes? <<https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>> Päivitetty 22.10.2020. Luettu 3.11.2020.
- 6 Developer Survey Results. 2019. Verkkoaineisto. <https://insights.stackoverflow.com/survey/2019#technology-_most-loved-dreaded-and-wanted-platforms> Luettu 4.11.2020.
- 7 Kubernetes Components. Verkkoaineisto. <<https://kubernetes.io/docs/concepts/overview/components/>> Päivitetty 28.8.2020. Luettu 4.11.2020.
- 8 Kubernetes User Case Studies. Verkkoaineisto. <<https://kubernetes.io/case-studies/>> Luettu 5.11.2020.
- 9 CASE STUDY: Spotify. Verkkoaineisto. <<https://kubernetes.io/case-studies/spotify/>> Luettu 5.11.2020.
- 10 Kubernetes. Verkkoaineisto. <<https://www.docker.com/products/kubernetes>> Luettu 6.11.2020.
- 11 Drivers. Verkkoaineisto. <<https://minikube.sigs.k8s.io/docs/drivers/>> Päivitetty 27.10.2020. Luettu 4.11.2020.
- 12 Service Name and Transport Protocol Port Number Registry. Verkkoaineisto. <<https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?search=3306>> Päivitetty 19.10.2020. Luettu 6.11.2020.