jamk

Kubernetes-klusterin tilan dokumentointi ohjelmallisesti

Mika Tonteri

Opinnäytetyö, AMK
Huhtikuu 2022
Liiketalouden ala
Tradenomi (AMK), tietojenkäsittelyn koulutusohjelma

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Tonteri, Mika

Kubernetes-klusterin tilan dokumentointi ohjelmallisesti

Jyväskylä: Jyväskylän ammattikorkeakoulu. Huhtikuu 2022, 41 sivua.

Liiketalouden ala. Tradenomi (AMK), tietojenkäsittelyn koulutusohjelma. Opinnäytetyö (AMK).

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

Web-sovellusten ja palveluiden paketointi ja käyttöönotto mullistui yhdessä yössä vuonna 2013, kun Docker julkaisi ensimmäisen helppokäyttöisen ja kokonaisvaltaisen konttialustaratkaisunsa. Tänä päivänä konttiteknologian osaajille on suuri tarve lähes kaikissa ohjelmistoalan yrityksissä.

Työssä tutustuttiin konttiteknologiaan ja konttien hallinnan automatisointiin eli konttiorkestraatioon keskittyen Googlen kehittämään, markkinoiden suosituimpaan orkestraattoriin, Kubernetesiin sekä Red Hatin kehittämään, Kubernetesin päälle rakennettuun OpenShift-alustaan.

Tutkimusta tehtiin tukemaan työn varsinaista toimeksiantoa. Työn toimeksiantaja, Kelan IT-palvelut, tarvitsi OpenShift-alustan ylläpitäjien ja asiakkaiden käyttöön raportointinäkymän, joka korvaisi virhealttiin manuaalisen dokumentaatioprosessin helppokäyttöisellä ja automaattisella ratkaisulla. Työn tavoitteena oli tämän raportointinäkymän datalähteenä toimivan ohjelmointirajapinnan kehittäminen.

Työn tuloksena oli toimeksiantajan vaatimusmääritysten mukainen ohjelmointirajapinta, jonka avulla voidaan hakea tietoa Kubernetes-klusterin resurssien tilasta. Tätä tietoa käytetään tarkoitusta varten kehitetyssä raportointinäkymässä. Rajapinta mahdollistaa toimeksiantajalle manuaalisesta dokumentaatioprosessista luopumisen säästäen OpenShift-alustan ylläpitäjien ja asiakkaiden työaikaa.

Avainsanat (asiasanat)

Kubernetes, OpenShift, Konttiteknologia, Konttiorkestraatio

Muut tiedot (salassa pidettävät liitteet)

Ei salassa pidettäviä liitteitä.

jamk

Description

Tonteri, Mika

Documenting Kubernetes clusters programmatically

Jyväskylä: JAMK University of Applied Sciences, April 2022, 41 pages.

Bachelor's degree programme in Business Information Technology

Permission for web publication: Yes

Language of publication: Finnish

Abstract

Packaging and deploying web applications and services was revolutionized overnight when Docker released their easy to use and feature packed container platform in 2013. Today, almost all companies deploying software are looking for professionals with skills in containers and related technologies.

The thesis dives deep into container technology and container orchestration while focusing on the most widely adopted orchestrator, Google's Kubernetes, and the SaaS-offering from Red Hat, which was built on top of it, OpenShift.

The study was conducted to aid the actual work requested by the client. The client, Kela IT-services, wanted a dashboard for their OpenShift platform administrators and clients that would replace manually updated Confluence pages with an easy to use and automated solution.

The result of the thesis was a REST API that was developed according to requirements of the client and could be used to query the Kubernetes API for data about the state of the cluster's resources. This data is used in a bespoke dashboard application, which enables the client to drop their manual Confluence documentation and saves both the administrator's and the client's valuable time.

Keywords/tags (subjects)

Kubernetes, OpenShift, Container technology, Container orchestration

Miscellaneous (Confidential information)

No confidential information.

Sisältö

Kuviot

Keskeiset käsitteet

Virtualisointi

Fyysisen resurssin kuten prosessorin, keskusmuistin tai verkkorajapinnan abstrahointi loogiseksi objektiksi. Käytetään yleisesti palvelintietokoneiden jakamiseksi useiksi virtuaalikoneiksi.

Virtuaalikone

Virtuaalinen tietokone, joka emuloi fyysisen tietokoneen toimintaa.

Hypervisor

Ohjelmisto, joka hallitsee isäntäkoneella suoritettavia virtuaalikoneita.

Kontti

Kontti on sovelluskoodia, kirjastoja ja muita riippuvuuksia sisältävä ohjelmistopaketti, joka suoritetaan eristettynä muista prosesseista.

Konttilevykuva

Tiedostopaketti, jota konttien ajonaikainen suoritusympäristö käyttää kontin käynnistämiseen

Konttiorkestraatio

Konttien käyttöönoton, ylläpidon, skaalauksen ja verkkoliikenteen hallintaan liittyvien tehtävien automatisointi.

Konttiorkestraattori

Konttien orkestrointia varten kehitetty järjestelmä kuten Docker Swarm, Kubernetes tai Apache Mesos.

Klusteri

Kokoelma tietokoneita, jotka toimivat yhdessä ikään kuin ne olisivat yksi laite.

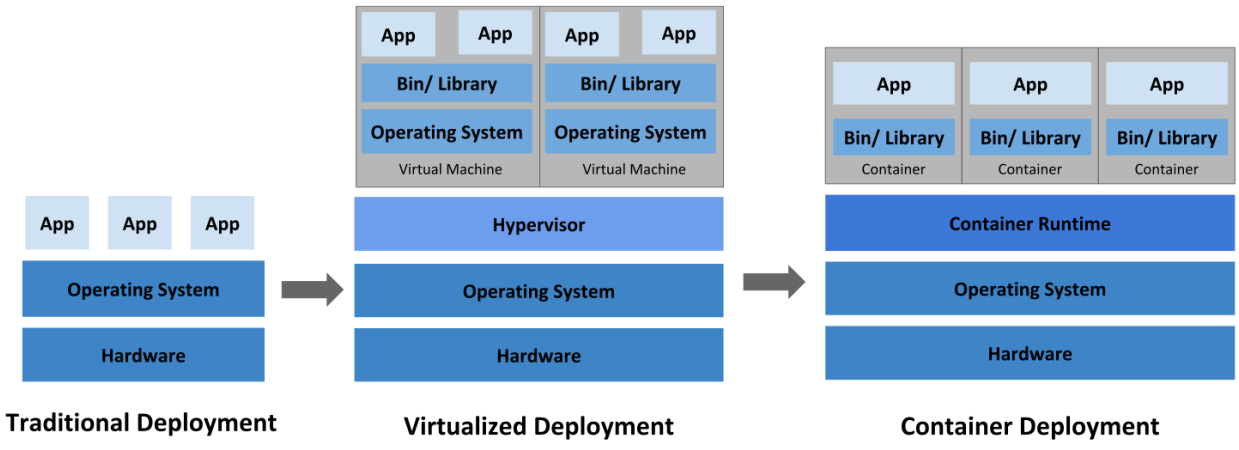
API

Lyhenne sanoista Application Programming Interface. Tarkoittaa ohjelmointirajapintaa, joka tarjoaa tietokoneille ja tietokoneohjelmille tavan keskustella ja vaihtaa dataa keskenään.

1 Johdanto

Web-sovellusten ja palveluiden paketoinnissa ja käyttöönotossa on ollut kolme selkeää aikakautta (Ks. Kuvio 1). Aluksi sovellus asennettiin suoraan fyysiselle palvelimelle, jolloin palvelimien tehon kasvaessa niiden resurssien käyttöaste jäi matalaksi. Tähän keksittiin lääkkeeksi virtualisointi, jonka avulla fyysisen palvelimen resurssit saatiin jaettua usean virtuaalisen palvelimen käyttöön. Virtualisointia hyödynnetään vielä tänäkin päivänä fyysisten palvelinten resurssien tehokkaan jakamisen apuna, mutta sovellukset pakataan entistä useammin kontteihin, jotka eivät tarvitse kokonaista käyttöjärjestelmää toimiakseen.

Konttiteknologian myötä suosituksi sovellusarkkitehtuurimalliksi noussut mikropalveluarkkitehtuuri nosti esiin toisen ongelman: Suurta määrää kontteja on hankala hallita. Mahdollisesti tuhansien konttien orkesterin hallintaan tarvittiin kapellimestari. Ohjelmistoyhtiö Google oli käyttänyt konttiteknologiaa sisäisesti jo kauan ennen kuin siitä tuli alan standardi. Se kehitti omien sisäisten ratkaisujensa pohjalta Kubernetes -konttiorkestraattorin, jonka se lahjoitti Cloud Native Computing Foundation -järjestölle vuonna 2014 avoimen lähdekoodin projektina (Kubernetes, N.d.a). Kubernetesista on sittemmin tullut maailman suosituin konttiorkestraattori (IBM, 2021).



Kuvio 1. Sovellusten käyttöönoton aikakaudet (Kubernetes, N.d.a)

Tämän opinnäytetyön toimeksiantaja Kelan IT-palvelut käyttää kontitettujen sovellusten käyttöönottoon ja ylläpitoon ohjelmistoyhtiö Red Hatin kehittämää OpenShift-alustaa, joka on

rakennettu Kubernetesin päälle laajentaen sen toiminnallisuutta. Alustan resurssit dokumentoidaan tällä hetkellä manuaalisesti Confluence-sivuille. Manuaalinen päivitysprosessi on altis virheille eikä dokumentaatio välttämättä vastaa todellisuutta. Työn tavoitteena on kehittää ohjelmointirajapinta, jonka avulla OpenShift-alustan resurssien tila voitaisiin dokumentoida automaattisesti, säästäen alustan ylläpitäjien ja asiakkaiden työaikaa.

2 Tutkimusasetelma

Tässä luvussa kuvataan tutkimuksen toimeksiantaja, tutkimusongelma ja tutkimuskysymykset sekä valitut tutkimustyön menetelmät.

2.1 Toimeksiantaja

Työn toimeksiantaja on Kansaneläkelaitoksen eli Kelan IT-palvelut. Kela on vastuussa Suomen sosiaaliturvasta ja IT-palveluiden vastuulla on kehittää ja ylläpitää tietojärjestelmiä, jotka mahdollistavat tämän tehtävän suorittamisen. Kelan IT-palveluissa työskentelee yli 800 IT-asiantuntijaa eri puolilla Suomea.

Kelalla on käytössä OpenShift-konttialusta, jonka päällä erilaisia tietojärjestelmäprojekteja ajetaan kontitettuna. Alustan resurssien dokumentaatio on tällä hetkellä levittäytynyt useille Atlassian Confluence-järjestelmän sivuille, joiden päivittäminen tapahtuu manuaalisesti. Manuaalisessa päivitysprosessissa on kuitenkin virhemahdollisuus, jolloin dokumentaatio ei välttämättä vastaa todellisuutta. Toimeksiantaja haluaa päästä eroon manuaalisesta päivitysprosessista ja keskittää OpenShift-alustan dokumentaation yhteen paikkaan, jolloin se on helposti alustan ylläpitäjien ja asiakkaiden saatavilla.

2.2 Kehitystyön tavoitteet

Opinnäytetyön tavoitteena on kehittää ohjelmointirajapinta, jonka avulla alustan resursseja voidaan dokumentoida automaattisesti. Rajapinnan tarjoamaa dataa tullaan ensisijaisesti näyttämään tätä tarkoitusta varten kehitetyssä web-pohjaisessa raportointinäkymässä, mutta rajapinta voisi tarjota tätä dataa myös muiden asiakasohjelmien käyttöön. Rajapinnan tulee toimia kontitettuna toimeksiantajan OpenShift-alustalla.

2.3 Tutkimuskysymykset ja rajaukset

Tämä työ on rajattu koskemaan konttiteknologiaa ja konttiorkestraatiota, koska niistä on tullut nopeasti standarditeknologioita sovellusten käyttöönotossa ja näiden teknologioiden osaajille on kasvava kysyntä työelämässä. Koska toimeksiantajalla on käytössään OpenShift-alusta joka on rakennettu Kubernetesin päälle, on työstä rajattu pois myös muut konttiorkestraattorit. Kehittämistyössä käytettävät web-kehityksen työkalut on nekin jätetty työstä pois, koska niistä on tehty vuosien saatossa runsaasti opinnäytetöitä, joten niitä tutkimalla ei syntyisi uutta tietoa.

Tutkimuskysymykset ovat siksi seuraavat:

1. Mikä on Kubernetes?
2. Miten Kubernetesin APIsta haetaan tietoa klusterin tilasta?
3. Miten klusterissa suoritettava kontitettu sovellus voi hyödyntää Kubernetesin APIa?

Tutkimuskysymykset on johdettu opinnäytetyön tavoitteesta. Kysymyksiin vastaaminen mahdollistaa toimeksiantajan vaatimusmäärittelyjen täyttämisen. Ensimmäiseen kysymykseen vastataan teoriaosan luvussa 4 Kubernetes. Toiseen ja kolmanteen kysymykseen vastataan työn empiirisessä osassa luvuissa 5.2 Rajapinnan toteutus ja 5.3 Rajapinnan käyttöönotto OpenShiftissä. Toinen ja kolmas kysymys ovat kehitystyön tavoitteiden saavuttamisen kannalta kriittisiä.

2.4 Tutkimusmenetelmät

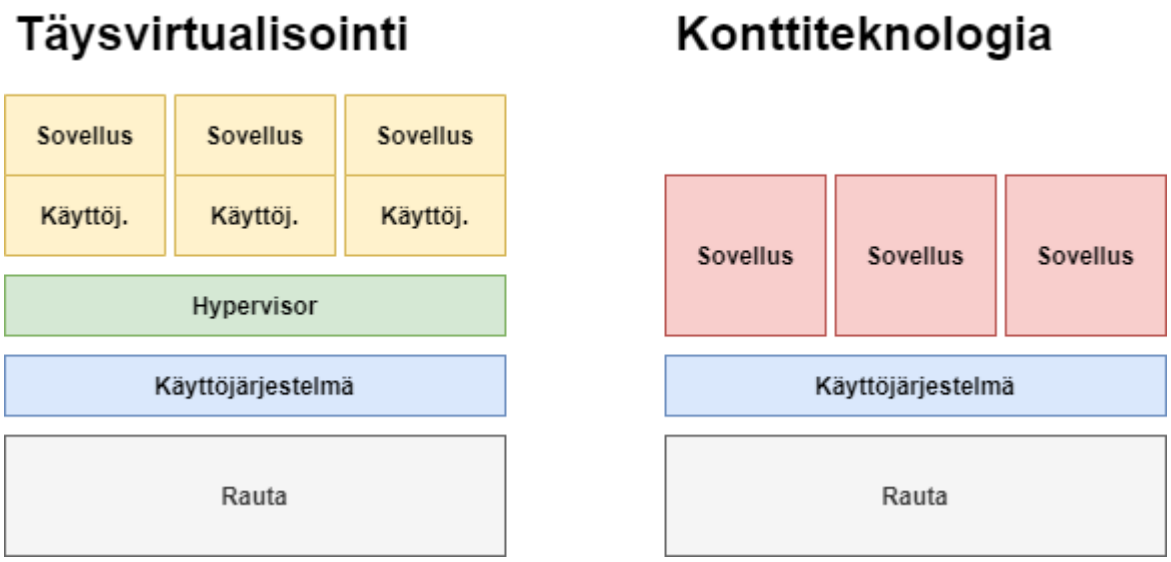
Kanasen (2015, 11) mukaan kehittämistutkimus pyrkii saamaan aikaan muutoksen joko poistamalla ongelman kokonaan tai parantamalla asioiden tilaa. Ojasalo, Moilanen ja Ritalahti (2014, 65) kirjoittavat, että mikäli tavoitteena on luoda jokin konkreettinen tuotos, on konstruktiivinen tutkimus sopiva lähestymistapa. Toisaalta taas Bister (2019, 48) kirjoittaa, että konstruktiivinen tutkimus pyrkii laajaan yleistettävyyteen, kun kehittämistutkimuksessa keskitytään yhden ongelman ratkaisuun. Koska opinnäytetyöni pyrkii ratkaisemaan toimeksiantajan dokumentaatioprosessin ongelman eikä ratkaisu ole yleistettävissä, on opinnäytetyöni kehittämistutkimusta.

Kananen (2015, 40) kuvailee, että kehittämistutkimus alkaa siitä, mihin perinteinen päättyy, koska perinteisessä tutkimuksessa ongelman poistaminen jää pois. Kehittämistutkimus perustuu kahteen

sykliin: tutkimussykliin ja muutossykliin. Tutkimussyklissä tutkimusongelmille etsitään vastaukset ja muutossyklissä muutokset toteutetaan ja arvioidaan. (Mts, 40)

3 Konttiteknologia

Konttiteknologialla tarkoitetaan virtualisointiratkaisua, jossa sovelluksen koodi ja sen riippuvuudet paketoidaan kevyeksi virtuaaliympäristöksi, kontiksi (Kuraman S, 2017). Konttipohjainen virtualisointi eroaa perinteisestä hypervisor-pohjaisesta virtualisoinnista siten, että kontit eivät sisällä kokonaista käyttöjärjestelmää, vaan ne jakavat käyttöjärjestelmäytimen isäntäkoneen kanssa (VMware, N.d). Näiden teknologioiden eroja on havainnoillistettu kuviossa 2.



Kuvio 2. Täysvirtualisointi verrattuna konttiteknologiaan

Linux-konttiteknologia hyödyntää kahta Linux-ytimen ominaisuutta: Nimiavaruuksia ja hallintaryhmiä. Nimiavaruuksien avulla prosesseja voidaan eristää niin, että ne eivät voi nähdä muita isäntäkoneella suoritettavia prosesseja. Hallintaryhmien avulla voidaan säädellä ja pitää kirjaa prosessien resurssikäytöstä, jolloin yksittäinen prosessi ei voi käyttää niin paljon resursseja, että niitä ei riitä toisille prosesseille. Lisäksi kontit hyödyntävät kerroksittaisia tiedostojärjestelmiä, jotka mahdollistavat tiedostojen jakamisen konttien välillä. Koska nämä tiedostojärjestelmät ladataan muistiin, ne mahdollistavat konttien nopean käynnistymisen. (Jain, 2020)

3.1 Konttiteknologian hyödyt ja huonot puolet

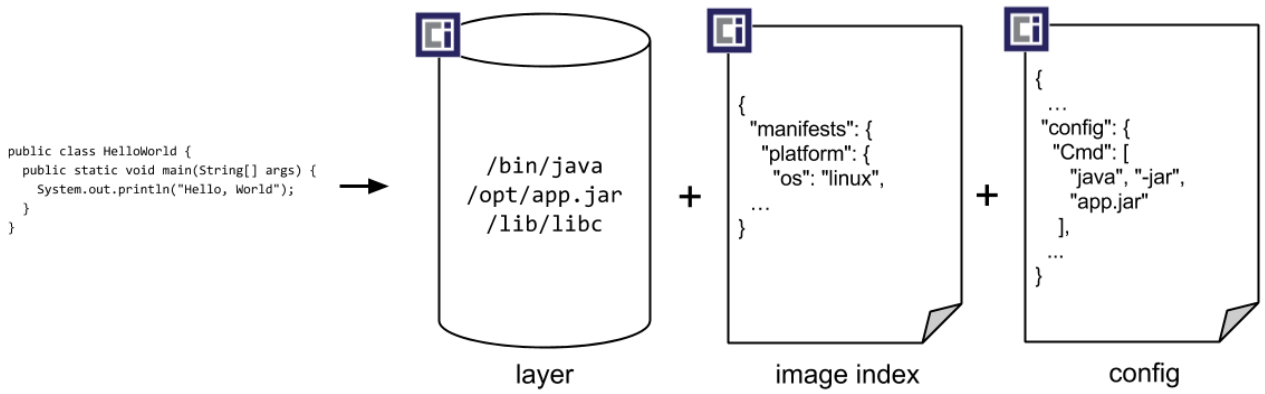
Konttiteknologialla on useita hyötyjä perinteiseen virtualisointiteknologiaan verrattuna. Kuraman S:n (2017) mukaan kontit ovat huomattavasti kevyempiä ja ne vievät vähemmän tallennustilaa kuin perinteiset virtuaalikoneet, koska ne eivät tarvitse kokonaista käyttöjärjestelmää tai emulaatiotasoja toimiakseen. Samasta syystä kontit myös käynnistyvät lähes välittömästi, kun taas virtuaalikoneen käynnistyminen voi kestää pitkään. Gandhi ja Szmrecsanyi (2019) listaavat hyödyiksi edellä mainittujen lisäksi myös kehitystyön ketteryyden, muutosten nopeuden sekä sovellusten helpon liikuteltavuuden eri alustojen ja pilvipalveluiden välillä.

Konttiteknologialla on myös huonoja puolia. PortWorxin (2021) tekemän konttiteknologian ja Kubernetesin käyttöönottoa selvittävän kyselytutkimuksen mukaan suurilla yrityksillä on vaikeuksia löytää osaavaa työvoimaa, etenkin tietoturva- ja verkkoliikenneosaajia, joka on hidastanut konttiteknologian käyttöönottoa.

Verkkoliikenne ja tietoturva ovat myös Guytonin (2019) listalla huolena, koska niiden hallinta on hankalaa konttien kanssa työskennellessä. Duncan ja Osborne (2018) nostavat esiin myös tarpeen erillisille tallennustilaratkaisuille, koska konttiteknologia on alun perin suunniteltu tilattomaksi, jonka vuoksi konttien sisäinen tallennustila ei ole pysyvää.

3.2 Open Container Initiative

Open Container Initiative eli OCI on nykyään Linux Foundation-järjestön hallinnoima projekti, jonka Docker ja CoreOS -projektit perustivat yhdessä muiden alan johtavien toimijoiden kanssa vuonna 2015. Sen tarkoituksena on ylläpitää standardimääritelmää konttilevykuvien formaateille ja ajonaikaisille suoritusympäristöille. OCI Image Format-määrittely perustuu Dockerin lahjoittamaan levykuvaformaattiin. Se määrittää OCI-yhteensopivien konttien levykuvien standardiformaatin. Levykuva itsessään koostuu indeksitiedostosta, konfiguraatiotiedostosta ja tiedostojärjestelmäkerroksista (Open Container Initiative, N.d.a). Levykuvan koostumusta havainnollistetaan kuviossa 3.



Kuvio 3. OCI Image Formatin yleiskuvaus (Open Container Initiative, N.d.b)

Indeksitiedostossa listataan levykuvien manifesteja. Manifesti kuvaa yksittäistä konttilevykuvaa ja se sisältää käytetyn levykuvaformaatin version, mediatyypin, levykuvan konfiguraation sekä levykuvan tasot (Open Container Initiative, N.d.c). Konfiguraatiotiedostossa määritellään parametrit kontin sisällä suoritettavalle sovellukselle. Tasot ovat tiedostokokoelmia, jotka kuvastavat tiedostojärjestelmää ja sen muutoksia (Open Container Initiative, N.d.d). Caban (2019) toteaa, että OCI-formaatti tarkoittaa kaikessa yksinkertaisuudessaan sitä, että kontti koostuu sovelluskoodista ja sen riippuvuuksista.

OCI Container Runtime-määrittely määrittää, kuinka OCI-konttilevykuva suoritetaan. OCI-konttilevykuva puretaan ajonaikaiseksi tiedostopaketiksi, jonka OCI-yhteensopiva ajonaikainen suoritusympäristö suorittaa (Open Container Initiative, N.d.a). OCI Container Runtime määrittää myös standardioperaatiot, joita konteille voidaan suorittaa. Näitä ovat mm. konttien luominen, käynnistäminen, pysäyttäminen ja tuhoaminen. (Open Container Initiative, N.d.d)

3.3 OCI-yhteensopivan konttilevykuvan luominen

Kuten yllä kerrottiin, OCI Image Format-levykuvamäärittely perustuu Dockerin levykuvaformaattiin. Tämän ansiosta OCI-yhteensopivia levykuvia voidaan määritellä Dockerfile-tekstitiedostossa ja rakentaa kontiksi myös muilla OCI-yhteensopivilla työkaluilla. Levykuvien luomiseen on myös muita tapoja riippuen valitusta työkalusta, mutta koska Dockerfilestä voidaan rakentaa levykuva miltei kaikilla OCI Container Runtime määrittelyn mukaisilla työkaluilla, käydään tässä työssä läpi vain Dockerfile.

Dockerfile on tekstitiedosto, joka sisältää komennot, joiden avulla konttilevykuva rakennetaan. Komennot esitellään Dockerfilessä siinä järjestyksessä, jossa niiden halutaan tapahtuvan. Dockerin (N.d) dokumentaation mukaan Dockerfilen formaatti vastaa Kuviossa 4 esiteltyä rakennetta.

```
1  # Tämä on kommentti. Docker ei huomio kommentteja.
2  FROM kuva
3
4  RUN komento
5
6  CMD [“komento”, “argumentti1”, “argumentti2”]
```

Kuvio 4. Dockerfilen perusrakenne

Konttien ajonaikainen suoritusympäristö ei huomioi risuaidalla alkavaa kommenttiriviä. Kommenttien tarkoituksena on toimia dokumentointityökaluna. Dockerfilen tulee aina alkaa FROM-käskyllä, jolla määritellään konttilevykuvan pohjana käytettävä levykuva. RUN-käskyllä suoritetaan jokin komento, esimerkiksi sovelluksen lähdekoodin kääntävä skripti. CMD-käskyllä määritellään komento, joka suoritetaan heti kun kontti käynnistetään. Muita Dockerfilen tukemia käskyjä ovat esimerkiksi COPY ja ADD joiden avulla tiedostoja voidaan lisätä konttilevykuvaan. (Docker, N.d)

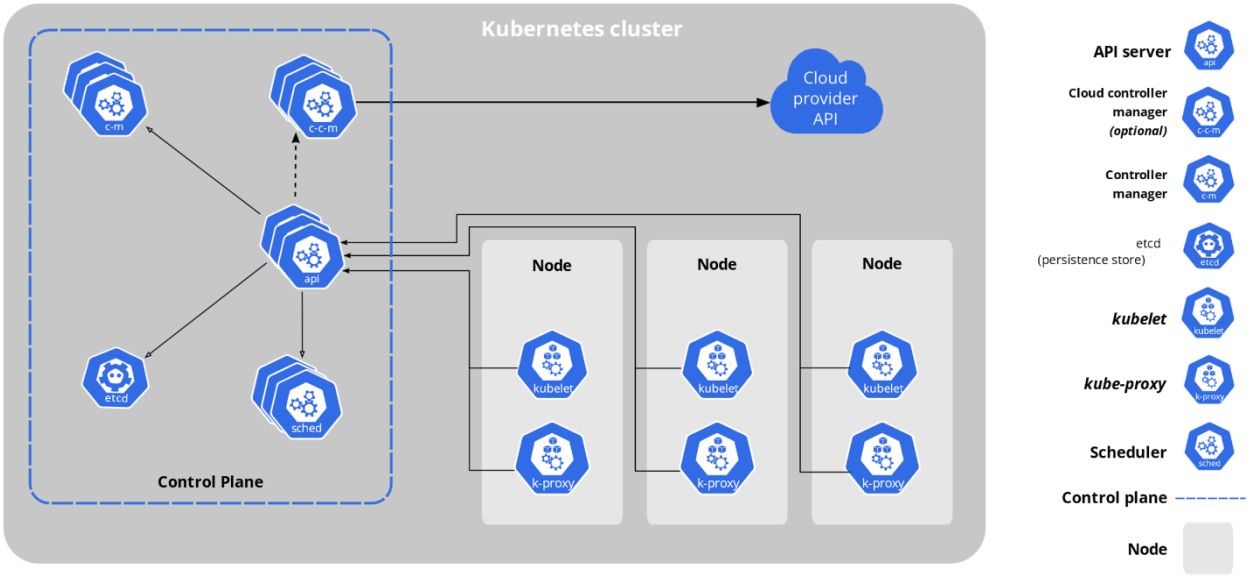
4 Konttiorkestraatio ja Kubernetes

Konttiorkestraatiolla tarkoitetaan konttien käyttöönoton, ylläpidon, skaalauksen ja verkkoliikenteen hallinnan automatisointia (Red Hat, 2019). Tähän tarkoitukseen on kehitetty niin sanottuja orkestraattoreita, jotka toteuttavat konttiorkestraation konseptin eri tavoilla. Tässä työssä keskitytään niistä suosituimpaan eli Kubernetesiin ja sitä laajentavaan OpenShift-alustaan.

Kubernetes on Googlen kehittämä avoimen lähdekoodin konttiorkestraattori, joka mahdollistaa monien kontitettujen sovellusten ja palveluiden hallintaan liittyvien tehtävien automatisoinnin. Kubernetesin ominaisuuksiin kuuluu helppo palveluiden löydettävyys, verkkoliikenteen kuormanjako, erityyppisten tallennustilaratkaisujen orkestrointi, automatisoitu sovellusten käyttöönotto- ja päivitysprosessi, automaattinen virheistä palautuminen ja salaisen tiedon kuten salausavainten ja käyttäjätunnusten sekä muun konfiguraation hallinta. (Kubernetes, N.d.a)

4.1 Kubernetesin toimintamalli

Kubernetes toimii klusterina, eli se koostuu kokoelmasta virtuaalisia tai fyysisiä laitteita, joita Kubernetesissa kutsutaan Nodeiksi. Jokaisessa Kubernetes-klusterissa on vähintään yksi hallintakerrospalvelin, joka suorittaa Control Planen eli hallintakerroksen muodostavia järjestelmäpalveluita. Näitä palvelimia kutsutaan nimellä Control Plane Node, Master tai Head. Lisäksi klusterissa on aina vähintään yksi Worker Node, joka vastaa kontitettujen sovellusten hyötykuorman suorittamisesta. Worker Nodeja kutsutaan yleisesti vain Nodeiksi (Poulton, 2021). Kubernetes -klusterin rakenne on esitelty kuviossa 5.



Kuvio 5. Kubernetes-klusterin komponentit (Kubernetes, N.d.b)

Kubernetes noudattaa niin sanottua deklaratiivista toimintamallia. Deklaratiivinen toimintamalli tarkoittaa sitä, että Kubernetesille esitetään asioiden haluttu tila ja jätetään asioiden muuttaminen sen huoleksi. Tämän tyylin etuna verrattuna perinteiseen imperatiiviseen malliin, jossa käytetään alustariippuvaisia komentoja ja skriptejä, on mallin yksinkertaisuus ja itsedokumentoiva tyyli. (Poulton, 2021)

4.2 Kubernetesin komponentit

Kuten kuviossa 4 havainnollistettiin, Kubernetes-klusteri koostuu kahdesta pääkomponentista: hallintakerroksesta ja Nodeista. Control Plane eli hallintakerros koostuu API-palvelimesta, tehtäväajastimesta, klusterin tietovarastosta sekä Controller managerista, joka hallitsee ja monitoroi klusterin kontrollereita. (Poulton, 2021)

API-palvelin on hallintakerroksen sydän, jonka läpi kaiken kommunikaation kaikkien komponenttien välillä tulee kulkea. Palvelin tarjoaa REST-arkkitehtuurityylin mukaisen rajapinnan, johon lähetetään HTTP POST-pyynnöllä manifesteja eli YAML-kuvauskielellä kirjoitettuja tekstitiedostoja, jotka kuvaavat klusterin haluttua tilaa. Manifesti validoidaan syntaksivirheiden varalta ja tallennetaan klusterin pysyvään tietovarastoon. Klusterin tietovarasto on sen ainoa totuuden lähde, jonne tallennetaan kaikki klusterin konfiguraatio ja tila. Se perustuu suosittuun hajautettuun etcd-tietokantaan. (Poulton, 2021)

Tehtäväajastin tarkkailee API-palvelinta uusien tehtävien varalta ja delegoi niitä Nodeille. Ajastin luokittelee Nodeja perustuen niiden kykyyn suorittaa tehtäviä. Mikäli tehtäväajastin ei löydä tehtävälle sopivaa Nodea, sitä ei suoriteta ja sen tila muutetaan odottavaksi. (Poulton, 2021)

Controller Manager vastaa kontrollereiden hallinnasta. Sen vastuulla on kontrollerien monitorointi ja käynnistäminen tarpeen mukaan. Kontrollerien tehtävänä on saattaa klusterin tila vastaamaan haluttua tilaa (Poulton, 2021). Kontrollerit seuraavat jonkin tietyn objektin sen hetkistä tilaa ja API-palvelimelle tulevia tilan muutospyyntöjä pitääkseen objektin tilan oikeana (Kubernetes, N.d.e).

Nodet koostuvat kolmesta komponentista: kubelet-agentista, konttien ajonaikaisesta suoritusympäristöstä sekä kube-proxy-palvelusta. Kubelet-agentti kuuntelee hallintakerroksen API-palvelinta uusien tehtävien varalta. Mikäli Node ei pysty suorittamaan tehtävää, se raportoi siitä hallintakerroksellelle, joka on vastuussa sopivan Noden löytämisestä. Kubelet-agentti käyttää konttien ajonaikaista suoritusympäristöä konttien hallintaan liittyviin tehtäviin, kuten konttien luomiseen, käynnistämiseen, pysäyttämiseen ja tuhoamiseen. Kube-proxy vastaa klusterin sisäisestä paikallisesta verkkoliikenteestä. Sen tehtävänä on varmistaa, että jokainen Node saa uniikin IP-osoitteen sekä hoitaa reititystä ja kuormanjakoa sovellusten välillä. (Poulton, 2021)

4.3 Kubernetesin objektit

Kubernetes käyttää erilaisia objekteja klusterin tilan kuvaamiseen. Objektit voivat kuvata mitä sovelluksia klusterissa on käynnissä, mitä resursseja niiden käytettävissä on sekä sovellusten käyttäytymiseen liittyviä toimintatapoja. Kun Kubernetes-objekti luodaan, kerrotaan Kubernetesille jotain klusterin halutusta tilasta. Kuten aiemmin kerrottiin, kaikki kommunikaatio Kubernetesissa kulkee API-palvelimen kautta, joten myös objektit luodaan lähettämällä YAML-kielellä kirjoitettu manifesti HTTP POST-pyyntönä API-palvelimelle. (Kubernetes, N.d.c)

Kubernetesin (N.d.c) dokumentaation mukaan Kubernetes-objektin kuvaava manifesti vaatii vähintään seuraavat tietueet (Ks. Kuvio 5):

apiVersion: Käytettävä Kubernetesin APIn versionumero.

kind: Minkä tyyppinen objekti on kyseessä.

metadata: Objektin yksilöivää tietoa kuten nimi, UID ja kohdenimiavaruus.

spec: Objektin haluttu tila.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Kuvio 6. Esimerkki objektin manifestitiedostosta (Kubernetes, N.d.c)

Seuraavassa käyn läpi joitakin Kubernetesin objekteja ja niiden käyttötarkoituksia tarkemmin. Kyseessä ei ole kaiken kattava lista, vaan keskityn tämän työn kannalta tärkeimpiin objekteihin.

Pod

Pod on Kubernetesin pienin yksikkö, joka koostuu yhdestä tai useammasta kontista. Podin sisällä olevat kontit jakavat suoritusympäristön, jolle on varattu tietty osa klusterin resursseista kuten taltioista, CPU-ajasta ja keskusmuistista. Podin sisällä olevilla konteilla on yhteinen ulkoinen IP-osoite. Mikäli konttien tarvitsee keskustella keskenään, se tapahtuu localhost-rajapinnan kautta. Podien elinkaari koostuu niiden luomisesta, käyttöönotosta ja tuhoamisesta, eli mikäli Pod kaatuu odottamattomasti, sitä ei yritetä käynnistää uudelleen, vaan se poistetaan ja tilalle luodaan uusi. (Poulton, 2021)

Deployment

Deployment on objekti, joka ikään kuin kääriytyy Podin ympärille. Se tarjoaa Podeille lisäominaisuuksia kuten automaattisen ongelmista palautumisen ja skaalautumisen. Yksittäinen

Deployment-objekti hallinnoi yhtä mallia, jonka pohjalta Pod luodaan. Deployment voi kuitenkin hallinnoida useita samasta mallista luotuja kopioita samanaikaisesti. Deployment mahdollistaa automaattisen ongelmista palautumisen siten, että Deployment-objekteja hallinnoiva DeploymentController-kontrolleri ottaa käyttöön uuden Podin vanhan kaatuessa. Samainen kontrolleri seuraa Podeille tulevaa liikennettä ja ottaa kopioita käyttöön tai poistaa niitä kuorman muuttuessa. (Poulton, 2021)

Service

Service on objekti, jonka avulla jokin sovellus, jota suoritetaan kokoelmassa Podeja, voidaan paljastaa klusterin ulkopuolelle (Kubernetes, N.d.d). Service-objekteilla on vakaa DNS-nimi, IP-osoite ja portti, jolloin niihin voidaan ottaa yhteys internetin kautta. Service-objektit myös jakavat verkkoliikenteen kuormaa tämän Pod-kokoelman sisällä. (Poulton, 2021)

ConfigMap ja Secret

ConfigMap ja Secret ovat objekteja, jotka mahdollistavat konfiguraatiodatan tallentamisen Podien ulkopuolella, sekä datan injektoinnin dynaamisesti Podin sisälle. ConfigMap ja Secret eroavat siten, että ConfigMapiin on tarkoitus tallentaa dataa, joka ei ole tietoturvan kannalta kriittistä, kuten ympäristömuuttujia ja kokonaisia konfiguraatiotiedostoja. Secret-objektiin on nimensä mukaisesti tarkoitus tallentaa salaista dataa kuten salausavaimia ja sertifikaatteja. (Poulton, 2021)

PersistentVolume ja PersistentVolumeClaim

PersistentVolume on objekti, joka kuvaa klusterissa käytettävissä olevaa tallennustilaresurssia. PersistentVolumeClaim on objekti, joka kuvaa pyyntöä pysyvästä tallennustilasta. PersistentVolumeClaim-objekti liitetään sellaiseen PersistentVolume-objektiin, jonka kuvaaman tallennustilan ominaisuudet ja pääsyoikeudet vastaavat pyyntöä. (Caban, 2019)

4.4 Kubernetes-klusterin resurssien jakaminen usealle projektille

Yksittäinen fyysinen Kubernetes-klusteri voidaan jakaa virtuaaliklustereiksi Namespace-ominaisuuden avulla. Namespacejen eli nimiavaruuksien tarkoituksena on mahdollistaa klusterin resurssien jakaminen usean käyttäjän kesken sekä tarjota vaikutusalue resurssien nimille, joka tarkoittaa sitä, että resursseilla voi olla klusterin tasolla sama nimi, mutta ei nimiavaruuden sisällä. (Kubernetes, N.d.e)

Poultonin (2021) mukaan nimiavaruudet ovat hyvä ratkaisu jakaa yksittäinen fyysinen klusteri usean osaston kesken, jolloin jokaisella osastolla on omat käyttäjät, pääsyoikeudet ja resurssirajoitukset. Poulton (Mt) huomauttaa myös että toisin kuin luvussa 3.1 esitellyt Linuxin nimiavaruudet, Kubernetesin vastaavia ei ole suunniteltu tehokkaaseen eristämiseen.

Rooliperustainen pääsyoikeuksien hallinta

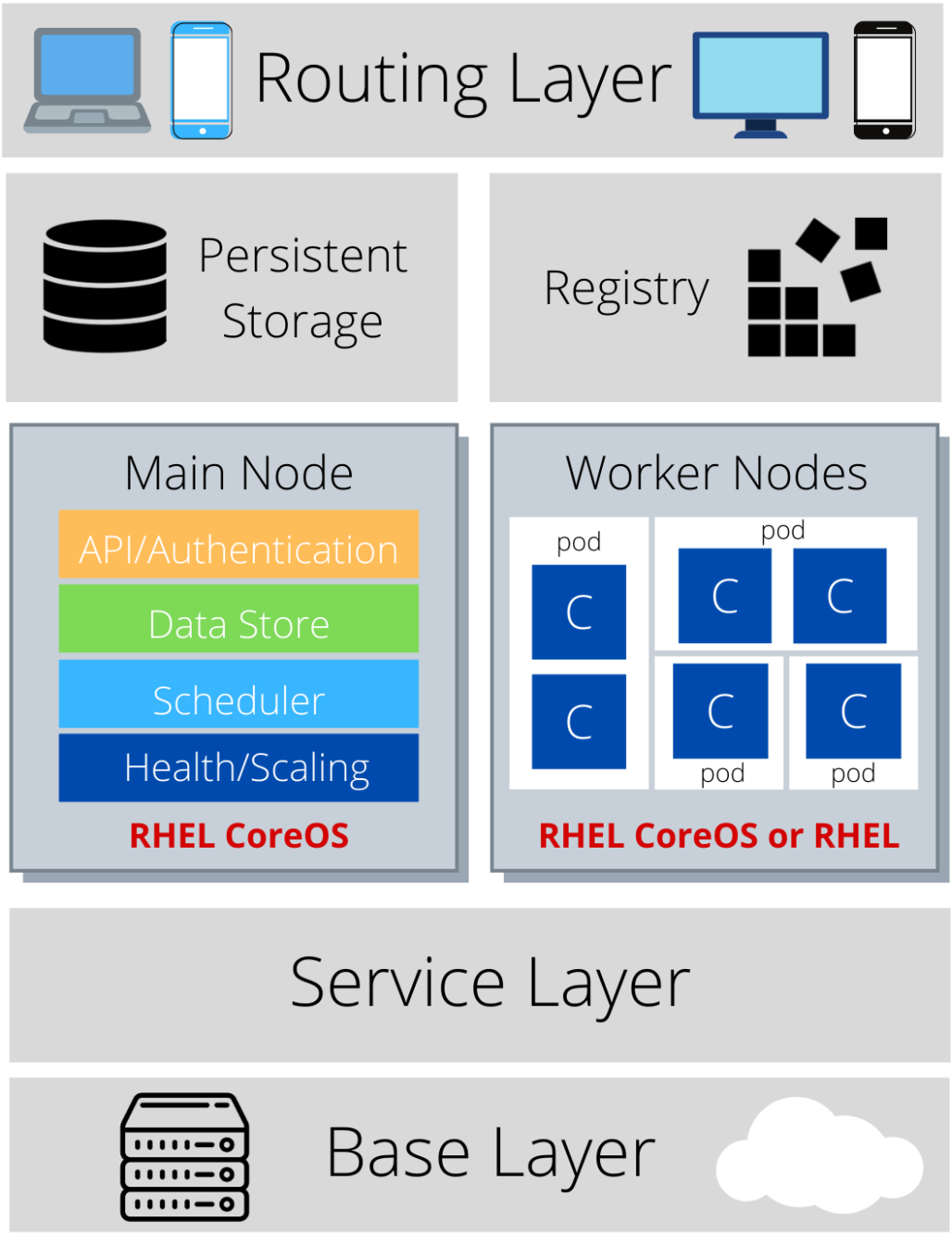
Kubernetesin APIn pääsyoikeuksia voidaan rajoittaa Role-based Access Control eli RBAC-ominaisuudella. RBAC mahdollistaa pääsyoikeuksien rajoittamisen yksittäisten käyttäjien roolien perusteella. Näitä rooleja määritellään Role-, ClusterRole-, RoleBinding-, ja ClusterRoleBinding-objekteilla. Role-objekti määrittää pääsyoikeudet tietyssä nimiavaruudessa, kun taas ClusterRole määrittelee ne koko klusterin laajuisesti. RoleBinding- ja ClusterRoleBinding-objektit liittävät nämä roolit tiettyihin käyttäjätunnuksiin tai ServiceAccount –objekteihin. (Kubernetes, N.d.f)

4.5 OpenShift

OpenShift on Red Hatin kehittämä Platform as a Service eli PaaS -tuote, joka on rakennettu Kubernetesin päälle. OpenShift tarjoaa ominaisuuksia, jotka Kubernetesista joko puuttuvat kokonaan tai ovat hankalasti käytettäviä (Zuev & Kropachev & Usov, 2018, 90). Cabanin (2019) mukaan OpenShift integroi yli 200 avoimen lähdekoodin projektia tarjotakseen tehokkaan ja helppokäyttöisen sovellusalustan.

OpenShiftin arkkitehtuuri ja komponentit

Abushamlehin (2020) mukaan OpenShiftin arkkitehtuuri koostuu seitsemästä kerroksesta. Nämä kerrokset ovat infrastruktuuri, palvelut, Main Node, Worker Nodet, pysyvä tallennustila, sisäinen levykuvarekisteri sekä reitityskerros. Arkkitehtuurimallia on havainnollistettu kuviossa 7.



Kuvio 7. OpenShiftin arkkitehtuurikerrokset (Abushamleh, 2020)

Infrastruktuurikerros käsittää klusterin fyysiset ja virtuaaliset palvelimet. Palvelukerros vastaa Podien ja pääsyoikeuksien määrittelystä. Main Node vastaa klusterin hallintakerroksesta kuten Kubernetesissakin. Worker Nodet, joita kutsutaan OpenShiftissä myös App Nodeiksi (Caban, 2019), vastaavat sovellusten hyötykuorman suorittamisesta. Pysyvän tallennustilan orkestraatiosta vastaa saman niminen kerros. Levykuvarekisteri vastaa klusterissa käytettävien konttilevykuvien tallentamisesta. Reitityskerros tarjoaa ulkoisen pääsyn klusterin sisällä suoritettaviin sovelluksiin sekä verkkoliikenteen kuormanjaon. (Abushamleh, 2020)

OpenShiftin erot Kubernetesiin verrattuna

OpenShift hyödyntää Kubernetesia sen konttiorkestraatio-ominaisuuksien osalta ja lisää sen päälle uusia ominaisuuksia. OpenShiftin arkkitehtuurikomponenteista Main Nodet, Worker Nodet ja Etcd-tietokanta ovat samoja kuin Kubernetesissa. OpenShiftin omia komponentteja ovat Router, joka vastaa klusterin sisään tulevan verkkoliikenteen hallinnasta sekä pääluvun alussa esitelty OpenShift Internal Registry –levykuvarekisteri, johon OpenShiftissä käytettävät konttilevykuvat tallennetaan. (Zuev ym., 2018, 90)

OpenShiftissä on myös parannettu monia Kubernetesista löytyviä ominaisuuksia. Autentikaatio voidaan toteuttaa esimerkiksi LDAP-protokollalla tai GitHub-tunnuksilla, OpenShiftin hienojakoinen käyttäjien ja projektien pääsynhallinta mahdollistaa multitenanttiuden, jolloin samalla alustalla voi työskennellä useita projektitiimejä samanaikaisesti. OpenShiftiin on myös integroitu konttilevykuvien rakennusominaisuus Source to Image eli S2i sekä sen toiminnan mahdollistava Git-integraatio. (Mts. 95)

OpenShiftissä on myös joitakin uusia objekteja verrattuna Kubernetesiin.

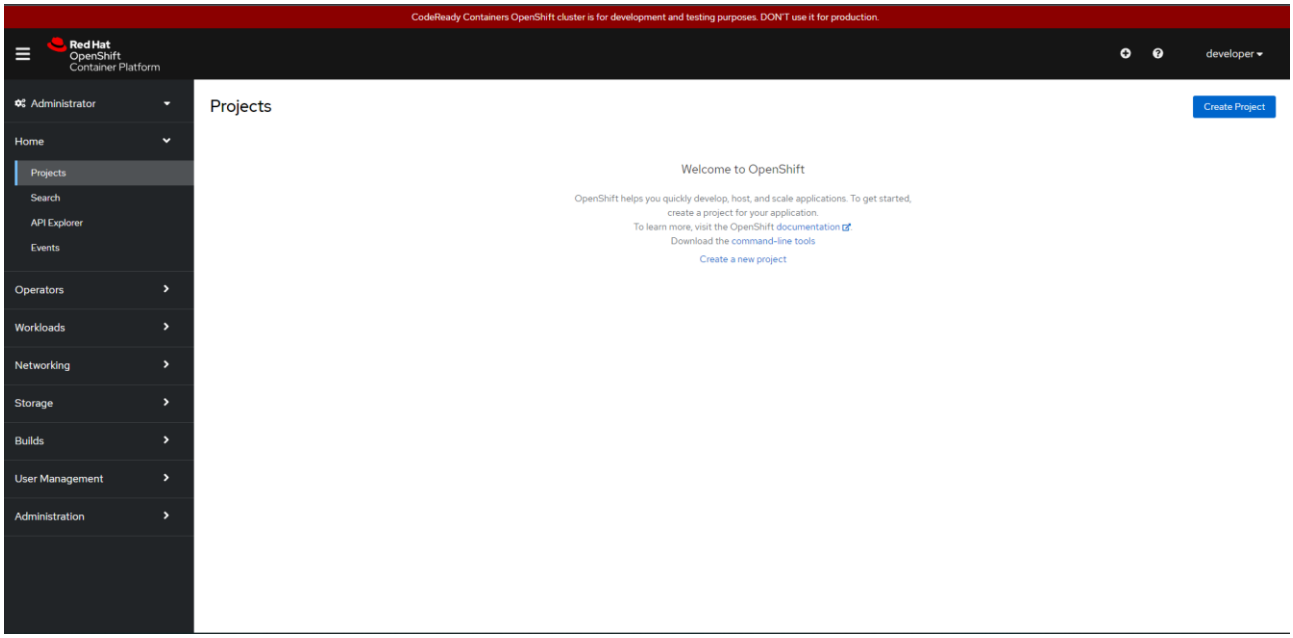
DeploymentConfig on Luvussa 4.2 esiteltyä Kubernetesin Deployment-objektia vastaava objekti. Uusina ominaisuuksina automaattinen palautus aikaisempaan versioon, muutosliipaisin, jonka avulla voidaan aloittaa uusi käyttöönotto sekä mahdollisuus liittää toimintoja perustuen Podien elinkaareen. (OpenShift, N.d)

BuildConfig in Source to Image –ominaisuuden konttilevykuvien rakentamiseen käyttämä objekti (Zuev ym., 2018, 95)

Route on DNS-nimipalvelu, jota OpenShiftin reititin käyttää OpenShiftissä sijaitsevien sovellusten sisääntulevan verkkoliikenteen sisääntulopisteenä. (Zuev ym., 2018, 95)

OpenShift Web Console ja OC-komentorivityökalu

OpenShiftiä voidaan hallita Kubernetesin tapaan komentoriviltä tai graafisesta selainkäyttöliittymästä, OpenShift Web Consolesta. Verkkokäyttöliittymässä on kaksi erillistä näkymää: Administrator- eli ylläpitäjän näkymä (Ks. Kuvio 8), sekä yksinkertaisempi, sovelluskehittäjille suunnattu Developer-näkymä.



Kuvio 8 OpenShift Web Consolen Administrator-näkymä

5 Kehitystyön toteutus

Rajapinta toteutettiin JavaScript-ohjelmointikielellä käyttäen Node.js-sovelluskehystä sekä Express.js ja kubernetes/client-node -kirjastoja. Node.js valittiin sovelluskehykseksi, koska sekä työn toteuttajalta että toimeksiantajalta löytyy siihen olemassa olevaa osaamista, jolloin rajapinnan jatkokehittäminen toimeksiantajan toimesta on mahdollista.

Rajapintaa päädyttiin kehittämään paikallisesti, koska virtuaalityöaseman kautta tapahtuva työskentely toimeksiantajan verkossa olisi ollut turhan hankalaa siitä saatuihin hyötyihin verrattuna. Kehitysvaiheessa ei ollut tarvetta testata rajapinnan toimintaa toimeksiantajan OpenShift-alustalla, koska OpenShift käyttää Kubernetesin APIa, jolloin toiminnallisuus oli mahdollista testata paikallisesti.

5.1 Kehitysympäristön asennus

Rajapintasovellus kehitettiin rinnakkain sitä hyödyntävän raportointinäkymän kanssa. Tämän vuoksi kehitysvaiheessa tarvittiin palvelin, jonne Kubernetes-klusteri ja rajapinta asennettiin, jotta paikalliselle työasemalle ei tarvinnut asentaa ja konfiguroida Kubernetes-klusteria. Tarkoitukseen valittiin pilvipalvelussa sijaitseva virtuaalinen palvelin, joka käytti Ubuntu 20.04 -käyttöjärjestelmää.

Kubernetes-jakeluksi valittiin k3s sen helpon asennusprosessin vuoksi. K3s:n ohjeen (Rancher, N.d.a) mukaisesti k3s asennettiin palvelimelle suorittamalla yksi ainoa komento (Ks. Kuvio 9).

```
root@ubuntu-2gb-hel1-1:~# curl -sfL https://get.k3s.io | sh -
[INFO]  Finding release for channel stable
[INFO]  Using v1.22.6+k3s1 as release
[INFO]  Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.22.6+k3s1/sha256sum-amd64.txt
[INFO]  Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.22.6+k3s1/k3s
[INFO]  Verifying binary download
[INFO]  Installing k3s to /usr/local/bin/k3s
[INFO]  Skipping installation of SELinux RPM
[INFO]  Creating /usr/local/bin/kubectl symlink to k3s
[INFO]  Creating /usr/local/bin/crictl symlink to k3s
[INFO]  Creating /usr/local/bin/ctr symlink to k3s
[INFO]  Creating killall script /usr/local/bin/k3s-killall.sh
[INFO]  Creating uninstall script /usr/local/bin/k3s-uninstall.sh
[INFO]  env: Creating environment file /etc/systemd/system/k3s.service.env
[INFO]  systemd: Creating service file /etc/systemd/system/k3s.service
[INFO]  systemd: Enabling k3s unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s.service → /etc/systemd/system/k3s.service.
[INFO]  systemd: Starting k3s
root@ubuntu-2gb-hel1-1:~# k3s kubectl get node
NAME               STATUS   ROLES                  AGE   VERSION
ubuntu-2gb-hel1-1   Ready    control-plane,master   41s   v1.22.6+k3s1
```

Kuvio 9. k3s:n asentaminen

Jotta palvelimella sijaitsevaa Kubernetes-klusteria voitiin käyttää paikallisella työasemalla, piti työasemalle asentaa kubectl-komentorivityökalu. Omalla työasemallani on käytössä Windows-käyttöjärjestelmä ja Git Bash-komentokehotteen kautta curl-työkalu, joten Kubernetesin dokumentaation (Kubernetes, N.d.g) ohjeiden mukaisesti myös kubectl voitiin asentaa yhdellä komennolla (Ks. Kuvio 10).

```
mton@desktop MINGW64 ~
$ curl -LO "https://dl.k8s.io/release/v1.23.0/bin/windows/amd64/kubectl.exe"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   154  100   154    0     0    750      0 --:--:-- --:--:-- --:--:--   754
100 45.6M  100 45.6M    0     0  9905k      0  0:00:04  0:00:04 --:--:-- 10.9M

mton@desktop MINGW64 ~
$ kubectl version --client
Client Version: version.Info{Major:"1", Minor:"22", GitVersion:"v1.22.5", GitCommit:"5c99e2ac2ff9a3c549d9ca665e7bc05a3e1
8f07e", GitTreeState:"clean", BuildDate:"2021-12-16T08:38:33Z", GoVersion:"go1.16.12", Compiler:"gc", Platform:"windows/
amd64"}
```

Kuvio 10. kubectl:n asentaminen

Jotta kubectl:n avulla voitiin komentaa etäpalvelimella sijaitsevaa klusteria, tuli klusterin konfiguraatiotiedosto tuoda paikalliselle työasemalle. K3s:n dokumentaation (Rancher, N.d.b) mukaan tämä tiedosto sijaitsee polussa /etc/rancher/k3s/k3s.yaml. Kubectl lukee konfiguraation käyttäjän kotikansiossa sijaitsevasta .kube -kansiosta tiedostosta config. Kubectl:n asennus ei luo

tätä kansiota automaattisesti, mutta kun se on luotu, tiedoston kopiointi oikeaan paikkaan onnistuu helposti scp-protokollan avulla:

scp käyttäjä@palvelin:/etc/rancher/k3s/k3s.yaml ~/.kube/config

Ladattuun tiedostoon tulee vaihtaa palvelimen IP-osoitteeksi sen palvelimen osoite, jossa klusteri sijaitsee. Tämän jälkeen tulee suorittaa vielä yksi komento, jonka jälkeen klusteria voidaan hallita paikallisella työasemalla suoraan ilman SSH-yhteyttä. Kuviossa 11 on esitelty komento, joka asettaa konfiguraatiotiedoston aktiiviseksi.

```
mton@desktop MINGW64 ~
$ kubectl config set-context default
Context "default" modified.

mton@desktop MINGW64 ~
$ kubectl get node
NAME                STATUS   ROLES                  AGE   VERSION
ubuntu-2gb-hel1-1   Ready    control-plane,master   17m   v1.22.6+k3s1
```

Kuvio 11. kubectl:stä yhteys etäpalvelimelle

Koodin kirjoittamiseen käytettiin Microsoftin kehittämää Visual Studio Code -koodieditoria. Visual Studio Codeen on saatavilla runsaasti lisäosia, jotka helpottavat eri ohjelmointikielien ja työkalujen kanssa työskentelyä. Versionhallintajärjestelmäksi valittiin Git ja repositorion tallennuspaikaksi GitHub. Konttilevykuvien rakentamiseen käytettiin Dockerin tarjoamaa Desktop-ohjelmaa, koska se oli helppo asentaa Windows-käyttöjärjestelmälle.

5.2 Rajapinnan toteutus

Toimeksiantajan vaatimuksena raportointinäkymälle oli kyky listata tietoja klusterin nimiavaruuksien eli eri projektien sekä Nodejen eli palvelimien tilasta. Rajapinnan tuli myös tallentaa resurssipyyntöjen vastaukset välimuistiin, jotta Kubernetesin APIin ei kohdistuisi suurta määrää http-pyyntöjä joka kerta, kun raportointinäkymä ladataan.

Projektin luonti ja kirjastojen asentaminen

Node.js -projekti aloitetaan komennolla npm init. Tämä komento luo package.json-tiedoston, jossa määritellään metatietoja projektista, skriptejä ja projektin tarvitsemat npm -pakettirekisterissä sijaitsevat kirjastot. Kun package.json on luotu, voidaan paketteja asentaa komennolla npm install –save paketinnimi.

Kuten luvun 5 alussa kerrottiin, käytettiin rajapinnan kehittämiseen Express.js ja kubernetes/client-node -kirjastoja. Express.js on ohjelmakirjasto, joka helpottaa Node.js:n kanssa työskentelyä. Kubernetes/client-node on Kubernetesin virallinen JavaScript-asiakaskirjasto, jonka avulla autentikaatio ja resurssipyyntöjen tekeminen Kubernetesin APIin helpottuu.

Yhteyden luonti Kubernetesin APIin

Kubernetesin JavaScript-kirjastolla yhteyden luonti Kubernetesin APIin onnistui helposti. Kirjasto huolehtii autentikaatiosta käyttäjän puolesta. Kirjasto lataa Kubernetesin tarvitsemat autentikaatiotiedot käytettävästä funktiokutsusta riippuen joko paikalliselta työasemalta tai hakemalla konfiguraatiotiedosto etäpalvelimelta.

```
3   const k8s = require('@kubernetes/client-node');
4
5   const kubeConfig = new k8s.KubeConfig();
6
7   if (process.env.NODE_ENV == 'production') {
8     kubeConfig.loadFromCluster();
9   } else {
10    kubeConfig.loadFromDefault();
11  }
12
13  const kubeApi = kubeConfig.makeApiClient(k8s.CoreV1Api);
14
15  module.exports = kubeApi;
```

Kuvio 12. Kubernetesin APIin yhdistävä koodi

Kuviossa 12 on esitelty kokonaisuudessaan koodi, jolla yhteyden ottaminen Kubernetesin APIin tapahtuu. Koodi tarkistaa NODE_ENV -ympäristömuuttujan arvon ja lataa konfiguraation joko

paikalliselta työasemalta tai etäpalvelimelta riippuen siitä, onko muuttujan arvona development vai production. Tämä moduuli voidaan ottaa käyttöön muissa tiedostoissa, joissa tehdään pyyntöjä Kubernetesin APIin. Kun moduuli on otettu käyttöön, voidaan Kubernetesin APIsta hakea tietoa yksinkertaisella funktiokutsulla. Kuviossa 13 on esitelty koodi, jonka avulla klusterilta pyydetään tiedot kaikista nimiavaruuksista.

```
8  namespaceRouter.get('/', async (req, res, next) => {
9    try {
10     const allNamespaces = await kubeApi.listNamespace();
```

Kuvio 13. Kaikkien Namespace-objektien hakeminen Kubernetesin APIsta

5.3 Rajapinnan käyttöönotto OpenShiftissä

Koska rajapinnan tuli toimia toimeksiantajan OpenShift-alustalla, halusin varmistua siitä, että se toimii kunnolla myös kyseisellä alustalla. Sovellusten käyttöönottoon OpenShiftissä on useita vaihtoehtoja. Tässä työssä käydään läpi niistä yksinkertaisin, Source to Image eli S2i, joka hakee sovelluksen lähdekoodin versionhallintajärjestelmästä ja luo siitä konttilevykuvan.

CodeReady Containersin asentaminen

OpenShift-alustasta on luotu paikallista kehitystä varten kevyempi versio, jonka voi asentaa omalle tietokoneelleen. Sen avulla ohjelmistokehittäjät voivat kehittää ohjelmistoja OpenShiftin päällä alusta alkaen. CodeReady Containersin voi ladata Red Hatin kehittäjäsivustolta osoitteesta https://console.redhat.com/openshift/create/local. Lataaminen vaatii Red Hat -tunnukset.

CodeReady Containers asennetaan ladatulla asennuspaketilla, jonka jälkeen se konfiguroidaan komentoriviltä komennolla crc setup. Kun asennus on valmis, paikallinen klusteri käynnistetään komennolla crc start. Klusterin käynnistyminen kestää muutaman minuutin. Kun klusteri on käynnistynyt, OpenShiftin käyttöliittymään pääsee menemällä selaimella osoitteeseen https://console-openshift-console.apps-crc.testing/.

Projektin luonti

Uusi namespace eli projekti voidaan luoda komentoriviltä tai käyttöliittymästä. Komentoriviltä se tapahtuu komennolla oc create project <projektin nimi>. Käyttöliittymästä se onnistuu painamalla Create Project-näppäintä.

ClusterRole ja ClusterRoleBinding

Koska toimeksiantajan alustalla on käytössä luvussa 4.4 esitelty rooliperustainen pääsynhallinta eli RBAC, piti rajapinnan toiminta testata rajoitetuilla oikeuksilla. Rajapinta tarvitsi ClusterRolen, joka on liitetty ServiceAccount-tunnukseen ClusterRoleBinding-objektilla. ClusterRole-objektissa määritellään ne Kubernetesin resurssit ja API-ryhmät, joihin sovelluksella on pääsyoikeus.

```
1  apiVersion: rbac.authorization.k8s.io/v1
2  kind: ClusterRole
3  metadata:
4    name: dashboard-api-user-role
5  rules:
6    - apiGroups: [""]
7      resources: ["namespaces", "nodes", "resourcequotas", "pods"]
8      verbs: ["get", "list"]
```

Kuvio 14. ClusterRolen määrittelevä yaml -manifesti

Kuviossa 14 esitelty YAML-kuvauskielellä kirjoitettu manifesti määrittää, että rajapinnalla on pääsy resursseihin namespaces, nodes, resourcequotas ja pods. Lisäksi roolissa määritellään, että rajapinnalla on lupa käyttää http-verbejä get ja list. Mikäli rajapinta yrittäisi käyttää esimerkiksi verbiä post, olisi tuloksena virhe. OpenShift luo automaattisesti kolme ServiceAccountia projektin luomisen yhteydessä: builder, deployer ja default. Yllä esitelty ClusterRole liitetään default-tunnukseen ClusterRoleBindingin avulla (Ks. Kuvio 15).

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: dashboard-api-user-role-binding
roleRef:
  kind: ClusterRole
  name: dashboard-api-user-role
  apiGroup: rbac.authorization.k8s.io
subjects:
  - kind: ServiceAccount
    name: dashboard-api-user
    namespace: default
```

Kuvio 15. ClusterRoleBinding -manifesti

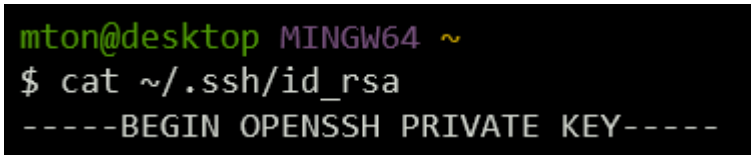
Manifestit voidaan luoda komennolla oc create -f <manifesti.yml> tai käyttöliittymän kautta Administrator-näkymässä menemällä User Management -> Roles/RoleBindings -> Create Role/RoleBinding ja kirjoittamalla tai liittämällä YAML-manifesti editoriin.

Koska ClusterRole ja ClusterRoleBinding ovat klusterin laajuisia resursseja, ei niitä voi luoda CodeReady Containersin developer tunnuksella. Admin-oikeudet saat kun kirjaudut sisään kubeadmin tunnuksilla. CodeReady Containersin ollessa käynnissä, saat salasanan näkyviin komennolla crc console --credentials.

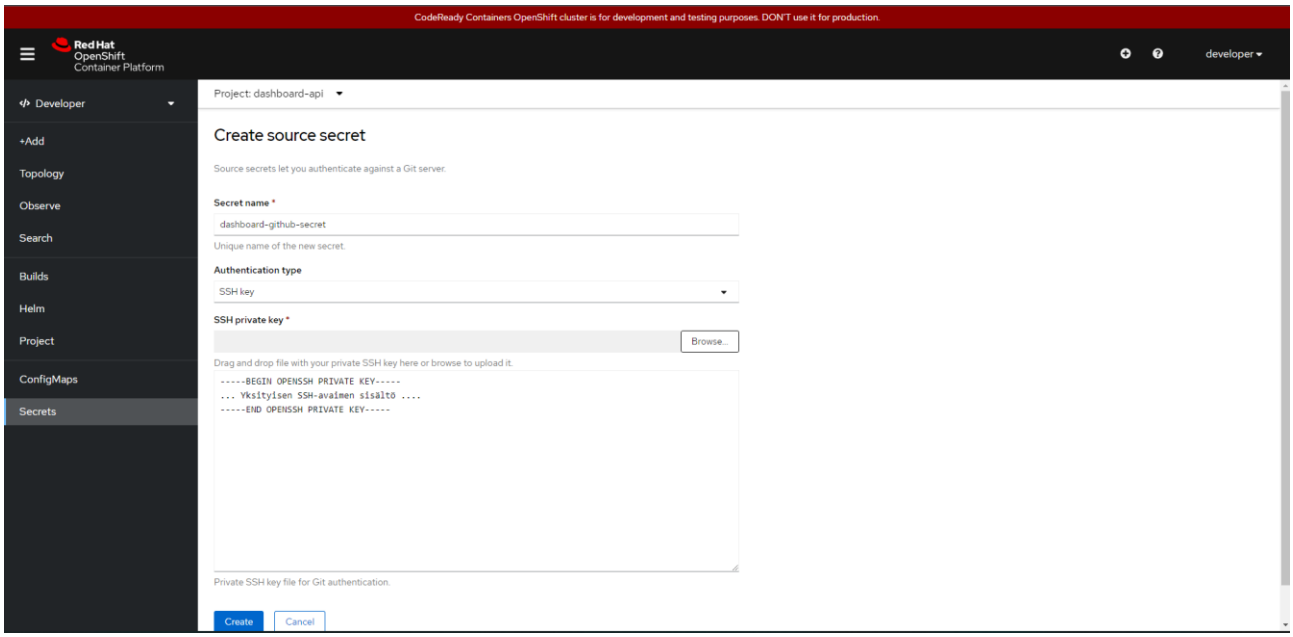
SSH-yhteys Githubiin

Koska rajapinnan lähdekoodi sijaitsee yksityisessä Github-repositoriossa, tulee OpenShiftin ja Githubin välille luoda SSH-yhteys, jotta OpenShift voi kloonata lähdekoodin sovelluksen käyttöönottovaiheessa. Tämä onnistuu OpenShiftin Developer-näkymässä valitsemalla vasemmasta sivupalkista Secrets -> Create a Secret -> Source Secret.

Uusi SSH-avain luodaan esimerkiksi Git Bashissa komennolla ssh-keygen. Kun avain on luotu, tulosta sen sisältö konsoliin komennolla cat /avaimen/sijainti, kopioi ja liitä se OpenShiftin käyttöliittymässä olevaan tekstilaatikkoon ja paina Create (Ks. Kuviot 16 ja 17).



Kuvio 16. Yksityisen SSH-avaimen tulostaminen konsoliin



Kuvio 17. Github -yhteyden tarvitseman Secret-objektin luominen

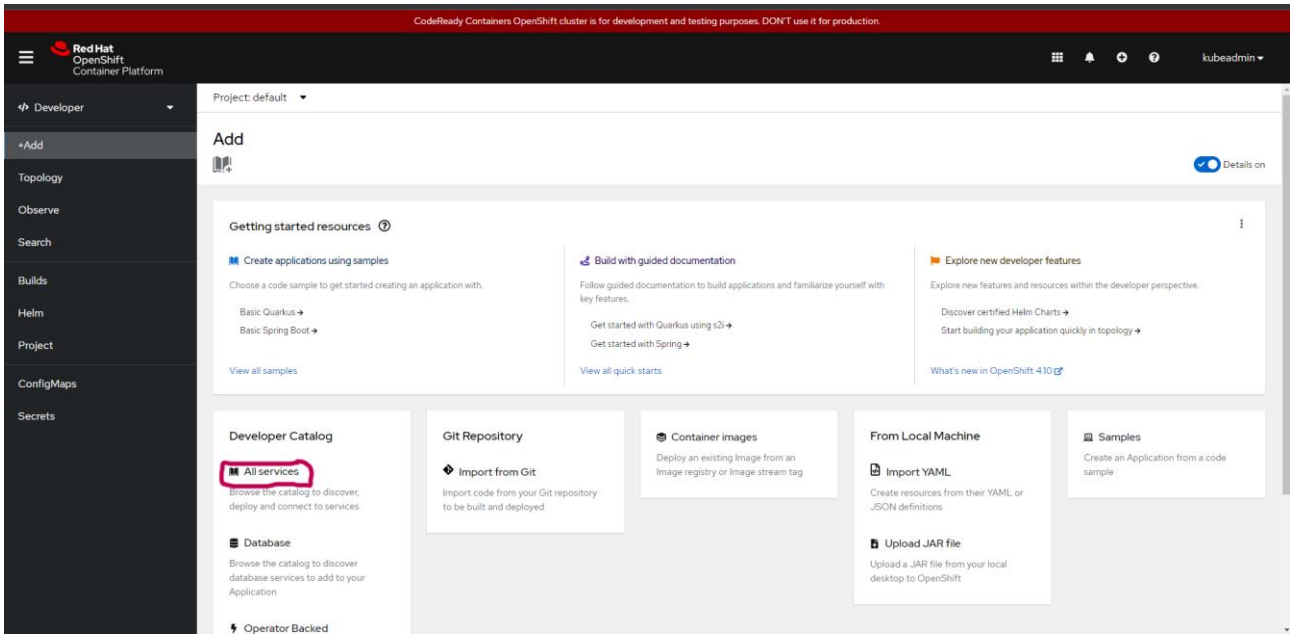
Seuraavaksi tulee SSH-avainparin julkinen osa lisätä siihen Github-repositorioon, johon haluat luoda yhteyden. Tulosta äsken luomasi avainparin julkinen osa konsoliin ja kopioi se leikepöydälle (Ks. Kuvio 18). Mene Githubissa repositorion asetuksiin ja valitse vasemmasta navigaatiopalkista kohdan Security alta Deploy Keys. Paina Add deploy key –painiketta ja liitä kopioimasi avain Key –tekstilaatikkoon ja paina Add key –painiketta.

```
mton@desktop MINGW64 ~
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDJ5V4zxMAyjTBWpacH0I4zCvVloIyRMtYljqTk8W4MLJY/9V1GvQZypcNRgmwBLMvfO/VapzkNkJaxInz3
oMICav8Xe+wITMVQr98hsKif8AFZHzj+a1YLE7gbqTfnUabMa4NU9kmizYIiPRatzyGIAhoBNjgbuTXP5lpCyuA/5quX5JWNi504Ouv0jC8n+VYzKF5mq4Wf
dPHi2GfQX+i/bkk8KcIpnU5GwVxsL/W6ncg86NVOVgyAYnpL4fGPDqky1ctfLqSTzkNUtYdRom52cV9ynhN3XqCiR0eQ/C20ZBUpJbAh0f+UYpAk1tFBHN6L
gfSw6x9TzFHpc195NQ2vx6o6II+Au2tTe5YEW4TnEwvX2ijjtMIgZ8GAo4nkkdMYl60URXapjiLVjD02EE/5P5j14NNViV2yIhbdmq5xU/dlZvutyWQ6L9Re
zgWutSCH94MoBUHQLRioAHHRnMM7sq4FZ9Ov/lENoIVardeZeBDF9yV8oVHrEuBhlmAfhO0= mton@desktop
```

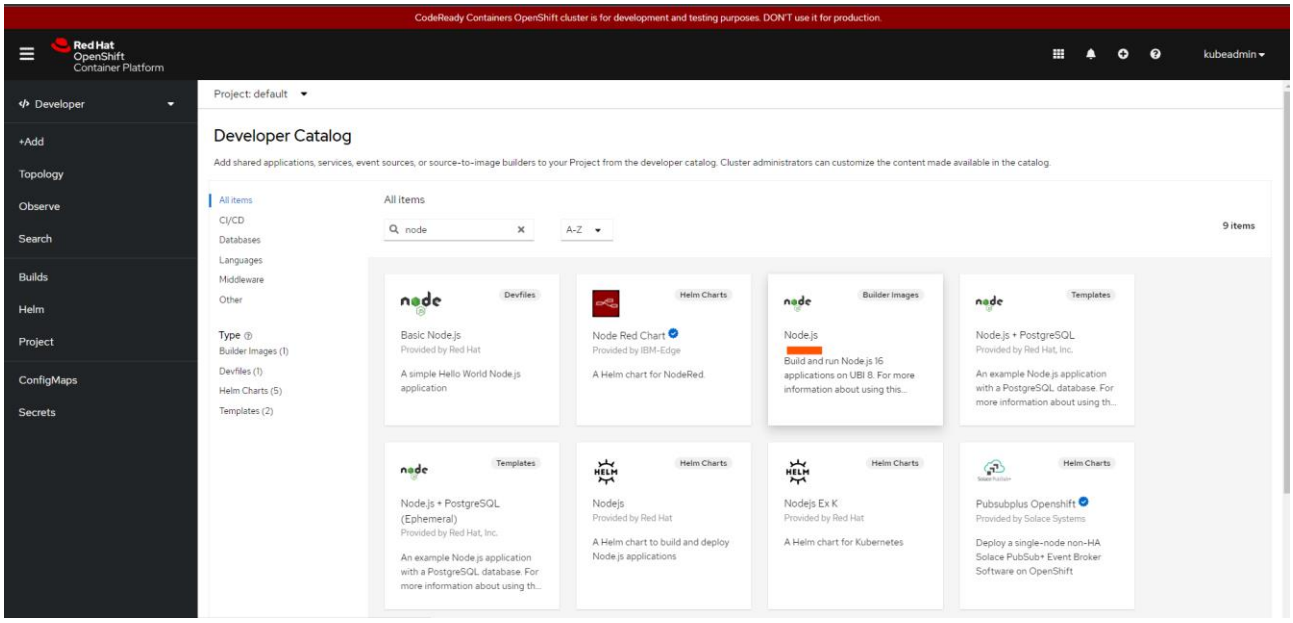
Kuvio 18. Julkisen SSH-avaimen tulostaminen konsoliin

Konttilevykuvan rakentaminen ja käyttöönotto S2i:n avulla

Kun kaikki sovelluksen tarvitsemat resurssit on luotu, voidaan siirtyä konttilevykuvan rakentamiseen ja lopulta sovelluksen käyttöönottoon OpenShift-alustalla. Uusi sovellus otetaan käyttöön Source to Image -ominaisuudella. OpenShiftin Developer -näkymässä valitse Add, Developer Catalog ja All Services (Ks. Kuvio 19.), kirjoita hakukenttään node, ja valitse Node.js Builder Image (Ks. Kuvio 20). Paina Create Application.



Kuvio 19. Sovelluksen lisäys OpenShiftiin



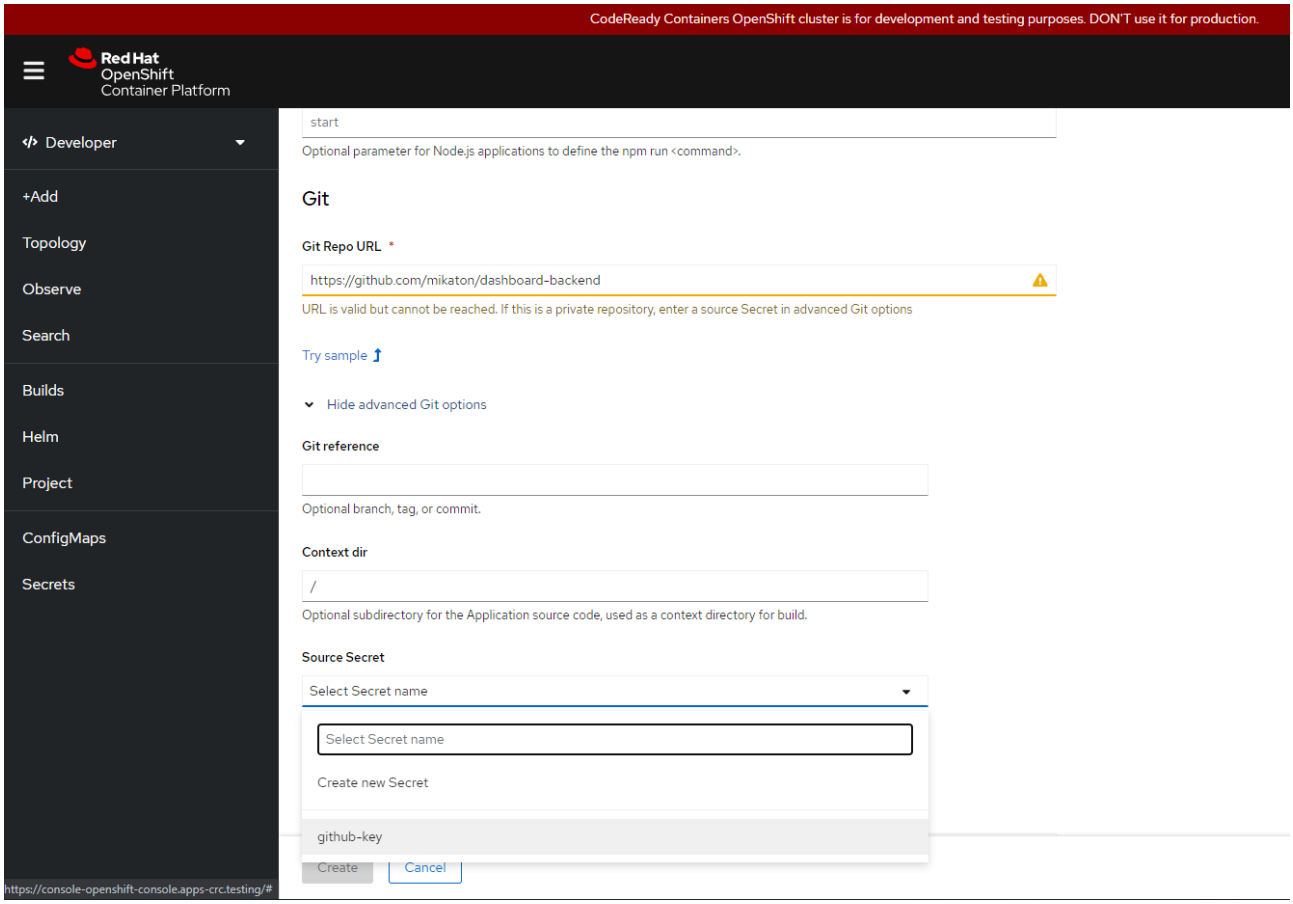
Kuvio 20. Developer Catalogista valitaan Node.js builder image

Aukeaa lomake, jonka avulla uusi Node.js -sovellus lisätään OpenShiftiin. Tällä lomakkeella asetetaan haluttu pohjana käytettävä levykuva. Vakiona kirjoitushetkellä on Node.js 16. Run command -kohtaan kirjoitetaan komento, joka kontin sisällä suoritetaan sen käynnistyttyä. Vakiona lomakkeella on start, joka käynnistää Node.js -sovelluksen.

Kohtaan Git lisätään käytettävän Git repositorion osoite. Koska kyseessä on yksityinen repositorio, ilmoittaa OpenShift tällöin osoitteen olevan oikea, mutta se ei ole saavutettavissa. Valitse kohdasta Source Secret aiemmin luomasi Secret-objekti, joka sisältää Githubiin yhteyden luovan SSH-avaimen (Ks. Kuvio 21). Kirjoitushetkellä virheilmoitus repositorion saavutettavuudesta pysyy, vaikka Secret lisätään tässä vaiheessa.

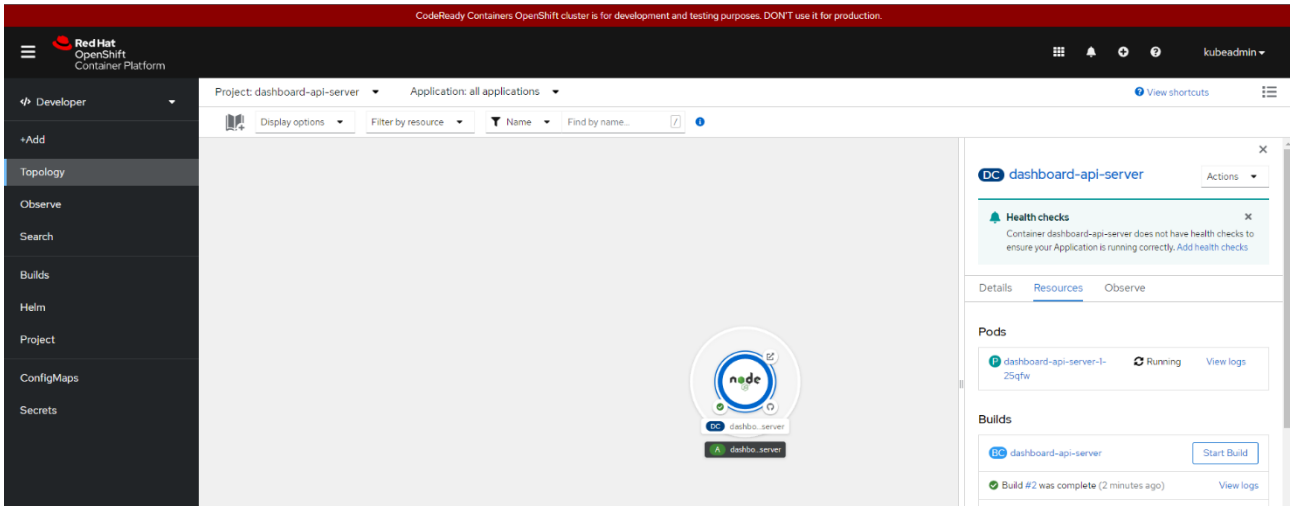
General -kohdassa aseta sovellukselle ja resurssille valitsemasi nimi kenttiin Application name ja Name. Valitse Resources -kohdassa DeploymentConfig luotavaksi resurssityypiksi. Advanced options -kohdassa asetetaan sovelluksen kohdeportti, jossa sovellus vastaa pyyntöihin. Valitse Create a route to the Application, jotta sovellus on saavutettavissa klusterin ulkopuolella. Mikäli sovellus ei kykene käsittelemään salattua liikennettä, pitää valinta kohdasta Secure Route poistaa, jotta se luodaan salaamattomana http-reittinä. Lopuksi paina Create, jolloin sovelluksen käyttöönotto käynnistyy.

Source to Image -ominaisuus kloonaa lähdekoodin ja luo siitä luvussa 3.3 esitellyn Dockerfilen. Tästä Dockerfilestä luodaan konttilevykuva, joka tallennetaan luvussa 4.5 esiteltyyn OpenShiftin sisäiseen levykuvarekisteriin. Tämän levykuvan perusteella luodaan sovelluksen tarvitsema kontti, jonka jälkeen se on käyttövalmis.



Kuvio 21. Source Secretin valinta

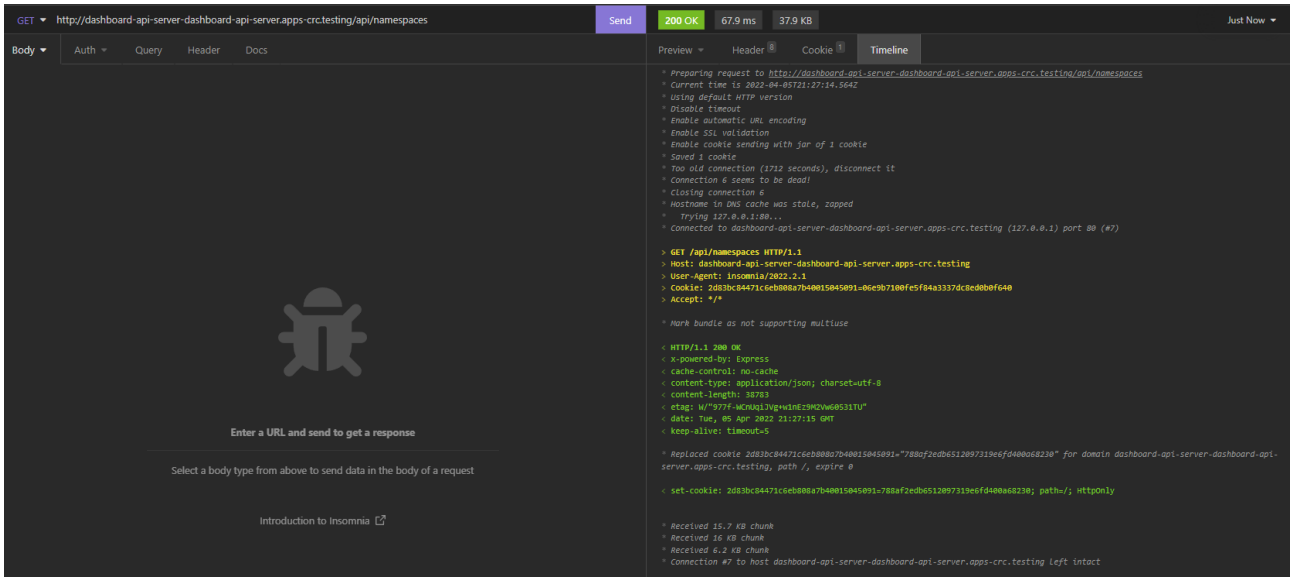
Käyttöliittymä siirtyy automaattisesti Topology-sivulle ja näyttää luodun DeploymentConfigin tiedot oikeassa reunassa sijaitsevassa ikkunassa (Ks. Kuvio 22). Mikäli sovelluksen kääntäminen ja käyttöönotto onnistui, on rajapinta nyt käytettävissä Route-objektin osoitteessa, joka löytyy samasta ikkunasta rullaamalla sitä alaspäin tai vaihtoehtoisesti Administrator-näkymästä kohdasta Network -> Routes.



Kuvio 22. DeploymentConfigin tietoikkuna

5.4 Rajapinnan testaaminen

Kun rajapinta oli otettu käyttöön OpenShiftissä, piti sen toimintaa testata. Rajapinnan toimintaa testattiin tekemällä http-pyyntöjä rajapinnan resursseja tarjoaviin reitteihin ja vertaamalla sen palauttamaa tietoa oletettuun. Tätä toistettiin, jotta voitiin varmistua siitä, että ohjelma toimii halutulla tavalla. Testaus toteutettiin REST APIen testaukseen kehitetyllä avoimen lähdekoodin Insomnia -ohjelmalla. Muita vaihtoehtoja REST APIen testaamiseen ovat esimerkiksi Postman ja komentorivityökalut curl ja HTTPie. Kuviossa 23 on esimerkki Insomnialla tehdystä http-pyynnöstä klusterin projekteista tietoa tarjoavaan reittiin.



Kuvio 23. Onnistunut HTTP-pyyntö OpenShiftissä sijaitsevaan rajapintaan

Rajapinta piti testata myös suuremman kuorman alla kuin manuaalisesti on mahdollista, koska virheelliset vastaukset suuren kuorman alla saattoivat kieliä ohjelmointivirheistä. Kuormatestaukseen käytettiin ApacheBench-työkalua, joka on saatavilla kaikille yleisimmille käyttöjärjestelmille. ApacheBenchiä käytetään komentorivin kautta. Sille annetaan argumenttina yhtäaikaisten yhteyksien lukumäärä ja pyyntöjen kokonaismäärä sekä osoite, jossa rajapinta vastaa pyyntöihin. Kuviossa 24 näytetään, kuinka ApacheBenchillä tehdään palvelimelle 1000 http-pyyntöä niin, että niitä lähetetään samanaikaisesti 10. Tuloksesta on luettavissa, että rajapinta toimii kuorman alla kuten pitääkin, koska pyynnöt menivät nopeasti läpi eikä epäonnistuneita pyyntöjä ollut.

```
PS C:\apache\Apache24> .\bin\ab.exe -n 1000 -c 10 http://dashboard-api-server-dashboard-api-server.apps-crc.testing/api/namespaces
This is ApacheBench, Version 2.3 <$Revision: 1879490 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking dashboard-api-server-dashboard-api-server.apps-crc.testing (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:
Server Hostname:        dashboard-api-server-dashboard-api-server.apps-crc.testing
Server Port:            80

Document Path:          /api/namespaces
Document Length:        38783 bytes

Concurrency Level:      10
Time taken for tests:   1.009 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      39166973 bytes
HTML transferred:       38783000 bytes
Requests per second:    991.31 [#/sec] (mean)
Time per request:       10.088 [ms] (mean)
Time per request:       1.009 [ms] (mean, across all concurrent requests)
Transfer rate:          37916.54 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       1
Processing:     4   10   3.9      8      34
Waiting:        4    9   3.9      8      34
Total:          4   10   3.9      9      34

Percentage of the requests served within a certain time (ms)
  50%      9
  66%     10
  75%     11
  80%     12
  90%     15
  95%     18
  98%     21
  99%     24
 100%     34 (longest request)
PS C:\apache\Apache24>
```

Kuvio 24. Kuormatestaus ApacheBenchillä

6 Pohdinta

6.1 Tulokset

Työn alussa asetettuihin tutkimuskysymyksiin löydettiin vastaukset. Ensimmäiseen kysymykseen vastattiin heti luvun 4 alussa. Kubernetes on konttiorkestraatioalusta, joka toimii klusterina. Jälkikäteen ajateltuna kysymys oli liian yksinkertainen eikä siihen vastaamisesta ollut merkittävää hyötyä työn toteutuksen kannalta. Toinen ja kolmas kysymys olivat parempia, koska niihin vastaaminen mahdollisti työn tavoitteeseen pääsemisen. Toisen kysymyksen vastaus oli, että Kubernetesin APIsta voidaan hakea tietoa tekemällä siihen http-pyyntöjä. Kolmannen kysymyksen vastaus oli luvussa 5.3 esitellyt ClusterRole ja ClusterRoleBinding-objektit, joiden piti olla klusterissa oikein asetettuna.

Työn tavoitteena oli kehittää toimeksiantajan vaatimusmäärittelyn täyttävä rajapintasovellus. Tähän tavoitteeseen päästiin ja rajapinta ja sen rinnalla kehitetty raportointinäkymä ovat toiminnassa toimeksiantajan OpenShift-alustalla. Toimeksiantajalla on kirjoitushetkellä käynnissä siirtyminen Confluence-dokumentaatiosta raportointinäkymän käyttöön alustan resurssien dokumentoinnissa.

Rajapinnalle on monta mahdollista jatkokehityssuuntaa. Yksi näistä on rooliperustaisten pääsyoikeuksien ulottaminen koskemaan myös rajapintaa ja raportointinäkymää, jolloin asiakas näkisi raportointinäkymässä vain ne resurssit, jotka hänellä on tehtäväroolinsa perusteella oikeus nähdä. Rajapinta on dokumentoitu niin, että toimeksiantajan omat kehittäjät voivat kehittää lisäominaisuuksia itsenäisesti.

Kehitystyö oli haastavaa, koska esimerkkejä ja käyttäjäkokemuksia Kubernetesin APIn hyödyntämisestä ohjelmallisesti oli rajallisesti saatavilla. Monesta ongelmasta selvittiinkin yrittämisen ja epäonnistumisen kautta. Automaattisten integraatiotestien luominen ja käyttäminen heti projektin alkuvaiheesta lähtien olisivat mahdollistaneet ohjelmointivirheiden ja muiden ongelmien löytymisen nopeammin. Myös tutkimuksen keskittäminen täysin Kubernetesiin ja sen APIn toimintaan olisi tukenut kehitystyötä. Konttiteknologian toiminnan tarkempi tutkiminen ei ollut kehitystyön kannalta oleellista, eikä siitä syntynyt kovin suurta lisäarvoa työlle.

Kehitystyötä olisi helpottanut DevOps –menetelmien hyödyntäminen. Kehitysvaiheessa konttilevykuvia rakennettiin ja otettiin käyttöön Kubernetes-klusterissa manuaalisesti. Koodimuutosten jatkuvalla integraatiolla eli CI-menetelmällä ja konttien jatkuvalla käyttöönotolla eli CD-menetelmällä merkittävä määrä manuaalista työtä olisi voitu automatisoida.

6.2 Luotettavuus

Vaikka työn tavoitteet saavutettiin, nousee siitä esille joitakin sen luotettavuutta heikentäviä asioita. Työstä puuttui jossain määrin suunnitelmallisuus ja tekijän työhön liittymättömät vaikeudet elämässä tekivät systemaattisesta etenemisestä vaikeaa.

Työn tutkimuksellisen osuuden laajuus ja aihepiiri muuttui prosessin alkuvaiheessa useasti. Työlle oli vaikeata löytää sellainen suunta, joka tukisi tavoitteiden saavuttamista ja olisi ajankohtaista. Tietoperusta ja työn empiirinen osuus ovat siksi hieman irrallisia kokonaisuuksia, jolloin työn toistaminen voi olla haastavaa.

Lähteiden valinta työn tietoperustan keräämiseksi oli myös haastavaa, koska kyseessä on sekä verrattain uusi, mutta samalla nopeasti eteenpäin liikkuva teknologia, joten lähteiden piti olla sekä riittävän uusia että täyttää lähteille asetetut kriteerit. Työssä pyrittiin käyttämään lähteenä ensisijaisesti järjestelmien virallista teknistä dokumentaatiota sekä ammattikirjallisuutta. Tietoperustassa käytettiin lähteinä myös joitakin blogiviestejä, vaikka niiden luotettavuutta on hankala arvioida. Näitä lähteitä pyrittiin vertaamaan muuhun aiheesta kirjoitettuun tietoon sekä arvioimaan kirjoittajan luotettavuutta muun muassa etsimällä kirjoittajasta tietoa internetistä.

6.3 Eettisyys

Työssä on pyritty noudattamaan tutkimuseettisiä periaatteita koko prosessin ajan. Toimeksantajan kanssa tehtyä salassapitosopimusta on noudatettu tarkasti, jotta salaista tietoa ei pääsisi vuotamaan. Lähteiden luotettavuutta on pyritty arvioimaan kriittisesti sekä vahvistamaan niiden esittämiä väitteitä muista lähteistä. Työn tekstissä lähdeviitteet on merkattu ohjeistuksen mukaisesti ja kaikki käytetyt lähteet on listattu lähdeluettelossa.

Lähteet

Abushamleh, M. 2020. OpenShift 101: Introduction, architecture, and operators. Blogiviesti IBM:n kehittäjäsivustolla. Viitattu 30.12.2021. https://developer.ibm.com/blogs/openshift-101-architecture/.

Caban, W. 2019. Architecting and Operating OpenShift Clusters: OpenShift for Infrastructure and Operations Teams. E-kirja. 1. Painos. Berkeley, CA: Apress. Viitattu 03.11.2021. https://janet.finna.fi, SkillSoft Books ITPro.

Docker. N.d. Dockerfile reference. Dockerin online-dokumentaatio. Viitattu 15.02.2022. https://docs.docker.com/engine/reference/builder/.

Gandhi, R., Szmrecsanyi, P. 2019. The Benefits of Containerization and What It Means for You. Blogiviesti IBM:n nettisivuilla. Viitattu 18.10.2021. https://www.ibm.com/cloud/blog/the-benefits-of-containerization-and-what-it-means-for-you.

Guyton, S. 2019. Containers & Containerization – Pros and Cons. Blogiviesti Atomic Objectin nettisivuilla. Viitattu 01.11.2021. https://spin.atomicobject.com/2019/05/24/containerization-pros-cons/.

IBM. 2021. Container Orchestration. Konttiorkestrointia esittelevä artikkeli IBM:n nettisivuilla. Viitattu 24.02.2022. https://www.ibm.com/cloud/learn/container-orchestration.

Jain, S. M. 2020. Linux Containers and Virtualization: A Kernel Perspective. E-kirja. 1. Painos. Berkeley, CA: Apress. https://janet.finna.fi, SkillSoft Books ITPro.

Kananen, J. 2015. Kehittämistutkimuksen kirjoittamisen käytännön opas. E-Kirja. Jyväskylä: Jyväskylän Ammattikorkeakoulu. Viitattu 03.11.2021. https://janet.finna.fi, Booky.

Kubernetes. N.d.a. What is Kubernetes? Kubernetesin online-dokumentaatio. Viitattu 10.11.2021. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.

Kubernetes. N.d.b. Kubernetes Components. Kubernetesin online-dokumentaatio. Viitattu 10.11.2021. https://kubernetes.io/docs/concepts/overview/components/.

Kubernetes. N.d.c. Understanding Kubernetes Objects. Kubernetesin online-dokumentaatio. Viitattu 10.11.2021. https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/.

Kubernetes. N.d.d. Service. Kubernetesin online-dokumentaatio. Viitattu 10.11.2021. https://kubernetes.io/docs/concepts/services-networking/service/.

Kubernetes. N.d.e. Namespaces. Kubernetesin online-dokumentaatio. Viitattu 12.11.2021. https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/.

Kubernetes. N.d.f. Using RBAC Authorization. Kubernetesin online-dokumentaatio. Viitattu 12.11.2021. https://kubernetes.io/docs/reference/access-authn-authz/rbac/.

Kubernetes. N.d.g. Install Tools. Kubernetesin online-dokumentaatio. Viitattu 23.02.2022. https://kubernetes.io/docs/tasks/tools/.

Kuraman S., S. 2017. Practical LXC and LXD: Linux Containers for Virtualization and Orchestration. 1. Painos. USA: Berkeley, CA: Apress. Viitattu 04.10.2021. https://janet.finna.fi, SkillSoft Books IT-Pro.

Ojasalo, K & Moilanen, T & Ritalahti, J. 2014. Kehittämistyön menetelmät: Uudenlaista osaamista liiketoimintaan. Kolmas, uudistettu painos. Helsinki: Sanoma Pro. Viitattu 03.11.2021. https://janet.finna.fi, Ellibslibrary.

Open Container Initiative. N.d.a. About the Open Container Initiative. Yleiskuvaus Open Container Initiativesta OCI:n nettisivuilla. Viitattu 01.11.2021. https://opencontainers.org/about/overview/.

Open Container Initiative. N.d.b. OCI Image Format build diagram. Kuva OCI:n Github-repositoriossa. Viitattu 01.11.2021. https://github.com/opencontainers/image-spec/blob/main/img/build-diagram.png.

Open Container Initiative. N.d.c. OCI Image Manifest Specification. Tekninen dokumentti OCI:n Github-repositoriossa. Viitattu 03.11.2021. https://github.com/opencontainers/image-spec/blob/main/manifest.md.

Open Container Initiative. N.d.d. Image Layer Filesystem Changeset. Tekninen dokumentti OCI:n Github-repositoriossa. Viitattu 03.11.2021. https://github.com/opencontainers/image-spec/blob/main/layer.md.

Open Container Initiative. N.d.e. Runtime and Lifecycle. Tekninen dokumentti OCI:n Github-repositoriossa. Viitattu 09.11.2021. https://github.com/opencontainers/runtime-spec/blob/main/runtime.md.

OpenShift. N.d. Understanding Deployments and DeploymentConfigs. OpenShiftin online-dokumentaatio. Viitattu 31.3.2022. https://docs.openshift.com/container-platform/4.8/applications/deployments/what-deployments-are.html.

PortWorx. 2021. 2021 Kubernetes Adoption Survey. Kyselytutkimusraportti. Viitattu 19.10.2021. https://www.purestorage.com/content/dam/pdf/en/analyst-reports/ar-portworx-pure-storage-2021-kubernetes-adoption-survey.pdf.

Poulton, N. 2021. The Kubernetes Book, 2021 edition. E-kirja. Itsejulkaistu. Viitattu 28.10.2021. https://janet.finna.fi. SkillSoft Books ITPro.

Rancher. N.d.a. K3s. K3s Kubernetes-jakelun online-dokumentaation etusivu. Viitattu 23.02.2022. https://k3s.io/.

Rancher. N.d.b. Cluster Access. K3s Kubernetes-jakelun online-dokumentaatio. Viitattu 23.02.2022. https://rancher.com/docs/k3s/latest/en/cluster-access/.

Red Hat. 2019. What is container orchestration? Artikkeli Red Hatin nettisivuilla. Viitattu 26.10.2021. https://www.redhat.com/en/topics/containers/what-is-container-orchestration.

Red Hat. 2019. What is the difference between OpenShift and Kubernetes? Artikkeli Red Hatin kehittäjäsivustolla. Viitattu 31.3.2022. https://developers.redhat.com/openshift/difference-openshift-kubernetes.

VMware. N.d. Why use containers vs VMs? Artikkeli VMwaren nettisivuilla. Viitattu 26.10.2021. https://www.vmware.com/topics/glossary/content/vms-vs-containers.

Zuev, D., Kropachev, A. & Usov, A. 2018. Learn OpenShift. E-kirja. 1. Painos. Birmingham, UK: Packt Publishing. Viitattu 30.12.2021. https://janet.finna.fi, ProQuest Ebook Central.