SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Joel Tikka

Varastonhallintajärjestelmän tuotantoympäristön kehitys

Palveluiden automatisointi Docker- ja Kubernetes-työkaluilla

Opinnäytetyö
Kevät 2022
Automaatiotekniikan tutkinto-ohjelma

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Tutkinto-ohjelma: Automaatiotekniikka

Suuntautumisvaihtoehto: Koneautomaatio

Tekijä: Joel Tikka

Työn nimi: Varastonhallintajärjestelmän tuotantoympäristön kehitys

Ohjaaja: Juha Yli-Hemminki

Vuosi: 2022 Sivumäärä: 68

Opinnäytetyön tavoitteena oli selvittää ja toteuttaa Pesmel Oy:n tarjoaman automatisoidun varastonhallintajärjestelmän toimintaympäristö Dockerilla ja Kuberneteksellä niin, että asiakaan tuotantoympäristön hallinnoimisesta ja ylläpitämisestä tulisi helpompaa. Työn toimeksiantaja Pesmel Oy on kansainvälinen materiaalin käsittelyn asiantuntija ja toimittaa maailmanlaajuisesti automatisoituja käsittely-, pakkaus- ja varastojärjestelmiä teollisuuteen.

Teoriaosassa käsiteltiin työhön liittyviä tekniikoita ja sovelluksia, kuten palvelinten virtualisointia, palvelinklustereita, Docker-konttiteknologiaa ja Kubernetes-orkestrointisovellusta yleisellä tasolla. Lisäksi tarkasteltiin myös näiden keskeisempiä käsitteitä ja toimintaperiaatteita.

Työn käytännönosuuden alussa selvitettiin, mistä sovelluksista ja osa-alueista Pesmelin toimittamat verkkosovellusympäristöt todellisuudessa koostuivat. Työssä selvisi, että varastonhallintajärjestelmät ovat laajoja kokonaisuuksia, ja vaativat useamman sovelluksen sekä palvelun yhteistoimintaa. Selvitysten jälkeen työhön tarvittavat sovellukset asennettiin virtuaalikoneympäristöön ja tämän jälkeen aloitettiin tuotantoympäristön Docker-levykuvien määrittäminen. Käyttömahdollisuuksien selvitys Docker- ja Kubernetes-työkaluilla toteutettiin vaiheittain.

Lopputuloksena saatiin tuotantoympäristö määritettyä Docker-työkalulla, jossa kaikki palvelut toimivat ja kykenivät kommunikoimaan keskenään lähes moitteettomasti. Kubernetes-työkalulla haluttuun tavoitteeseen päästiin vain osittain, sillä tuotantoympäristön sovelluksia ei saatu kommunikoimaan keskenään käytetyssä ajassa.

Toimeksiantaja pystyy hyödyntämään tämän työn käytännön tutkimusosuutta arvioidessaan työkalujen soveltuvuutta tuotantoympäristönsä toteuttamiseksi. Pohdinnassa ja yhteenvedossa esitettiin työn aikana ilmenneitä havaintoja työkalujen soveltuvuudesta toimeksiantajan tuotantoympäristön kehittämiseksi.

[1] Asiasanat: Varastonhallintajärjestelmä, Docker, Kubernetes, sovellus, virtualisointi

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Degree programme: Automation Engineering

Specialisation: Machine Automation

Author: Joel Tikka

Title of thesis: Development of Warehouse Management System in Production Environment

Supervisor: Juha Yli-Hemminki

Year: 2022 Number of pages: 68

Docker and Kubernetes are two of the fastest developing modern technologies which blend in perfectly with each other's operating environments. The aim of the thesis was to gain hands on experience in software containerization by exploring open-source software development platforms from Docker and Kubernetes in a web application environment. Another aim was, to be able to use the gathered information in the development of the automated production environment of Pesmel Oy. The company specializes in industrial automated processing, packaging, and storage systems.

The thesis was started by finding information on the important concepts of server clusters, the virtualization of servers, and the basics of Docker containerization technology and Kubernetes orchestration tool. Then followed the practical part which handled software implementation. It was studied how the required configurations and installations had been made in the production environment. After that, the installation of the main functional components of Docker and Kubernetes was made. Next, Docker images were created, and the management control of containers was implemented with Docker. In the last phase of the practical part containers were attempted to operate with Kubernetes. The testing was done with the VMware virtual machine in Windows operating system.

As the result of the thesis an open-source container orchestration system was created, and information was collected for the development of the company's warehouse management system. The principal of the thesis will be able to utilize the practical part of the thesis when assessing the suitability of the tools for their production environment.

[1] Keywords: Docker, Kubernetes, open source, containerization, implementation

SISÄLTÖ

Kuva- ja kuvioluettelo

Käytetyt termit ja lyhenteet

API	Ohjelmointirajapinta (Application Programming Interface) on rajapinta, jossa eri ohjelmat ja sovellukset voivat tehdä pyyntöjä ja vaihtaa tietoja keskenään. Esimerkiksi ohjelmointirajapinta mahdollistaa käyttöjärjestelmän ja sovelluksen välisen tiedon siirtämisen automaattisesti.
CPU	Suoritin tai prosessori (Central Processing Unit) on tietokoneen osa, joka suorittaa tietokoneohjelman sisältämiä konekielisiä
I/O	Siirräntä (Input / Output) tarkoittaa tietokonelaitteen ja toisen tietokonelaitteen tai ulkoisen maailman välillä tapahtuvaa tiedonsiirtoa.
Kernel	Kernel on tietokoneohjelma tietokoneen käyttöjärjestelmän (OS) ytimessä ja tarjoaa peruspalvelut kaikille muille käyttöjärjestelmän osille. Se toimii käyttöjärjestelmän ja laitteiston välillä, sekä auttaa prosessien ja muistin hallinnassa, tiedostojärjestelmissä, laiteohjauksessa ja verkkotoiminnassa.
Käynnistystiedosto-järjestelmä	Käynnistystiedostojärjestelmä (boot filesystem) tallentaa käynnistysjärjestelmäprosessin edellyttämät tiedostot. Sitä käytetään UnixWare-käyttöjärjestelmässä. Se ei tue hakemistoja ja sallii vain vierekkäisen allokoinnin tiedostoille, jotta käynnistys olisi yksinkertaisempaa.
OS	Tietokoneen käyttöjärjestelmä (Operating System, OS) on järjestelmäohjelmisto, joka suorittaa kaikki perustehtävät, kuten tiedostonhallinnan, muistinhallinnan, prosessinhallinnan, syötteiden ja tulosteiden käsittelyn sekä oheislaitteiden, kuten levyasemien ja tulostimien ohjauksen.
PaaS	Platform as a Service. Sovellusalusta palveluna -mallissa käyttäjän ei tarvitse huolehtia käyttöjärjestelmän tai varusohjelmien ylläpidosta, vaan voi keskittyä sovellusten tekemiseen. Tällaisia tarjoavat

	esimerkiksi useat pilvipalvelut, kuten Microsoft Azure, Google Cloud ja Amazon Web Services (AWS)
Pistoke	Ohjelmointirajapinta (socket) tiedon lähettämiseen ja vastaanottamiseen päätepisteiden välillä joko verkossa tai prosessien välisessä kommunikaatiossa (IPC).
RAM	Keskusmuisti (Random Access Memory) on tietokoneen käyttömuisti. Se on käytännössä työmuisti, johon latautuvat käyttöjärjestelmän lisäksi suoritettavat sovellukset sekä näiden tarvitsemat tiedot.
REST API	Sovellusohjelmointirajapinta (API tai web API), joka noudattaa REST-arkkitehtonisen tyylin rajoituksia ja mahdollistaa vuorovaikutuksen RESTful-verkkopalveluiden kanssa. REST on lyhenne sanoista Representational State Transfer.
Websocket	Kaksisuuntainen tietokoneviestintäprotokolla, joka tarjoaa verkkoselaimelle yhden TCP-yhteyden kautta kanavan kaksisuuntaiseen viestintään, sekä mahdollistaa myös asynkronisen kommunikoinnin.

1 JOHDANTO

1.1 Yritysesittely

Työn toimeksiantajana toimi Pesmel Oy. Pesmel on kansainvälinen materiaalin käsittelyn asiantuntija, joka toimittaa automatisoituja käsittely- pakkaus ja varastojärjestelmiä teollisuuteen (Pesmel, i.a.). Pesmel perustettiin 1978 ja se työllistää maailmanlaajuisesti noin 250 työntekijää. Suomessa virallinen päätoimipaikka on Kauhajoki.

Markkinoilla olevia tuotteita ovat (Pesmel, i.a.) muun muassa:

- Varastonhallintajärjestelmät (WMS – Warehouse Management System)
- Automaattivarastoja paperi-, rengas-, metalli- ja autoteollisuuteen
- Paperi- ja metalliteollisuuden logistiikka- ja pakkausjärjestelmiä.

Pesmelin kilpailukyky perustuu innovatiivisiin ratkaisuihin sekä vahvaan insinööri- ja järjestelmäosaamiseen kansainvälisillä ja kotimaisilla markkinoilla (Pesmel, i.a.).

1.2 Työn tausta

Työn taustalla on tarve selvittää toimeksiantajan ICT-palveluiden tuotantoympäristön automatisointia. Ratkaisua haettiin palveluiden automaattiseen päivittämiseen, pysäyttämiseen ja ylläpitämiseen. Asiakasrajapinnassa toimivat palvelut olivat työläitä päivittää ja päivitykset tehtiin yleensä manuaalisesti. Lisäksi päivitysten tai muutosten osalta ei ollut jäljitettävyyttä. Asiakaan varastonhallintajärjestelmän palvelut saattoivat kaatua ilman erillistä ilmoitusta. Tieto ongelmista saatiin asiakkaan yhteydenoton yhteydessä. Myös tähän etsittiin ratkaisua ja yhtenä ratkaisuvaihtoehtona ajateltiin pakata palvelut Docker-kontteihin ja hallinnoida niitä Kubernetes-ohjelmistolla.

1.3 Työn tavoite

Tavoitteena on selvittää, voiko Pesmelin tarjoaman varastonhallintajärjestelmän automatisoitua toimintaympäristöä toteuttaa Dockerilla ja Kuberneteksellä niin, että asiakaan tuotantoympäristön hallinnointi ja ylläpitäminen ei vaikeutuisi, vaan siitä tulisi helpompaa. Tarvittavien resurssien määrä ei tulisi kasvattaa, vaan pitää ne mahdollisimman samanlaisina tai jopa

vähentää niitä. Päivitys- ja sovellushistorian osalta tavoitteena on tutkia lokitietojen keräämisen toteutuksia.

1.4 Työn rakenne

Teoriaosassa käsitellään työhön liittyviä tekniikoita ja sovelluksia, kuten virtualisointia, klustereita, Dockeria ja Kubernetestä. Virtualisointia ja klustereita käsitellään luvuissa 2 ja 3 yleisellä tasolla, sekä tarkastellaan myös niiden keskeisempiä käsitteitä ja toimintaperiaatteita. Luvussa 5 tarkastellaan Docker-sovelluksen historiaa ja toiminnallisuutta sekä sovelluksien keskeisimpiä toimintoja, tehtäviä ja vastuualueita. Viimeisessä teoriaosan luvussa 6 käsitellään Kubernetes-konttiorkestrointisovelluksen toiminnallisuutta ja paneudutaan sen historiaan, sekä käsitellään Kubernetekseen liittyviä komponentteja ja minikube-ympäristöä.

Tämän jälkeen siirrytään käytännönosan alkuselvitykseen, jossa käydään työn tavoitteet tarkemmin läpi ja selvitetään nykyisen tuotantoympäristön tilanne, käytettävät sovellukset ja palvelut. Seuraavaksi käsitellään tutkimus- ja selvitystyössä tarvittavien sovelluksien asennusta ja tarvittavien Docker-levykuvien määrittämistä. Luvussa 9 kerrotaan itse tutkimustyön osuudesta eli siitä, kuinka käyttömahdollisuuksia testattiin ja tutkittiin. Lopuksi on tulosten käsittely, yhteenveto ja pohdintaa käyttömahdollisuuksista sekä jatkokehitysideoita tälle työlle.

2 VIRTUALISOINTI

Aikaisemmin organisaatiot käyttivät fyysisiä palvelimia sovellusten suorittamiseen (Kubernetes, 2021-a). Fyysisten palvelimien sovelluksille ei voitu määrittää resurssirajoja, mikä aiheutti ongelmia resurssien allokoinnissa. Saattoi esimerkiksi olla tapauksia, joissa yksi sovellus vei suurimman osan fyysisen palvelimen resursseista, jolloin muut sovellukset eivät toimineet oletetusti. Ongelma yritettiin ratkaista suorittamalla jokainen sovellus omalla fyysisellä palvelimellaan (Kubernetes, 2021-a). Ratkaisu ei kuitenkaan skaalautunut, koska resurssit olivat vajaakäytössä ja organisaatioille oli kallista ylläpitää monia fyysisiä palvelimia. Siksi kehitettiin virtualisointi, joka mahdollistaa useiden virtuaalikoneiden suorittamisen yhdellä fyysisellä palvelimella (Kubernetes, 2021-a). Virtualisoinnin tekniikoita on kehitetty jo 1960-luvulta lähtien. Esimerkiksi IBM:n CP-40-laitteen käyttöjärjestelmässä voitiin jokaiselle käyttäjälle luoda oma virtuaalinen tietokone (Heino, 2010, s. 59–60).

Virtualisoinnin tarkoituksena on rakentaa tehtävien suorittamiseen eristetty virtuaalinen ympäristö isäntäkoneen tai palvelimen resursseja hyödyntäen tehtävien suorittamiseen (Heino, 2010, s. 60). Virtuaalisen ympäristön käyttöturvallisuus on fyysistä parempi, sillä jokainen käyttäjä työskentelee käyttäen täysin erillistä käyttöjärjestelmäänsä (Conroy, 2018). Käyttäjät eivät pysty kaatamaan koko järjestelmää, vaan pelkästään oman käyttöjärjestelmänsä. Lisäksi virtualisointi mahdollistaa sovellusten eristämisen virtuaalikoneiden välillä (Kubernetes, 2021-a).

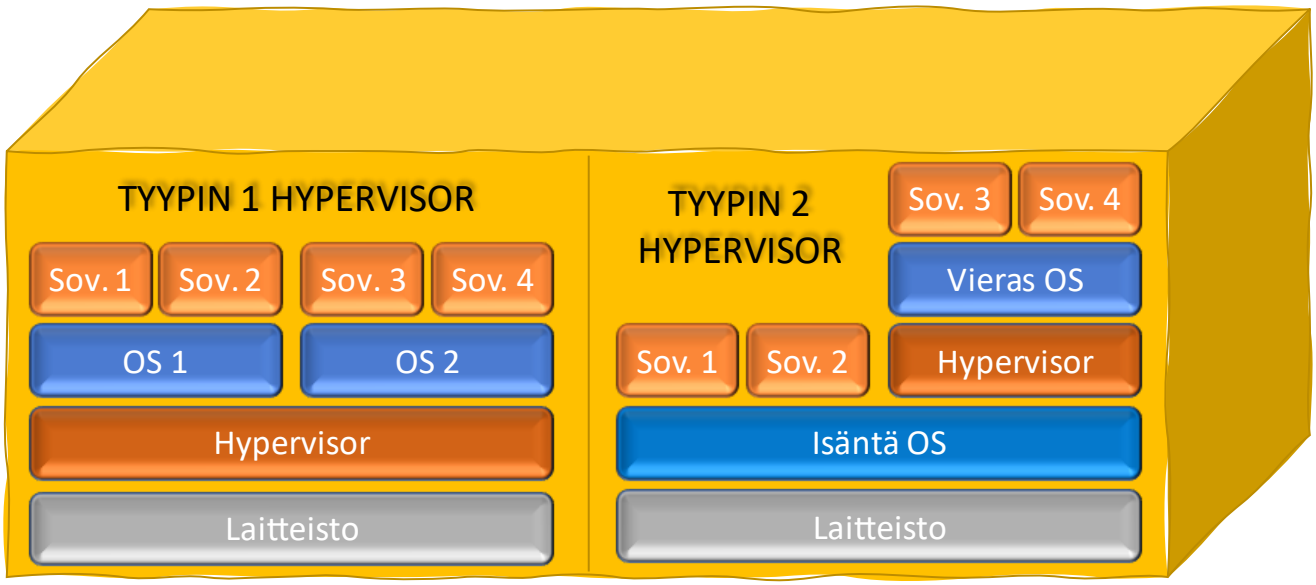
Heinon (2010, s. 59) mukaan virtualisoinnilla säästetään pääoma- ja ylläpitokustannuksissa. Vanhoista järjestelmistä luopuminen virtuaaliympäristöissä on helpompaa ja kustannustehokasta verrattuna fyysisiin laitteisiin. Virtualisointi voidaan tarpeen mukaan toteuttaa joko ohjelmisto- tai käyttöjärjestelmäpohjaisesti hypervisor-ohjelman päälle rakennettuna (mts. 61).

2.1 Hypervisor

Käyttöjärjestelmään perustuva virtualisointi voi olla tehokasta, sillä se voidaan toteuttaa ilman emulointia ja välikonepalveluita (Heino, 2010, s. 61). Hypervisor-ohjelmaa voidaan suorittaa käyttöjärjestelmän, kuten Windowsin päällä tai ilman, riippuen virtualisoinnin hypervisor-tyypistä (Conroy, 2018). Hypervisor siis jaetaan kahteen tyyppiin, joista ensimmäinen voidaan optimoida tehokkaammaksi eikä se vaadi isäntäkäyttöjärjestelmää. Hypervisorin

toimintaperiaatteena on toteuttaa ja simuloida sen päälle asennettavien virtuaalikoneiden palvelut (Heino, 2018, s. 60). Tämä järjestää palvelut ja toisaalta myös estää isäntäkäyttöjärjestelmää käyttämästä oheislaitteita, jotka on varattu virtuaaliselle käyttöjärjestelmäinstanssille.

Hypervisor-tason ja käyttöjärjestelmän väliin virtuaalikoneeseen asennetaan laiteajuri, jonka avulla saadaan tarvittavat resurssit ja oheislaitteet käyttöön (Heino, 2018, s. 60). Taso toimii näin mahdollistajana virtualisoinnille, eli useamman eri palvelimen tai käyttöjärjestelmän suorittamisen yhden fyysisen palvelimen resursseilla.



Kuvio 1. Hypervisor-tyyppien rakenteet (mukaillen Portnoy, 2016, s. 22–25).

Hypervisor-tyyppien keskeisimmät erot voidaan huomata kuviossa 1. Ensimmäisen tyypin sovellukset vaativat hypervisoria isäntäkäyttöjärjestelmältä, toisin kuin kakkostyypissä, jossa sovelluksia voidaan suorittaa joko hypervisorin kanssa tai ilman (Portnoy, 2016, s. 22–25). Lisäksi tyypin yksi virtualisointi toimii niin sanotusti yhdellä tasolla ja asennetaan suoraan laitteiston (hardware) päälle, josta tulee sen nimitys bare metal ja full virtualization (Bernstein, 2014, s. 83; Heino, 2018, s. 60–61). Täydessä virtualisoinnissa hypervisor kykenee suorittamaan virtuaalisen käyttöjärjestelmän täysin alkuperäisen kaltaisena (Heino, 2018, s. 61).

Tyypin 1 täysivirtuaalisointipalveluita (Heino, 2018, s. 61) tarjoavat:

- WMware vSphere
- Microsoft Hyper-V.

Näiden lisäksi tyypin 1 täysvirtualisointipalveluita löytyy muitakin, yhtenä esimerkkinä QEMU (Software Testing Help, 2022). Ja toisena Red Hat Enterprise Virtualization, RHEV (Actual-Tech Media, 2021).

Tyypin kaksi osittaisessa, eli paravirtualisoidussa (para virtualization), ympäristössä hypervisor tarjoaa virtuaalikoneelle API-rajapinnan laitteistopalveluiden toteuttamiseksi (Heino, 2018, s. 61). Tämän tyypin virtuaalista käyttöjärjestelmää täytyy muokata paravirtualisointia tukevaksi (mts. 61). Esimerkiksi ytimen virtualisoimattomat kutsut korvataan hyperkutsuilla, jotka kykenevät kommunikoimaan suoraan virtualisoitavan hypervisor-tason kanssa (VMware Inc, 2008, s. 5).

Tyypin 2 paravirtualisointipalveluita (Heino, 2018, s. 61) tarjoavat:

- Xen
- KVM.

Lisäksi tyypin 2 paravirtualisointipalveluiden joukkoon kuuluu olennaisesti myös Oracle VM VirtualBox (Software Testing Help, 2022). Hypervisorit soveltuvat myös kuormantasaajiksi yllättävien vikatilanteiden tai huoltotoimenpiteiden sattuessa (Heino, 2018, s. 63). Lisäksi tarpeettomia isäntäkoneita voidaan ajaa hallitusti alas.

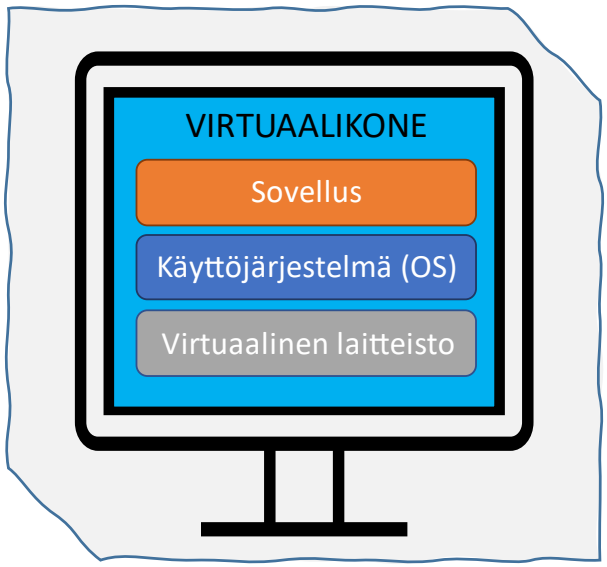
2.2 Virtuaalikone VM

Virtuaalikoneet (Virtual Machine) ovat virtualisoinnin peruskomponentteja. Ne ovat säiliöitä perinteisille käyttöjärjestelmille ja sovelluksille, jotka toimivat fyysisen palvelimen hypervisorin päällä (Portnoy, 2016, s. 37). Virtuaalikoneen sisällä asiat muistuttavat hyvin paljon fyysistä palvelinta ja varsinaisen palvelimen tavoin virtuaalikone tukee käyttöjärjestelmää. Niille voidaan myös määrittää resursseja, joihin virtuaalikoneessa toimivat sovellukset voivat pyytää pääsyä (mts. 37).

Useita virtuaalikoneita voi toimia samanaikaisesti yhdessä fyysisessä palvelimessa, ja nämä virtuaalikoneet voivat myös käyttää monia erilaisia käyttöjärjestelmiä, jotka tukevat monia erilaisia sovelluksia (Portnoy, 2016, s. 37). Fyysisillä palvelimilla asia on toisin, sillä käyttöjärjestelmien määrä on rajattu yhteen ja sovelluksiakin on rajallisesti käytettävissä.

Virtuaalikone on todellisuudessa vain joukko tiedostoja, jotka muodostavat virtuaalipalvelimen (Portnoy, 2016, s. 37). Tärkeimmät virtuaalikoneen muodostavat tiedostot ovat määritystiedosto ja virtuaalilevytiedostot. Määritystiedosto kuvaa virtuaalilaitteiston käyttämiä resursseja luettelomaisesti (mts. 37).

Virtuaalikoneesta on kaksi erilaista näkymää: toinen sisältä ja toinen ulkopuolelta (Portnoy, 2016, s. 38). Ulkopuolelta nähdään isäntäpalvelimen kokoonpano ja rakenne. Näkymä virtuaalikoneen sisältä on samanlainen kuin fyysisen koneen sisällä. Käyttöjärjestelmän tai sovelluksen näkökulmasta on vaikea havaita eroa fyysisen ja virtuaalisen ympäristön välillä, koska näkymä, tallennus, muisti, verkko ja käsittely ovat samanlaisia molemmissa (mts. 39).



Kuvio 2. Yksinkertaistettu virtuaalikoneen näkymä (mukaillen Portnoy, 2016, s. 38).

Kuviossa 2 nähdään yksinkertaisen virtuaalikoneen rakenne. Virtuaalikoneet ovat täysin eristettyjä, joten ne ovat erittäin turvallisia (Krochmalski, 2016, s. 8). Virtuaalikoneilla on myös haittoja. Nimittäin ne tarvitsevat kaikki samat ominaisuudet käyttöönsä kuin käyttöjärjestelmätkin, eli laiteohjaimet, ydinjärjestelmäkirjastot, tiedostot ja niin edelleen. Ne ovat hyvin raskaita suorittaa ja vievät isäntäkäyttöjärjestelmältä paljon laskentaresursseja. Näin ollen virtuaalikoneiden määrä rajautuu luonnollisesti vain muutamaan per laite, koska isäntäkoneen suorituskyky heikkenee ja jaettava resurssienkapasiteetti pienenee jokaisen virtuaalikoneen käynnistämisen myötä. Lisäksi ne eivät ole yksinkertaisia asentaa, vaan vaativat täydellisen asennuksen. Myös sovelluksen suorittaminen virtuaalikoneella vaatii isäntäkäyttöjärjestelmältä toimivan hypervisorin.

3 PALVELINKLUSTERI

Fyysiset palvelimet saattavat vikaantua käytettäessä, ja siksi näille kehitetään kaupallisesti vikasietoisempia ratkaisuja ohjelmistojen ja laitteistojen osalta (Portnoy, 2016, s. 246). Vikasietoisuuden saavuttamiseksi ja palvelinkatkojen välttämiseksi käytetään redundanttisia järjestelmiä. Redundanttisissa järjestelmissä yksittäisen komponentin vika ei kaada palvelinta tai sen tukemaa sovellusta. Alun perin 1980-luvulla nämä järjestelmät olivat tarkoitettu organisaatioille, jotka käyttivät hätäpalveluita tai lennonohjausjärjestelmiä. Pian yritykset, joilla oli kriittisiä transaktioikkunoita, alkoivat myös käyttää niitä. Tällaisia olivat esimerkiksi rahoituspalveluyritykset, joukkoliikennepalvelut ja ostoskanavat. Näiden yritysten seisokit tai palvelukatkot johtivat väistämättä merkittäviin taloudellisiin tappioihin. Järjestelmät ovat edelleen olemassa, mutta niiden tilalle on ilmaantunut parempia ratkaisuja. Yksi näistä ratkaisuista on klusterointi.

Klusteri syntyy yhdistämällä kaksi palvelinta tai vaihtoehtoisesti useampia palvelimia yhteen fyysiseen verkkoon (Portnoy, 2016, s. 246). Palvelimet jakavat yhteiset tallennusresurssit ja klusteriohjelmiston. Klusteriohjelmisto reitittää sovellusliikennettä palvelimien välillä. Kun esimerkiksi ensisijainen palvelin epäonnistuu käsittelyssä, ohjelmisto reitittää sovellusliikenteen toissijaiselle palvelimelle ja näin käsittely voi jatkua. Klusterin avulla sovellus voi toipua palvelinkatkoksesta nopeammin. Klusteriohjelmisto saa palvelimet näyttämään yhdeltä yksittäiseltä resurssilta, mutta sen hallinta on todellisuudessa monimutkaista ja vaatii erikoisosaamista sovelluksien määrittämisessä. Klustereita voidaan käyttää myös virtuaaliympäristön sisällä tai klusteroida virtuaalikoneesta toiseen. Lisäksi voidaan klusteroida fyysisestä koneesta virtuaalikoneeseen.

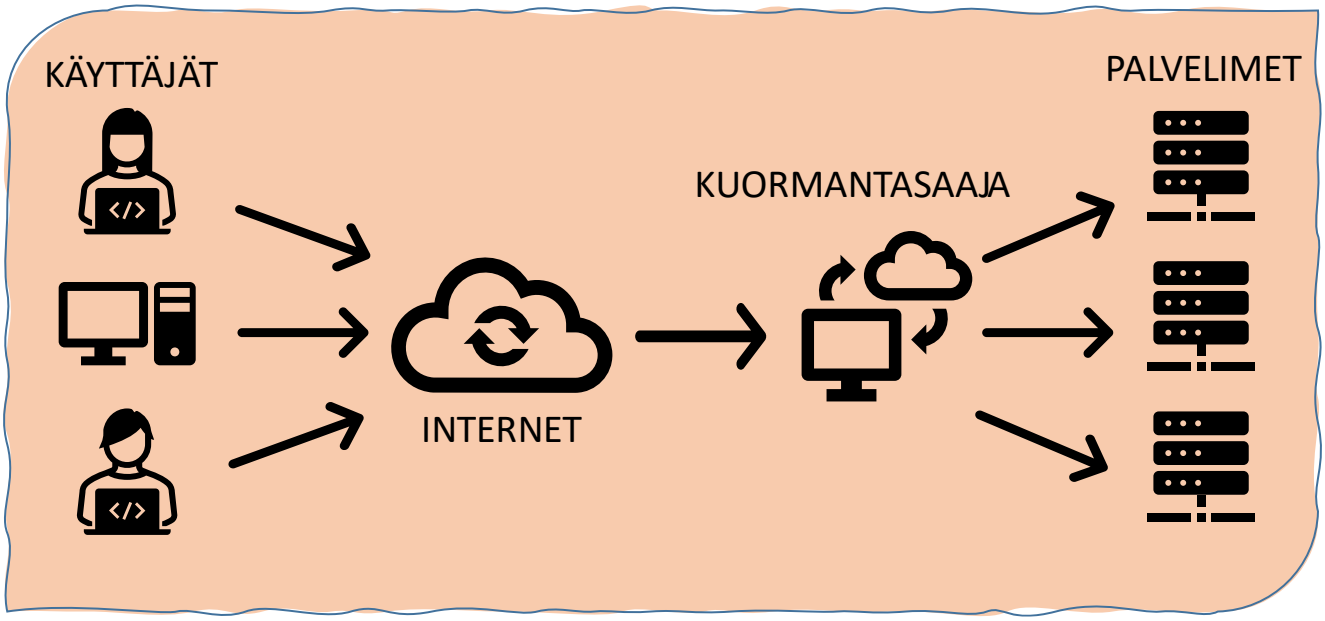
Esimerkkejä klusterointi sovelluksista (Portnoy, 2016, s. 246) ovat:

- Microsoft Cluster Service (MSCS)
- Oracle Real Applications Clusters (Oracle RAC).

3.1 Kuormantasaus

Sovelluksien suorituskykyvaatimukset elävät jatkuvassa muutoksessa. Osan sovelluksista on tuettava useita käyttäjiä samanaikaisesti (Portnoy, 2016, s. 263). Siksi ne suorittavat useita kopioita verkkopalvelimesta tai sovelluspalvelintasosta. Perinteisesti tämän kaltaisia

sovelluksia on fyysisesti saatettu toteuttaa varustamalla palvelimet kymmenillä eri korttipalvelimilla. Kuormantasaajat sijoitetaan (kuvio 3) tasojen väliin tasaamaan liikennevirtaa ja ohjaamaan se uudelleen verkkopalvelimen tai sovelluspalvelimen vian sattuessa. Virtuaaliympäristössä sama voidaan toteuttaa ottamalla kuormanjako käyttöön omana virtuaalikoneena. Kasvavaa kuormitusta voidaan virtuaaliympäristössä helpottaa kloonaamalla uusia koneita ja ottamalla ne nopeasti käyttöön.



Kuvio 3. Kuormantasaajan toimintaympäristö (mukaillein Portnoy, 2016, s. 261–262).

Kuormien jakaminen on valtava voimavara muistiresurssien säästämisessä, kun isäntäkäyttöjärjestelmässä on useita kloonattuja virtuaalikoneita (Portnoy, 2016, s. 263). Resurssikiistan ilmentyessä voidaan virtuaalikoneita automaattisesti siirtää palvelimelta toiselle kaikkien fyysisten resurssien parhaan käytön takaamiseksi. Palvelinvian sattuessa verkkopalvelimen ja sovelluspalvelimen lisäkopiot muissa virtualisointikoneissa pitävät sovelluksen saatavilla. Kaatuneet virtuaalikoneet palautetaan ja siirretään muualle klusteriin. Tämä takaa sovelluksille korkean käytettävyyden.

4 DOCKER-KONTTITEKNOLOGIA

Docker esiteltiin ensimmäisen kerran maailmalle Kaliforniassa, 15. päivä maaliskuuta vuonna 2013 (Kane & Matthias, 2015, s. 1). Kane ja Matthiaksen (2015, s. 1) mukaan esittelijänä toimi dotCloudin toimitusjohtaja ja ohjelman kehittäjä Solomon Hykes. Ainoastaan viiden minuutin pituinen hissipuhe pidettiin konferenssissa, joka oli suunnattu Python-koodikielen kehittäjille. Noihin aikoihin vain neljäkymmentä ulkopuolista oli päässyt kokeilemaan ohjelmaa.

Muutaman viikon kuluessa julkistamisesta ohjelma oli kerännyt huomattavan suuren yleisön sekä median huomion (Kane & Matthias, 2015, s. 1). Tuolloin yksityisessä projektissa kehitetyn ohjelman lähdekoodi muutettiin nopeasti avoimeksi ja laitettiin julkiseen jakoon GitHubin kautta, josta kuka tahansa pystyi sen lataamaan ja osallistumaan mukaan projektiin.

Kuukausien kuluessa yhä useammat kuulivat tästä vallankumouksellisella tavalla rakennetusta ja tuotetusta ohjelmasta, jonka avulla pystyi luomaan helposti jaettavan kontin mille tahansa sovellukselle (Kane & Matthias, 2015, s. 1). Ohjelman luvattiin olevan työkalu organisaatioiden työnkulun ja reagointikyvyn virtaviivaistamiseen. Niinpä vuoden kuluessa melkein kaikki ohjelmistoalalla työskentelevät tiesivät Dockerin, mutta eivät välttämättä tunteneet sen käyttötarkoitusta riittävän hyvin sen hyödyntämiseksi.

4.1 Konttisovelluksien historia

Docker ei ollut ensimmäinen, joka keksi hyödyntää konttiteknologiaa, vaan sen juuret ylettyvät aina vuoteen 1979 asti (Bernstein, 2014, s. 82). Tuolloin kehitettiin Unix-versiota 7, jonka osana otettiin käyttöön Unix chroot -komento. Vuonna 1998 FreeBSD toteutti laajennetun version chrootista ja sitä kutsuttiin nimellä jail. Vuonna 2004 suorituskykyä parannettiin ja markkinoille tuli Solaris 10 tuotteella zones. Myöhemmin Solaris 11 myötä, vyöhykkeisiin perustuva täysimittainen suorituskyky valmistui ja sitä kutsuttiin nimellä containers. Tuolloin myös patentoidut Unix-myyjät tarjosivat samanlaisia ominaisuuksia tuotteissaan, kuten esimerkiksi HP-UX-kontteja ja IBM AIX -työkuormaosioita.

Kun Linux nousi hallitsevaksi avoimeksi alustaksi, korvasi se kaikki aikaisemmat muunnelmat markkinoilla. Tekniikka nimeltä LXC (Linux Containers) löysi tiensä tuolloin alan standardiksi (Bernstein, 2014, s. 82). Perustusvaiheessa Docker laajensi LXC-konttiteknologian käyttöä lisäämällä ytimen (kernel) ja sovellustason sovellusliittymä (API) kontin sisälle. Yhdessä

nämä kaikki tasot suorittavat prosesseja eristettynä, kuten CPU, muisti, I/O ja verkkoprosessit.

4.2 Sovellus

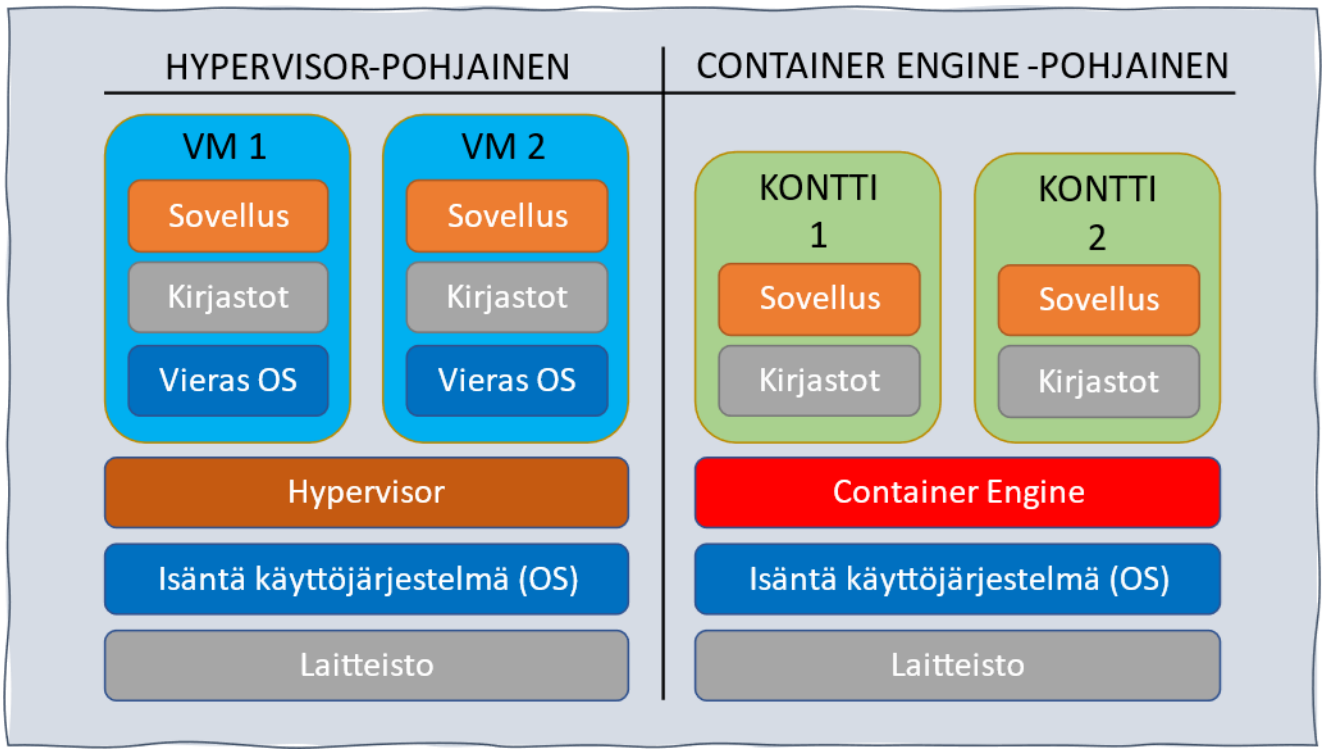
Docker on siis avoin standardialusta siirrettävien hajautettujen sovellusten kehittämiseen, pakkaamiseen ja käyttämiseen (Vohra, 2016, s. 1). Siirrettävä sovellus on ohjelmistotuote, joka on suunniteltu helposti siirrettäväksi tietokoneympäristöstä toiseen (TechTarget, 2012). Vohran (2016, s. 1) mukaan Dockeria käyttämällä kehittäjät ja järjestelmäadministraattorit voivat luoda, lähettää ja suorittaa sovelluksia millä tahansa alustalla, kuten tietokoneella, pilvipalvelussa, konesalissa tai virtuaalikoneella.

Kaikkien tarvittavien riippuvuuksien saaminen sovellukseen, kuten koodi, ajonaikaiset kirjastot ja järjestelmätyökalut, on usein haaste sovelluksen kehittämisessä ja käytössä (Vohra, 2016, s. 1). Docker yksinkertaistaa sovelluskehittämisen ja suorittamisen pakkaamalla kaikki ohjelmiston tarvitsemat sovellukset yhdeksi ohjelmistoyksiköksi, jota kutsutaan Docker-levykuvaksi (image), jota voidaan käyttää millä tahansa alustalla. Docker-levykuvassa kaikki virtuaalikoneen kuvat toimivat samassa virtuaalikoneen ytimessä.

4.3 Hyödyntäminen tuotantoympäristössä

Kuviossa 4 verrataan hypervisior- ja konttisovellusten käyttöönottoa. Hypervisor-pohjainen käyttöönotto on ihanteellinen silloin, kun samassa pilvessä tai laitteessa olevat sovellukset vaativat erilaisia käyttöjärjestelmiä (Bernstein, 2014, s. 82). Konteilla toteutetut käyttöönotot ovat huomattavasti pienempiä kuin hypervisorien mahdollistamat käyttöönotot.

Konttiteknologia mahdollistaa satojen säiliöiden tallentamisen fyysiselle isännälle ja konttien uudelleenkäynnistys ei vaadi isäntäkäyttöjärjestelmän uudelleenkäynnistämistä (Bernstein, 2014, s. 82). Kontti on käytännöllinen tapa säilöä sovellus sekä jakaa sen tiedostot ja kirjastot.



Kuvio 4. Hypervisorin ja konttitoteutuksen eroavaisuudet (mukaillen Bernstein, 2014, s. 82).

4.3.1 Tehtävienhallinta

Tehtävienhallintaa tarvitaan jokaisessa nykyaikaisessa modernissa toteutuksessa ja näin on myös Dockerin tapauksessa (Kane & Matthias, 2018, s. 216). Periaatteessa ei voi olla olemassa minkäänlaista järjestelmää ilman tehtävienhallintaa. Yleensä tämän hallinnan tekee käyttöjärjestelmä. Esimerkiksi tarkastellessa Linuxin init-järjestelmää, sisältää se seuraavia komentoja tehtäville, kuten ’upstart’, ’systemd’, ’System V init’ ja ’runit’.

Tehtävienhallinnassa käyttöjärjestelmälle kerrotaan prosessista, jonka halutaan olevan käynnissä, ja tämän jälkeen määritetään, miten sen tulisi käyttäytyä eri tilanteissa. Tilanteita voivat olla prosessin uudelleenkäynnistys, käynnistys, pysäyttäminen, sulkeminen, konfiguraatioiden uudelleen lataaminen ja elinkaaren hallinnoiminen (Kane & Matthias, 2018, s. 216). Tehtävienhallintaa tarvitaan myös erikoistilanteissa, kuten prosessin uudelleenkäynnistämisessä silloin kun sen havaitaan kaatuneen. Erilaiset sovellukset tarvitsevat erilaisia tehtävienhallintatyökaluja. Esimerkiksi Linuxia hallinnoidessa otetaan käyttöön cron, tehtävien aikataulutuksien hallinnoimiseksi.

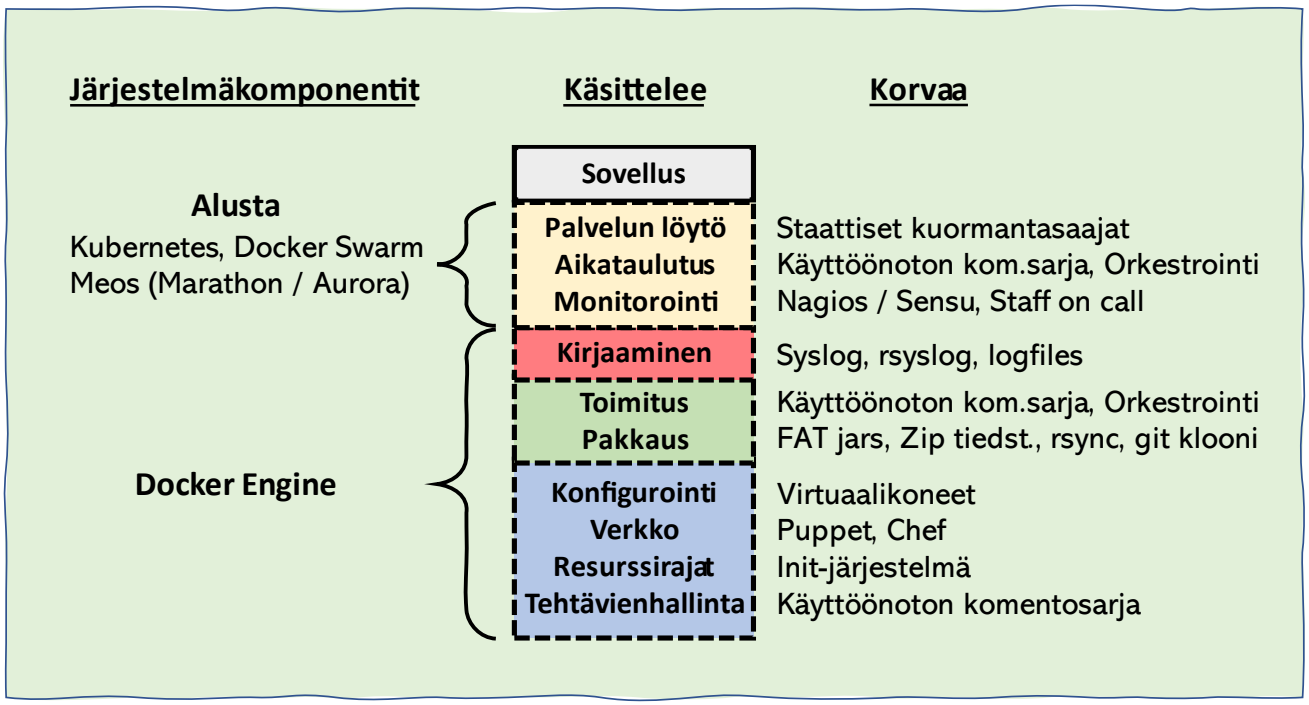
Konttitoteutuksessa tehdään samanlaista tehtävienhallintaa, mutta se tapahtuu pääsääntöisesti kontin ulkopuolelta (Kane & Matthias, 2018, s. 216). Tarvitaan kuitenkin enemmän

metatietoja, jotta saadaan prosessit tekemään oikeita asioita. Docker Engine tarjoaa joukon työkaluja tehtävienhallinnan tukemiseksi, kuten ’docker start’, ’docker stop’, ’docker run’ ja ’docker kill’.

4.3.2 Resurssien käyttö

Resurssirajat sijaitsevat kuvion 5 tuotantorakenteessa tehtävienhallinnan yläpuolella. Tuotantoympäristöt ovat hallinnoineet resursseja ennen kuin Docker popularisoi resurssien käytön (Kane & Matthias, 2018, s. 217). Perinteisesti käytettiin Linuxin ulimit-toimintoa. Lisäksi erityyppisiä asetuksia sovelluksien ajonaikaisessa ympäristössä toteutettiin mm. Java-, Ruby- tai Python-pohjaisilla virtuaalikoneilla. Yksi varhaisten pilvipohjaisten järjestelmien läpimurroista oli käyttää yksittäistä virtuaalista palvelinta rajoittamaan yhden yrityssovelluksen resurssien käyttöä, mikä saattaa nykyisellä mittapuulla kuulostaa aika vähäiseltä konttiteknologian rinnalla.

Dockerin avulla konteille voidaan helposti lisätä erilaisia työkaluja resurssien hallinnointiin (Kane & Matthias, 2018, s. 217). Sovelluksien muistia, tallennustilaa tai I/O-asetuksia voidaan vapaasti joko rajoittaa tai lisätä konttien tarpeiden mukaan. Tämä ydinominaisuus tekee konttiteknologiasta arvokkaan lisän tuotantoympäristöön. Lisäksi konttien ansiosta itse sovellus tarvitsee vähemmän resursseja (Wallenius, 2022-b).



Kuvio 5. Dockerin rooli tuotantoympäristössä (mukaillen Kane & Matthias, 2018, s. 215).

4.3.3 Konfigurointi

Kaikilla sovelluksilla on oltava jonkinlainen pääsy niiden määritystiedostoihin (Kane & Matthias, 2018, s. 217). On olemassa kaksi tasoa sovelluksen konfiguroinnissa. Alimmalla tasolla havaitaan, miten sovellus olettaa sitä ympäröivän ympäristön olevan määritettynä. Kontit käsittelevät tämän konfiguroinnin Dockerfilen avulla, joka mahdollistaa ympäristön rakentamisen joka kerta samalla tavalla. Perinteisimmissä järjestelmissä tulisi käyttää määrittämisen hallintajärjestelemiä, kuten Chef, Puppets tai Ansible, jotta saavutetaan sama lopputulos. Perinteisiä hallintajärjestelmiä voidaan käyttää myös konttiteknologian kanssa, mutta ei sovelluksien riippuvuussuhteiden hallinnoimiseksi. Se työ kuuluu Dockerin ja Dockerfilen tehtäväksi. Vaikka Dockerfilen sisältö on erilainen eri sovelluksilla, ovat työkalut ja mekanismit niiden taustalla samat.

Seuraava konfiguroinnin taso on konfiguroinnin soveltaminen suoraan sovellukseen (Kane & Matthias, 2018, s. 217). Dockerin natiivi mekanismi on käyttää ympäristömuuttujia ja tämä toteutuu myös kaikilla nykyaikaisilla alustoilla. Perinteisemmät järjestelmät luottavat määritystiedostoihin, joilla saattaa kuitenkin olla negatiivisia vaikutuksia sovelluksen havaittavuuteen ja käytettävyyteen. Ympäristömuuttujien avulla sovelluksen testaus helpottuu, kun muuttujia ei määritetä sovelluksen koodin sisälle valmiiksi, vaan ne voidaan tarvittaessa asettaa erilaisiksi joka kerta (mts. 338).

4.3.4 Monitorointi

Tuotantojärjestelmän tärkeimpiin vaatimuksiin kuuluu, että niiden toimintaa voidaan seurata ja monitoroida eri mittareilla (Kane & Matthias, 2018, s. 151). Tuotantojärjestelmä ei tule toimimaan hyvin pitkällä aikavälillä, mikäli toimintaa ei voida monitoroida tai seurata. Nykyaikaisissa toimintaympäristöissä kerätään paljon tietoa järjestelmistä, sekä tilastoja raportoidaan ja havainnollistetaan mahdollisimman paljon toimintaympäristön kehittämiseksi. Docker tukee konttien eheystarkastuksia ja joitakin alkeellisia monitorointiominaisuuksia, kuten ’docker stats’ sekä ’docker events’. Kuitenkin parempia monitorointiominaisuuksia saadaan käyttämällä kolmannen osapuolen ohjelmia, kuten Prometheus-monitorointijärjestelmää.

4.3.5 Tehtävien aikataulutus

Kontit ovat siirrettävissä, koska Docker tarjoaa hyvät työkalut siihen. Se avaa paljon mahdollisuuksia resurssien parempaan käyttöön, luotettavuuteen, itsestään toipuviin palveluihin ja dynaamiseen skaalautuvuuteen (Kane & Matthias, 2018, s. 219). Vanhemmissa järjestelmissä palveluita käsiteltiin usein erillisillä palvelimilla ja jokaisella palvelimella oli parinaan vain yksi palvelu. Docker tarjoaa palveluiden hallintaan huomattavasti tätä kehittyneemmän vaihtoehdon.

Hajautetut tehtävien aikataulutuspalvelut Dockerin avulla saavat koko palvelinverkoston vaikuttamaan siltä, kuin se olisi vain yksi tietokone (Kane & Matthias, 2018, s. 219). Ajatuksena oli määrittää sovellukselle halutut toiminnot sekä käytännöt eri tilanteisiin. Järjestelmän tehtäväksi jää selvittää, missä sovellus suoritetaan ja miten monta eri kopiota sovelluksesta ajetaan samanaikaisesti. Ajoituspalvelut hallitsevat myös sovelluksien ohjaamista ja uudelleenkäynnistymistä virhetilanteissa. Tällöin siirretään sovellukset toimimaan sellaisilla resursseilla, jotka ovat eheitä. Vanhan ja uuden sukupolven sovellukset voidaan käynnistää myös rinnakkain, ja siirtää hallitusti vanhan sovelluksen tehtäviä uudelle sovellukselle. Tällä tavalla saavutetaan käyttöönottotilanteessa mahdollisimman vähäiset seisokkiajat.

4.4 Palvelu ja asiakasohjelma

Palvelimella ajettavia kontteja suoritetaan ohjelmalla nimeltä Docker Engine (Wallenius, 2022-a). Docker Engine on rakennettu Linux-ytimen päälle ja se hyödyntää sen

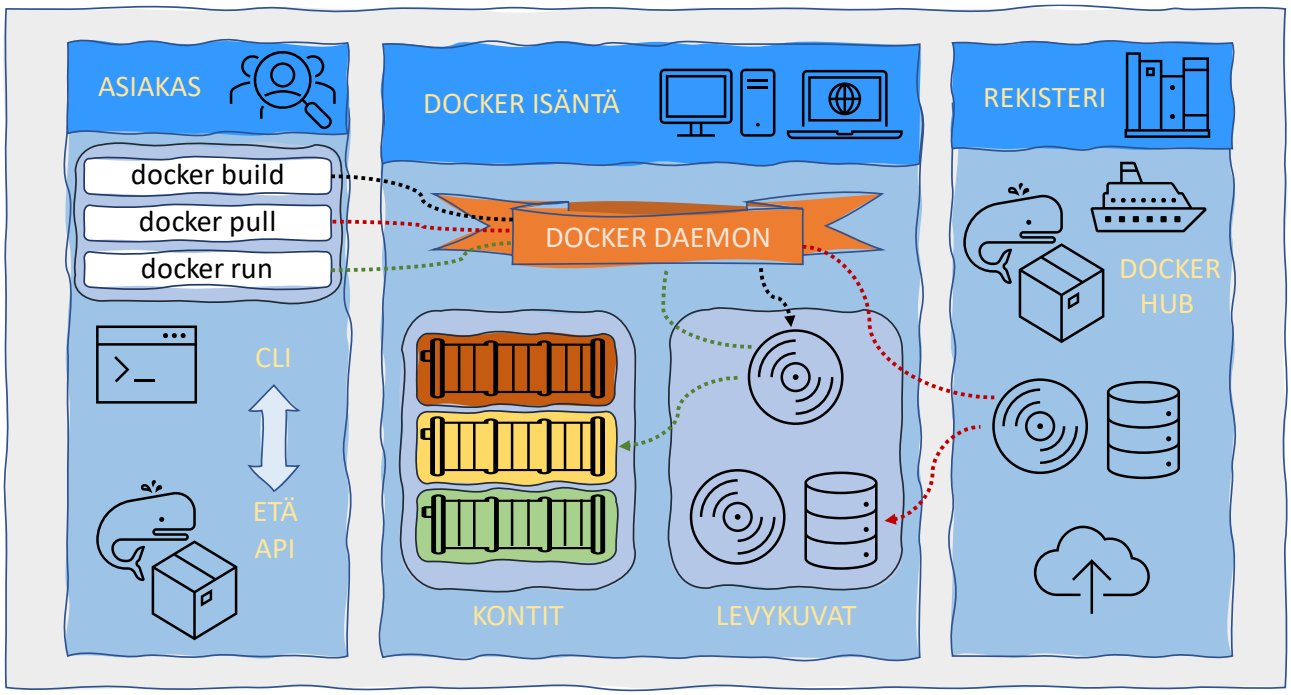
ominaisuuksia laajasti (Pethuru, 2005, s. 7). Sen vuoksi sitä voidaan käyttää suoraan Linux-käyttöjärjestelmän yhteydessä. Docker Engine -ohjelmaa voidaan käyttää myös Mac- ja Windows-käyttöjärjestelmissä sovittimen välityksellä.

Engineä hallitaan Docker API -rajapintojen kautta ja ne tarjoavat vuorovaikutuskanavan Docker daemonin kanssa (Wallenius, 2022-a; Docker Inc, i.a.-a). Docker daemon kuuntelee Docker API -rajapinnan pyyntöjä ja hallitsee Docker-objekteja, kuten levykuvia (image), konteja, levyasemia (volume) ja verkkoja (Docker Inc, i.a.-b). Daemon voi myös kommunikoida muiden taustaprosessien kanssa hallitakseen Docker-palveluita (Docker Inc, i.a.-b). Docker Engine API on RESTful API, jota käyttää HTTP-asiakas (client), kuten wget tai curl. Vaihtoehtoisesti voidaan käyttää myös HTTP-kirjastoa (Docker Inc, i.a.-a). Enginen avulla voidaan pakata sovellusten tarvitsemia kokonaisuuksia ja suorittaa niitä. Pakattuja kokonaisuuksia kutsutaan nimellä levykuva (Wallenius, 2022-a).

Docker-asiakas (Command line interface, CLI) käyttää API-rajapintoja kommunikoimaan Docker Hubin ja Docker-rekisterien (registry) kanssa (Khare, 2015, s. 133). Näiden API-rajapintojen lisäksi Dockerilla on omat sidokset eri ohjelmointikielille. API-rajapintojen tuntemusta tarvitaan esimerkiksi konttien hallintaan tai hyvän käyttöliittymän (GUI) rakentamiseen Docker-levykuville.

4.5 Arkkitehtuuri

Docker käyttää asiakas-palvelin-arkkitehtuuria (Docker Inc, i.a.-b). Kuviossa 6 havainnollistetaan asiakas-palvelin-arkkitehtuurin rakennetta ja toimintaympäristön sovelluksia. Asiakasohjelma keskustelee Docker-daemonin kanssa, joka suorittaa konttien rakentamisen, käytön ja jakelun. Docker-asiakas ja daemon voivat toimia samassa järjestelmässä. Vaihtoehtoisesti voidaan yhdistää asiakas Docker-etä-daemoniin. Docker-asiakas ja -daemon kommunikoivat REST API -rajapinnan avulla, UNIX-pistokkeiden (socket) tai verkkoliitännän kautta. Toinen Docker-asiakasohjelma on Docker Compose. Sen avulla voidaan työskennellä sovellusten kanssa, jotka koostuvat konteista.



Kuvio 6. Dockerin arkkitehtuuri (mukaillen Docker Inc, i.a.-b).

4.6 Docker image

Docker image eli levykuva koostuu tiedostojärjestelmistä, jotka on kerrostettu toistensa päälle (Turnbull, 2019, s. 2). Image on lukutiedosto, joka sisältää ohjeet sovellukseen tarvittavista ohjelmistoista ja käyttöjärjestelmästä Docker-kontin luomiseksi. (Wallenius, 2022-b; Docker Inc, i.a.-b). Image on käytännössä itsenäinen ja siirrettävä kokonaisuus (Wallenius, 2022-b). Tarkastellessa esimerkiksi Windowsin pohjalta rakennettua imagea, sisältäisi se esimerkiksi seuraavat komponentit: Windows Server Core -käyttöjärjestelmän, IIS-verkkosovelluspalvelimen, ASP.NET-komponentit, ajettavan sovelluksen ja sen tarvitsemat konfiguraatiot pakattuna yhteen tiedostoon. Tällaisella imagella voitaisiin käyttää verkkosovellusta, sillä se sisältää kaiken tarvittavan sovelluksen suorittamiseksi.

Dockerilla voidaan luoda omia levykuvia tai vaihtoehtoisesti käyttää muiden luomia, rekisterissä julkaistuja levykuvia (Docker Inc, i.a.-b). Levykuvia luodaan määrittämällä Dockerfile-tiedosto ja siihen kirjoitetaan suorittamiseen tarvittavat ohjeet. Dockerfile-tiedostoa muutettaessa tulee levykuva rakentaa uudelleen tietojen päivittymiseksi. Vain muuttuneet tasot rakennetaan uudelleen, mikä tekee levykuvista kevyitä, pieniä ja nopeita verrattuna muihin virtualisointiteknologioihin.

Kun suoritetaan levykuvia Dockerilla, jokainen suoritettu komento muodostaa sovelluspinossa uuden kerroksen edellisen päälle (Bernstein, 2014, s. 82). Komennot voidaan suorittaa manuaalisesti tai automaattisesti käyttämällä apuna edellä mainittua Dockerfile-tiedostoa (mts. 82). Dockerfile-tiedosto on automaattisesti suoritettava komentosarja uuden levykuvan luomiseksi, tiedosto koostuu eri komennoista ja argumenteista lueteltuna peräkkäin (mts. 83). Niitä käytetään järjestämään käyttöönottoartefakteja ja yksinkertaimaistamaan käyttöönottoprosessia alusta loppuun saakka (mts. 83).

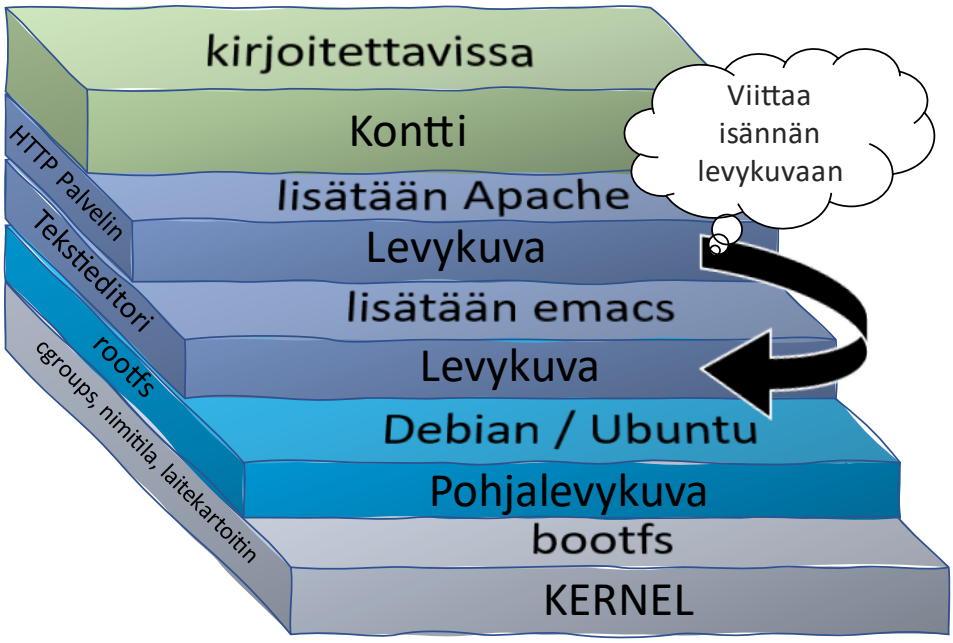
4.6.1 Rakenne

Tarkastellessa levykuvien rakennetta (kuvio 7) huomataan, että kaiken perusta on käynnistystiedostojärjestelmä bootfs, joka muistuttaa tyypillistä Linux/Unix- käynnistystiedostojärjestelmää (Turnbull, 2019, s. 2). Käyttäjä ei kuitenkaan ole suoraan vuorovaikutuksessa käynnistystiedostojärjestelmän kanssa. Kontin käynnistyttyä siirretään bootfs muistiin ja vapautetaan initrd-levykuvan käyttämää RAM-muistia muuhun käyttöön.

Seuraava kerros on rootfs-juuritiedostojärjestelmä ja se voi käyttää yhtä tai useampaa järjestelmää, kuten Debian- tai Ubuntu-tiedostojärjestelmiä (Turnbull, 2019, s. 2). Docker hyödyntää juuritiedostojärjestelmää vain lukutilassa (read-only), toisin kuin Linux-käyttöjärjestelmässä voidaan käynnistyksen jälkeen myös kirjoittaa (read-write) tiedostojärjestelmään.

Docker hyödyntää union mount -tiedostojen käyttöönottotapaa tiedostojärjestelmässään (Turnbull, 2019, s. 2). Tässä käyttöönottotavassa sulautetaan kaikki tiedostojärjestelmät päällekkäin niin, että ne näyttävät yhdeltä kokonaiselta tiedostojärjestelmän hakemistolta. Todellisuudessa se kuitenkin sisältää useampia tiedostoja ja alihakemistoja kaikista taustalla olevista tiedostojärjestelmistä.

Docker kutsuu kutakin tiedostojärjestelmänkuvaa erikseen ja ne voidaan asetella kerroksittain toistensa päälle (Turnbull, 2019, s. 2). Edellistä kerrosta alempana oleva tiedostojärjestelmänkuvaa kutsutaan vanhemmaksi (parent image) ja pinon pohjimmaista kutsutaan peruskuvaksi (base image). Docker kiinnittää luku-kirjoitus-tiedostojärjestelmätason (read-write) kaikkien edellä mainittujen kerroksien päälle. Haluttu prosessi tai sovellus suoritetaan tässä tiedostojärjestelmän osassa.



Kuvio 7. Docker-levykuvan rakenne (mukaillen Turnbull, 2019, s. 3).

4.7 Docker-kontti

Kontti (container) on levykuvasta muodostettu suoritettava instanssi (Docker Inc, i.a.-b). Tarkemmin sanottuna Docker rakentaa kontin levykuvan järjestelmätiedostokuvien pinosta (Turnbull, 2019, s. 3). Kontin ensimmäisen kerran käynnistyessä ylimmäinen luku-kirjoitustaso on tyhjä, ja muutokset kontissa kohdistuvat tähän tasoon. Käytännössä siis kopioidaan alemmasta vain luku -tasosta (read-only) tiedot ylempään lukukerrokseen. Kopioitava vain lukutaso ei kuitenkaan katoa kokonaan, vaan tämä toiminto piilottaa alla olevan vain lukutason kopion alle. Tätä ominaisuutta kutsutaan perinteisesti nimellä kopiointi kirjoituksessa (copy on write), ja se on yksi Dockerin vahvuustekijä.

Rakenteen avulla levykuvien rakentamiseen ja sovelluksien tai palveluiden käynnistämiseen vaadittu aika lyhenee (Turnbull, 2019, s. 3). Rakennettuja kontteja voidaan myös pysäyttää ja muokata nopeasti. Konttien yleistä turvallisuutta voidaan helposti lisätä skannaamalla ne tunnettujen haavoittuvuuksien ja altistumisten varalta (Schenker, 2018, s. 11).

Docker hyödyntää konteissa Linux-ytimen (kernel) mekanismeja, kuten iptables, virtuaalinen silta ja erilaisia tiedostojärjestelmän ohjaimia (Schenker, 2018, s. 11; Kane & Matthias, 2018, s. 17). Kontit käyttävät myös Linux-turvallisuusprimitiivejä, kuten Linux-ytimen nimitiloja (namespaces) hiekkalaatikkoihin (sandbox), kun eri sovellukset toimivat samassa tietokoneessa (Schenker, 2018, s. 11).

Linuxiin pohjautuvat kontit käyttävät myös yhtenä turvallisuusprimitiivinä ohjausryhmiä (cgroups) (Pethuru, 2015, s. 180). Ohjausryhmät eivät vain seuraa prosessiryhmiä, vaan mahdollistavat muistin, CPU:in ja I/O-resurssien rajoittamisen ja priorisoinnin (mts. 6, 180). Ohjausryhmien tehtävänä on varmistaa, että jokainen Docker-kontti saa kiinteän määrän muistia, CPU:ta ja levy I/O:ta (mts. 180). Näin toimimalla kontit eivät pysty sammuttamaan isäntäkonetta missään olosuhteissa. Ohjausryhmät ovat välttämättömiä joidenkin palvelinestohyökkäyksien torjumiseksi. Lisäksi cgroups-hallintaa käytetään välttämään meluisa naapuriongelma, jossa yksi sovellus käyttää kaikki palvelimen käytettävissä olevat resurssit, ja näännyttää muut sovellukset (Schenker, 2018, s. 11).

4.8 Docker-rekisteri

Rekisteri (registry) on avoimeen lähdekoodiin perustuva ja erittäin skaalautuva palvelinpuolen sovellus, joka tallentaa ja jakaa Docker-levykuvia (Docker Inc, i.a.-c). Rekisterit voivat olla julkisia tai yksityisiä säilytyspaikkoja (Wallenius, i.a.-a). Tunnetuin julkinen rekisteri on Docker Hub, jota kuka tahansa voi käyttää (Wallenius, i.a.-a; Docker Inc, i.a.-b). Docker on määritetty oletuksena etsimään levykuvia Docker Hubista (Docker Inc, i.a.-b).

Myös Docker Hubissa voidaan ylläpitää yksityistä tai julkista rekisteriä (Docker Inc, i.a.-b). Palvelu tarjoaa käyttäjille huoletonta ja käyttövalmista ratkaisua levykuvien tallennukseen ja jakeluun (Docker Inc, i.a.-c). Rekisterin tallennuspaikkaa ja jakelukanavaa voidaan hallita tarpeiden mukaan (Docker Inc, i.a.-c). Organisaatioille on saatavilla lisäominaisuuksia, jotka sisältävät muun muassa automaattisen koontityökalun (Docker Inc, i.a.-c). Docker Hub -rekisteriin ei tällä hetkellä ole saatavilla käyttöliittymää, vaan sitä käytettään API-palvelun kautta (Turnbull, 2019, s. 66). Kolmannen osapuolen kehittämiä sovelluksia kuitenkin löytyy esimerkiksi Portus, joka tarjoaa rekisterien turvalliseen hallinnointiin kehitetyn käyttöliittymän (Portus, i.a.).

Rekisterin käyttö vaatii käyttäjätunnuksen ja kirjautumisen Docker Hub -rekisteriin (Turnbull, 2019, s. 14). Sisäänkirjautuminen onnistuu komennolla ’docker login’ ja uloskirjautuminen komennolla ’docker logout’ (mts. 14). Käytettäessä Docker-komentoja ’docker pull’ tai ’docker run’ kopioidaan vaaditut levykuvat määritetystä rekisteristä (Docker Inc, i.a.-b). Komennolla ’docker push’ luotu levykuva tallennetaan määritettyyn rekisteriin.

5 KUBERNETES

Nimi Kubernetes tulee kreikasta ja tarkoittaa ruorimiestä tai lentäjää. Kuitenkin lyhennettynä sitä kutsutaan nimellä K8s, ja lyhenne muodostuu laskemalla kahdeksan kirjainta K- ja s-kirjaimien välillä (Kubernetes, 2021-a). Kubereneteksen kehitti Google, ja projektin lähdekoodi julkistettiin avoimeksi DockerCon-tapahtumassa, sekä levitettiin yleiseen jakoon vuonna 2014 (Kane & Matthias, 2018, s. 262; Kubernetes, 2021-a). Siihen tiivistyy Googlen yli 15 vuoden kokemus tuotantotyökuormien suorittamiseksi laajassa mittakaavassa ja yhteisön parhaat ideat sekä käytännöt (Kubernetes, 2021-a). Vuosien saatossa Googlen kehittämä Kubernetes liittyi Cloud Native Computing Foundation (CNCF) -yhteisöön, ja siitä tuli selkeä markkinajohtaja konttipohjaisten sovellusten alalla (Baier ym., 2019, s. 64).

Kubernetes on nopeasti kasvava ekosysteemi, ja nykyään se on luultavasti laajimmin käytetty konttialusta (Kubernetes, 2021-a; Kane & Matthias, 2018, s. 262–263). Kuitenkin markkinoilla on myös muita vastaavia tuotteita, kuten Mesos, joka julkaistiin vuonna 2009 (Kane & Matthias, 2018, s. 263). Tuolloin se oli laajemmassa käytössä, ja kypsempi alusta kuin Kubernetes. Kuitenkin Kuberneteksen käytännöllisyys teki vaikutuksen moneen sen aikaiseen Docker-käyttäjään, ja sen tunnettavuus on noussut vuosien saatossa. Sen seurauksena vuonna 2017 Docker ilmoitti DockerCon-EU-tapahtumassa lisäävänsä Kubernetes tuen Docker Engine -työkaluun. Docker asiakkaat pystyivät tämän jälkeen sekoittamaan sovelluksien käyttöönotoissa eri variaatioita Docker Swarmin ja Kuberneteksen välillä saman työkalun avulla, mikä teki Kuberneteksestä entistä halutumman vaihtoehdon.

5.1 Sovellus

Kubernetes on siirrettävä, laajennettava, avoimen lähdekoodin alusta konttityökuormien ja -palveluiden hallintaan, mikä mahdollistaa sekä deklaratiivisen konfiguroinnin että automatisoinnin (Kubernetes, 2021-a). Lisäksi sen palvelut, tuki ja työkalut ovat laajasti saatavilla. Deklaratiivisella konfiguroinnilla tarkoitetaan työkalujen luokkaa, jonka avulla käyttäjät voivat ilmoittaa jonkin järjestelmän halutun tilan (Elhage, 2019). Kuberneteksen ensisijainen vastuualue on konttien orkestrointi, eli auttaa konttipohjaisten sovellusten käyttöönoton järjestämisessä, skaalauksessa ja hallinnassa, sekä varmistaa niiden toimivuus sovellusympäristössä (Baier ym., 2019, s. 64, 66). Kubernetes tarkkailee käynnissä olevien konttien eheyttä ja korvaa kaatuneet, reagoimattomat, tai muuten vialliset kontit uusilla (mts. 66).

Alustan tarjoamia palveluita (Baier ym., 2019, s. 65) on:

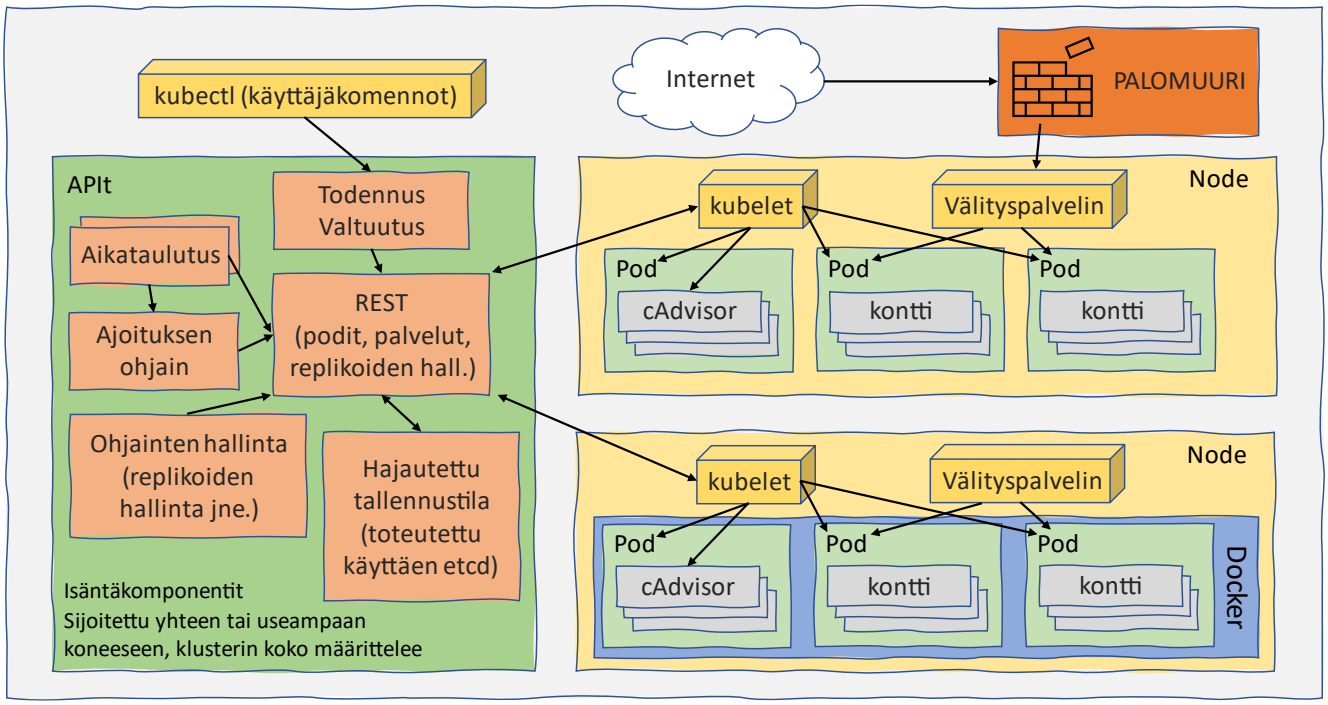
- Varastointijärjestelmien asentaminen
- Sovellusten eheyden tarkistaminen
- Sovellusinstanssien monistaminen
- Kuormien tasaaminen
- Päivitysten suorittaminen
- Resurssien seuranta
- Lokien keräys ja hyödyntäminen
- Sovelluksien virheiden etsiminen
- Todennuksien ja valtuutuksien tarjoaminen

Kubernetes ei kuitenkaan ole PaaS-palvelualusta (Baier ym., 2019, s. 65). Se ei määrää käyttöjärjestelmän tärkeitä ulkomuotoseikkoja, vaan jättää ne käyttäjälle tai muille Kubernetes päälle rakennetuille järjestelmille, kuten Deis, Openshift ja Eldarion. Esimerkiksi Kubernetes:

- Ei vaadi mitään tiettyä sovellusta tai kehystä
- Ei vaadi mitään tiettyä ohjelmointikieltä
- Ei tarjoa tietokantoja tai viestijonoja
- Ei erottele sovelluksia palveluista
- Ei omista kauppatoria valmiiden palveluiden hyödyntämiseksi
- Sallii käyttäjän valita omat loki, seuranta ja hälytysjärjestelmät

5.2 Komponentit

Kubernetes koostuu eri komponenteista ja yksiköistä, joita voidaan ikään kuin rakennuspalikkojen tavoin rakentaa päällekkäin tai sarjaan (Baier ym., 2019, s. 68). Komponentit voivat myös sijaita abstraktion eri tasoilla. Esimerkiksi node ja master ovat joukko Kubernetes-komponentteja, jotka sijaitsevat eri tasoilla. Komponentit liitetään yhteen ja organisoidaan API-ryhmiin sekä resurssiluokkiin parhaan toiminnallisuuden saavuttamiseksi. Kuviossa 8 havainnollistetaan edellä mainittua komponenteista rakennettua monimutkaista Kubernetes arkkitehtuuria syvemmän ymmärryksen saavuttamiseksi.



Kuvio 8. Kuberneteksen arkkitehtuuri (mukaillen Baier ym., 2019, s. 69).

5.2.1 Node

Node on fyysinen tai virtuaalinen kone, jota tyypillisesti käytetään Kubernetes-klusterissa Linux-pohjaisella käyttöjärjestelmällä (Heck, 2018, s. 17). Node toimii isäntänä sen sisällä toimiville komponenteille (Sayfan, 2018, s. 12). Jokaisella nodella on useita Kubernetes-komponentteja, kuten kube-välityspalvelin (proxy) ja kubelet (mts. 12). Kubernetes-isäntä hallinnoi nodeja, ja käytännössä noden tehtävänä on toimia isäntänsä työntekijänä, eli suorittaa podeja (mts. 12). Aikaisemmin nodeja kutsuttiin kätyreiksi ja tähän nimitykseen voi törmätä vanhemmissa artikkeleissa. Kuviossa 8 kuvataan noden toimintaympäristöä ja vastuualueita, sekä voidaan huomata sen hallinnoivan useita komponentteja.

Kubernetes hallinnoi resurssejaan nodeissa seuraamalla niiden resurssien käyttöä aikataulut-tamalla, käynnistämällä ja uudelleenkäynnistämällä podeja (Heck, 2018, s. 17). Lisäksi hallintaa voidaan toteuttaa koordinoimalla muita mekanismeja, jotka yhdistävät podit yhteen tai julkistavat ne klusterin ulkopuolelle. Nodet sisältävät merkityksellisiä metatietoja, joiden avulla Kubernetes pystyy esimerkiksi huomioimaan eroja podien ajoituksissa suorittaessaan niitä isäntäkomponentissa (mts. 18). Kubernetes tukee useiden erilaisten koneiden yhteistyötä, joten ohjelmistoja voidaan suorittaa koneiden välillä tehokkaasti, tai asettaa podien suorittamisen vain niihin koneisiin, joilla on riittävästi resursseja.

5.2.2 Master

Isäntä (master) on Kubernetes-klusterin ohjaustaso ja se koostuu useista komponenteista, kuten API-palvelimesta, aikataulutuksesta, valtuutuksesta ja todennuksesta sekä ohjainten skaalauksesta ja hallinnasta (Sayfan, 2018, s. 12; Saito ym., 2018, s. 9). Isäntä on vastuussa podien tapahtumien globaalista käsittelystä, ja yleensä kaikki pääkomponentit on määritetty yhdelle isännälle (Baier ym., 2019, s. 70). Kuitenkin käytettäessä korkean saatavuuden, tai erittäin suuria klustereita, suositellaan redundanttisen isännän käyttöä (mts. 70). Kuviosta 8 voidaan tarkastella kuinka isännän sisällä olevat komponentit linkittyvät toisiinsa.

Kubectl on komentoriviliittymä (command line interface), joka saadaan käyttöön asentamalla Kubernetes-isäntä (Saito ym., 2018, s. 10). Sitä voidaan käyttää Kubernetes-klusterin ohjaamiseen, komponenttien tilojen tarkasteluun tai luettelointiin.

Aikataulutus (scheduler) auttaa isäntää valitsemaan, mikä kontti ajetaan missäkin nodessa (Saito ym., 2018, s. 10). Se rakentuu yksinkertaisesta algoritmista, joka määrittelee prioriteetin konttien lähettämiseksi ja sitomiseksi node-komponentteihin. Määritettävät asiat koskevat yleensä käytettäviä resursseja, kuten muistia, suoritinta (CPU) ja konttien määrää.

Ohjainten hallinta (controller manager) suorittaa klusterin hallintatoimintoja (Saito ym., 2018, s. 10). Esimerkiksi se hallinnoi Kubernetes-nodeja, luo ja päivittää Kuberneteksen sisäisiä tietoja, ja yrittää muuttaa nykyistä tilaa haluttuun tilaan (mts. 10). Replikoiden hallinta (replication controller) kuuluu myös näihin hallintatoimintoihin (Kubernetes, 2022-a). Replikoiden määrää valvotaan ja hallinnoidaan niin, että saatavilla ja käytettävissä on aina tarvittava määrä podeja (Kubernetes, 2022-a).

5.2.3 Pod

Pod on pienin komponentti, jota Kubernetes hallinnoi (Heck, 2019, s. 15). Se on varsinainen työyksikkö, jonka ympärille muu järjestelmä rakentuu (Heck, 2019, s. 15; Sayfan, 2018, s. 12). Pod koostuu yhdestä tai useammasta kontista ja näihin liittyvistä tiedoista (Heck, 2019, s. 15). Pod-yksiköt ajoitetaan aina ajettavaksi samalla koneella yhtä aikaa (Sayfan, 2018, s. 12). Kontit podin sisällä jakavat saman IP-osoitteen ja portit, sekä kommunikoivat paikallisesti (localhost) tai prosessien välisellä (interprocess) viestinnällä (mts. 12).

Podien tietorakennetta voidaan tarkastella ja kysyä Kubernetekseltä komentorivin (CLI) komennoilla (Heck, 2019, s. 15). Tietorakenne sisältää tietoja yhdestä tai useammasta kontista, sekä erilaisia metatietoja, joita Kubernetes käyttää podin koordinoimiseen muiden podien kanssa. Lisäksi metatiedot sisältävät toimintaperiaatteet siitä, kuinka Kuberneteksen tulisi toimia ja reagoida, jos ohjelma epäonnistuu tai sitä pyydetään käynnistämään ohjelma uudelleen. Metatiedoissa voidaan määritellä myös asioita, kuten affiniteetti. Affiniteetti vaikuttaa muun muassa siihen, kuinka saadaan konttien levykuvat käyttöön ja Podin ajoitukseen klusterissa eli siihen, mihin se voidaan aikataulutuksellisesti sijoittaa. Podia ei ole tarkoitettu pitkäikäiseksi ja kestäväksi kokonaisuudeksi, vaan lyhytaikaisiksi yksiköiksi (mts. 15–16). Toisin sanoen niitä luodaan ja tuhotaan tarpeen mukaan, mikä mahdollistaa ohjelman paremman hallittavuuden ja saatavuuden (mts. 16).

Sovelluksia voidaan suorittaa yhden Docker-kontin sisällä (kuvio 8) käyttämällä Docker-sovellusta valvontaohjelmana ja asettamalla se suorittamaan useita prosesseja kontin sisällä (Heck, 2019, s. 70). Tätä käytäntöä kuitenkin paheksutaan yleensä seuraavien asioiden puutteesta (mts. 70–71):

- Läpinäkyvyys. Kun kontit podien sisällä tehdään näkyviksi koko infrastruktuurille, mahdollistaa se palveluiden paremman tarjoamisen konteille, kuten prosessienhallinnan ja resurssien seurannan. Tämä helpottaa käyttäjää toimintojen määrittämisessä.
- Ohjelmiston riippuvuuksien irrottaminen. Yksittäiset kontit voidaan helpommin versioida ja rakentaa uudelleen sekä käyttää uudelleen itsenäisesti.
- Helppokäyttöisyys. Käyttäjien ei tarvitse käyttää omia sovelluksia prosessinhallintaan, koodin ja signaalien käsittelyyn.
- Tehokkuus. Infrastruktuurin ottaessa enemmän vastuuta voivat kontit olla kevyempiä.

Podit tarjoavat erinomaisen ratkaisun toisiinsa läheisesti liittyvien konttiryhmien hallintaan, jotka ovat toisistaan riippuvaisia ja joiden on tehtävä yhteistyötä suorittaakseen niiden isännän asettaman tehtävän (Sayfan, 2018, s. 13). Lisäksi jokainen pod saa ainutlaatuisen yksilöivän tunnuksen (UID), jonka avulla ne voidaan tarvittaessa erottaa toisistaan.

5.2.4 Volume

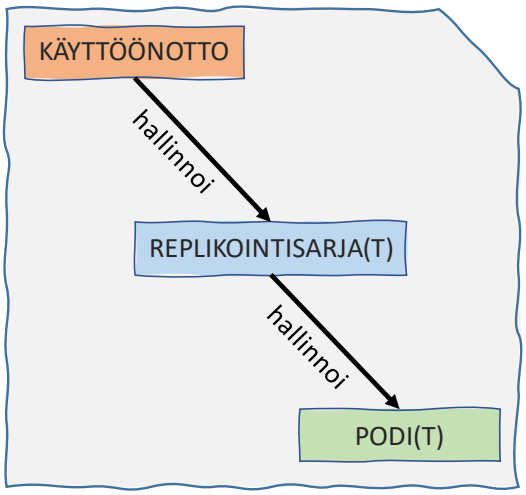
Paikallinen tiedon varastointi podissa tai kontissa on lyhytaikaista ja katoaa komponenttien tuhoutumisen mukana (Sayfan, 2018, s. 13, 15). Ratkaisuksi tähän käytetään volyymejä (volume) varastoimaan podin sisällä ajettavien konttien tuottamat tiedot paikalliseen tallennustilaan, joko koko podin käyttöiän ajaksi, tai vaihtoehtoisesti myös pidemmäksikin aikaa (Lukša, 2020, s. 181). Lisäksi podissa voidaan käyttää jaettua tallennustilaa, ja liittää se jokaiseen konttiin (Baier ym., 2019, s. 70; Lukša, 2020, s. 181). Volyymit tarjoavat myös pysyvää levytilaa osaksi sovellusten tietojen tallennusta (Brendan ym., 2019, s. 59).

Dockerilla on samantyyppinen volyymikonsepti, mutta se ei ole yhteensopiva Kuberneteksen tarjoaman vaihtoehdon kanssa, ja siksi näitä ei tule sekoittaa keskenään (Sayfan, 2018, s. 15). Volyymityyppejä on tarjolla monia ja Kubernetes tukee osaa niistä suoraan. Modernein tapa käyttää volyymejä Kuberneteksessä on konttien varastointikäyttöliittymä (CSI), jota käytetään konttijärjestelijöiden ja varastointeja tarjoavien palveluiden välisen vuorovaikutuksen standardoimiseksi (mts. 15, 212).

5.2.5 Nimikkeet, replikointisarjat ja replikointiohjaimet

Nimikkeet (labels) tarjoavat metatietoja objektien tunnistamiseen, ja näitä perusominaisuuksia käytetään objektien ryhmittelyyn, katseluun ja toiminnallisuuteen (Brendan ym., 2019, s. 65). Nimikkeiden valitsimet määritetään yksinkertaisilla avainarvopareilla, jossa molemmat arvo ja avain esitetään merkkijonoina (Sayfan, 2018, s. 14; Brendan ym., 2019, s. 66). Nimikkeiden vertailussa on olemassa kaksi eri operaatiota, yhtä suuri kuin tai erisuuri kuin, osoittamaan nimikkeiden samanarvoisuutta tai eriarvoisuutta (Sayfan, 2018, s. 14). Nimikkeisiin pohjautuvassa valinnassa voidaan useita arvoja erottaa pilkulla. Sarjaan perustuvat valitsimet kykenevät ja sallivat edellisen lisäksi myös valinnan useista eri nimikkeiden arvoista.

Replikointiohjaimet (replication controllers) ja replikointisarjat (replica sets) hallitsevat molemmat pod-ryhmiä nimikkeiden valitsimien avulla ja varmistavat, että tietty määritetty määrä podeja on aina toimintakunnossa (Sayfan, 2018, s. 14). Suurin ero on, että replikointiohjaimet valitsevat jäsenyyden testaamalla nimikkeiden samanarvoisuuden, ja replikointisarjat käyttävät sarjaan perustuvaa valintaa. Replikointisarjat ovat kriittisessä roolissa edistämässä vakaasuoraa skaalausta, eli rinnakkaisten podien määrän hallinnoimisessa (Heck, 2019, s. 19). Lisäksi ne kertovat klusterin sisällä suoritettavien podien määrän Kubernetekselle.



Kuvio 9. Käyttöönotto objektin, replikointisarjojen ja podien hallintakaavio (mukaillen Lukša, 2018, s. 261).

5.2.6 Service ja Deployment

Palvelu (service) on Kuberneteksen resurssi, jota käytetään avuksi podien käsittelyssä (Heck, 2018, s. 72). Palvelut tarjoavat esimerkiksi kerroksen konttien avulla tuotetun verkkosovelluksen käyttöliittymän ja tietokannan välille. Lisäksi Kubernetes mahdollistaa muun muassa palveluiden skaalauksen itsenäisesti, päivittämisen ja skaalauksen ongelmien käsittelemisen. Palvelu voi sisältää tiedonsiirron toimintaperiaatteet eli tiedon siitä, mikä data pitää siirtää. Palvelu toimii avainasemassa podien julkistamisessa toisillensa, ja myös konttien julkistamisessa klusterin ulkopuolelle. Kubernetes hallinnoi palveluiden avulla podien välistä koordinointia sekä liikennettä sisään ja ulos. Palvelun edistyneellä käytöllä voidaan määrittää palvelu myös resurssille, joka on kokonaan klusterin ulkopuolella.

Käyttöönotto-objektit (deployment objects) hallinnoivat Kuberneteksessa uusien versioiden (kuvio 9) julkaisua (Brendan ym., 2019, s. 113). Niiden avulla voidaan helposti siirtyä sovelluksen versiosta toiseen, muuttamalla versionumeroa määritetyssä käyttöönotto-objektissa. Tämä käyttöönoton muuttamisprosessi on hienovarainen ja säädettävissä. Tehdessä muutoksia käyttöönotto-objektiin Kubernetes odottaa sopivaa hetkeä tehdä muutokset jokaiseen podiin, ja varmistaa samalla uusien sovellusten eheyden ja soveltuvuuden uuteen ympäristöön. Mikäli virheitä ilmaantuu riittävästi, käyttöönottoprosessi pysäytetään. Käyttöönoton avulla voidaan siis yksinkertaisesti ja luotettavasti ottaa käyttöön uusia ohjelmistoversioita ilman seisokkiaikoja tai virheitä.

Todellista toiminnallisuutta ohjataan taustalla vaikuttavalla käyttöönoton ohjaimella (deployment contoller), jota suoritetaan klusterissa (Brendan ym., 2019, s. 113). Tämän ansiosta käyttöönotot on helppo integroida lukuisien jakelutyökalujen ja palveluiden kanssa. Palveluiden kanssa käyttöönotot voidaan suorittaa myös paikoissa, joissa on ajoittaisesti huono internetyhteys, koska käyttöönotto huolehtii uusien versioiden turvallisesta päivittymisestä (mts. 113). Käyttöönotto-objektit voidaan kirjoittaa YAML-merkintäkielellä tai JSON-muodossa (Lukša, 2020, s. 72).

5.2.7 Namespaces

Linuxiin pohjautuva nimitilat (namespaces) varmistavat, että jokaisella prosessilla on oma näkemyksensä järjestelmästä (Lukša, 2020, s. 45). Tämä tarkoittaa, että konteissa käynnissä olevat prosessit näkevät vain joitakin tiedostoja, kuten järjestelmän- ja verkonprosessit, ikään kuin ne olisivat käynnissä erillisessä virtuaalikoneessa. Aluksi kaikki järjestelmän resurssit on kerätty samaan paikkaan kaikkien käytettäväksi, mutta Linux-ytimen avulla luodaan nimitiloja erottamaan resursseja toisistaan, ja järjestelemällä ne pienempiin, toisistaan eristettyihin ryhmiin. Näin tehdessä voidaan asettaa yksittäisiä tai useita prosesseja näkyväksi tietyille ryhmille. Prosessia luodessa voidaan määrittää, mitä nimitilaa sen tulee käyttää. Luotu prosessi näkee vain niitä resursseja, jotka ovat tässä nimitilassa ja jättää muut huomioimatta.

Nimitiloja on olemassa useampaa eri tyyppiä ja tässä on niitä esitettynä (Lukša, 2020, s. 45):

- Mount-nimitila eristää kiinnityspisteet eli tiedostojärjestelmät
- Prosessi-ID-nimitila eristää prosessien tunnisteet
- Verkko-nimitila eristää verkkolaitteet, pinot ja portit.
- Prosessien välinen kommunikaatio (IPC) -nimitila eristää prosessien välistä kommunikaatioita, mukaan lukien viestijonoja ja jaettua muistia.
- UNIX-aikajakamisjärjestelmän (UTS) nimitila eristää järjestelmän isäntänimen ja verkkotietopalvelun (NIS) verkkotunnuksen
- Käyttäjä-ID-nimitila eristää käyttäjä- ja ryhmätunnukset
- Cgroup-nimitila eristää ohjausryhmän juurihakemiston.

5.2.8 ConfigMap ja Secrets

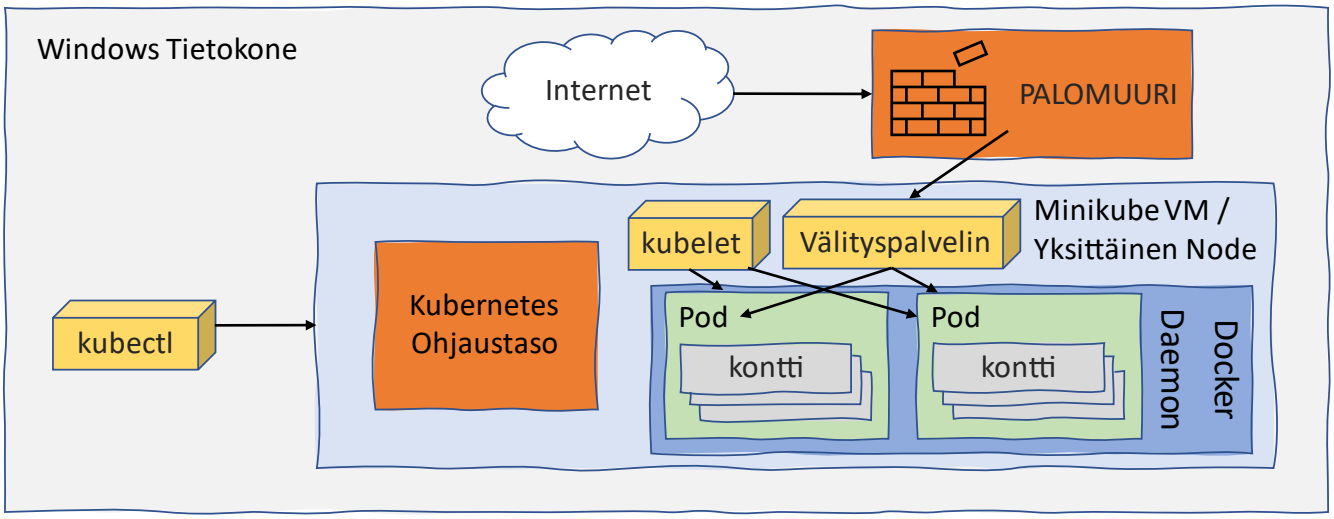
ConfigMap on API-objekti, ja sitä käytetään tallentamaan konfiguraatioita muiden objektien käyttöä varten (Kubernetes, 2022-b). ConfigMap yhdistetään podiin ennen sen suorittamista, jolloin kontin levykuvaa ja podin määritystiedostoa voidaan käyttää uudelleen muuhun tarkoitukseen, sekä muutokset voidaan kohdistaa ainoastaan ConfigMap-objektiin (Brendan ym., 2019, s. 153). Podit voivat käyttää ConfigMap-objektin tietoa monessa eri muodossa, kuten lyhyinä merkkijonoina, komentorivin argumentteina tai yhdistelmänä avainarvopareja määritystiedoston muodossa (Brendan ym., 2019, s. 153; Kubernetes, 2022-b). ConfigMap ei tarjoa salausta sisältämälleen tiedolle, joten luottamuksellisten tietojen suojaamiseksi on käytettävä secret-objektia tai kolmannen osapuolen työkaluja (Kubernetes, 2022-b).

Salaisuudet (secrets) ovat samanlaisia kuin ConfigMaps-objektit, mutta keskittyvät arkaluonteisten todennus- ja tunnistetietojen tarjoamiseen (Brendan ym., 2019, s. 153; Heck, 2018, s. 110). Ne voivat olla esimerkiksi tunnistetietoja, salasanoja, tunnuksia, avaimia tai TLS-sertifikaatteja (Brendan ym., 2019, s. 153; Kubernetes, 2021-b). Kubernetes luo ja hallinnoi salaisuuksia yksitellen, ja käyttää niiden tallentamisessa base64-koodausmenetelmää (Heck, 2018, s. 119).

5.3 Minikube

Kun tarvitaan paikallista kehitysympäristöä tai halutaan säästää pilviresurssien käytössä, voidaan tietokoneelle asentaa yhden noden klusteri käyttäen Kuberneteksen tarjoamaa minikube-klusteria (Brendan ym., 2019, s. 29). Minikube simuloi Kubernetes-klusteria, se soveltuu oppimiseen, kokeiluun ja sovellusten kehittämiseen paikallisesti.

Minikuben tapauksessa kaikki Kubernetes-ohjelmat ja -komponentit ajetaan yhdessä (kuvio 10) virtuaalikoneessa (Heck, 2018, s. 17). Laajemmissa Kubernetes-klustereissa voi olla yksi tai useampi kone määritettynä pelkästään klusterin hallinnoimiseen, sekä erilliset koneet, jossa työkuormaosiot ajetaan (mts. 17). Minikube ei kuitenkaan tarjoa laajojen klustereiden tavoin hajautettua klusteriympäristöä (Brendan ym., 2019, s. 29).



Kuvio 10. Minikuben Kubernetes-klusterin ympäristö Windowsilla (mukaillen Lukša, 2020, s. 56).

Linux-pohjaisilla käyttöjärjestelmillä voidaan Dockerin avustuksella käyttää Kubernetestä suoraan käyttöjärjestelmän päällä (Lukša, 2020, s. 57). Windowsia käytettäessä tulee koneessa olla hypervisor asennettuna ja käytettävissä (Brendan ym., 2019, s. 30). Vaihtoehtoisesti operoidessa macOs- tai Linux-käyttöjärjestelmillä, voidaan myös käyttää virtualbox-virtuaalikoneohjelmaa (mts. 30).

6 TUOTANTOYMPÄRISTÖ

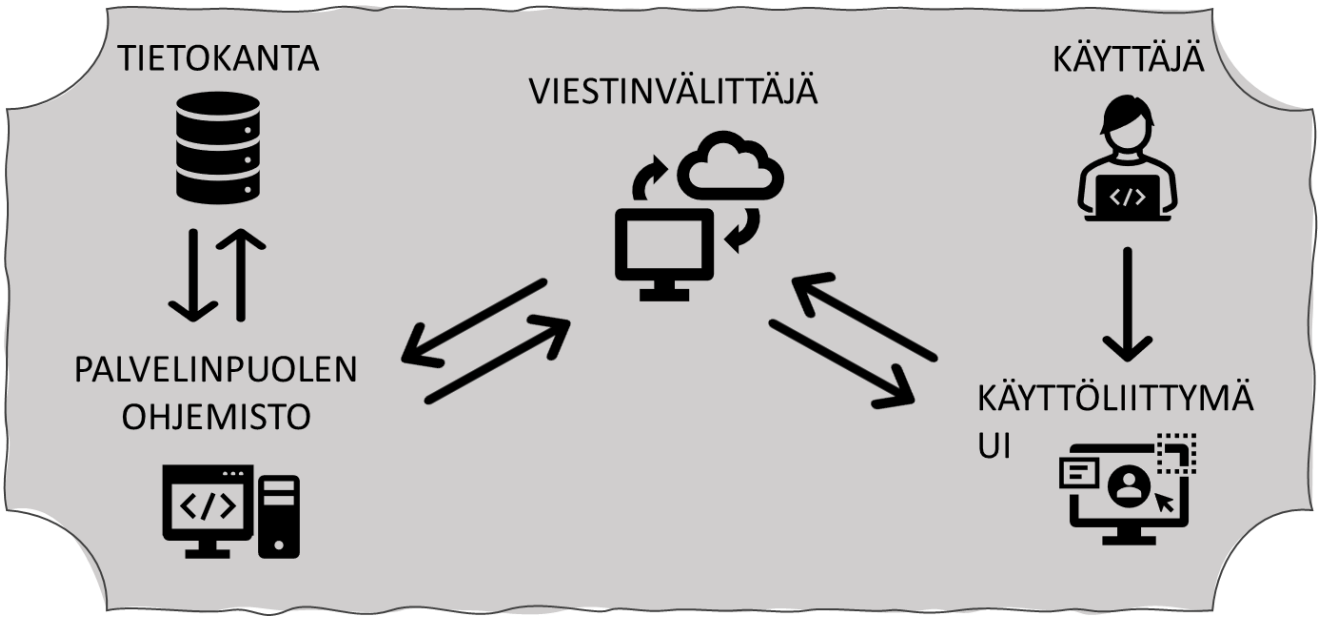
6.1 Tavoitteet

Tavoitteena oli selvittää ja toteuttaa Pesmelin tarjoaman varastonhallintajärjestelmän automatisoitu toimintaympäristö Dockerilla ja Kuberneteksellä. Keskeisimpiä selvitettäviä asioita olivat sovelluksien ja palveluiden päivitettävyys, ajoympäristön hallinta, tuotantoympäristön konfigurointi, korkean saatavuuden soveltaminen, vaadittavien resurssien selvitys ja käyttö, sovellusten versionhallinta sekä lokitietojen keräys. Tuotantoympäristön kehittämistä haluttiin selvittää ja tarkoituksena oli löytää siihen soveltuvia työkaluja. Yksi vaihtoehdoista oli Docker ja Kubernetes, jonka tarjoamia palveluita ja työkaluja tässä opinnäytetyössä selvitettiin ja toteutettiin käytännössä.

6.2 Tuotantoympäristön kartoitus

Käytännönosuuden alussa selvitettiin, mistä sovelluksista ja osa-alueista Pesmelin toimittamat verkkosovellusympäristöt todellisuudessa koostuvat. Varastonhallintajärjestelmät ovat laajoja kokonaisuuksia, ja vaativat useamman sovelluksen sekä palvelun yhteistoimintaa. Asiakkaalle usein näkyvä kokonaisuus on selaimen käyttöliittymä, josta sovellusta voidaan käyttää ja hallinnoida sovelluksen sisältämiä ominaisuuksia. Tämä on kuitenkin vain jäävuoren huippu (kuvio 11), koska todellisuudessa rakenteeseen tulee vielä lisää osa-alueita, kuten relaatiotietokanta (SQL), palvelinpuolen ohjelmisto (backend) ja tietoliikenteen viestinkäsittelijä. Sovellukset on toteutettu Windows-pohjaisella käyttöjärjestelmällä, mikä täytyy huomioida toteutusvaiheessa.

Tuotantoympäristö ei sisältänyt Dockeriin tai Kubernetekseen liittyviä ohjelmia, kuten Docker Desktop -sovellusta tai Kubernetes minikube -klusteria, joten kaikki tässä tutkimustyöosiossa tarvittavat työkalut asennettiin ja otettiin käyttöön tutkimustyön suorittamiseksi. Käytännön tutkimus suoritettiin VMware-virtuaalikoneympäristössä, koska sinne asennettuna sovellukset eivät häirinneet muita tietokoneen toiminnallisuuksia. Lisäksi vaadittavat muokkaukset sovelluksiin onnistuivat jouhevammin samassa virtuaaliympäristössä kehitystyökalujen kanssa. Todellisuudessa tuotantoympäristössä ei käytetä virtuaalikoneita, vaan kaikki sovellukset asennetaan suoraan käytettävän laitteiston käyttöjärjestelmän päälle.



Kuvio 11. Tuotantoympäristön rakenne.

6.2.1 Käyttöliittymä

Käyttöliittymää, tai puhekielessä käytettynä UI, käytetään verkkoselaimesta käsin joko paikallisessa tai verkkosivuja tarjoavan palvelun osoitteessa. Verkkosovellus on varastonhallintajärjestelmän näkyvin osa-alue. Varastonhallintajärjestelmä on verkkosivujen tapainen käyttöliittymä, mutta sisältää kuitenkin enemmän dynaamisia toiminnallisuuksia kuin tavalliset verkkosivut. Käyttöliittymän visuaalisuus rakentuu verkkosivun tavoin HTML-ohjelmointikielen koodista. Tämän lisäksi voidaan käyttää tyylikirjastoja, kuten Bootstrap, visuaalisen muotoilun tukena ja Aurelia.io-alustaa selaimen toiminnallisuuden rakentamiseksi.

Verkkosovelluksen alustana käytettiin avoimen lähdekoodin Node.js-ympäristöä. Node.js on asynkroninen tapahtumapohjainen JavaScript ajoympäristö, joka on suunniteltu skaalautuvien verkkosovellusten rakentamiseen (OpenJs Foundation, i.a.). JavaScript on yleinen ohjelmointikieli, jota myös Pesmel käyttää verkkosovelluksien ohjelmoimiseksi. Verkkosovellus kommunikoi ja vaihtoi tietoja palvelinpuolen sovelluksen kanssa REST API -rajapinnan välityksellä.

6.2.2 Tietokanta

Tietokannat ovat tallennuspaikkoja, joissa säilytetään dataa ja metadataa (Taylor, 2013, s. 7). Metadata on tietoa, joka sisältää kuvauksen datarakenteista tietokannassa. Tietorakenteet koostuvat useista eri tauluista, jotka voivat sisältää toisistaan riippuvaista tietoa, kuten sarakkeita (mts. 12). Tietokantoja hallinnoidessa käytetään SQL-ohjelmointikieltä (mts. 7). Tietokantoja voidaan käyttää eri tarkoituksiin, kuten henkilökohtaiseen käyttöön, ryhmätyöhön tai yrityskäyttöön (mts. 7–8). Tietokantoja hallinnoidaan tietokannan hallintajärjestelmällä (Database management system, DBMS) ja niitä voidaan skaalata moneen eri tarkoitukseen (mts. 8). Relaatiotietokanta on tietokantamalli, jossa taulujen välille luodaan niin sanottuja äiti-lapsi-suhteita eli viite-eheyksiä (mts. 12). Tietokannan rakenteita voidaan muuttaa ilman, että tarvitsee tehdä muutoksia sovelluksiin, mikäli tietokannan ja sovelluksissa käytettyjen taulujen väliset suhteet eivät ole vaarassa korruptoitua (mts. 11–12). Tässä opinnäytetyössä käytettiin Microsoftin SQL Serveriä 2019 hallinnoimaan relaatiotietokannan rakennetta, käyttäjiä ja tauluja.

6.2.3 Palvelinpuolen ohjelmisto

Palvelinpuolen ohjelmisto, jota myös kutsutaan nimellä backend, koostui tässä työssä Java-sovelluksesta, joka rakennettiin Hibernaten ja avoimen lähdekoodin Spring Boot -verkkosovellusalustan avulla. Java-sovelluksen avulla toteutettiin kaikki kirjoitukset tietokantoihin ja palvelinpuolen tarvittavat ohjaustoiminnot ja kommunikaatiot muiden palveluiden kanssa.

Java-sovellukseen kirjoitetaan haluttu koodi ja liitetään siihen tarvittavat liitännäiset, luokkamuuttujat ja ajastimet huolehtimaan ajettavista tehtävistä (Williams, 2014, s. 13). Käytännössä palvelinpuolen ohjelmistokokonaisuus on sovelluksen aivot. Lopuksi kokonaisuus paketoidaan yhdeksi tiedostokokonaisuudeksi (mts. 13). Tässä työssä käytettiin ajettavaa Java JAR -tiedostoa.

Spring Bootin avulla Java-kokonaisuus paketoitiin yhdeksi projektiksi. Työkalua nimeltä spring initializr käytettiin sovelluksen verkkosivuilla, jossa voitiin myös valita sovellukseen tarvittavat liitännäiset mukaan. Tässä työssä käytettiin apuna Pesmelin valmiiksi rakentamaa Java-sovellusta, koska aika ei olisi riittänyt ohjelmoimaan palvelinpuolen-sovellusta.

6.2.4 Tietoliikenteen viestien käsittely

Tietoliikennettä ohjaamaan käytettiin Apachen ActiveMQ-viestinvälittäjää. ActiveMQ on suosituin avoimen lähdekoodin ja usean protokollan Java-pohjainen viestinvälittäjä (The Apache Software Foundation, i.a.). Se tukee useampaa standardiprotokollaa ja kykenee yhdistämään JavaScript-, C-, C++-, Python- ja .Net-ohjelmointikielillä kirjoitetut asiakasohjelmat. Viestien tietoliikenne sovelluksien välillä voidaan toteuttaa käyttäen websocket-kommunikointiprotokollaa. Käytännön osuudessa tietoliikenteelle täytyi määrittää omat portit, joita käytettiin websocket-kommunikoinnissa.

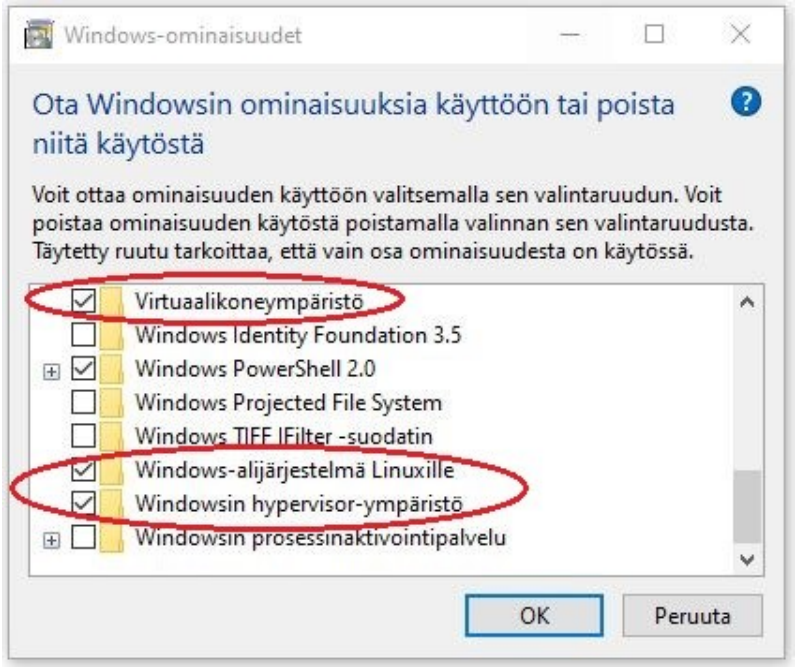
7 SOVELLUKSIEN ASENNUS

7.1 Taustatiedot

Työssä tarvittavat sovellukset ja ohjelmat asennettiin ensin, jotta itse tutkimus- ja selvitystyön tekemiseen oli vaadittavat edellytykset ja työkalut. Asennettavat sovellukset koostuivat pääosin Dockerista ja Kuberneteksestä, sekä niiden ohessa toimivista aliohjelmista, kuten Docker-compose, kubectl ja minikube.

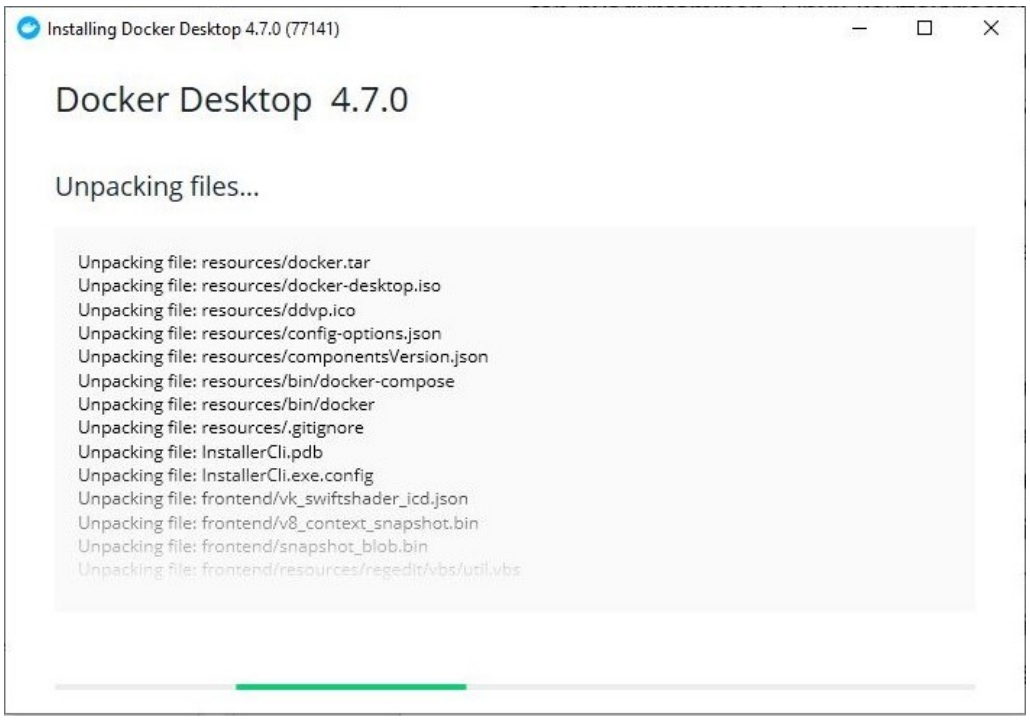
7.2 Docker

Ennen asennusta tarkistettiin Windowsin ominaisuuksista, löytyikö tietokoneesta virtuaalisointiin tarvittavat ominaisuudet, kuten Windowsin hypervisor-ympäristö, Windowsin alijärjestelmä Linuxille ja virtuaalikoneympäristö (kuva 1). Kuvassa 1 punaisella ympyröidyt ominaisuudet valittiin aktiiviseksi, mikäli näin ei ollut. Windows-ominaisuudet saatiin näkyviin suorittamalla komentosuoritteessa (⊞+R) komento ’optionalfeatures’. Lisäksi käytössä tuli olla Windows 11 -, Windows 10 Home- tai Pro -käyttöjärjestelmä ja RAM-muistia tarvittiin 4 GB. Windows-version tuli olla 2004 tai uudempi. Windowsin versio voitiin tarkistaa Windows-asetuksista. Asetukset-valikosta valittiin kohta Tietoja ja otsikon Windows-määritykset alta löytyivät käyttöjärjestelmän versiotiedot.



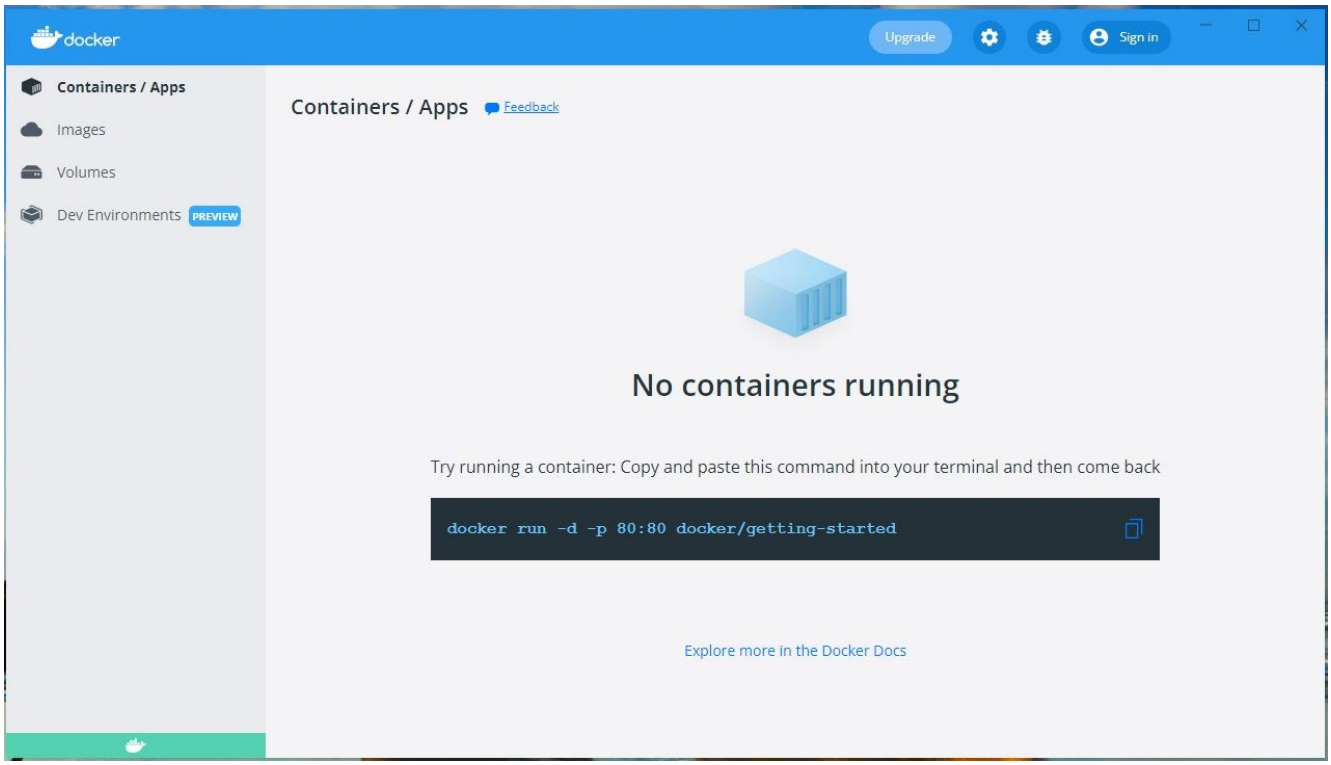
Kuva 1. Kuvankaappaus Windows-ominaisuuksista.

Docker asennettiin Windows-käyttöjärjestelmälle Docker Desktop -asennusohjelman avulla (kuva 2). Asennusohjelma ladattiin Dockerin virallisilta verkkosivuilta ja asennettiin asennusohjelman ohjeiden mukaisesti. Vaihtoehtoinen tapa asentaa ja suorittaa Linux-käyttöjärjestelmään pohjautuva Docker ilman Desktop-sovellusta, on Windowsin alijärjestelmän Linuxille WSL-kehitysympäristön hyödyntäminen. Tämä kuitenkin vaatii enemmän perehtyneisyyttä komentoriviliittymän käyttämisestä ja edellyttää Visual Studio Code -ohjelmaan Docker-laajennuksen asentamista. Kolmannen osapuolen sovellukset, kuten Podman, tarjoavat vielä yhden mielenkiintoisen ratkaisun, joka ei vaadi lainkaan palvelinpuolen taustaprosesseja toimiakseen. Kaikkiin edellä mainittuihin tapoihin löytyi kattavasti materiaalia asennuksen tueksi internetistä.



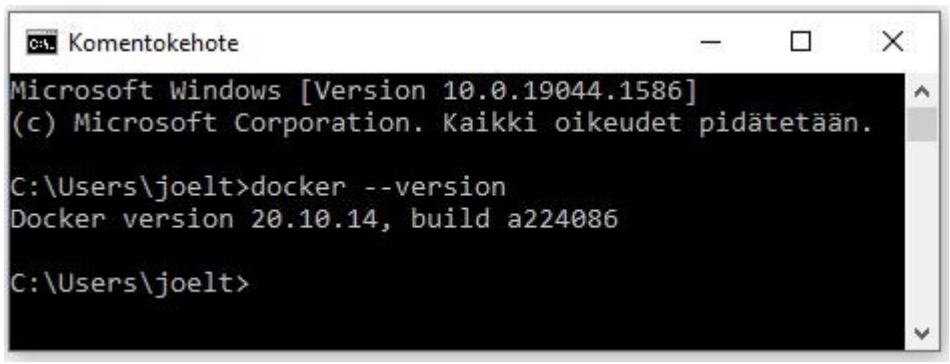
Kuva 2. Docker Desktop Installer.

Tässä työssä käytettiin Docker Desktop -sovellusta (kuva 3). Docker Desktop -asennusohjelmaan sisältyi myös Windowsin alijärjestelmä Linuxille, mutta kehittyneempi versio 2 (WSL 2). Desktop -asennusohjelman käyttäminen on suositelluin tapa asentaa Docker Windows-käyttöjärjestelmälle, sillä siinä mukana tuleva graafinen käyttöliittymä helpottaa Dockerin hallinnoimista (Microsoft, 2022). Kuitenkaan Windowsiin integroitua Desktop-sovellusta ei voida samanaikaisesti käyttää WSL-kehitysympäristön kanssa, vaan ne tulee suorittaa eristettynä toisistaan (Microsoft, 2022). Esimerkiksi WSL-instanssin tarvitsema Docker Engine voidaan suorittaa omassa Ubuntu-pohjaisessa kontissa, erillään Windowsista (Microsoft, 2022). Ubuntu on Linux-käyttöjärjestelmä.



Kuva 3. Docker Desktop -käyttöliittymän näkymä.

Desktop-sovelluksen asennuksen yhteydessä päivitettiin vielä Linux-ydin (kernel), WSL 2 -kehitysympäristön asennuksen loppuun saattamiseksi. Ytimen päivityspaketti ladattiin ja asennettiin Docker Desktop -sovelluksen osoittamasta Microsoftin verkko-osoitteesta. Päivityksen ja uudelleenkäynnistyksen jälkeen Docker oli asennettu onnistuneesti Windows-käyttöjärjestelmälle. Tämä voitiin tarkistaa komentokehotteella suorittamalla komento ’docker --version’, jolloin näytölle tulostui asennetun Docker-ohjelman versio (kuva 4).

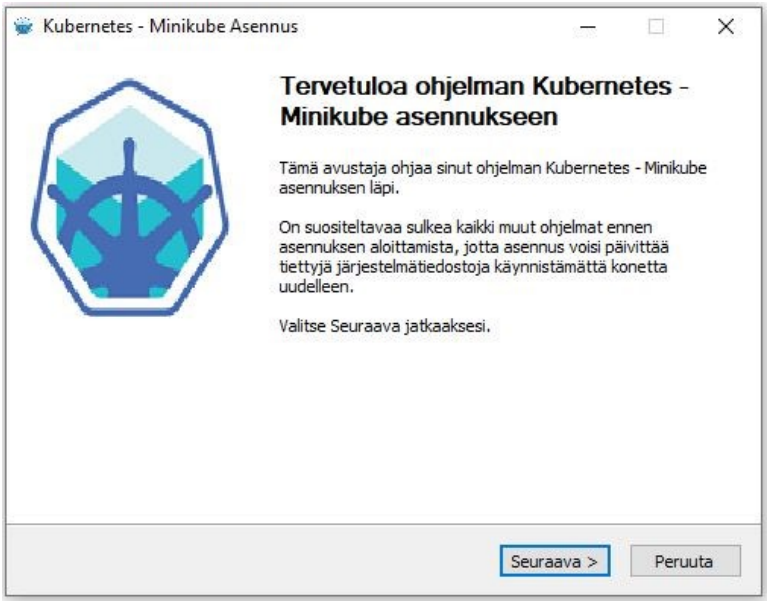


Kuva 4. Docker-version tarkistaminen komentokehotteesta.

7.3 Kubernetes-orkestrointisovellus

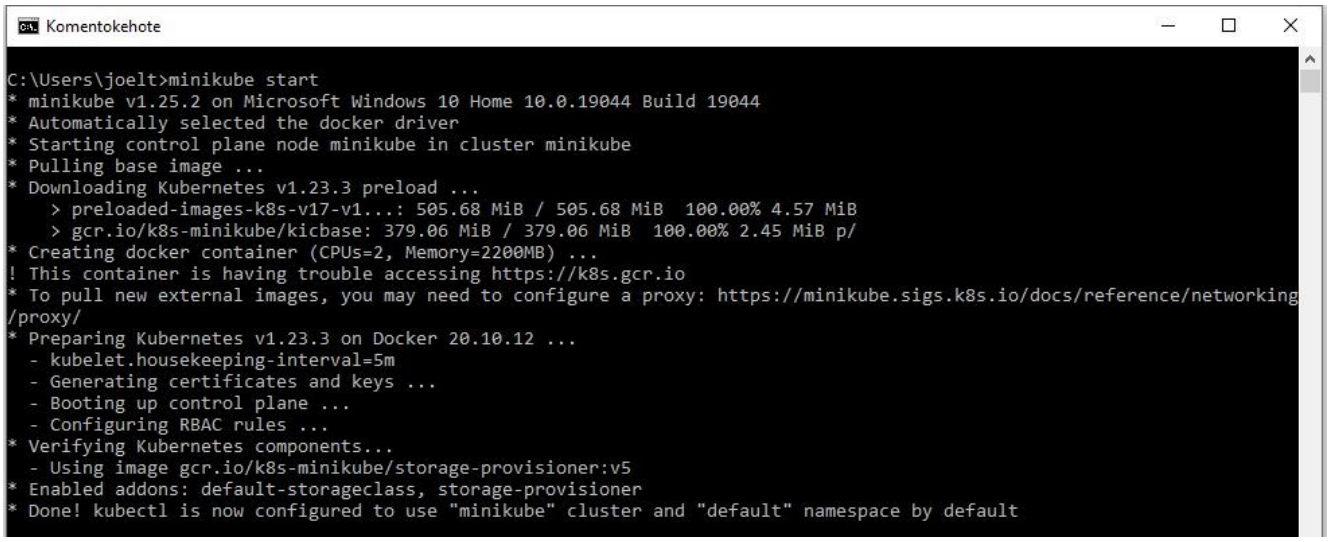
Kuberneteksen tapauksessa ei ollut saatavilla graafista käyttöliittymää sovelluksen ohjaamiseksi, vaan Kubernetestä hallinnoidaan komentokehotteella ympäristöihin määritetyillä komennoilla. Kuberneteksen asennuksen voi toteuttaa muutamalla eri tavalla riippuen käyttötarkoituksesta ja käyttöjärjestelmästä. Esimerkiksi olemassa on kind, jonka avulla voidaan paikallisesti suorittaa klusterien hallinnoiminen ja määrittäminen. Toiseksi voidaan käyttää kubeadm-työkalua, jonka avulla suoritetaan Kubernetes-klusteria suoraan laitteiston päällä. Kolmantena voidaan käyttää minikube-ympäristöä, joka luo virtuaalisen ympäristön klusterin suorittamiseksi.

Tässä työssä asennettiin ja käytettiin Kubernetes minikube -ympäristöä. Minikuben asennusohjelman ladattiin Kuberneteksen verkkosivuilta ja se vaatii laitteistolta RAM-muistia 2 GB, 2 CPU:ta ja 20 GB vapaata tallennustilaa levyasemalta. Verkkosivulta valittiin eri käyttöjärjestelmien, julkaisutyyppien ja asennustapojen väliltä sopivin vaihtoehto. Tässä työssä käytettiin verkkosivuilta ladattavaa Windowsille sopivaa asennusohjelmaa (kuva 5). Minikuben asentaminen onnistui seuraamalla asennusohjelman ohjeita.



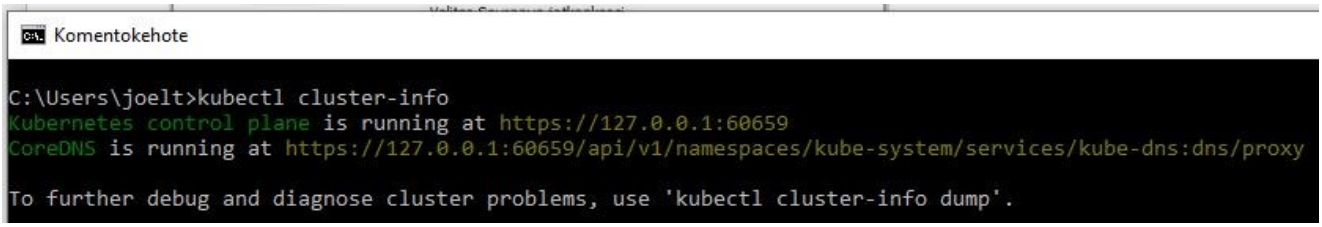
Kuva 5. Kubernetes minikube -asennusohjelma.

Onnistuneen asennuksen jälkeen avattiin komentokehote (⊞+R) komennolla ’cmd’ ja suoritettiin siinä komento ’minikube start’. Ensimmäisellä kerralla tämä komento latasi viimeisimmän Docker-levykuvan ja asensi tarvittavat tiedostot minikube-instanssin suorittamiseksi (kuva 6).



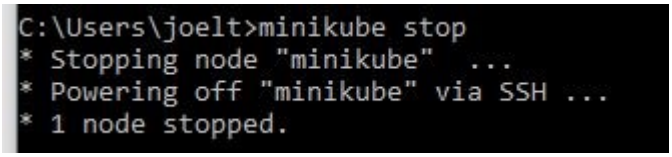
Kuva 6. Komentokehote ja minikuben käynnistäminen.

Tämän jälkeen tarkistettiin käyttäjäkomentotyökalun (kubectl) toimivuus ja klusterin tila komennolla ’kubectl cluster-info’ (kuva 7). Tätä käyttämällä komentokehotteeseen tulostui klusterin nykyinen tila ja IP-osoitteen tiedot.



Kuva 7. Klusterin tilan tarkistaminen.

Asennuksien todettiin onnistuneen ja osa-alueiden varmistettiin toimivan oikein. Minikube-ympäristö suljettiin ja sammutettiin komennolla ’minikube stop’ (kuva 8).



Kuva 8. Minikuben pysäyttäminen.

8 LEVYKUVIEN MÄÄRITYS

8.1 Ympäristö

Sovelluksien suorittaminen kontissa vaati kunkin sovelluksen kopioimisen ja käytön määrittämisen Docker-levykuvassa. Levykuvat luotiin Dockerfile-tiedostojen avulla omiksi kokonaisuuksiksi, jotka yhdistettiin suoritettavaksi samaan ympäristöön. Dockerfile-tiedostolla ei ole tiedostopäätettä, joka tekee siitä erityisen muihin tiedostomuotoihin nähden. Tällainen tiedosto voidaan luoda esimerkiksi komentokehotteessa komennolla ’type nul>Dockerfile’.

Suoritettavia Docker-levykuvia muodostettiin yhteensä 8 kappaletta, joista 5 sisältyi palvelinpuolen ohjelmistolle ja loput 3 muodostuivat toisten sovelluksien levykuvista. Seuraavissa kappaleissa paneudutaan konttiympäristön levykuvien määrittämiseen Dockerfile-tiedostojen ja ’docker build’ -komennon avulla.

8.2 Palvelinpuolen sovellus

Palvelinpuolen sovelluksen sisältö oli ohjelmoitu valmiiksi toimeksiantajan puolesta ja siihen ei ollut tarkoitus tässä työssä perehtyä sen tarkemmin. Sovelluksen rakennetta kuitenkin tarkasteltiin ja tutkittiin, mistä eri komponenteista se rakentui. Tällöin selvisi, että suoritettavia kokonaisuuksia löytyi 5 kappaletta, joista jokainen teki omaa erityistä tehtävää. Tehtäviin lukeutui kommunikointi eri laitteistoille ja järjestelmille, käyttöliittymän API, tietokannan migraatio, yrityksen kirjastot ja palvelinpuolen toimintalogiikka. Jokainen osa-alue paketoitiin ja rakennettiin omiin JAR-tiedostoihin, koska kyseessä oli Java-ohjelmointikieleen pohjautuva sovellus. JAR-tiedostoihin muutoksia tehdessä suoritettiin komento ’mvn clean package -am -DskipTests’, joka suoritti tiedoston uudelleen rakentamisen. Tämän jälkeen tuli luoda uusi levykuva, että uudet tiedot päivittyisivät myös sinne. Palvelinpuolen Dockerfile-tiedostot olivat keskenään hyvin samankaltaisia, koska niissä kaikissa määritettiin aluksi pohjalevykuva ja määritetty JAR-tiedosto lisättiin tämän päälle (kuva 9).

```
1  # BASE IMAGE
2  FROM openjdk:8
3  # ONT UI_API PORTS
4  EXPOSE 8081
5  # LISÄTÄÄN HAKEMISTOON
6  ADD /1.0-SNAPSHOT/ont-ui-api-1.0-SNAPSHOT.jar ont-ui-api-1.0-SNAPSHOT.jar
7  # SUORITETAAN KOMENTO
8  ENTRYPOINT ["java","-jar","ont-ui-api-1.0-SNAPSHOT.jar"]
```

Kuva 9. Dockerfile-tiedoston sisältö UI API -kontin osalta.

Tiedostojen suurimmat eroavaisuudet olivat lähinnä käytettävien porttien numeroissa ja JAR-tiedostojen nimissä. Määritetyt Dockerfile-tiedostot muutettiin Docker-levykuviksi komennolla ’docker build’ (kuva 10), jolloin ne siirtyivät koneen paikalliseen muistiin.

```
C:\Users\wms\.m2\repository\com\ont\ont-ui-api>docker build -f Dockerfile -t ont-ui-api:v1.0 .
[+] Building 7.1s (7/7) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 311B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/openjdk:8
 => [internal] load build context
 => => transferring context: 57.73MB
 => [1/2] FROM docker.io/library/openjdk:8@sha256:c498405e558f67cea05d04d63c5daf930d0c1fbbf451c8f04f0e2321d740cb83
 => CACHED [2/2] ADD /1.0-SNAPSHOT/ont-ui-api-1.0-SNAPSHOT.jar ont-ui-api-1.0-SNAPSHOT.jar
 => exporting to image
 => => exporting layers
 => => writing image sha256:e98fe1e2afa30ea106555bf2e58d97c7012a3244d695c21cfd1cdd0390769b01
 => => naming to docker.io/library/ont-ui-api:v1.0
```

Kuva 10. Docker-levykuvan luominen.

Luomisen yhteydessä annettiin haluttu levykuvan nimi ’-t’-merkinnän jälkeen, merkintä ’-f’ viittaa haettavaan Dockerfile-tiedostonimeen. Piste komennon perässä kertoo Dockerille, että suoritukseen tarvittava ’Dockerfile’-tiedosto sijaitsee komentoa suoritettavassa hakemistossa. Levykuvien luominen toteutettiin samalla tavalla myös lopuille palvelinpuolen tiedostoille.

8.3 Viestinvälittäjä

Tietoliikenteen viestinvälittäjälle tehtiin samat toimenpiteet kuin edellä, eli määritettiin Docker-file-tiedosto ja suoritettiin se ’docker build’ -komennolla. Kuvassa 11 ilmenee viestinvälittäjän levykuvan rakenne. Levykuvassa määritettiin muihin verrattuna enemmän käytettäviä portteja erityyppisille kommunikointitavoille, kuten STOMP, AMQP, MQTT ja TCP. Kaikkia portteja ei kuitenkaan käytetty, vaan ne määritettiin varmuuden vuoksi ennakkoon.

Toimeksiantajalla oli valmiiksi konfiguroitu viestinvälittäjäpalvelu Apache ActiveMQ, joten ri-villä 4 (kuva 11) tiedostosijainnista kopioitiin kaikki viestinvälittäjän tiedostosijainnin tiedostot

yhdeksi levykuvaksi. Peruslevykuvana käytettiin Alpine Linux -käyttöjärjestelmäpohjaista ratkaisua, koska tämän työn tekijän mielestä tämä versio toimi parhaiten viestinvälittäjä-sovelluksen kanssa. Alpine on pienin ja kevyin mahdollinen käyttöjärjestelmä, joka voidaan suorittaa kontin sisällä.

```
1  # Using jdk as base image
2  FROM openjdk:8-jdk-alpine
3  # Copy the whole directory of activemq into the image
4  COPY apache-activemq-5.15.12 /opt/apache-activemq-5.15.12
5  # Set the working directory to the bin folder
6  WORKDIR /opt/apache-activemq-5.15.12/bin
7  # Start up the activemq server
8  EXPOSE 8161 61616 61613 5672 1883
9  ENTRYPOINT ["./activemq","console"]
```

Kuva 11. Viestinvälittäjän Dockerfile.

8.4 Käyttöliittymä

Käyttöliittymän peruskuvana käytettiin node-levykuvaa (kuva 12) ja olemassa olevat tiedostot kopioitiin tämän päälle omaan määritettyyn tiedostosijaintiin. Kopioidut Node.js-komponentit levykuvan sisällä asennettiin komennolla ’npm install’. Tämän jälkeen asennettiin vielä Aurelia-komentorivityökalu. Levykuvaan asennetut tiedostot valmisteltiin suorittamista varten ’npm run build’ -komennolla. Kommunikointia varten avattiin portti 9000, ja rivillä 9 (kuva 12) levykuvan sisältö käynnistettiin halutulla IP-osoitteella.

```
1  FROM node:14
2  RUN mkdir -p /home/app
3  COPY . /home/app
4  WORKDIR /home/app
5  RUN npm install
6  RUN npm install aurelia cli -g
7  RUN npm run build
8  EXPOSE 9000
9  CMD ["npm","start","--","--host", "0.0.0.0"]
```

Kuva 12. Käyttöliittymän Dockerfile.

8.5 Tietokanta

Tietokannan määrittäminen konttiin toteutettiin useamman eri tiedoston yhdistelmällä, koska tietokantaan haluttiin määrittää oletuskäyttäjä ja itse nimetty tietokanta Java-sovelluksien kommunikointia varten. Tietokannan tuli olla määritettynä ensin, jotta tiedon tallennus ja kommunikointi muiden sovelluksien kanssa oli ylipäätänsä mahdollista (kuva 13). Tietokantaa

määritettäessä tuli ympäristömuuttujiin syöttää salasana ja hyväksyä käyttöehdot. Salasanan tuli olla riittävän vahva sekä sisältää erikoismerkkejä ja erikokoisia kirjaimia. Tässä työssä käytettiin kuitenkin mahdollisimman yksinkertaista salasanaa esimerkkikuvan takia.

```
1  # Download base image and set variables
2  FROM mcr.microsoft.com/mssql/server:2019-latest
3  ENV ACCEPT_EULA=Y
4  ENV SA_PASSWORD=TestPassword
5  VOLUME /ont-mssql /var/lib/mssqlql/data
6  EXPOSE 1433/tcp
7  # Switch to root user for admin access
8  USER root
9  # Create directory and copy
10 RUN mkdir -p /usr/src/app
11 COPY . /usr/src/app
12 WORKDIR /usr/src/app
13 # Give permissions and execute
14 RUN chmod +x /usr/src/app/data.sh
15 ENTRYPOINT /bin/bash ./entrypoint.sh
```

Kuva 13. Tietokannan levykuvan määritys.

Levykuvan määrittämisen lopuksi (kuva 13) annettiin kontin sisällä käyttäjälle oikeus suorittaa ’entrypoint.sh’-tiedosto, joka sisälsi (kuva 14) halutut tietokannan määritykset ja tietokannan käynnistämisen.

```
1  /opt/mssql/bin/sqlservr & /usr/src/app/data.sh
```

Kuva 14. Entrypoint.sh-tiedoston sisältö.

Kutsutussa ’data.sh’-tiedostossa (kuva 15) tehtiin vielä tarkasteluja ennen varsinaista määritystiedoston suorittamista. Sh-tiedostot oli muotoiltu Unix-käyttöjärjestelmän hyväksymäksi koodiksi. Rivillä 6 (kuva 15) suoritetaan tietokannan sqlcmd-komentorivityökalun avulla määritystiedosto ’setup.sql’ levykuvassa määritetyllä salasanalla.

```
#run the setup script to create the DB and the schema in the DB
#do this in a loop because the timing for when the SQL instance is indeterminate
sleep 20
for i in {1..25};
do
    /opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P Test1Passw0rd -d master -i setup.sql
    if [ $? -eq 0 ]
    then
        echo "setup.sql completed"
        break
    else
        echo "not ready yet..."
        sleep 1
    fi
done
```

Kuva 15. data.sh-tiedosto.

Varsinainen käyttäjän lisääminen ja oikeuksien määrittäminen suoritettiin vasta siis tiedostossa ’setup.sql’ (kuva 16), joka sisälsi pelkästään SQL-kielen komentoja. Tiedostossa luotiin tietokanta ’WMS_database_ONT’ ja sille käyttäjä ’testuser’, sekä asetettiin tarvittavat oikeudet.

```
CREATE DATABASE [WMS_database_ONT]
GO
USE [WMS_database_ONT]
GO
CREATE LOGIN [testuser] WITH PASSWORD=N'testPass', DEFAULT_DATABASE=[master], CHECK_EXPIRATION=OFF, CHECK_POLICY=OFF
GO
CREATE USER [pesmel] FOR LOGIN [testuser]
GO
USE [WMS_database_ONT]
GO
ALTER USER [testuser] WITH DEFAULT_SCHEMA=[dbo]
GO
USE [WMS_database_ONT]
GO
ALTER ROLE [db_owner] ADD MEMBER [testuser]
GO
ALTER DATABASE [WMS_database_ONT] SET AUTO_CLOSE OFF
GO
```

Kuva 16. Tietokannan määritystiedosto 'setup.sql'.

Muokattu tietokanta tallennettiin erilliseksi levykuvaksi ja nimettiin komennolla ’docker commit mcr.microsoft.com/mssql/server:2019-latest ont-mssql:x0’, tämän jälkeen tietokannan levykuvaa ei tarvinnut määrittää enää uudestaan, vaan voitiin käyttää valmiiksi määritettyä tietokantaa apuna seuraavissa vaiheissa.

9 KÄYTTÖMAHDOLLISUUKSIEN SELVITYS

9.1 Docker-ympäristö

Levykuvien määrityksien jälkeen yritettiin suorittaa kaikki levykuvat samaan aikaan Docker Compose -tiedoston avulla. Näin varmistuttiin levykuvien toimivuudesta ja yhteensopivuudesta. Compose-tiedosto (kuva 17) on YAML-merkintäkielellä kirjoitettu tiedosto, jossa listataan suoritettavat levykuvat. Tiedostoon määritettiin myös käytettäville levykuville oma verkko, jossa sovellukset pystyivät kommunikoimaan keskenään samalla IP-osoitteella. Sovelluksille tuli kuitenkin määrittää käytettävät portit erikseen.

Compose-tiedostossa voitaisiin määrittää myös ehtoja uudelleenkäynnistykselle, mikäli levykuvan suorittaminen epäonnistuisi, ja lisäksi voidaan asettaa eheystarkastuksia halutuille levykuville. Lisäksi toisistaan riippuvaiset kontit tulee linkittää keskenään attribuutilla ’links’ (liite 1), joka määrittää niiden välille linkin, jotta ne kykenevät keskustelemaan keskenään ja vaihtamaan tietoa sovelluksien välillä.

```
version: '3'
services:
  node:
    image: ontui:master1.0
    ports:
      - 9000:9000
      - 8080:8080
  mssql:
    image: ont-mssql:x0
    ports:
      - 1433:1433
  ont-wms-rulla:
    image: ont-wms-rulla:x0
    ports:
      - 8079:8079
  ont-ui-api:
    image: ont-ui-api:x0
    ports:
      - 8081:8081
  ont-comm:
    image: ont-comm:x0
  ont-base:
    image: ont-base:x0
  ont-db-migration:
    image: ont-db-migration:x0
    ports:
      - 1443:1443
  ont-activemq:
    image: ontactivemq:x0
    ports:
      - 8161:8161
      - 61616:61616
      - 61613:61613
      - 5672:5672
      - 1883:1883
networks:
  ont-network:
    driver: bridge
```

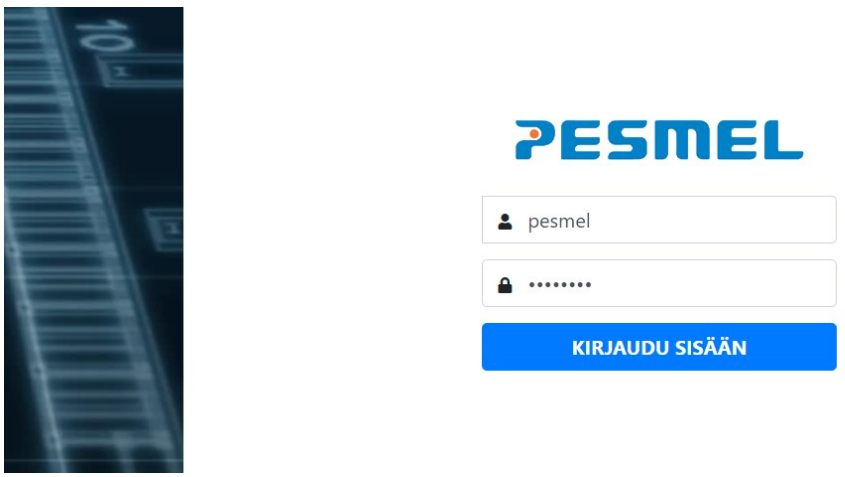
Kuva 17. Docker Compose -tiedoston sisältö.

Compose-tiedostossa levykuvan versiota muutettiin muuttamalla levykuvan nimen perässä kaksoispisteellä alkavaa versionumeroa joksikin toiseksi, mikä nopeutti toimintaa testausvaiheessa. Tiedoston määrityksen jälkeen suoritettiin komento ’docker-compose -f ont-compose.yaml up’ (kuva 18), joka käynnisti kaikki tiedostossa määritetyt levykuvat samanaikaisesti.

```
C:\Users\wms\Documents\ont-wms>docker-compose -f ont-compose.yaml up
[+] Running 8/8
 - Container ont-wms-ont-db-migration-1  Created                                                    0.3s
 - Container ont-wms-ont-comm-1          Created                                                    0.3s
 - Container ont-wms-node-1              Created                                                    0.3s
 - Container ont-wms-ont-wms-rulla-1     Created                                                    0.3s
 - Container ont-wms-ont-base-1          Created                                                    0.3s
 - Container ont-wms-ont-ui-api-1        Created                                                    0.3s
 - Container ont-wms-mssql-1             Created                                                    0.3s
 - Container ont-wms-ont-activemq-1      Created                                                    0.3s
Attaching to ont-wms-mssql-1, ont-wms-node-1, ont-wms-ont-activemq-1, ont-wms-ont-base-1, ont-wms-ont-comm-1, ont-wms-ont
-db-migration-1, ont-wms-ont-ui-api-1, ont-wms-ont-wms-rulla-1
```

Kuva 18. Compose-tiedoston käynnistäminen.

Määritetyt levykuvat käynnistyivät oletetusti, mutteivat ongelmitta. Esimerkiksi websocket-yhteyden muodostaminen Noden ja käyttöliittymän API-kontin välille osoittautui ongelmalliseksi, johtuen määritystiedostojen monimutkaisuudesta. Käynnistyksen yhteydessä näytölle tulostui konttien sisältämien sovelluksien tietoja (liite 2), joka helpotti virheiden havainnointia. Sovelluksien käynnistyttyä selaimella siirryttiin verkkosoiteeseen ’https://localhost:9000’, jossa verkkosovelluksen käyttöliittymän kirjautumissivu (kuva 19) oli käynnistyneenä.



Kuva 19. Verkkosovelluksen käyttöliittymän kirjautumissivu.

Tässä vaiheessa todettiin kuitenkin levykuvien määrityksen onnistuneen riittävän hyvin työn jatkamista varten. Seuraavassa vaiheessa määritettiin käytettävät levykuvat Kuberneteksen Minikube-ympäristöön varsinaista Kuberneteksen tutkimustyövaihetta varten.

9.2 Kubernetes-ympäristö

Minikube-ympäristön määrittämisessä käytettiin apuna Kubernetes Kompose -työkalua, joka kääntää Docker Compose -tiedoston yhteensopivaksi Kubernetekselle. Kompose-työkalun käyttötapa on samanlainen kuin Compose-työkalulla, ja tästä syystä Kubernetes onkin tarkoituksenmukaisesti nimennyt oman työkalunsa muuttamalla C-kirjaimen, K-kirjaimeksi. Kompose-työkalun käytössä tulee kuitenkin huomioida, ettei se ole täydellinen, mutta se luo hyvän pohjan tarvittaville tiedostoille. Lisäksi alkumäärityksissä säästetään aikaa. Kompose-työkalun voi ladata Windowsille komennolla ’choco install kubernetes-kompose’, mikäli koneelle on asennettu Chocolatery-asennusohjelmisto. Chocolatery-asennusohjelmiston kautta useat työkalut ja sovellukset ovat helpommin ladattavissa suoraan komentokehotteesta.

Varsinainen Kompose-työkalun käyttö suoritetaan komennolla ’kompose convert -f ont-compose.yaml’ (kuva 20). Komento suorittaa määritetyn YAML-tiedoston nimeltä ’ont-compose.yaml’ ja muuntaa sen Kubernetekseen sopiviksi tiedostoiksi. Komentoon voidaan lisätä myös muita haluttuja parametrejä, kuten muunnetaanko kaikki tiedot yhteen tiedostoon vai siirretäänkö ne johonkin tiettyyn hakemistoon. Tällaisena komento luo jokaiselle sovellukselle oman Deployment ja Service -tiedoston samaan hakemistoon YAML-tiedoston kanssa.

```
C:\Users\wms\Documents\ont-wms>kompose convert -f ont-compose.yaml
INFO Network ont-network is detected at Source, shall be converted to equivalent NetworkPolicy at Destination
WARN Service "ont-base" won't be created because 'ports' is not specified
INFO Network ont-network is detected at Source, shall be converted to equivalent NetworkPolicy at Destination
INFO Network ont-network is detected at Source, shall be converted to equivalent NetworkPolicy at Destination
INFO Network ont-network is detected at Source, shall be converted to equivalent NetworkPolicy at Destination
INFO Create kubernetes pod instead of pod controller due to restart policy: on-failure
INFO Network ont-network is detected at Source, shall be converted to equivalent NetworkPolicy at Destination
INFO Network ont-network is detected at Source, shall be converted to equivalent NetworkPolicy at Destination
INFO Network ont-network is detected at Source, shall be converted to equivalent NetworkPolicy at Destination
INFO Kubernetes file "ont-activemq-service.yaml" created
INFO Kubernetes file "ont-comm-service.yaml" created
INFO Kubernetes file "ont-db-migration-service.yaml" created
INFO Kubernetes file "ont-mssql-service.yaml" created
INFO Kubernetes file "ont-node-service.yaml" created
INFO Kubernetes file "ont-ui-api-service.yaml" created
INFO Kubernetes file "ont-wms-rulla-service.yaml" created
INFO Kubernetes file "ont-activemq-deployment.yaml" created
INFO Kubernetes file "ont-network-networkpolicy.yaml" created
INFO Kubernetes file "ont-base-deployment.yaml" created
INFO Kubernetes file "ont-comm-deployment.yaml" created
INFO Kubernetes file "ont-db-migration-deployment.yaml" created
INFO Kubernetes file "ont-mssql-pod.yaml" created
INFO Kubernetes file "mssqlvolume-persistentvolumeclaim.yaml" created
INFO Kubernetes file "ont-node-deployment.yaml" created
INFO Kubernetes file "ont-ui-api-deployment.yaml" created
INFO Kubernetes file "ont-wms-rulla-deployment.yaml" created
```

Kuva 20. Kompose-komennon käyttö.

Komento loi myös konfiguraatiot verkkoasetuksille ja määritti tallennustilan Kubernetes-ympäristölle (volume). Kuvassa 21 on esimerkki Deployment-tiedoston sisällöstä, jossa määritetään ajettavien kopioiden lukumäärä, käytettävän levykuvan nimi sekä sille määritetyt asetukset.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    io.kompose.service: ont-node
  name: ont-node
spec:
  replicas: 1
  selector:
    matchLabels:
      io.kompose.service: ont-node
  strategy: {}
  template:
    metadata:
      labels:
        io.kompose.service: ont-node
    spec:
      containers:
        - image: docker.io/joketti/ont:ui
          imagePullPolicy: IfNotPresent
          name: ont-node
          ports:
            - containerPort: 9000
            - containerPort: 8080
          resources: {}
      imagePullSecrets:
        - name: regcred
      restartPolicy: Always
status: {}
```

Kuva 21. Ont-node-deployment.yaml -tiedosto.

Levykuvat päätettiin tallentaa Docker Hub -rekisteriin sujuvampaa käyttöä varten. Lisäksi levykuvista jäi talteen useampia eri versioita ja koneen levytilaa vapautettiin muuhun käyttöön. Deployment-tiedoston määrityksiin sisältyy myös Kubernetes-salaisuus (secret) nimeltä ’regcred’, jossa sijaitsee salatut käyttäjätunnisteet Docker Hub -rekisteriin. Kubernetes-salaisuus luodaan komennolla ’kubectl create secret docker-registry regcred --docker-server= https://index.docker.io/v1/ --docker-username=testuser --docker-password=TestPassword--docker-email=test.user@hotmail.com’. Salaisuuden määrittäminen oli tarpeellinen, koska Kubernetes ei osannut ladata levykuvia yksityisestä rekisteristä ilman käyttäjätunnistetietoja. Lisäksi tietoturva tiedoston käsittelyssä parantui, koska käyttäjätunnukset eivät olleet näkyvissä.

Määritetyt tiedostot otettiin käyttöön komennolla ’kubectl apply -f ./k8s’ (kuva 22). Tämä komento suorittaa määritetyn k8s-kansion sisältöineen, näin säästytään useamman komennon kirjoittamiselta.

```
C:\Users\wms\Documents\ont-wms>kubectl apply -f ./k8s
persistentvolumeclaim/mssqlvolume configured
deployment.apps/ont-activemq created
service/ont-activemq created
deployment.apps/ont-db-migration created
deployment.apps/ont-mssql created
service/ont-mssql created
networkpolicy.networking.k8s.io/ont-network configured
deployment.apps/ont-node created
service/ont-node created
deployment.apps/ont-ui-api created
service/ont-ui-api created
deployment.apps/ont-wms-rulla created
service/ont-wms-rulla created
```

Kuva 22. Kubectl apply -komento.

Seuraavaksi tarkistettiin, ovatko service-, deployment- ja pod-objektit käynnistyneet. Tämä suoritettiin komennolla ’kubectl get deployment’, ’kubectl get service’ tai ’kubectl get pods’. Komennon loppuun voidaan lisätä ’-o wide’ (kuva 23), joka antaa vielä laajemmin tietoa objekteista. Samalla nähtiin objekteihin viittaavien levykuvien nimet.

```
C:\Users\wms\Documents\otk-wms>kubectl get deployment -o wide
NAME              READY   UP-TO-DATE   AVAILABLE   AGE    CONTAINERS        IMAGES                                SELECTOR
ont-activemq      1/1     1            1           2m5s   ont-activemq      docker.io/joketti/ont:activemq        io.kompose.service=ont-activemq
ont-db-migration  1/1     1            1           2m4s   ont-db-migration  docker.io/joketti/ont:db-migration    io.kompose.service=ont-db-migration
ont-mssql         1/1     1            1           2m4s   ont-mssql         docker.io/joketti/ont:mssql           io.kompose.service=ont-mssql
ont-node          1/1     1            1           2m4s   ont-node          docker.io/joketti/ont:ui              io.kompose.service=ont-node
ont-ui-api        1/1     1            1           2m3s   ont-ui-api        docker.io/joketti/ont:ui-api          io.kompose.service=ont-ui-api
ont-wms-rulla     0/1     1            0           2m3s   ont-wms-rulla     docker.io/joketti/ont:wms-rulla       io.kompose.service=ont-wms-rulla
```

Kuva 23. Laajempi tieto deployment-komennosta.

Kaikki osiot saatiin käynnistettyä, mutta yhteyksien määrittämisessä oli ongelmia, minkä vuoksi sovelluksia ei saatu kommunikoimaan keskenään. Docker Compose -tiedostoissa tämä ongelma ratkesi links-attribuutilla, mutta Kubereneteksen tapauksessa tulisi määrittää manuaalisesti IP-osoitteet tai oma DNS-palvelin huolehtimaan kommunikoinnista. Lisäksi Ingress-tiedoston määrittämisellä voitaisiin julkaista Kubernetes-objekteja minikube-noden ulkopuolelle. Nämä osiot jäivät tässä työssä tekemättä ajan loppumisen vuoksi.

10 TULOKSET

Selvitettäviä asioita oli kohtuullisen paljon ja työn laajuudesta kertovat asetetut tavoitteet, kuten sovelluksien ja palveluiden päivitettävyys, ajoympäristön hallinta, tuotantoympäristön konfigurointi, korkean saatavuuden soveltaminen, vaadittavien resurssien selvitys ja käyttö, sovellusten versionhallinta sekä lokitietojen keräys. Näihin kaikkiin tavoitteisiin ei päästy käytännön osalta, vaan niitä ehdittiin tutkimaan ainoastaan teorian tasolla. Esimerkiksi sovellusten versionhallinta, korkean saatavuuden soveltaminen ja lokitietojen keräys jäivät käytännön osalta toteuttamatta.

Dockerin osalta tavoitteisiin päästiin ja tuotantoympäristö saatiin määritettyä. Yhtenä poikkeuksena oli kuitenkin käyttöliittymä, joka lähti käyntiin, mutta websocket-kommunikointi käyttöliittymän API-kontille jäi uupumaan. Tämän vuoksi varastonhallintajärjestelmää ei päästy ohjaamaan käyttöliittymältä.

Taustalla suoritettavat palvelinpuolen ohjelmistot käynnistyivät ja kykenivät kommunikoimaan tietokannan kanssa, joten ne saatiin onnistuneesti määritettyä. Palvelinpuolen ohjelmiston osioita olivat kommunikointi, käyttöliittymän API, tietokannan migraatio, yrityksen kirjastot ja palvelinpuolen toimintalogiikka.

Tietokanta saatiin määritettyä ja sinne luotiin käyttöoikeudet. Tietokannan rakenteeseen määritettiin uusi tietokanta nimeltä WMS_database_ONT. Lisäksi Docker Hub -rekisteriin tallennettiin käytetyt levykuvat ja rekisterin avulla voitiin ainakin osittain toteuttaa versionhallintaa, mutta vain levykuvien osalta.

Kuberneteksen osalta tavoitteisiin päästiin osittain. Suorittamiseen tarvittavat tiedostot saatiin luotua, mutta tuotantoympäristön määritys jäi puutteelliseksi. Tämän vuoksi sovellukset eivät kyenneet kommunikoimaan keskenään. Korkean saatavuuden soveltaminen ei käytännössä onnistunut, koska ilman määritettyä ympäristöä ei pystytty kokeilemaan tilannetta, jossa Kubernetes olisi ylläpitänyt useampaa sovelluskopioita kerralla. Lokitietoja pystyttiin seuramaan aina kulloisenkin käynnissä olevan Podin osalta, mutta tietoja ei tallennettu mihinkään fyysiseen tallennustilaan.

11 POHDINTA JA YHTEENVETO

Työn tavoitteena oli tutkia varastonhallintajärjestelmän tuotantoympäristön automatisoinnin kehittämistä Docker- ja Kubernetes-työkalujen avulla. Työ aloitettiin kirjallisuuden etsimisellä tutkimustyön aiheista ja saatavilla olevien materiaalien tutkimisella. Alkuvaiheessa paneuduttiin selvittämään työssä käytettävien työkalujen soveltuvuutta toimeksiantajan tuotantoympäristöön, eli perehdyttiin tarkemmin tuotantoympäristön rakenteeseen ja tutustuttiin työkalujen sisältämiin toimintoihin. Tämän jälkeen kirjoitettiin työn teoriaosuus, ja samalla ymmärrys käytännön työhön tarvittavien työkalujen toimintaan syventyi.

Teoriaosuuden jälkeen asennettiin käytettävät työkalut omaan virtuaalikoneympäristöön ja aloitettiin käytettävien levykuvien määrittäminen. Itse käytännön tutkimustyönosuus toteutettiin vaiheittain, ensiksi tuotantoympäristö määritettiin Docker-työkalun avulla ja tämän jälkeen sama pyrittiin toteuttamaan Kubernetes-työkalulla. Lopputuloksena saatiin tuotantoympäristö määritettyä Dockerilla, jossa kaikki palvelut toimivat ja kykenivät kommunikoimaan keskenään. Yhtenä poikkeuksena tähän oli käyttöliittymän API-kontin websocket-kommunikointi. Kubernetes-työkalulla päästiin vain osittain tavoitteeseen, sillä tuotantoympäristön sovelluksia ei saatu määritettyä kommunikoimaan keskenään.

Työn loppuun saattaminen vaatisi muutaman palaverin toimeksiantajan kanssa ja arviolta kuukauden lisää aikaa tuotantoympäristön määrittämiseen, lokitietojen keräämiseen ja versionhallintatyökalujen käyttöönoton toteuttamiseen. Yksi osasyy tavoitteiden savuttamattomuuteen oli myöhästynyt työn aloitusajankohta. Työ suunniteltiin aloitettavaksi vuoden alusta, jolloin aikaa työn tekemiselle olisi ollut useampi kuukausi enemmän. Alkuvuoden opinnot kuitenkin sotkivat alustavat aikataulusuunnitelmat, joten todellisuudessa koko työn tekemiseksi oli aikaa vain noin kaksi kuukautta.

Toimeksiantaja pystyy hyödyntämään tämän työn tutkimusosuutta arvioidessaan työkalujen soveltuvuutta tuotantoympäristönsä toteuttamiseksi. Työssä käytetyt sovellusinstanssit olivat raskaita suorittaa, koska jokainen erillinen virtuaaliympäristö vaatii laitteistolta resursseja toimiakseen. Näin ollen isäntälaitteiston suorituskyky heikkeni huomattavasti konttien käyttöönoton myötä, mikä heijastui verkkoselaimen toimintaan hidastumisena. Käytännössä koneelta vaadittavat resurssit tulisivat tällaisella käytöllä kasvamaan, mikä ei ole tavoiteltu ominaisuus. Lisäksi käytettävät levykuvat kasvoivat suuriksi, joten monistaessa niistä useampia sovelluskokonaisuuksia, tulee resurssien tarve kasvamaan entisestään.

Kubernetes-klusterin käyttötarkoitus toteutuu parhaiten useamman fyysisen koneen klusterikokonaisuudessa, jota varten se on luotu. Silloin useamman eri sovelluksen hallinnoiminen tehostuu, kuten esimerkiksi mikropalveluiden ylläpitämiseen, missä Kubernetes-orkestrointisovelluksen hallinnoimia sovelluksia voi olla kymmeniä tai satoja. Yhden koneen klusterista ei tähän käyttötarkoitukseen saada tarpeeksi tehokasta ratkaisua, eikä myöskään saavuteta toimintaympäristön saatavuutta parantavaa ratkaisua.

Työn käytännönosuutta yritys voisi hyödyntää pienten muutosten myötä, mutta on tarpeen huomioida työkalujen käyttötarkoitus, joka soveltuu paremmin kehitysvaiheeseen kuin tuotantoympäristön palveluiden suorittamiseen. Toimeksiantajalle luovutetaan tarkempi sisäinen dokumentti toteutetun kokonaisuuden käyttämiseksi.

Tulevaisuudessa olisi mielenkiintoista selvittää, voiko kolmansien osapuolien kehittämiä työkaluja käyttää tämän työn jatkokehittämisessä. Esimerkiksi tietokannan paikallisen tallennustilan käytön sijasta voitaisiin käyttää Microsoft Azure -pilvitallennuspalvelua, tai vaihtoehtoisesti levykuvien ja tietokannan tallennustilana voitaisiin käyttää myös Amazon-pilvipalveluita.

Lisäksi Docker-levykuvien versionhallinnan osalta olisi mielenkiintoista soveltaa tähän työhön avoimen lähdekoodin CI/DI-käännösautomaation palvelinohjelmistoa nimeltä Jenkins. Jenkins on jatkuvan integraation työkalu, eli käytännössä se automatisoi levykuvien kääntämisen ja tallentamisen rekisteriin. Työkalu vertailee tallennettuja sovellusversioita keskenään ja havaitessaan muutoksia se suorittaa ohjelmoidut prosessit uuden levykuvaversion tallentamiseksi.

Työn Kubernetes-osuuden kehittämiseksi olisi mielenkiintoista selvittää työkalua nimeltä Skaffold. Skaffold on CI/DI jatkuvan integraation komentorivityökalu, mutta suunniteltu käytettäväksi Kubernetes-pohjaisten sovelluksien tukena. Skaffold-työkalua ohjataan syöttämällä komennot komentokehotteeseen. Työkalu suorittaa Kubernetes-artefaktien rakentamisen, testauksen, tallentamisen ja käyttöönoton. Skaffold määritetään YAML-tiedoston avulla.

Kokonaisuudessa työ olisi saattanut valmistua nopeammin, mikäli lähtötaso olisi ollut korkeampi työkalujen käytön ja tietotaidon osalta. Lisäksi tiiviimpi yhteistyö toimeksiantajan kanssa olisi saattanut nopeuttaa työn valmistumista. Työn käytännönosuudessa ongelmatilanteiden selvittelyyn ja ratkaisuun kului eniten aikaa. Annettuun haasteeseen tartuttiin kuitenkin ennakkoluulottomasti ja työssä pyrittiin kaikkia osapuolia hyödyttävään ratkaisuun.

LÄHTEET

ActualTech Media. (12.4.2021). *The Top 5 Enterprise Type 1 Hypervisors You Must Know.* https://www.actualtechmedia.com/io/top-5-enterprise-type-1-hypervisors/

Baier, J., Sayfan, G., & White, J. (20.5.2019). *The Complete Kubernetes Guide : Become an Expert in Container Management with the Power of Kubernetes.* Packt Publishing.

Bernstein, D. (9.2014). Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing (1)*3, 81–84. https://doi.org/10.1109/MCC.2014.51

Brendan, B., Beda, J., & Hightower, K. (4.10.2019). *Kubernetes: Up and Running* (2 p.). O'Reilly Media.

Conroy, S. (25.1.2018). *History of Virtualization.* https://www.idkrtm.com/history-of-virtualization/

Docker Inc. (i.a.-a). *Develop with Docker Engine API*. https://docs.docker.com/engine/api/

Docker Inc. (i.a.-b). *Docker overview.* https://docs.docker.com/get-started/overview/

Docker Inc. (i.a.-c). *Docker registry.* https://docs.docker.com/registry/

Elhage, N. (12.11.2019). *The architecture of declarative configuration management.* https://blog.nelhage.com/post/declarative-configuration-management/

Heck, J. (6.4.2018). *Kubernetes for Developers : Use Kubernetes to Develop, Test, and Deploy Your Applications with the Help of Containers.* Packt Publishing.

Heino, P. (2010). *Pilvipalvelut*. Talentum.

Kane, S., & Matthias, K. (2015). *Docker: Up and Running: Shipping Reliable Containers in Production*. O'Reilly Media.

Kane, S., & Matthias, K. (2018). *Docker: Up & Running: Shipping Reliable Containers in Production* (2 p.). O'Reilly Media.

Khare, N. (6.2015). *Docker Cookbook.* Packt Publishing.

Kubernetes. (23.7.2021-a). *What is Kubernetes?* https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Kubernetes. (20.12.2021-b). *Secrets.* https://kubernetes.io/docs/concepts/configuration/secret/

Kubernetes. (6.3.2022-a). *ReplicationContoller.* https://kubernetes.io/docs/concepts/workloads/controllers/replicationcontroller/

Kubernetes. (2.4.2022-b). *ConfigMaps.* https://kubernetes.io/docs/concepts/configuration/configmap/

Krochmalski, J. (11.2016). *Developing with Docker.* Packt Publishing.

Lukša, M. (2018). *Kubernetes in Action.* Manning Publications.

Lukša, M. (2020). *MEAP Edition: Kubernetes in Action* (2 p.). Manning Publications.

Microsoft. (15.2.2022). *Get started with Docker remote containers on WSL 2.* https://docs.microsoft.com/en-us/windows/wsl/tutorials/wsl-containers

OpenJs Foundation. (i.a.). *About Node.js.* https://nodejs.org/en/about/

Pesmel Oy. (i.a). *About Us.* https://pesmel.com/about-us/

Pethuru, R. (6.2015). *Learning Docker.* Packt Publishing.

Portnoy, M. (29.8.2016). *Virtualization essentials.* John Wiley & Sons, Incorporated.

Portus. (i.a.). *Claim control of your Docker Images.* http://port.us.org/

Saito, H., Lee, H., & Hsu, K. (30.5.2018). *Kubernetes Cookbook : Practical Solutions to Container Orchestration,* 2nd Edition. Packt Publishing.

Sayfan, G. (27.4.2018). *Mastering Kubernetes : Master the Art of Container Management by Using the Power of Kubernetes,* 2nd Edition. Packt Publishing.

Schenker, G. (26.4.2018). *Learn Docker - Fundamentals of Docker 18. x : Everything You Need to Know about Containerizing Your Applications and Running Them in Production.* Packt Publishing.

Software Testing Help. (3.3.2022). *Top 10 Most Popular Virtualization Software.* https://www.softwaretestinghelp.com/virtualization-software/

Taylor, A. (3.9.2013). *SQL for Dummies.* John Wiley & Sons.

TechTarget. (5.2012). *Portable app.* https://www.techtarget.com/searchcloudcomputing/definition/portable-app#:~:text=A%20portable%20application%20(portable%20app,one%20computing%20environment%20to%20another.&text=Because%20files%20and%20data%20are,host%20operating%20system%20(OS

The Apache Software Foundation. (i.a.). *Flexible & Powerful Open Source Multi-Protocol Messaging.* https://activemq.apache.org/

Turnbull, J. (26.8.2019). *The Docker Book: Containerization is the new virtualization* (v18.09.2 : c2c5fa8). James Turnbull.

VMware Inc. (11.3.2008). *Understanding Full Virtualization, Paravirtualization, and Hardware Assist.* https://www.vmware.com/techpapers/2007/understanding-full-virtualization-para-virtualizat-1008.html

Vohra, D. (2016). *Pro Docker.* Apress.

Wallenius, N. (23.2.2022-a). *Mikä on Docker ja mitä hyötyä siitä on?* https://niklaswallenius.fi/mika-on-docker/

Wallenius, N. (23.2.2022-b). *Konttiteknologia – mitä kontit ovat ja mitä hyötyä niistä on?* https://niklaswallenius.fi/konttiteknologia-mita-hyotya/

Williams, N. (10.3.2014). *Professional Java for Web Applications.* John Wiley & Sons.

LIITTEET

Liite 1. Ont-compose.yaml -tiedosto.

Liite 2. Docker-konttien lokitietoja käynnistysvaiheessa.

Liite 1. Ont-compose.yaml -tiedosto.

```
version: '3'
services:
  ont-node:
    image: docker.io/joketti/ont:ui
    labels:
        kompose.image-pull-policy: IfNotPresent
        kompose.image-pull-secret: regcred
    ports:
      - "9000:9000"
      - "8080:8080"
    links:
      - ont-activemq
      - ont-ui-api
      - ont-wms-rulla
  ont-mssql:
    image: docker.io/joketti/ont:mssql
    labels:
        kompose.image-pull-policy: IfNotPresent
        kompose.image-pull-secret: regcred
        kompose.volume.size: 2Gi
    # Define volume first then remove comment
    #volumes:
    #    - 'mssqlvolume:/var/opt/mssql-vol'
    networks:
      - ont-network
    ports:
      - "1433:1433"
    # Remove comment if used base and comm
    #links:
    #  - ont-comm
    #  - ont-base

  ont-wms-rulla:
    networks:
      - ont-network
    image: docker.io/joketti/ont:wms-rulla
    labels:
        kompose.image-pull-policy: IfNotPresent
        kompose.image-pull-secret: regcred
    environment:
       DATABASE_HOST: ont-mssql
       ACTIVEMQ_HOST: ont-activemq
    ports:
      - "8079:8079"
      - "1431:1433"
    links:
      - ont-mssql
    depends_on:
      - ont-mssql

  ont-ui-api:
    image: docker.io/joketti/ont:ui-api
    labels:
        kompose.image-pull-policy: IfNotPresent
        kompose.image-pull-secret: regcred
    networks:
```

```
      - ont-network
    environment:
       DATABASE_HOST: ont-mssql
       ACTIVEMQ_HOST: ont-activemq
    depends_on:
      - ont-mssql
    ports:
       - "8081:8081"
       - "1434:1433"
    links:
      - ont-mssql
  # Remove comments if used
  #ont-comm:
  #  image: docker.io/joketti/ont:comm
  #  labels:
   #     kompose.image-pull-policy: IfNotPresent
   #     kompose.image-pull-secret: regcred
   # ports:
   #    - "8083:8083"
   # networks:
   #   - ont-network
  #ont-base:
  #  image: docker.io/joketti/ont:base
   # labels:
   #     kompose.image-pull-policy: IfNotPresent
   #     kompose.image-pull-secret: regcred
   # networks:
   #   - ont-network
  ont-db-migration:
    networks:
      - ont-network
    image: docker.io/joketti/ont:db-migration
    labels:
        kompose.image-pull-policy: IfNotPresent
        kompose.image-pull-secret: regcred
    depends_on:
      - ont-mssql
    links:
      - ont-mssql
  ont-activemq:
    image: docker.io/joketti/ont:activemq
    labels:
        kompose.image-pull-policy: IfNotPresent
        kompose.image-pull-secret: regcred
    ports:
      - "8161:8161"
      - "61616:61616"
      - "61613:61613"
      - "5672:5672"
      - "1883:1883"
    networks:
        - ont-network
    links:
      - ont-ui-api
      - ont-wms-rulla
networks:
  ont-network:
    driver: bridge
```

Liite 2. Docker-konttien lokitietoja käynnistysvaiheessa.