Opinnäytetyö (AMK)

Tietojenkäsittely

2019

Antti Leskinen

MIKROPALVELU-ARKKITEHTUURIN KÄYTTÄMINEN PILVIPALVELUSSA

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely

2019 | 54 sivua

Antti Leskinen

MIKROPALVELUARKKITEHTUURIN KÄYTTÄMINEN PILVIPALVELUSSA

Perinteisesti sovellukset tehdään yhtenä kokonaisuutena. Tätä kokonaisuutta kutsutaan monoliittiseksi arkkitehtuuriksi. Monoliittisessa arkkitehtuurissa yksi virhe sovelluksessa voi estää koko sovelluksen toiminnan.

Mikropalveluarkkitehtuurissa sovellus tehdään monen itsenäisen palvelun avulla. Mikropalveluarkkitehtuurilla pyritään minimoimaan monoliittisen arkkitehtuurin ongelmaa, sillä mikropalveluarkkitehtuurissa virheen sattuessa se vaikuttaa vain kyseiseen palveluun koko sovelluksen sijasta, jolloin muu sovellus jatkaa toimintaansa normaalisti. Mikropalveluarkkitehtuurissa pitää ottaa huomioon sovelluksen eri osien vaikutuksia kokonaisuuteen ennen käyttöönottoa.

Opinnäytetyön tarkoitus on kertoa, mitä teknologioita ja sovelluksia voidaan käyttää mikropalvelua tehtäessä. Olennaisina asioina ovat konttiteknologia ja pilvipalvelut. Konttiteknologia on kasvattanut Dockerin myötä suosiota ja helpottaa mikropalvelujen tekemistä sijoittamalla sovelluksen osat hallittaviin itsenäisiin kontteihin. Lisäksi opinnäytetyön tavoitteena on tehdä toimiva yksinkertainen sovellus hyödyntäen konttiteknologiaa, Azure-pilvipalvelua sekä Kubernetes-kontinhallinta-alustaa. Azure-pilvipalvelu helpottaa mikropalveluarkkitehtuurin tekemistä hallitsemalla mikropalvelujen resursseja, joista maksetaan vain käytön mukaan.

Opinnäytetyön tuloksena on kokonaisuus mikropalveluarkkitehtuurin käytöstä ja huomioitavista asioista mikropalveluarkkitehtuurin käyttöönotossa. Tuloksena kehitettiin toimiva mikropalvelu, joka integroitiin Azure-pilvipalveluun onnistuneesti ja jota kutsuttiin julkisesti internetosoitteen avulla.

ASIASANAT:

mikropalveluarkkitehtuuri, mikropalvelu, konttiteknologia, pilvipalvelu, kontinhallinta

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business information technology

2019 | 54 pages

Antti Leskinen

USING MICROSERVICES ARCHITECTURE WITH A CLOUD SERVICE

Applications are created traditionally as one wholeness. This wholeness is called monolithic architecture. In the monolithic architecture, one mistake in the application can prevent the operation of the whole application. On the contrary, application architecture in microservices is created with the help of many independent services. Microservices architecture attempts to minimise the problem of the monolithic architecture because when a mistake happens in the microservices architecture, it will affect only the service in question whilethe rest of the application continues its operation normally. In the microservices architecture, the impact of the independent services to the wholeness of the application must be planned before implementation.

The purpose of the thesis was to investigate what technologies and applications can be used when creating a microservice. The essential matters are container technology and cloud services. The container technology expanded in popularity with the usage of Docker and facilitates the microservices by placing parts of the application in independent containers to be commanded. The Azure cloud service facilitates the microservices architecture by controlling the resources of the microservices which are paid only according to the usage. The company does not need to buy expensive resources such as servers for its own use.

The objective of the thesis was to make a functional simple application utilising container technology, Azure cloud service and Kubernetes container management. For the theoretical part of the thesis, the material was collected from numerous internet sources.

The objective of the thesis was achieved. The functional microservice was developed as a result and was successfully integrated into the Azure cloud service and it was publicly called from an Internet address.

KEYWORDS:

microservices architecture, microservice, container technology, cloud service, container management

SISÄLTÖ

KUVAT

KÄYTETTY SANASTO

Monoliittinen arkkitehtuuri	Yksi iso kokonaisuus, johon sovellus perustuu (Rouse ym. 2018c).
Mikropalveluarkkitehtuuri	Sovellus pilkotaan pieniin ja itsenäisiin palveluihin, mistä sovelluksen kokonaisuus koostuu (Rouse ym. 2018c).
Kontti	Kokoelma kirjastoja, sovelluksia ja konfiguraatiotiedostoja, mitkä paketoidaan konttiin ja se toimii aina samalla tavalla, vaikka ympäristö vaihtuisi kehittäjän tietokoneelta testi- tai tuotantoympäristöön (Rubens 2017).
Palvelu	Pieni yksinkertainen sovellus, mikä tekee vain tiettyä tehtävää (Rouse ym. 2018c).
Virtuaalikone	Virtuaalinen tietokone simuloi oikeaa tietokonetta käyttöjärjestelmineen kaikkineen. Mahdollistaa monen tietokoneen suorittamisen käyttäjän tietokoneen laitteiston avulla (Bauer 2018).
Docker	Avoimen lähdekoodin sovellus, mikä hyödyntää kontteja ja niissä käytettäviä teknologioita (Opensource 2019).
Azure	Microsoftin julkaisema pilvessä eli internetin välityksellä toimiva alusta, missä on joukko pilvipalveluja yritysten tarpeisiin (Martin 2014).
Kubernetes	Konttien hallintaan tarkoitettu alusta (Google 2019a).
Nodejs	Mahdollistaa JavaScript-ohjelmointikielen käyttämisen erillisenä sovelluksena pelkän selaimen sijaan (Patel 2018).
RBAC	Role-based access control. Roolipohjainen pääsynhallinta Azuressa (Lyon ym. 2019).

1 JOHDANTO

Perinteinen tapa tehdä sovelluksia on tehdä yksi iso kokonaisuus, johon sovellus perustuu. Tätä kutsutaan nimellä monoliittinen arkkitehtuuri. Tällaisessa arkkitehtuurissa yksi virhe sovelluksessa voi estää koko sovelluksen toimimisen. Mikropalveluarkkitehtuurissa pyritään ratkaisemaan kyseinen ongelma tekemällä sovellus useammista pienistä palveluista, jolloin yksi virhe palvelussa ei estä koko palvelun toimimista.

Opinnäytetyössä käydään läpi, mitä mikropalveluarkkitehtuuri on, mitä asioita mikropalveluarkkitehtuurin käyttöönotossa kannattaa ottaa huomioon, miksi mikropalveluarkkitehtuuri kannattaa ottaa käyttöön sovellusta tehtäessä ja mitä tekniikoita ja sovelluksia mikropalveluarkkitehtuurin hyödyntämisessä voidaan käyttää. Opinnäytetyön tarkoitus on kertoa mikropalveluarkkitehtuurista ja siihen liittyvien tekniikoiden hyödyntämisestä aina käyttöönottoon asti tekemällä toimiva mikropalvelu. Opinnäytetyön toimeksiantaja on Tieto Finland Oy. Tieto on yksi Suomen suurimpia ohjelmistoyrityksiä.

Mikropalveluarkkitehtuurilla sovelluksista halutaan tehdä modulaarisia, joita voidaan käyttää myöhemmin uudestaan. Opinnäytetyön aihe on valittu ajankohtaisuuden ja monipuolisuuden takia. Mikropalveluarkkitehtuuri voidaan ottaa käyttöön yrityksen koosta riippumatta.

Teoriaosuudessa käydään läpi mikropalveluarkkitehtuurin lisäksi, mitä Docker on ja miten Docker liittyy kontteihin, mitä kontit ovat ja miksi niitä kannattaisi käyttää sovellusta tehtäessä, mitä Kubernetes on ja miten se liittyy Dockeriin ja kontteihin, mitä Azure-pilvipalvelu on, miten Azurea hyödyntämällä voidaan tehdä mikropalveluarkkitehtuurin tekemisestä helpompaa ja turvallisempaa yhdistämällä kaikkia yllämainittuja sovelluksia ja tekniikoita Azuren avulla.

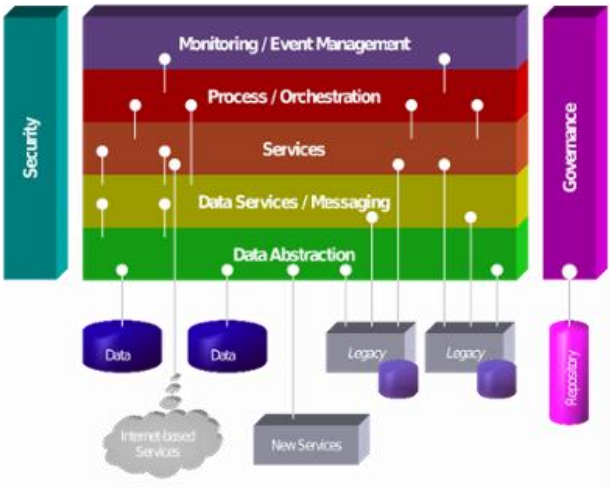
Käytännön osuudessa tehdään mikropalvelu, joka liitetään Docker-konttiin, joka liitetään Azure-pilvipalveluun. Tämän jälkeen kontti liitetään Kubernetesiin, minkä jälkeen mikropalvelua kutsutaan julkisesti internetosoitteella. Mikropalveluarkkitehtuuri on kasvattanut suosiotaan viime vuosina www-sovelluksia tehtäessä Docker-konttien ja pilvialustojen, kuten Azuren, ansiosta.

2 MIKROPALVELUARKKITEHTUURI

Tässä luvussa käydään läpi mitä mikropalveluarkkitehtuuri on, mikä on mikropalveluarkkitehtuurin historia ja mistä se on peräisin, mikropalveluarkkitehtuurin määritelmä, sekä miten mikropalveluarkkitehtuuria käyttöönotettaessa pitäisi sovelluksen tekemistä ajatella pilkkomalla sovellus pieniin osiin uusiksi palveluiksi.

2.1 Mikropalveluarkkitehtuurin historia

Palvelukeskeinen arkkitehtuuri oli suosittua 2000-luvun alussa, jos haluttiin tehdä sovellus, missä palvelut ja itse sovellus olivat eriytetty toisistaan. Palvelukeskeisessä arkkitehtuurissa on tarkoituksena tehdä sovellus omana kokonaisuutena ja siihen liittyvät palvelut omana kokonaisuutena, mitkä keskustelevat keskenään, joita kutsutaan sovelluksessa tarvittaessa. (Wright 2019). Kuvassa 1 on palvelukeskeisen arkkitehtuurin rakennemalli.



Kuva 1. Palvelukeskeinen arkkitehtuuri (Corbasson 2007).

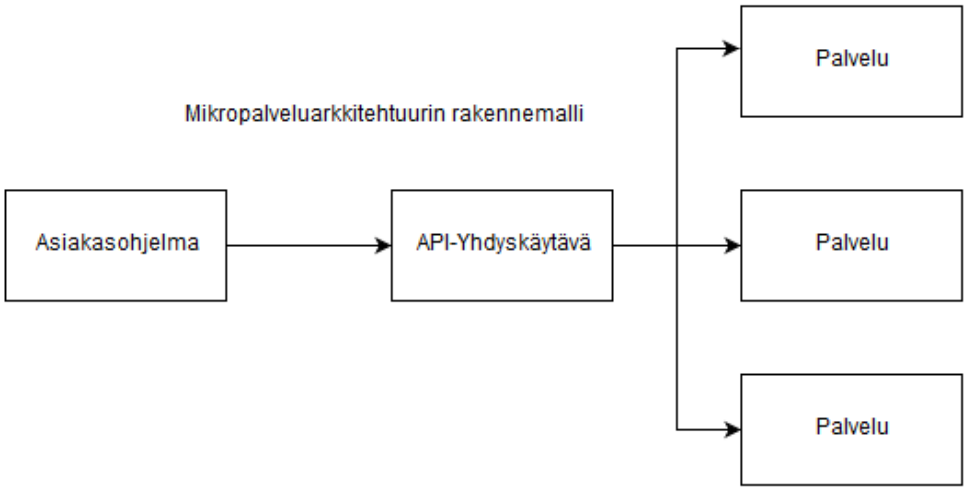
Palvelukeskeisen arkkitehtuurin ongelmana oli, että yhden palvelun sisällä voi olla monta pienempää taustalla olevaa palvelua. Jos joku taustalla oleva palvelu lakkasi toimimasta. Samalla lakkasi koko palvelu toimimasta, vaikka palvelun muut osat toimisivat hyvin. (Wright 2019). Mikropalveluarkkitehtuuri eroaa palvelukeskeisestä arkkitehtuurista, siinä, että jokaisella palvelun sisällä ei ole pienempiä palveluita ja palvelut eivät saa riippua toisistaan.

Palvelukeskeisen arkkitehtuurin kehityksen tuloksena syntyi mikropalveluarkkitehtuuri. Termi mikropalvelut tuli yleiseen tietoisuuteen vuonna 2011 sovellusarkkitehtien konferenssissa. (Mauersberger 2017)

Netflix (Netflix.com) on ensimmäisiä yhtiöitä, jotka ottivat mikropalveluarkkitehtuurin käyttöön. Netflix on harvoja mikropalveluarkkitehtuurin menestystarinoita ja menestyksen ansiosta Netflix on julkaissut monia palveluita avoimena lähdekoodina. (SmartBear Software 2015). Tämä auttaa monia mikropalveluarkkitehtuurin käyttöönotossa.

2.2 Mikropalveluarkkitehtuurin määritelmä

Mikropalveluarkkitehtuuri on erilainen lähestymistapa sovelluksen kehittämiseen. Mikropalveluarkkitehtuurissa sovellus pilkotaan pienempiin palveluihin, joita kutsutaan ison sovelluksen sijasta. Jokainen palvelu suorittaa vain tiettyä tehtävää ja käyttää yksinkertaista hyvin määritettyä rajapintaa muiden palveluiden kanssa kommunikoimiseen. (Rouse ym. 2018c). Kuvassa 2 on mikropalveluarkkitehtuurin rakennemalli.



Kuva 2. Mikropalveluarkkitehtuuri (Wasson ym. 2018c).

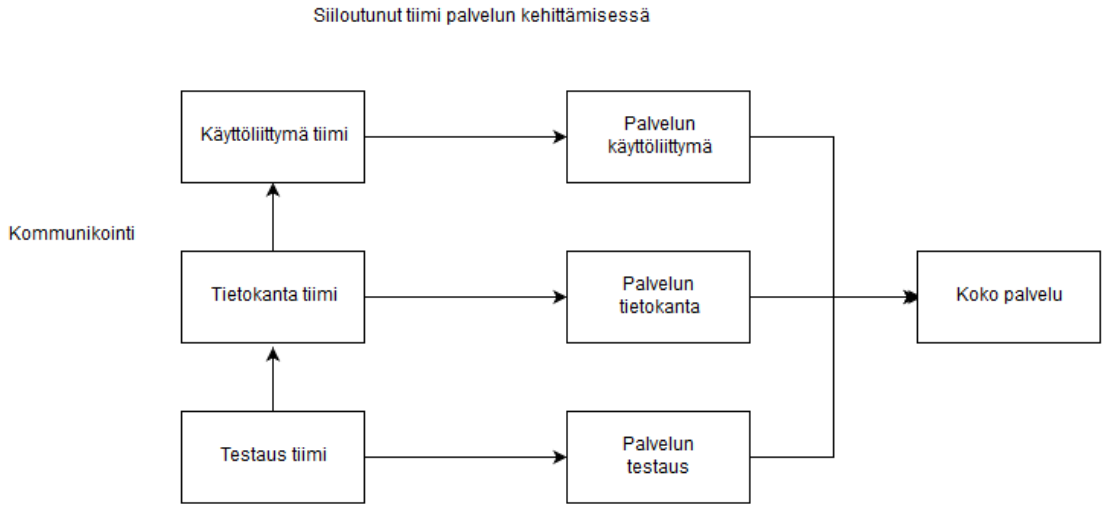
Rajapinnan avulla saadaan palvelun sisältämät tiedot helposti ja nopeasti saataville ilman, että tarvitsee erikseen kutsua kyseistä palvelua. (Rouse ym. 2018c)

Mikropalveluarkkitehtuurissa palvelu ei saa olla riippuvainen toisesta palvelusta. Palvelu vastaa oman tiedon hallinnasta ja sen säilymisestä. Tämä voidaan saavuttaa tekemällä jokaiselle palvelulle oma tietokanta. Palveluiden ei tarvitse olla tehty samalla teknologialla, koska jokainen palvelu on oma itsenäinen kokonaisuutensa. (Wasson ym. 2018c)

2.3 Mikropalveluarkkitehtuuri pilkkominen

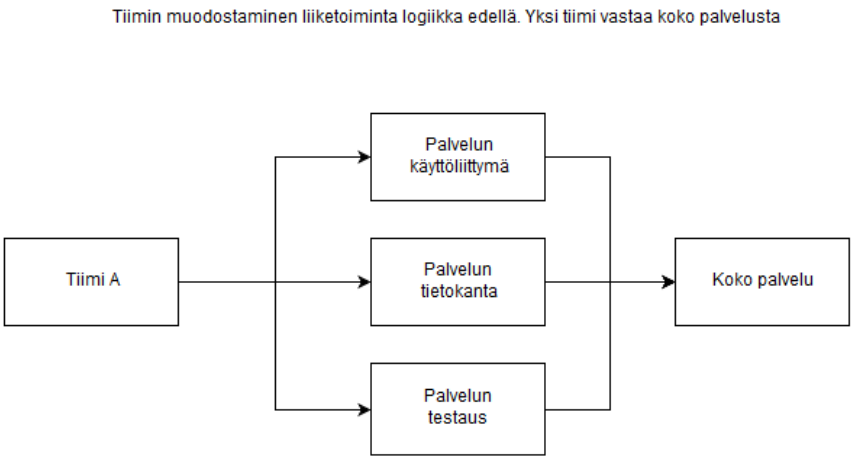
Mikropalveluarkkitehtuurin pilkkominen osiin voi olla hankalaa, jos ei tiedä mitä kriteereitä kannattaa käyttää sovelluksen pilkkomiseen. Pilkkomista voidaan ajatella perinteiseen tapaan, missä palvelu jaotellaan osiin niin, että eri tiimit tekevät tietyn osan palvelusta. Näitä osia voivat olla käyttöliittymä, tietokanta ja palvelinlogiikka. (Fowler & Lewis 2014)

Tässä tavassa ongelmana on siiloutuminen, jossa jokaisen pienen muutoksen takia, pitää ottaa yhteyttä toiseen tiimiin. Muutoksen tekeminen vie paljon aikaa ja jokainen tiimi on erillinen kokonaisuutensa ja muut tiimit eivät välttämättä tiedä, mitä kyseinen ryhmä tekee. Siiloutuminen tekee palvelun kehittämisestä hankalaa ja kallista. (Fowler & Lewis 2014). Kuvassa 3 on siiloutunut tiimi palvelun kehittämisessä.



Kuva 3. Siiloutunut tiimi (Fowler & Lewis 2014).

Palvelun pilkkominen kannattaa tehdä liiketoimintalogiikka edellä. Tällöin jokainen liiketoimintaan liittyvä asia on oma palvelu. (Fowler & Lewis 2014). Kuvassa 4 on tiimin muodostaminen liiketoimintalogiikka edellä, missä yksi tiimi vastaa koko palvelusta.



Kuva 4. Tiimin muodostaminen liiketoimintalogiikka edellä. (Fowler & Lewis 2014).

Yrityksellä voi olla esimerkiksi viisi tuotetta, jolloin jokainen tuote olisi oma palvelunsa, missä kukin tiimi pitää huolen palvelun kehityksestä ja ylläpidosta. Tämä vaatii tiimiltä enemmän, koska jokainen osa-alue oli oman tiimin alaisuudessa, mutta nyt yhden tiimin pitää hallita palvelun kokonaisuus. Tämä tekee palvelun kehittämisestä nopeampaa ja halvempaa, mutta samalla vaativampaa. (Fowler & Lewis 2014)

2.4 Mikropalveluarkkitehtuurin hyödyt ja haitat

Mikropalveluarkkitehtuurin hyödyt

Mikropalveluarkkitehtuurin hyödyt tulevat pääasiassa hyvästä suunnittelusta ja palvelujen pienuudesta, koska sen tuloksena palvelut ovat helppo ottaa käyttöön sovelluksessa. (Rouse ym. 2018c)

Sovellus koostuu monesta palvelusta, mitkä tekevät vain tiettyä asiaa. Palvelut vaativat vähemmän kehittämisaikaa yksinkertaisuuden myötä. Palveluiden ollessa itsenäisiä niitä voidaan käyttää muissa projekteissa tarpeen vaatiessa. (Rouse ym. 2018c)

Jokaisella palvelulla on oma tietokantansa, jos palvelu on pilvipalvelussa kuten esimerkiksi Azuressa, niin skaalautuminen hoidetaan automaattisesti käytön mukaan. (Wasson ym. 2018c)

Pienet tiimit ovat vastuussa koko palvelun elinkaaren kaikissa vaiheissa, mikä mahdollistaa innovaation paremmin ja niiden käyttöönoton olettaen, että tiimit saavat olla mahdollisimman itsenäisiä palvelun kehittämisessä. Sovelluksen jakaminen useampaan pieneen palveluun johtaa pa-

rempaan laatuun ja ymmärrettävyyteen, koska koodista tulee lyhyempi ja helpommin ylläpidettävä ja kehittäjien on helpompi työstää ja ajatella pientä kokonaisuutta, kuin ymmärtää kokonaisen sovelluksen arkkitehtuuria ennen kehittämisen jatkamista. (Rouse ym. 2018c)

Blogitekstissä Antti Toivanen sanoo, että mikropalvelu kannattaa tehdä silloin, jos tekee palvelun kokonaan alusta ja tarkoituksena ei ole käyttää muita valmiiksi löytyviä ohjelmistoja. Muita syitä tehdä mikropalveluja ovat riippumaton skaalautuvuus ja palvelun saaminen markkinoille nopeasti. (Toivanen 2018)

Mikropalveluarkkitehtuurin haitat

Mikropalveluarkkitehtuuri vaatii enemmän suunnittelua isoon sovellukseen verrattuna, koska jokainen palvelu on oma kokonaisuutensa ja palveluja pitää miettiä itse palvelun toiminnan suhteen ja miten palvelu keskustelee muiden palvelujen kanssa rajapinnan kautta. (Rouse ym. 2018c)

Palvelujen kokonaisuuden hallinta voi olla hankalampaa, koska palvelut voivat koostuva useista eri tekniikoista ja tekniikoiden hallinta voi olla tiimiltä hankalaa. Palveluilla on isompi viive kuin yhdellä isolla sovelluksella. Isolla sovelluksella voi olla yksi pyyntö, mutta mikropalveluarkkitehtuurissa jokainen palvelu on oma pyyntönsä. (Wasson ym. 2018c)

Jokaisen palvelun tietoa ja tiedon eheyttä on hankala varmistaa. Mikropalveluarkkitehtuuri sovelluksen toiminta voi olla hitaampaa, koska jokainen palvelu on oma prosessinsa, mikä kuluttaa palvelimen muistia ja tehoa. (Wasson ym. 2018c)

Palvelujen testaaminen on monimutkaisempaa. Yhden palvelun testaaminen voi olla helpompaa, koska testataan yhtä asiaa ja yhdistetään yhteen palvelimeen. (Wasson ym. 2018c)

Mikropalveluarkkitehtuuri voi olla turvattomampi väärin tehtynä, koska hyökkääjällä on enemmän mahdollisuuksia läpäistä sovelluksen tietoturva. Jos hyökkääjä pääsee yhteen palveluun sisään, niin samalla hän pääsee moneen palveluun samaan aikaan. Oikein tehty mikropalveluarkkitehtuurisovellus on turvallisempi, koska hakkeri pääsee vain yhteen palveluun. (Rouse ym. 2018c)

3 KONTTITEKNOLOGIA

Tässä luvussa käydään läpi mitä konttiteknologia on, miksi kontteja kannattaisi käyttää, miksi konteista on tullut niin suosittuja, miten Docker liittyy kontteihin ja miksi konteista puhuttaessa moni puhuu Dockerista, vaikka kontit ja Docker ovat eri asioita.

3.1 Määritelmä ja historia

Kontit ovat kokoelma kirjastoja, sovelluksia ja konfiguraatiotiedostoja, jotka paketoidaan konttiin ja se toimii aina samalla tavalla, vaikka ympäristö vaihtuisi kehittäjän tietokoneelta testi- tai tuotantoympäristöön. Konttiteknologian ideana on ratkaista sovelluskehityksessä tuttu ongelma, että ohjelma toimii omalla tietokoneella, mutta toisessa ympäristössä ei toimikaan eli miten saadaan ohjelma toimimaan luotettavasti myös muissa, kuin omassa ympäristössä. Tämän takia kontit paketoidaan samalla tavalla. Käyttöjärjestelmään liittyvistä asioista ei tarvitse suurimmassa osassa tapauksia välittää. (Rubens 2017)

Konttiteknologia on ollut olemassa 1970-luvulta asti. Vuonna 1979 julkaistiin Unix-versio 7:nteen chroot-järjestelmäkomento, mikä muutti juurihakemiston prosessin ja hakemiston aliprosessit uuteen paikkaan kiintolevyllä. Tästä alkoi konttiteknologia, mikä eristi prosessit ja niiden oikeudet omaksi kokonaisuudeksi. 2000-luvulle mentäessä oli julkaistu monta ohjelmaa, mitkä hyödynsivät konttiteknologiaa kuten Solaris Containers vuonna 2004, Open VZ vuonna 2005 ja LXC vuonna 2008, LXC oli ensimmäinen Linuxin konttihallintaohjelma ja se käyttää Linuxin kerneliä. (Osnat 2018). LXC mahdollisti virtualisoinnin käyttöjärjestelmätasolla, mikä teki mahdolliseksi monen Linux-ympäristön käytön samalla kernelillä samaan aikaan. (Marquez 2018)

Kuitenkin vasta vuonna 2013 julkaistu Docker teki konttiteknologiasta suositun. Se käytti aluksi LXC-teknologiaa konttien hallitsemisessa, mutta lopulta korvasi LXC:n omalla teknologialla nimeltään libcontainer. Dockerin suosion ansiosta konttien käyttäminen on yleistynyt ja moni puhuu tämän takia konteista tarkoittaessaan oikeasti Dockerin kontteja. (Osnat 2018)

Dockerin eroaa merkittävästi LXC:stä, sillä Dockerilla pystyi siirtämään kontin ympäristöstä toiseen helpommin. Dockerin oman teknologian ansiosta se paransi konttiteknologiaa jatkokehittämällä libcontaineriin muun muassa turvaprofiilit, verkkorajapinnat, palomuurisäännöt ja käyttäjäryhmät. (Marquez 2018)

3.2 Konttistandardi

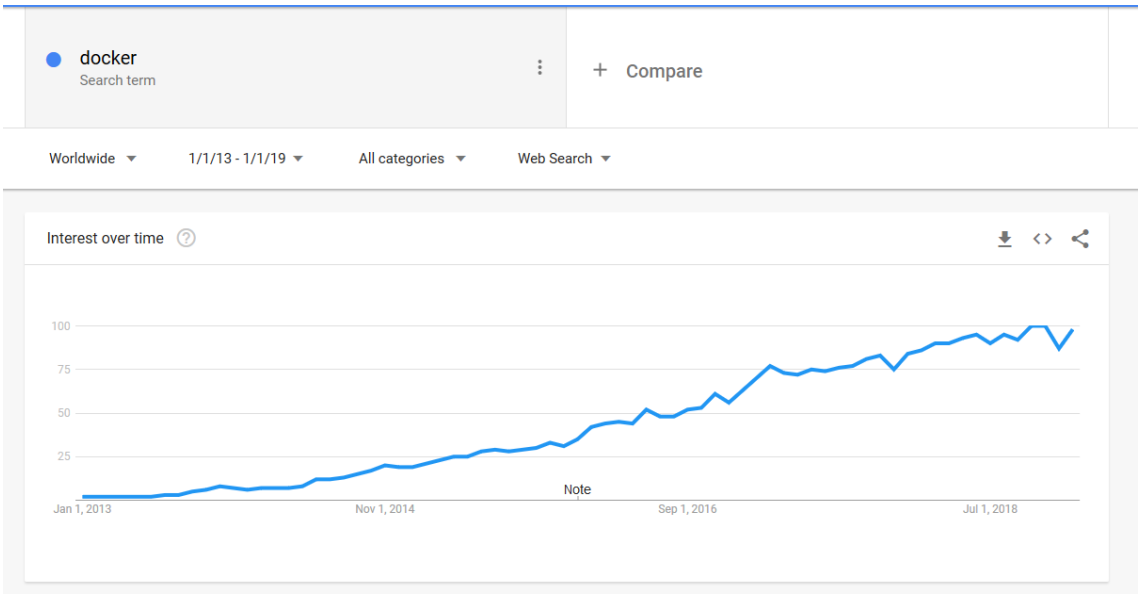
Mitä suositummaksi konttiteknologia on tullut, sitä enemmän kaivattiin yhteistä tapaa tehdä kontteja. CoreOS (coreos.com) tuotti oman tavan tehdä kontteja vuonna 2015, mitä kutsuttiin nimellä App Container Image (ACI). Tämän pelättiin syrjäyttävän Dockerin tavan tehdä kontteja. Jokaisella ohjelmalla oli oma tapansa tehdä kontti, jonka pelättiin johtavan konttiformaattisotaan. Tämä johtaisi konttiformaattien pirstaloitumiseen. (Rubens 2017)

Myöhemmin vuonna 2015 perustettin Open Container Project (Avoin kontti projekti, opencontainers.org) hillitsemään eri konttiformaattien syntyä ja siitä toivottiin yhteistä tapaa tehdä konttiformaattia. Open Container Initiative (entinen Open Container Project) tarkoituksena oli varmistaa, että konttiteknologian perusta kuten konttiformaatti oli standardisoitu, jolloin kaikki voivat hyödyntää niitä. (Rubens 2017)

Teknologian perustana käytettiin Dockerin teknologioita, jotta projekti saatiin alkuun. Konttistandardin hyötynä on, että yritykset eivät tuhlaa resursseja kilpailevien konttiteknologioiden tekemiseen vaan voivat keskittyä kehittämään sovelluksia, mitkä hyödyntävät olemassa olevaa konttiteknologiaa paremmin. Sovellukset voivat keskittyä esimerkiksi konttien turvallisuuteen ja konttien hallintaan. (Rubens 2017)

3.3 Konttiteknologian suosio

Konttiteknologian suosio on rajahtänyt muutaman vuoden aikana suosiossa Dockerin ansiosta. Dockerin suosio selittyy sen helppokäyttöisyydellä ja skaalautuvuudella pienelläkin teholla. (Swersky 2018). Kuvassa 5 on Google trendsin tietoja sanalle Docker.



Kuva 5. Google trends -kuvaaja hakusanalle Docker.

Ennen konttiteknologiaa palvelimia vuokrattiin tai ostettiin ja käytettiin paljon rahaa vain siihen, että sovellus pysyi toiminnassa kävijöiden kasvaessa. Nykyään ne voidaan korvata ostamalla pilvipalvelusta palvelimia tai käyttämällä kontteja tekemään vastaava murto-osalla kuluista ja laitteistosta. (Swersky 2018)

Docker aloitti avoimena lähdekoodina alusta alkaen, mikä teki sen käyttöönotosta nopeampaa. Kontit käynnistyvät ja pysähtyvät paljon nopeammin kuin virtuaalikoneet ja ovat siirrettävämpiä verrattuna virtuaalikoneisiin. Kontteja voi lisätä ja poistaa ympäristöstä nopeammin. Kontit sopivat hyvin mikropalveluarkkitehtuurille. (Tozzi 2017c)

Dockerin suosio jatkaa kasvamistaan, mitä enemmän yritykset tajuavat sen mahdolliset hyödyt automatisoinnin suhteen. Dockerin voi integroida jatkuvaan integrointi työkaluihin, mikä tekee prosessien automatisoinnista yrityksessä helpompaa.

3.4 Ero konttiteknologian ja virtuaalikoneen välillä

Virtuaalikoneen ja kontin välillä on tärkeä ero. Jokainen virtuaalikone joutuu simuloimaan koko käyttöjärjestelmän laitteistoineen ja ohjelmistoineen. Tämä vie yleensä useita gigatavuja tilaa. Virtuaalikoneessa voidaan määrittää, kuinka paljon tietokoneen tehoja käytetään virtuaalikoneen käynnissä pitämiseen nopeasti. (Bauer 2018)

Kontit jakavat tietokoneen käyttöjärjestelmän resursseja, jolloin kontin koko on useita megatavuja gigatavujen sijaan. Kontteja voi pitää useampaa samaan aikaan päällä viemättä kuitenkaan enemmän tilaa kuin virtuaalikone. Konteilla voi tehdä helpommin yhtenäisen sovellusympäristön, mitä pystyy siirtämään tietokoneesta toiseen vaivattomasti, koska kontit hoitavat sovelluksen toimimisen, eikä käyttäjän tai kehittäjän tarvitse itse muokata asetuksia saadakseen kontin toimimaan. (Bauer 2018)

Kontin toimiminen edellyttää samaa käyttöjärjestelmää. Kontteja kannattaa käyttää, jos sovelluksesta täytyy olla monta versiota samaan aikaan päällä, mutta haluat kuluttaa mahdollisimman vähän tilaa ja resursseja tietokoneelta.

3.5 Konttiteknologian hyvät ja huonot puolet

Konttiteknologian hyvät puolet

Konttiteknologian yksi hyvä puoli on, että ei tarvitse virtuaalikonetta kontin toimimiseen. Kontti toimii nopeammin kuin virtuaalikone.

Kontit mahdollistavat sovellusten julkaisemisen nopeammin, koska kontit ovat itsenäisiä ja siirrettäviä kokonaisuuksia, joissa on omat eristetyt kiintolevytilat. Sovelluksessa tarvittavat riippuvuudet laitetaan konttiin ja kontti toimii samalla tavalla omalla kuin muidenkin tietokoneilla. Tämä tekee konttien konfiguraatiosta helpompaa, koska kontin tarvitsee konfiguroida vain kerran ja se toimii kaikilla samalla tavalla.

Kontit voidaan yhdistää kontinhallintajärjestelmään kuten Kubernetesiin, mikä huolehtii useiden tuhansien konttien käynnissä pitämisestä ja valvonnasta. (Swersky 2018)

Konttiteknologian huonot puolet

Konttiteknologiaa tulee ymmärtää hyvin ennen sen käyttöönottoa. Aikaa tulee käyttää sen kouluttamiseen yrityksessä. (Swersky 2018)

Sovellus kannattaa ensin muuttaa kokoelmaksi mikropalveluja ja kokeilla, että kokonaisuus on hallinnassa ja toiminnassa.

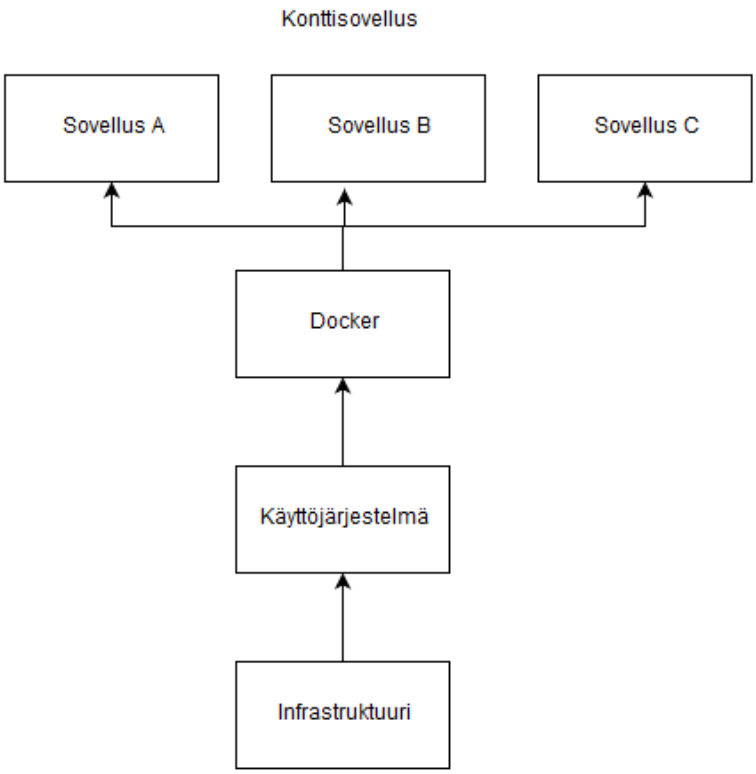
Konttiteknologiaa ei kannata ottaa käyttöön, jos yrityksen järjestelmät ovat yksinkertaisia ja konttien käyttöönotto tuottaisi liikaa ongelmia käyttöönoton tai monimutkaisuuden suhteen ja konttien käyttöönotolle ei ole varsinaista tarvetta. (Swersky 2018)

4 DOCKER

Tässä luvussa käydään läpi, mikä Docker sovellus on ja mitä sillä on tarkoitus tehdä, mistä asioista Docker koostuu ja miten Dockerin on tarkoitus toimia.

4.1 Määritelmä

Docker (docker.com) on vuonna 2013 julkaistu avoimen lähdekoodin sovellus, mikä tekee sovelluksen tekemisestä ja käyttöönotosta helpompaa käyttämällä kevyempää versiota virtualisoinnista, mitä kutsutaan kontiksi. Kontit auttavat sovelluksen kehittäjiä helpottamaan käyttöönottoa. Kehittäjä voi paketoida kaikki vaadittavat kirjastot ja riippuvuudet yhdeksi paketiksi, mikä julkaistaan Docker-konttina, jota kutsutaan myös Docker-imageksi. (Opensource 2019). Kuvassa 6 on Docker-kontin sovelluksen visualisointi.



Kuva 6. Docker-kontin sovelluksen visualisointi (Docker 2019h).

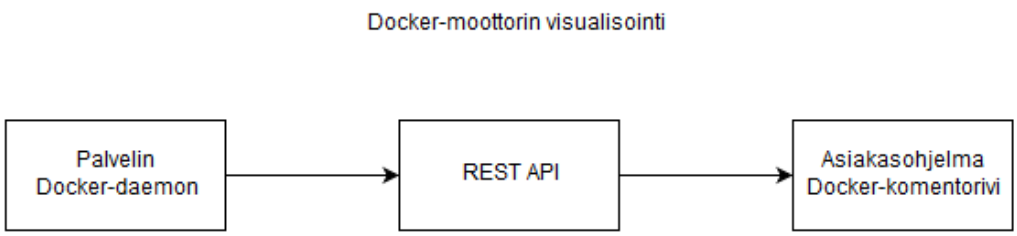
Kontin avulla kehittäjän ei tarvitse huolehtia jokaisesta käyttöjärjestelmästä tai tietokoneiden asetuksista, koska Docker hoitaa sen puolen automaattisesti. Docker on virtuaalikoneen kaltainen, mutta ei tee kokonaan uutta virtuaalikonetta, niin kuin tehtäisiin normaalin virtuaalikoneen kanssa. Docker käyttää kehittäjän tietokoneen tai palvelimen käyttöjärjestelmän ydintä (kernel) osana Docker-imagea. Käyttöjärjestelmän ydin ei vaadi kokonaista uutta käyttöjärjestelmää toimimiseen vaan pelkästään asioita, mitä ei löydy käyttöjärjestelmästä valmiiksi, kuten kehittäjän lisäämät kirjastot, mitä tarvitsee sovelluksen toimimiseen. Tämän ansiosta Docker-imagen koko vähenee ja teho paranee huomattavasti, koska ei tarvitse virtualisoida koko käyttöjärjestelmää. (Opensource 2019)

4.2 Dockerin toiminta

Docker koostuu Docker-moottorista, Docker-palvelimesta, Docker-asiakasohjelmasta, Docker-rekisteristä ja Docker-objekteista. Docker ja Docker-moottori perustuvat asiakasohjelma palvelin (client-server) arkkitehtuuriin. Arkkitehtuurissa Dockerin-asiakasohjelma keskustelee Docker-palvelimen kanssa. Docker-palvelin tekee kaiken raskaan työn liittyen Docker-konttien rakentamiseen, käynnissä pitämiseen ja jakamiseen. (Docker 2019c)

Docker-asiakasohjelma ja palvelin kommunikoivat keskenään käyttäen standardia ohjelma rajapintaa, jota kutsutaan REST APIksi. Tämän kaiken mahdollistaa Docker- moottori. Moottori koostuu kolmesta osasta, joista ensimmäinen on Docker-palvelin, jota kutsutaan Docker-palvelin prosessiksi. Tämä pyörii koko ajan taustalla Docker-kontin ollessa käynnissä. (Docker 2019c)

Toinen osa on REST API-ohjelma rajapinta, mikä kertoo, mitä rajapintoja ohjelmat voivat käyttää palvelimen kanssa kommunikoimiseen ja mitä oikealla voi tehdä. Kolmas osa on komentokehote rajapinta-asiakasohjelma, mitä käytetään palvelimen ja rajapinnan kommunikointiin, jotta saadaan Docker tekemään tarvittavat asiat sillä hetkellä. Asiakasohjelma pystyy kommunikoimaan useamman palvelimen kanssa. (Docker 2019c). Kuvassa 7 on Docker-moottorin visualisointi.



Kuva 7. Docker-moottori (Docker 2019c).

Docker-palvelin kuuntelee Dockerin-APIa eli rajapintaa pyyntöihin vastaamiseen ja Dockerin-objektien hallinnoimiseen. Palvelin voi myös kommunikoida toisten palvelimien kanssa, mitkä hallinnoivat Dockerin-palveluja. (Docker 2019c)

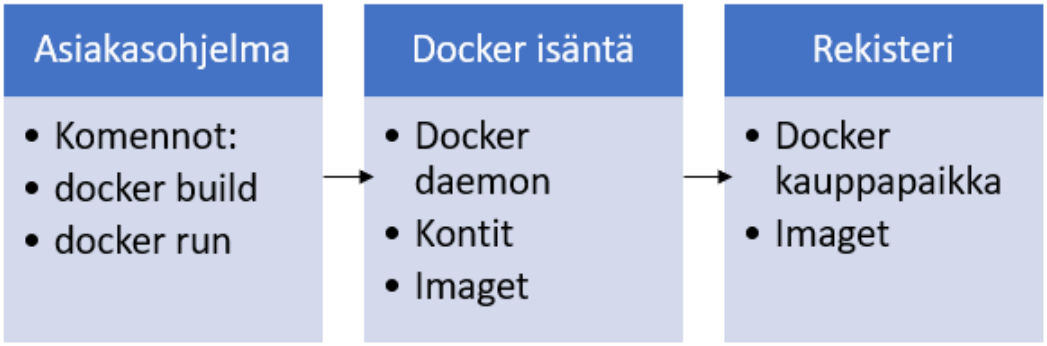
Dockerin rekisterit varastoivat Docker-imageja. Dockerin kauppapaikka (eng. Docker Hub) on julkinen rekisteri, mitä kaikki voivat käyttää tarvittaessa. Dockerin kauppapaikassa voi tarvittaessa ylläpitää yksityistä rekisteriä. Kauppapaikka käyttää samankaltaisia komentoja kuin muut versionhallinta ohjelmat kuten GitHub. Komento kuten ”Docker pull”, hakee uusimman version Docker-imagesta, mikä on saatavilla Dockerin kauppapaikassa sijaitsevassa rekisterissä. Docker oletuksena hakee haettavia imageja Dockerin kauppapaikasta, eikä sitä tarvitse erikseen konfiguroida. (Docker 2019c)

Docker-objektit ovat kaikki asiat Dockerissa, joilla on oma nimensä. Joka kerta kun tehdään tai käytetään jotain Dockerin toimintoa, sillä on oma Docker-objekti. Jokainen objekti on oma kokonaisuutensa ja sisältää objektin tiedot. (Docker 2019c)

Seuraavassa esitellään Docker-objektit image, kontti ja palvelu.

Image on vain luettavissa oleva runko, joka sisältää ohjeita Docker-kontin tekemisestä. Oman imagen tekemiseen vaaditaan Docker-tiedosto. Docker-tiedosto kertoo mitä vaiheita vaaditaan imagen tekemiseen ja käynnistämiseen. Jokainen ohje tiedostossa on oma kerros imagessa. Docker-tiedoston muuttuessa vain kyseiset kerrokset vaihtuvat tiedostossa koko tiedoston sijasta. Docker-image voi perustua toisiin imageihin, joissa on omia muokkauksia. (Docker 2019c)

Kontti on käynnistettävissä oleva instanssi imagesta. Kontin voi luoda, käynnistää, pysäyttää, siirtää toiseen kansioon tai poistaa käyttämällä Docker-asiakasohjelmaa. Kontin voi yhdistää useampaan verkkoon, johon kiinnittää vapaata kovalevytilaa tai siitä voi tehdä uuden imagen perustuen nykyiseen tilaan. Kontti määritellään siihen perustuvan imagen avulla ja omiin konfiguraatioihin perustuen. (Docker 2019c). Kuvassa 8 on Dockerin arkkitehtuurin visualisointi.



Kuva 8. Docker-arkkitehtuuri (Docker 2019c).

Palvelut auttavat useiden konttien hallitsemisessa jakamalla kontin useaan Docker-palvelimeen. Docker-palvelimet työskentelevät yhdessä joukkona. Jokainen joukon jäsen on oma palvelimensa. Kaikki palvelimet keskustelevat käyttämällä Dockerin APIa. (Docker 2019c)

Palvelu helpottaa esimerkiksi konttien tasaisessa latauskuorman hallinnassa, jos yhteen konttiin tulee enemmän pyyntöjä. Palvelu jakaa kuorman automaattisesti kaikkien konttien kesken, mikä vähentää hidastumisia liiallisen kuorman takia. (Docker 2019c)

4.3 Docker-imagen ja instanssin ero

Docker-image on kokoelma kirjastoja ja muita riippuvuuksia, mitä tarvitaan kontin toimimiseen, sitä voisi ajatella ohjelman asentajana. Docker-instanssi eli kontti on oma eristetty prosessi, mikä hyödyntää käyttöjärjestelmän ja Docker-imagen resursseja Docker-kontin käynnistyksessä. Instansseja voi olla useampia käynnissä samaan aikaan, mutta kaikki kontit käyttävät samoja käyttöjärjestelmän resursseja. Tämä voi hidastaa tietokonetta. Docker-instansseja on turvallista käyttää monta samaan aikaan, koska jokainen prosessi on eriytetty toisista instansseista ja tietokoneen omista prosesseista. Jokainen Docker-instanssi on oma yksittäinen kokonaisuutensa. Joka kerta, kun käynnistetään Docker-image, tehdään Docker-instanssi eli kontti. (Leoni 2018)

4.4 Dockerin hyvät ja huonot puolet

Dockerin hyvät puolet

Docker-kontit ovat luotu juuri yhden sovelluksen ajamiseen ja sen ajamiseen nopeasti ja turvallisesti. Virtuaalikoneet taas ajavat oman käyttöjärjestelmän joka kerta, missä voi olla monia ohjelmia käynnissä samaan aikaan, mikä hidastaa konetta. Docker-kontti on samanlainen joka kerta, vaikka vastaanottajalla olisi eri asetukset. Tämä takaa, että tuloksiin voi luottaa omassa koneessa ja muiden koneissa. (Kobos 2018)

Käyttöönoton ja konfiguroinnin helppous. Jokaisessa Docker-imagessa on Docker-tiedosto, mikä kertoo mitä image tekee missäkin vaiheessa ja sen konfigurointiin löytyy ohjeita Dockerin dokumentaatiosta. Dockerin kauppapaikasta voi ladata tarvittavia imageja. (Kobos 2018)

Docker-konttien eristäminen. Jokainen Docker-kontti on oma eristetty kokonaisuutensa, jolloin voi ajaa monta kertaa saman sovelluksen yhdessä palvelimessa. Yksinkertaistaen jokainen kontti on oma itsenäinen versio sovelluksesta. Tämä lisää turvallisuutta, koska jos vahinkoa syntyy, niin vahinko kohdistuu vain kyseiseen konttiin. (Kobos 2018)

Dockerin kauppapaikka on kätevä monien sovellusten jakamiseen. Sovelluksista on Docker-image tehtynä ja se säästää aikaa, jos sitä voi käyttää sellaisenaan tai pienin muokkauksin tehdä oman versionsa. Sovelluksen päivittäminen helpottuu, koska voidaan ladata viimeisin versio imagesta ja sen jälkeen varmistaa sovelluksen toimivuus ja turvallisuus. Tämän jälkeen sovellus voidaan päivittää ja muut voivat ladata uusimman toimivan version. Jos sovellusta ei löydy kauppapaikasta joutuu sovelluksen tekemään alusta asti itse kirjastoineen, ympäristöineen ja riippuvuuksineen. Tämä voi viedä paljon aikaa. (Kobos 2018)

Docker tekee versionhallinnasta ja jatkuvasta integroinnista helpompaa. Jatkuvalla integroinnilla voi automaattisesti tehdä uuden version imagesta aina kuin muutoksia on tehty sovellukseen, sen jälkeen voidaan laittaa imagelle tarvittavat tunnisteet ja ladata uusi versio Dockerin kauppapaikkaan. (Kobos 2018)

Docker tekee kehittämisestä helpompaa, koska voi olla oma kontti testaukselle ja tuotannolle. Testauksessa voidaan testata sovelluksen ominaisuuksia. Kun ne toimivat ne voidaan kopioida tuotantokonttiin. Tuotanto- ja testauskontit ovat itsenäisiä, koska ne eivät ole tietoisia toisistaan, vaikka käyttävät samaa sovellusta. (Kobos 2018)

Dockerin huonot puolet

Dockeria ei kannata käyttää, jos olemassa oleva sovellus on monimutkainen. Jos käyttäjä ei ole kaikkien oikeuksien ylläpitäjä. On vaikeampaa sovittaa uutta asiaa olemassa olevaan sovellukseen, koska siihen voi mennä enemmän aikaa kuin sovelluksen uudelleen tekemiseen. (Kobos 2018)

Docker toimii Linux-käyttöjärjestelmässä ilman virtualisointia. Windows- ja Mac-koneessa Docker toimii vain virtualisoinnin avulla, mikä on paljon hitaampi Linuxiin verrattuna ja kaikki asiat ei välttämättä toimi samalla tavalla. Jos sovellus on esimerkiksi verkkosovellus, missä palvelimet yleensä toimivat Linux-käyttöjärjestelmässä, ei ongelmia pitäisi syntyä. Pilvipalvelujen avulla ongelma on ratkaistu, kunhan tuki löytyy Docker-konteille. (Tozzi 2016a)

Dockerin oppiminen voi olla haastavaa, koska monet asiat eroavat virtuaalikoneista ja monimutkaisten konseptien hahmottaminen voi olla vaikeata kehittäjille. (Labrecque 2018)

Dockeria ei kannata käyttää, jos sovelluksen suorituskyky on tärkeää. Docker-kontit ovat nopeita, mutta sovelluksen ajaminen oikeassa käyttöjärjestelmässä kaikki resurssit varattuna sovelluksen pyörittämiseen ilman Dockeria on yleensä vielä nopeampaa. (Kobos 2018). Tätä vaihtoehtoa kannattaa käyttää vain silloin, jos on täysin varma, että sovellus toimii samassa ympäristössä ilman muutoksia useita vuosia ja sovellusta ylläpidetään aktiivisesti.

Turvallisuudessa on omat riskinsä, koska Docker käyttää tietokoneen omaa käyttöjärjestelmää konttien resurssien hallitsemiseen. Hyökkääjä voi pahimmassa tapauksessa ottaa tietokoneen hallintaan hyökkäämällä konttiin, mikä ei ole turvattu oikein. Hyökkääjä voi kaataa vain tietyn kontin tai kaikki kontit kerralla riippuen hyökkäyksen onnistumisesta. Virtuaalikoneen avulla voidaan lisätä turvallisuutta. Jos virtuaalikone hakkeroidaan, menetetään vain kyseinen virtuaalikone ja uusi voidaan laittaa pystyyn, kun vika on korjattu. (Runnable 2019)

Docker on tehty pääasiassa sovelluksilla, jotka vaativat pelkästään komentokehotetta. Graafista käyttöliittymää ei voida käyttää suoraan, koska sitä ei näytetä. (Tozzi 2017b)

5 KUBERNETES

Tässä luvussa käydään läpi mitä Kubernetes on, sen ominaisuudet, arkkitehtuuri ja hyvät ja huonot puolet.

5.1 Määritelmä

Kubernetes (kubernetes.io) on kokoelma palveluja, missä on ominaisuuksia kuten salaisien koodien hallinta, palvelujen paljastaminen julkisesti verkkoon, skaalautuminen eli uusien konttien lisääminen tai poistaminen käytön mukaan ja kuorman hallinnan tasapainottaminen. Kubernetes automatisoi sovelluksen päivittämisen. Kubernetes automaattisesti käynnistää uudelleen tai korvaa kontin sen epäonnistuessa. (Google 2019a)

Kubernetes julkaistiin avoimena lähdekoodina vuonna 2014 Googlen toimesta, jota ennen Kubernetes on ollut Googlen sisäisessä käytössä. Kubernetesia käytetään useiden konttien kuormien tasapainottamiseen, konttien toimintojen hallitsemiseen ja konttien palvelujen automatisointiin. Kubernetesin käyttöönottaminen vaatii sovelluksen laittamisen konttiin, koska Kubernetes on konttikeskeinen ympäristö. (Google 2019b)

Kubernetes on tehty mikropalveluarkkitehtuuria hyväksi käyttäen pilkkomalla Kubernetesin eri palvelut yksittäisiksi palveluiksi, mitkä voi ottaa käyttöön halutessaan. (Google 2019b)

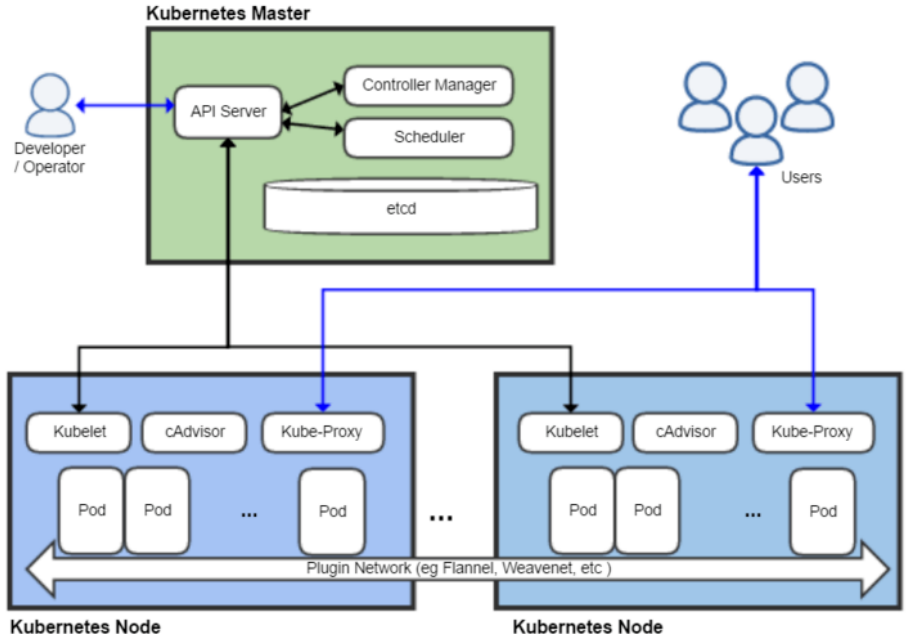
5.2 Kubernetesin arkkitehtuuri

Kubernetesissa kontit on käynnissä säiliöissä (pods). Säiliöitä käytetään konttien aikataulutukseen. Säiliöt sijaitsevat tietokoneella ja tietokoneen resursseja voi jakaa säiliöiden kesken. Kubernetes etsii tietokoneen, missä on tarpeeksi resursseja säiliön käynnissä pitämiseen ja käynnistää säiliöön liittyvät kontit. Ristiriitojen välttämiseksi jokaisella säiliöllä on oma uniikki IP-osoitteensa. (Rouse ym. 2017b)

Kubelet on pääsilmukka, joka hallitsee säiliöitä. Kubelet hoitaa säiliöiden käynnistämisen automaattisesti virhetilanteissa. Säiliöiden manuaaliseen hallintaan voidaan käyttää Kubernetesin rajapintaa. (Rouse ym. 2017b)

Replikaatiokontrolleri pitää huolen, että asetusten määrittämä säiliöiden lukumäärä on toiminnassa kaiken aikaa. Kontrollerilla voi myös tehdä uusia säiliöitä, kopioida säiliöitä ja skaalata säiliöitä. Kubernetes skaalaa kontrollerin avulla säiliöitä instanssien lukumäärän mukaan. (Rouse ym. 2017b)

Kubelet suorittaa Kubernetesin rajapintaa ja hallitsee klustereita. Klusterit ovat joukko kontteja. Kuvassa 9 selvennetään Kubernetesin arkkitehtuuria.



Kuva 9. Kubernetesin arkkitehtuuri (Khtan66 2016).

Rajapinta on osa Kubernetesin ohjaamiskeskusta, mikä ohjaa komentoja ja kuormia klustereiden kesken. Kubernetesin komponenteissa käytetään säiliön nimeä palvelun paljastamiseen, automaattisen kuorman toteuttamiseen ja konttien linkittämiseen osaksi ryhmiä. (Rouse ym. 2017b)

5.3 Kubernetesin hyvät ja huonot puolet

Kubernetesin hyvät puolet

Kubernetes helpottaa useiden konttien hallintaa samanaikaisesti pitämällä kirjaa konteista ja hallitsemalla konttien kuormia tasaisesti. (Google 2019b)

Kontin päivittämisen nopeus ja virheistä palautuminen on nopeaa, koska Kubernetes hoitaa sen automaattisesti. Tarkoituksena on päivittää palveluita ilman niiden rikkoutumista ja vähentämään päivittämisestä aiheutuvia käyttökatkoksia. (Jayanandana 2018)

Kubernetesissa on asetukset tietylle tilalle ja Kubernetes pitää huolen, että tila pysyy samana, vaikka kontteja tuhottaisiin ajan kuluessa. Esimerkiksi voidaan määritellä, että säiliöstä on kolme kopiota aina saatavilla.

Kubernetesin huonot puolet

Kubernetesin monimutkaisuus, asetusten laittaminen oikein ja kaikkien käsitteiden ymmärtäminen täysin ennen käyttöönottoa voi nostaa kynnystä ottaa Kubernetes käyttöön. (Hirschauer 2018)

Kubernetesissa voi olla replikaatio-ongelmia. Tietokannassa oleva tieto voidaan replikoida eli kopioida Kubernetesin avulla monta kertaa tiedon säilyttämisen helpottamiseksi. Tällöin tieto voi korruptoitua ja klusterin käynnissä pitäminen voi vaatia tarkkoja asetuksia Kubernetesissa tai tietokannan asetuksissa, mitkä ei välttämättä sovi Kubernetesin dynaamiseen tapaan tehdä klusterin käyttöönottoja. (Bakker 2019)

Kubernetesin säiliöiden resurssirajat pitää olla määritetty tarkasti suorittaessaan monia kontteja samanaikaisesti tehokkaasti, muuten resurssit voivat loppua säiliöstä kesken. Tällöin säiliö voi kaatua, jollei se saa resursseja muualta. (Bakker 2019)

6 AZURE

Tässä luvussa käydään läpi mitä Azure on ja mitä sillä voi tehdä. Azure on todella laaja, mutta luvussa tarkastellaan Azurea mikropalveluarkkitehtuurin näkökulmasta.

6.1 Määritelmä

Azure (azure.microsoft.com) on vuonna 2010 Microsoftin julkaisema joukko pilvipalveluja yritysten tarpeisiin (Hauger 2010). Pilvi-sanalla tarkoitetaan internetin välityksellä toimivia palveluita. Azure julkaistiin alun perin Windows Azure nimellä, mutta nimi muutettiin Microsoft Azureksi vuonna 2014, koska Microsoft huomasi Azuren kehittyneen käsittämään paljon muuta kuin pelkästään Windows-käyttöjärjestelmää. (Martin 2014)

Azure toimii monella ohjelmointikielellä. Tuki löytyy myös Windows ja Linux-käyttöjärjestelmille, SQL-relaatiotietokannoille ja NoSQL-tietokannoille. (Microsoft 2019g)

Azuressa on palveluja infrastruktuuripalveluille kuten palvelimille, virtuaalikoneille ja tietojen varastoinnille. Palveluja löytyy myös sovelluksen tekemiseen palveluna. (Microsoft 2019i)

6.2 Azure ja mikropalveluarkkitehtuuri

Azuressa palvelujen löytämisen helpottamiseksi Microsoft on jakanut Azuren palvelut 20 kategoriaan, joita selaamalla palvelut on helpompi löytää. Seuraavassa joitakin kategorioita, jotka liittyvät mikropalveluarkkitehtuuriin eniten.

Ensimmäinen kategoria on **analytiikka**. Analytiikan ideana on saada kaikki mahdollinen data kerättyä, analysoitua ja visualisoitua. Analytiikan avulla voidaan tehdä parempia päätöksiä tai löytää uusia mahdollisuuksia yrityksissä (Microsoft 2019a). Päätöksen tai mahdollisuuden tueksi on faktatietoa. Mikropalveluarkkitehtuurissa jokaisesta palvelusta tulee paljon tietoa, mitä voi analysoida joko palvelutasolla tai kokonaisuutena. Jos tätä tietoa ei kerättäisi mitenkään olisi vaikea tietää mahdollisia parannuskeinoja tai ongelmakohtia, joihin puuttua, koska asioista ei jäisi jälkeä mihinkään. Analytiikka on yksi tapa hallita palveluista tulevaa tietomäärää.

Toinen kategoria on **laskeminen** (eng. compute). Laskeminen käsittää sovelluksen kehittämisen infrastruktuurin esimerkiksi virtualisoinnin, kontit, konttien hallitsemisen ja palvelimet. Tämä kategoria on arkkitehtuurin pääkategorioista, mikä mahdollistaa arkkitehtuurin toimimisen ja tiedon liikkumisen palveluissa ja niiden välillä. (Microsoft 2019c)

Kolmas kategoria on **kontit**. Azuressa kontit kategoria helpottaa konttien kehittämistä, julkaisua ja päivittämistä alusta loppuun. Kontteja voi säilyttää Dockerin kauppapaikassa tai Azuren omassa konttirekisterissä. Azuressa voi käyttää Azuren Kubernetesta konttien hallinnan helpottamiseksi. (Microsoft 2019e)

Mikropalveluarkkitehtuuri perustuu erilaisiin palveluihin, jotka laitetaan kontteihin helpommin hallittaviksi ja niihin voi integroida jatkuvat päivitykset tai lokitietojen keruujärjestelmän helpommin kontti kerrallaan.

6.3 Azure mikropalveluarkkitehtuurin käytössä

Azure sopii hyvin konttien ja mikropalveluarkkitehtuurin kanssa työskentelyyn olemassa olevien palvelujen ansiosta, mitkä on tehty näitä silmällä pitäen. Azuressa on tuki Dockerille ja Kubernetesille ja monille muille palveluille, mitkä helpottavat mikropalveluarkkitehtuurilla työskentelemistä. Azuressa palvelut ovat yksittäisiä itsenäisiä kokonaisuuksia, jotka tekevät yhtä asiaa ja toimivat mikropalveluarkkitehtuurin kanssa loistavasti. Azure helpottaa mikropalveluarkkitehtuurin tekemistä, koska jokainen palvelu voidaan tehdä erillisenä Docker-konttina, jota hallitaan Azuren ja Kubernetesin avulla. (Microsoft 2019h)

Arkkitehtuurin hallintaan tiedonhallinnan kuuluu olennaisena osana palvelujen lokititetojen ja jatkuvan integroinnin hallinta. Lokitiedot ovat tekstitiedostoja ja näiden tietojen hallintaan Azuressa on Azure Monitor, joka kerää lokitietoja. Lokitietojen kerääminen on hyödyllistä, koska silloin tietää paremmin mitä palveluissa tapahtuu tiettynä hetkenä, esimerkiksi vikatilanteissa. Lokit sisältävät tietoja sovelluksen toiminnasta, järjestelmätietoja esimerkiksi prosessorista ja palvelimeen tehtyjä pyyntöjä esimerkiksi lomaketietojen lähettäminen palvelimelle ja niiden tilannekoodeja mahdollisten virhesisältöjen kera. Lokitiedot pitäisi siirtää pysyvään tallennuspaikkaan esimerkiksi tietokantaan lokitietojen eheyden ja pitkäaikaisen säilyttämisen vuoksi. Lisäksi voidaan tarvittaessa analysoida palvelun toimimista tai toimimattomuutta pitkän ajan kuluttua ja syiden etsimisessä ja korjaamisessa. (Wasson ym. 2018b)

Jatkuva integrointi tai jatkuva kehittäminen auttaa mikropalveluarkkitehtuurissa huomattavasti, koska sillä voidaan automatisoida palvelun julkaisu ja päivittäminen. Jatkuva integrointi on yksi isoimmista syistä mikropalveluarkkitehtuurin käyttöönottamiseen. Isoimpana haasteena on integrointi yhtenäisesti eri palvelujen välillä, mikä ei välttämättä ole mahdollista, koska eri palveluilla voi olla eri teknologioita ja integroinnin voi joutua hoitamaan erikseen jokaiselle palvelulle. Tämä lisää integroinnin haastavuutta ja vie enemmän aikaa kuin monoliittisessa sovelluksessa. (Wasson ym. 2018a)

Mikropalvelu kannattaa laittaa Azuren Docker-konttiin, mihin talletetaan kaikki tarvittavat riippuvuudet, jotta palvelun käyttäminen ja jatkuva integrointi/kehittäminen olisi mahdollisimman helppoa. (Wasson ym. 2018a)

6.4 Azuren hyvät ja huonot puolet

Azuren hyvät puolet

Azurea kannattaa käyttää, jos Microsoftin palvelut ja työkalut ovat tuttuja ja niitä pystyy hyödyntämään Azuren käytössä. Microsoftilla on myös yhteistyökumppaneita esimerkiksi RedHatin (redhat.com) ja Ciscon (cisco.com) kanssa, mikä parantaa Azuren palveluja entisestään. (Harvey 2017)

Azuressa maksetaan käytön mukaan, josta Azuren sivuilla on laskuri hinnan arviointia varten. Tämä on hyvä, jos tiedetään etukäteen kuinka paljon käyttöä palveluilla on. (Microsoft 2019b)

Azure varmuuskopioi tiedon kolmeen eri paikkaan tietokeskuksessa, mikä helpottaa tiedon palauttamista tiedon hävitessä jostain paikasta. Tiedon varmuuskopioinnin tiheyttä voi säätää haluamaansa tiheyteen, esimerkiksi päivän tai viikon välein. Toistuva tiedon varmuuskopiointi pitää huolen ajantasaisesta tiedon säilytyksestä. (Shortslef 2019)

Azuren huonot puolet

Käytön mukaan maksaminen on myös huono puoli, koska kulujen arviointi on vaikeaa monien palvelujen kustannusten yhteisarvioinnin kanssa ja palvelut sisältävät monia hinnoitteluluokkia, joista oikean hintaluokan valitseminen voi olla haastavaa. (Rouse & Bigelow 2018)

Pilvipalveluilla ei ole standardia, mitä kaikki pilvipalvelun tarjoajat noudattaisivat, vaan pilvipalvelujen tarjoajat tekevät omat versionsa palveluista omalla tavallaan. Tämä johtaa siihen, että palvelut pitää opetella uudestaan, jos vaihtaa palveluntarjoajaa tulevaisuudessa. Tämä ongelma ei ole vain Azuren, vaan yleisesti pilvipalvelujen ongelma. (Rouse & Bigelow 2018)

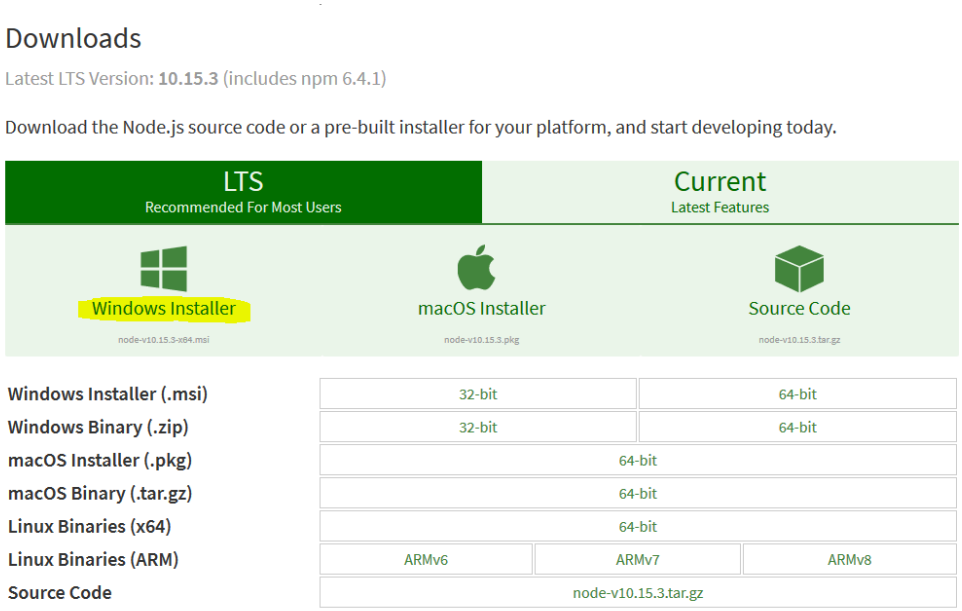
7 KONTIN LUOMINEN DOCKERILLA

Tässä luvussa käydään läpi tarvittavien sovelluksien asentaminen, oman sovelluksen tekeminen sekä mitä vaatimuksia sovelluksen tekemiseen vaaditaan ja miten kontti luodaan Dockerin avulla.

7.1 Nodejs-ympäristön asentaminen

Esimerkkisovellus tehdään käyttämällä Nodejs-ympäristöä, JavaScript-ohjelmointikieltä ja Windows-käyttöjärjestelmää. Nodejs mahdollistaa JavaScriptin käyttämisen selaimessa ja palvelimella. (Patel 2018).

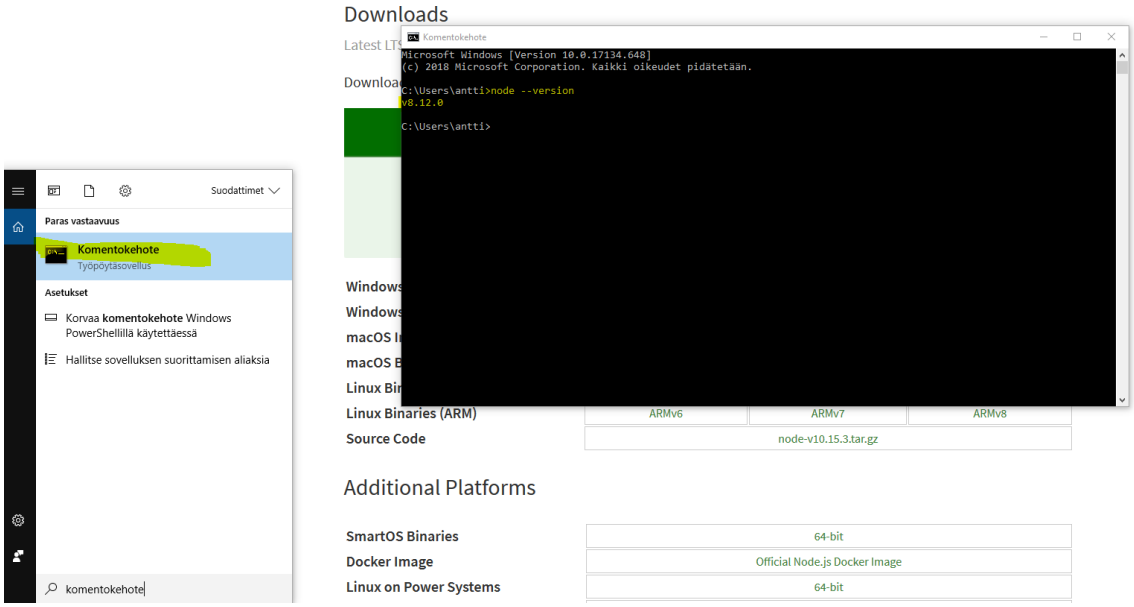
Nodejs on käyttöjärjestelmäriippumaton ja se voidaan ladata omalle tietokoneelle Nodejs sivulta (https://nodejs.org/en/download/) ja painamalla asennusnappia oikean käyttöjärjestelmän kohdalta. Koska käytetään Windows-käyttöjärjestelmää, painetaan ”Windows Installer”-näppäintä. Kuvassa 10 näytetään Nodejs asentaminen sivustolta.



Kuva 10. Nodejs-ympäristön asentaminen.

Tiedoston lataamisen jälkeen kaksoisklikataan tiedostoa ja asennusohjelma avautuu, jolla asennetaan Nodejs painamalla alakulmassa olevaa ”Next”-näppäintä. Tämän jälkeen pitää hyväksyä lisenssiehdot ja painaa ”Next”-näppäintä muutaman kerran uudestaan Nodejs-ympäristön asentamiseksi.

Nodejs asentamisen varmistaminen tapahtuu avaamalla komentokehote. Komentokehote avataan kirjoittamalla Windowsin-hakupalkkin ”komentokehote” ja painamalla hakupalkin yläreunassa näkyvää sovellusta. Komentokehotteelle kirjoitetaan ”node --version”. Asennettu versio pitäisi näkyä komentokehotteessa. Kuvassa 11 näytetään asennettu Nodejs-versio.



Kuva 11. Asennetun Nodejs-version varmistaminen komentorivin avulla.

7.2 Sovelluksen tekeminen

Nodejs-version varmistamisen jälkeen voidaan aloittaa sovelluksen tekeminen JavaScript-ohjelmointikielellä. Sovellus on yksinkertainen ohjelma, mikä tulostaa selaimen ikkunaan tekstin ”Tervetuloa mahtavaan sovellukseeni!”. Tätä varten tehdään uusi kansio C-aseman juureen nimeltä ”sovellus”. Tämä tapahtuu klikkaamalla oikeaa hiiren näppäintä ja valitsemalla uusi kansio. Kansioon tehdään kaikki sovelluksessa tarvittavat tiedostot. Kansio voi sijaita missä vaan tietokoneella ja olla minkä tahansa niminen, mutta kansion sijainti ja nimi pitää tietää. Kansion polku on: ”C:\sovellus”. Hiirellä kaksoisklikataan kansiota ja avataan muistiosovellus. Muistion avaaminen toimii samalla tavalla kuin komentokehote, mutta nimi vaihtuu kehotteesta muistioksi ja kirjoitetaan kuvassa 12 näkyvä sisältö. Kirjoittamisen jälkeen tallennetaan tiedosto nimellä index.js.

index.js – Muistio

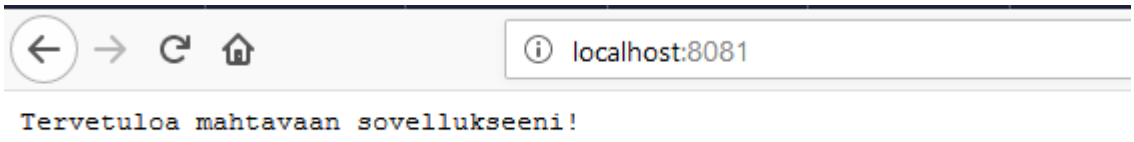
Tiedosto Muokkaa Muotoile Näytä Ohje

```
// importtaa http kirjasto
let http = require('http');

http.createServer(function luoPalvelin (request, response) {
    // Lähetä the HTTP header
    // HTTP Status: 200 : OK
    // Sisältötyyppi: text/plain
    response.writeHead(200, { 'Content-Type': 'text/plain' });

    // Lähetä vastaus "Tervetuloa mahtavaan sovellukseeni!"
    // palvelimen portti on 8081
    response.end('Tervetuloa mahtavaan sovellukseeni!');
}).listen(8081);

// komentorivi näyttää seuraavanlaisen tekstin
console.log('palvelin on käynnissä osoitteessa http://localhost:8081/');
```

Kuva 12. index.js-tiedoston sisältö.

7.3 Sovelluksen käynnistäminen

Esimerkkisovellus käynnistetään avaamalla komentokehote samassa kansiossa, missä sovellus sijaitsee. Se tehdään kirjoittamalla kansion yläpalkissa näkyvään polkuun: ”cmd”, mikä tarkoittaa komentokehotteen avaamista kyseisen kansion ollessa suoraan aktiivinen. Komentokehotteen ollessa auki oikeassa polussa, polun pitäisi olla ”C:\sovellus”. Komentokehotteeseen kirjoitetaan: ”node index.js” ja kehotteessa lukee ”palvelin on käynnissä osoitteessa http://localhost:8081”. Varmistetaan avaamalla nettiselain ja kirjoittamalla selaimen yläkulmassa olevaan osoitepalkkiin: ”localhost:8081”. Selaimessa näkyy teksti: ”Tervetuloa mahtavaan sovellukseeni!” (Kuva 13).



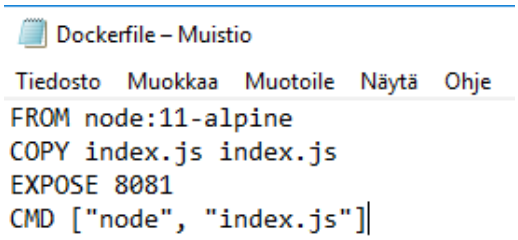
Kuva 13. Sovelluksen toiminta selaimessa.

7.4 Dockerin asentaminen

Kontin tekemiseen tarvitaan Docker-sovellus. Sovelluksen asentaminen onnistuu helpoiten Dockerin kauppapaikasta: https://hub.docker.com/editions/community/docker-ce-desktop-windows. Asentamista varten luodaan Docker-tunnukset seuraamalla ohjeita sivuston ”signup”-osiolla ja kirjautumalla sisään juuri tehdyllä Docker-tunnuksella sovelluksen lataamista varten. Lataamiseen jälkeen kaksoisklikataan tiedostoa ja oletusasetuksilla klikataan alakulmassa näkyvää ”ok”-näppäintä, jonka jälkeen Docker on asennettu. Asennuksen viimeistelemiseksi tarvitsee kirjautua ulos tietokoneelta painamalla keskellä näkyvää ”close and logout”-näppäintä.

7.5 Kontin tekeminen ja suorittaminen

Kontin tekemiseen vaaditaan Docker-tiedosto samaan kansioon, missä sovellus sijaitsee. Mennään kansioon polussa: ”C:\sovellus”, jonka jälkeen avataan muistio ja tehdään tiedosto nimeltä ”Dockerfile” ilman tiedostopäätettä sekä kirjoitetaan kuvassa 14 oleva sisältö.



Kuva 14. Docker-tiedoston sisältö.

Docker-tiedoston tekemisestä, yleisesti Dockerin komennoista ja siihen liittyvästä kieliopista löytyy lisää tietoa Dockerin dokumentaatiosta: https://docs.docker.com/engine/reference/builder/ ja https://docs.docker.com/engine/reference/run/, tai valitsemalla vasemmalta näkyvästä valikosta haluamansa komennon tai käyttämällä komentokehotteen ”docker –help” komentoa. Docker-tiedoston pitää alkaa FROM-komennolla, mikä kertoo mistä imagesta pohja otetaan sovellusta varten. Kuvan 14 Docker-tiedostossa otetaan Nodejs versio 11 pohjaksi, minkä jälkeen käytetään COPY-komentoa kopioimaan aikaisemmin tehty index.js-tiedosto. EXPOSE-komento paljastaa portin 8081, jotta sovellus voi toimia. Lopuksi käytetään CMD-komentoa, joka suorittaa Nodejs:n avulla index.js:n samalla tavalla kuin aikaisemmin tehtiin komentokehotteen kautta. (Docker 2019g)

Docker-image tehdään avaamalla komentokehote samassa kansiossa kuin sovellus. Docker-tiedosto on kontekstipohjainen. Konteksti otetaan siitä, missä kansiossa komentokehotteella komento ”docker build” suoritetaan. (Docker 2019g) Tässä tapauksessa komento suoritetaan sovellusprojektin juurikansiossa samassa paikassa, missä Docker-tiedosto sijaitsee, joten pelkkä ”docker build .” riittää ja ei tarvitse erikseen kirjoittaa tiedostopolkua komentoon.Tämän jälkeen avataan komentokehote sovelluskansiossa ja kirjoitetaan ”docker build . -t omasovellus” eli ”docker build piste -t omasovellus”. Piste kirjoitetaan, koska halutaan ottaa nykyinen kansio kontekstina ja kopioida kaikki polut kansion sisällä ja -t:llä voidaan antaa oma nimi imagelle löytämisen helpottamiseksi. (Docker 2019a) Kuvassa 15 on Dockerin näkymä onnistuneen Docker-imagen luonnin jälkeen.

```
C:\sovellus>docker build . -t omasovellus
Sending build context to Docker daemon  3.584kB
Step 1/4 : FROM node:11-alpine
11-alpine: Pulling from library/node
8e402f1a9c57: Pull complete
682f5b61546c: Pull complete
c482ca3e1e59: Pull complete
Digest: sha256:5fec7c5da14d7ce1e27247cce8889f51d2cf97f6aa73511ccc9a8944f066d625
Status: Downloaded newer image for node:11-alpine
 ---> 953c516e1466
Step 2/4 : COPY index.js index.js
 ---> f9bb7c2e2d15
Step 3/4 : EXPOSE 8081
 ---> Running in 3c0da4fec855
Removing intermediate container 3c0da4fec855
 ---> 898360d440ce
Step 4/4 : CMD ["node", "index.js"]
 ---> Running in 9873a361a250
Removing intermediate container 9873a361a250
 ---> b7fb1c465a2a
Successfully built b7fb1c465a2a
Successfully tagged omasovellus:latest
```

Kuva 15. Docker-imagen tekeminen onnistuneesti.

Ennen komennon suorittamista imagen olemassaolon voi tarkistaa kirjoittamalla komentokehotteeseen ”docker images”. Silloin pitäisi näkyä kaksi Docker-imagea: yksi node image, mikä on pohjaimage, ja toinen juuri tehty omasovellus-image (Kuva 16).

```
C:\sovellus>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
omasovellus         latest              b7fb1c465a2a        7 seconds ago       76.1MB
node                11-alpine           953c516e1466        8 days ago          76.1MB
```

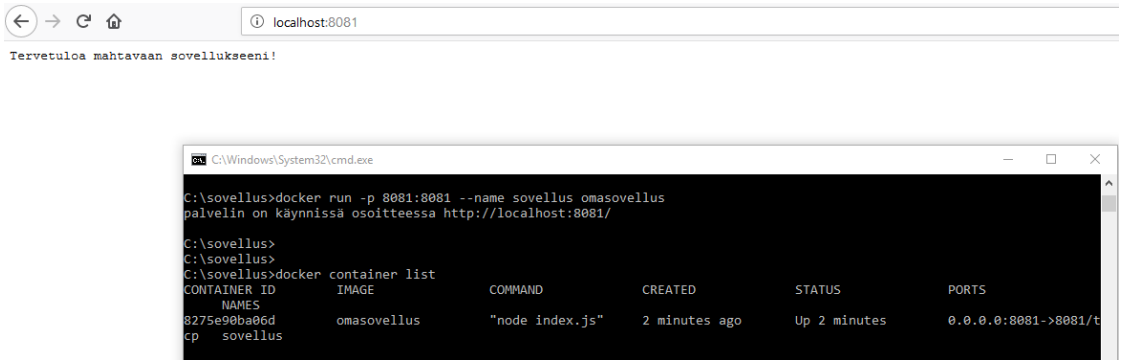
Kuva 16. Docker-imagen olemassaolon varmistaminen.

Docker-kontti suoritetaan kirjoittamalla komentokehotteeseen ”docker run -p porttinumero --name konttinimi imagenimi” eli tässä tapauksessa

”docker run -p 8081:8081 --name sovellus omasovellus”.

Parametri -p tarkoittaa portin julkaisemista mainitussa portissa. Komennnon ensimmäinen numero tarkoittaa tietokoneen porttia ja toinen numero tarkoittaa kontin porttia. Name-parametrilla annetaan nimi kontille komentojen suorittamisen helpottamiseksi. (Docker 2019d)

Suorittamalla edellä mainittu komento ja avaamalla selain osoitteessa ”localhost:8081” näkyy teksti ”Tervetuloa mahtavaan sovellukseeni!”. Docker-kontin olemassaolon voi varmistaa kirjoittamalla komentokehotteeseen ”docker container list”, jonka jälkeen näkyy käynnissä oleva kontti. Kuvassa 17 näkyy sovelluksen suorittaminen konttina selaimen kautta ja konttien listaus komentokehotteessa.



Kuva 17. Sovelluksen suorittaminen konttina ja konttien listaaminen.

Docker kontin voi myöhemmin käynnistää komennolla ”docker start konttinimi” (Docker 2019e). tai pysäyttää komennolla ”docker stop konttinimi” (Docker 2019f). Kontit voi listata komennolla: ”docker container ls” tai ”docker container list”. (Docker 2019b)

8 KONTIN ASENTAMINEN AZUREEN

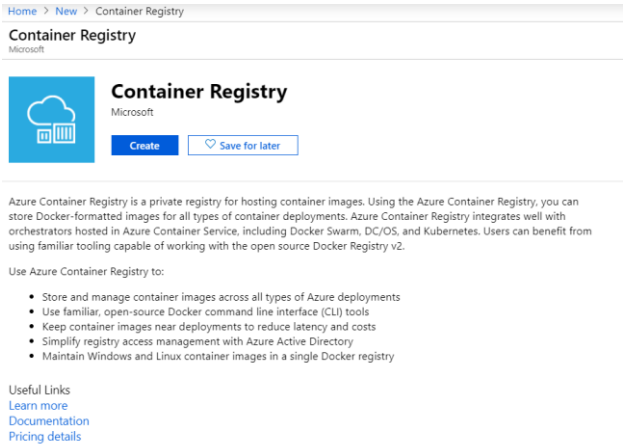
Tässä luvussa käydään läpi Azureen kirjautuminen, Azure-CLI-paketin asennus, Azuressa tehtävän konttirekisteri-resurssin tekeminen ja konttien lisääminen konttirekisteriin.

8.1 Azureen kirjautuminen

Azuren käyttämiseen vaaditaan Microsoft-tili ja luottokortti tai pankkikortti ilmaisen rahan käyttämistä varten ensimmäisen 30 päivän ajan. Azurea aloitetaan käyttämään menemällä osoitteeseen: https://azure.microsoft.com/en-us/ ja klikkaamalla oikeassa yläkulmassa vihreällä näkyvää ”free account”-näppäintä. Sen jälkeen ”start free”-näppäintä painamalla saadaan ilmainen raha. Azurea käytetään kirjautumalla olemassa olevaan Microsoft-tiliin tai luomalla uusi käyttäjä klikkaamalla ”No account? Create one!” ja seuraamalla ohjeita. (Microsoft 2019f)

8.2 Azure konttirekisterin tekeminen

Konttirekisteri on yksityinen rekisteri omien tai yrityksen konttien hallintaan. Konttirekisteri toimii Docker-kauppapaikan tavoin. Konttirekisteriä voidaan käyttää myöhemmin Kubernetes clusterin luonnissa. (Microsoft 2019d). Azuren hallintapaneelin vasemmalta valikosta klikataan ”create a resource”, jonka jälkeen avautuvaan hakupalkkiin kirjoitetaan: ”container registry”. Sitten täytetään omat tiedot sivun http://techgenix.com/azure-container-registry/ esimerkin mukaisesti. Tämän jälkeen avautuu konttirekisterin tiedot (Kuva 18).



Kuva 18. Konttirekisterin tiedot.

Konttirekisterin ikkunassa klikataan ”create”-näppäintä ja aloitetaan konttirekisterin luonti kirjoittamalla seuraavat tiedot avautuvaan näkymään:

Registry name (rekisterin nimi): oppari.

Rekisterin nimi voi olla mikä vaan ja on tärkeää kirjoittaa se muistiin, koska nimeä käytetään rekisteriin yhdistämiseen.

Resource group (resurssiryhmä)**:** oppariresurssit.

Painikkeella ”Create new” luodaan uusi resurssiryhmä, missä kaikki tämän työn aikana olevat resurssit sijoitetaan hallinnan helpottamiseksi ja myöhemmin resurssiryhmän poistamiseksi. Lopuksi varmistetaan oikea tilaus eli subscription.

Location (konttirekisterin maantieteellinen sijainti): West Europe.

Suositellaan, että kaikki resurssiryhmän sisällä olevat resurssit ovat samassa paikassa nopeamman haun vuoksi.

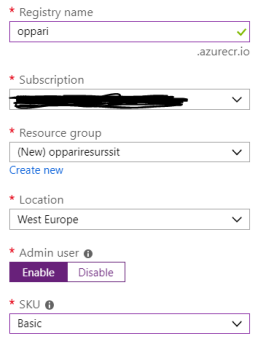
Admin user (ylläpitäjäkäyttäjän aktivointi)**:** Enable.

Ylläpitäjäkäyttäjän aktivoinnilla voidaan käyttää resurssin nimeä ja ylläpitäjän avainta konttirekisterin yhdistämiseen Docker-komentokehotteen kautta.

SKU (palvelun taso): Basic

Palvelun tasona käytetään alinta mahdollista eli basic-tasoa, mikä kuluttaa vähiten rahaa, tarjoaa samat ominaisuudet, kuin muut tasot. Tasojen erona on vain teho ja skaalaaminen.

Lopuksi klikataan ”create” näppäintä, jonka jälkeen näkyvät tehdyn konttirekisterin tiedot (Kuva 19).



Kuva 19. Azure portaalissa luodun konttirekisterin tiedot.

Konttirekisterin luonnin jälkeen tuomme rekisteriin oman kontin.

8.3 Kontin lisääminen Azure-konttirekisteriin

Kontti lisätään konttirekisteriin käyttämällä Dockeria, tietokoneen komentokehotetta ja siihen asennettua Azure-CLI-pakettia. Azure-CLI asennetaan omalle koneelle linkistä https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?view=azure-cli-latest ja painamalla ”Download the MSI installer”-näppäintä, jonka jälkeen tuplaklikkataan ladattua tiedostoa ja painetaan ”install”.

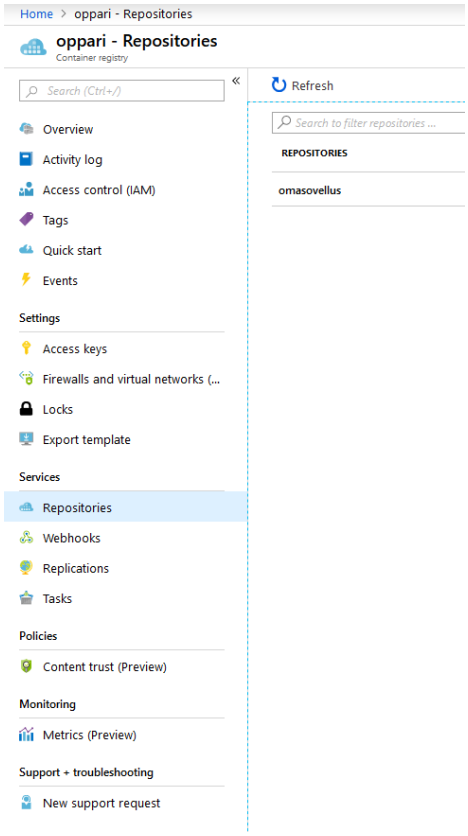
Tämän jälkeen käynnistetään komentokehote ja kirjaudutaan Azureen sisään kirjoittamalla komento ”az login”. Sitten kirjoitetaan selainikkunaan kirjautumistiedot. Tämän jälkeen voidaan kirjautua Azure-konttirekisteriin komennolla: ”az acr login –name oppari”, joka käyttää ”az login” komennon kirjautumistietoja konttirekisteriin yhdistämiseen. Kirjautuminen on onnistunut, kun komentokehoitteessa lukee teksti ”Login succeeded”. Paikallinen kontti nimetään ”docker tag” komennolla:”docker tag omasovellus oppari.azurecr.io/omasovellus:v1”. Tämän jälkeen paikallinen juuri tehty kontti tuodaan omalta tietokoneelta Azuren konttirekisteriin komennolla ”docker push oppari.azurecr.io/omasovellus:v1”. (Anderson 2018)

Kuvassa 20 on näkymä Azuressa onnistuneen konttirekisteriin kirjautumisen ja kontin lisäämisen konttirekisteriin jälkeen.

```
C:\sovellus>az acr login --name oppari
Login Succeeded

C:\sovellus>docker push oppari.azurecr.io/omasovellus:v1
The push refers to repository [oppari.azurecr.io/omasovellus]
d111e3384cd1: Pushed
1861d43e0a1b: Pushed
48f0cd99b58b: Pushed
bcf2f368fe23: Pushed
v1: digest: sha256:eaa5e51c17f831c2ffca18ea6c4ae8c5aaac86d67ba3ea6ba8a7479e3aac06c9 size: 1158
```

Kuva 20. Onnistunut kirjautuminen ja kontin lisääminen konttirekisteriin.

Kontin lisäämisen onnistuminen voidaan tarkistaa Azuren käyttöliittymästä klikkaamalla tehtyä konttirekisteriä ja menemällä valikkoon palvelut (services). Sitten painetaan repositories-näppäintä, jonka jälkeenpitäisi näkyä juuri lisätty omasovellus kansio (Kuva 21). (Anderson 2018).



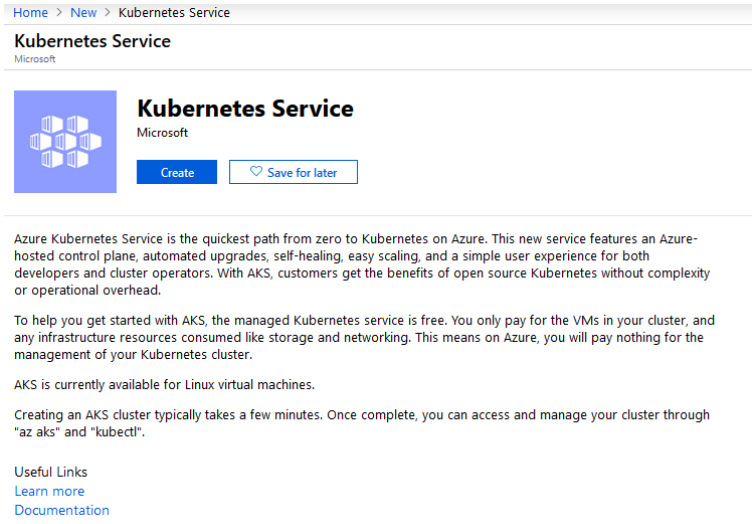
Kuva 21. Azure-portaalin näkymä konttirekisterissä sijaitsevasta kontista.

9 KONTIN LIITTÄMINEN KUBERNETESIIN

Tässä luvussa luodaan Azure Kubernetes-palvelu ja palveluun liitetään aikaisemmassa luvussa tehty kontti Azure-konttirekisteristä. Lopuksi palvelua kutsutaan julkisesti internetosoitteen välityksellä.

9.1 Azure Kubernetes-palvelun luominen

Kubernetes palvelu luodaan menemällä Azure-portaalin valikkoon ja klikkaamalla ”create a resource”. Hakupalkkiin kirjoitetaan ”kubernetes” ja klikataan Kubernetes Service ja painetaan ”create”-näppäintä. Kuvassa 22 näytetään Kubernetes-palvelun tiedot palvelun luonnin jälkeen.



Kuva 22. Kubernetes-palvelun tiedot.

Palvelun tiedot lisätään seuraamalla harjoitusta sivulla https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough-portal. Kubernetes-palveluun tiedot täydennetään Azure-portaalin kautta ohjeiden mukaan:

Resourcegroup: oppariresurssit

Kubernetes cluster name (Kubernetes-klusterin nimi): oppariaksklusteri

Region: West Europe

Kubernetes version (Kubernetes-versio): 1.11.9

Tämän tiedon voi vaihtaa myöhemmin.

DNS name prefix (nimipalvelimen nimen etuliite): oppariaksklusteri-dns

Tätä käytetään Kubernetes-API yhdistämiseen, sekä konttien hallintaan klusterin tekemisen jälkeen.

Node size (virtuaalikoneen koko): Standard DS1 v2 (1cvpu, 3,5 GB memory)

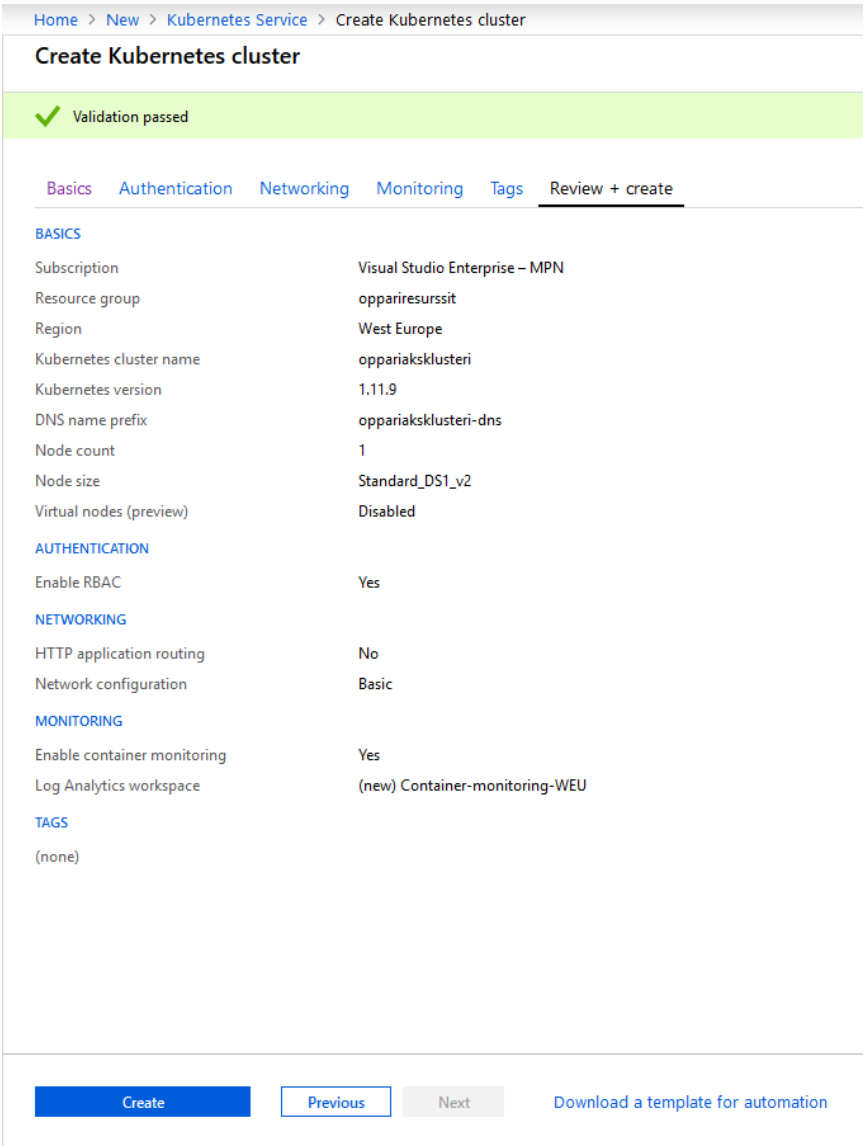
Virtuaalikoneen koko, mikä muodostaa solmut klusterissa. Tietoa ei voi muuttaa luonnin jälkeen.

Node count (solmujen lukumäärä)**:** 1

Solmujen lukumäärä, jonka voi muuttaa myöhemmin.

Tämän jälkeen toimitaan seuraavasti: Ensin klikataan ”Next: Authentication” näppäintä. Sen jälkeen hyväksytään oletusasetukset ja klikataan ”Next:Networking”-näppäintä. Seuraavaksi hyväksytään oletusasetukset ja klikataan ”Next:Monitoring”-näppäintä. Sen jälkeen hyväksytään konttien monitorointi ja tehdään lokitus analytiikan resurssiryhmä klikkaamalla ”create new”, valitaan Regioniksi West Europe ja nimetään se ”container-monitoring-WEU” ja klikataan ”Create”. Sitten klikataan ”Next:Tags”-näppäintä. Tämän jälkeen hyväksytään oletusasetukset ja klikataan ”Next:Review + Create”-näppäintä. Lopuksi varmistetaan oikeat tiedot ja klikataan ”Create”-näppäintä klusterin luomiseksi. (Foulds ym. 2018b).

Kuvassa 23 näytetään laaditun Kubernetes-palvelun tiedot.



Kuva 23. Azure portaalissa luotu Kubernetes-palvelu.

9.2 Kontin lisääminen Kubernetes-palveluun

Kubernetes-klusterin luonnin jälkeen lisätään konttirekisteristä kontti Kubernetes-klusteriin käyttämällä kubectl-pakettia. Kubectl-paketti asennetaan kirjoittamalla koneen komentokehotteella ”az aks install-cli”. Asennuksen jälkeen kirjoitetaan kehotteeseen ”set PATH=%PATH%;C:\Users\antti\.azure-kubectl”, jotta ”kubectl” komentoa voidaan käyttää tässä komentokehoitteen sessiossa. Toinen tapa tehdä sama asia on asettaa järjestelmänpoluksi kubectl.exe tiedoston polku tietokokoneella.

Sitten kirjoitetaan komentokehotteeseen ”az aks --help”, jotta löydetään tarvittava komento Kubernetes hallintapaneelin aukaisemiseksi. Ennen hallintapaneelin käyttöä pitää yhdistää kubectl-

paketti Kubernetes-klusteriin, mikä tehdään komennolla: ”az aks get-credentials—resource-group <resurssiryhmänimi> --name <aksnimi>” eli ” az aks get-credentials --resource-group oppariresurssit --name oppariaksklusteri”.

Hallintapaneelissa kaikki on lukittu, koska roolipohjainen pääsynhallinta (RBAC) on voimassa. Roolipohjaisessa pääsynhallinnassa annetaan käyttäjille roolien mukaan oikeudet yksittäisten käyttäjäoikeuksien sijaan. Hallintapaneeli saadaan avattua tekemällä sidos klusterin rooliin sivun: https://docs.microsoft.com/en-us/azure/aks/kubernetes-dashboard ohjeiden mukaan komennolla: ”kubectl create clusterrolebinding kubernetes-dashboard --clusterrole=cluster-admin --serviceaccount=kube-system:kubernetes-dashboard”. (Foulds ym. 2018a)

Tämän jälkeen tehdään uusi rooli, jolla sallitaan Kubernetes-palvelulle pääsy luotuun konttirekisteriin näillä ohjeilla: https://docs.microsoft.com/en-us/azure/container-registry/container-registry-auth-aks. Komentokehotteeseen kirjoitetaan seuraavat komento: ”az aks show --resource-group oppariresurssit --name oppariaksklusteri --query "servicePrincipalProfile.clientId" --output tsv” ja otetaan arvo muistiin. Esimerkin tapauksessa arvo: ”57c94903-5aec-40bf-b64d-17f3e10295ff”. (Lepow ym. 2018)

Sitten syötetään komento: ” az acr show --name oppari --resource-group oppariresurssit --query "id" --output tsv” ja otetaan arvo muistiin. Esimerkissä:”/subscriptions/438b9962-1c2f-4a02-92cd-5fc8fada3a46/resourceGroups/oppariresurssit/providers/Microsoft.ContainerRegistry/registries/oppari”. (Lepow ym. 2018)

Sitten syötetään komento, jolla luodaan rooli Azure-konttirekisterin liittämisen Kubernetesiin: ”az role assignment create --assignee 57c94903-5aec-40bf-b64d-17f3e10295ff --role acrpull --scope /subscriptions/438b9962-1c2f-4a02-92cd-5fc8fada3a46/resourceGroups/oppariresurssit/providers/Microsoft.ContainerRegistry/registries/oppari”.

Parametsin assignee kirjoitetaan ensimmäisen komennon tulos (57c94903-5aec-40bf-b64d-17f3e10295ff) ja Scopeen kirjoitetaan toisen komennon tulos (/subscriptions/438b9962-1c2f-4a02-92cd-5fc8fada3a46/resourceGroups/oppariresurssit/providers/Microsoft.ContainerRegistry/registries/oppari). Vastauksena pitäisi tulla satunnaisia id-numeroita aaltosulkeiden sisällä JSON-muodossa eli avain arvo pareina. (Lepow ym. 2018)

Kubernetes-hallintapaneeli aukeaa kirjoittamalla komentokehotteeseen: ”az aks browse -g <resurssiryhmänimi> -n <aksnimi>” eli ”az aks browse -g oppariresurssit -n oppariaksklusteri”. Hallintapaneelin avauduttua luodaan sovellus klikkaamalla oikeassa yläkulmassa ”+ Create”-näppäintä ja klikataan ”create an app”. Tämän jälkeen syötetään kenttiin seuraavat tiedot:

App name (sovelluksen nimi)**:** omasovellusaks

Container image (kontin URL)**:** oppari.azurecr.io/omasovellus:v1

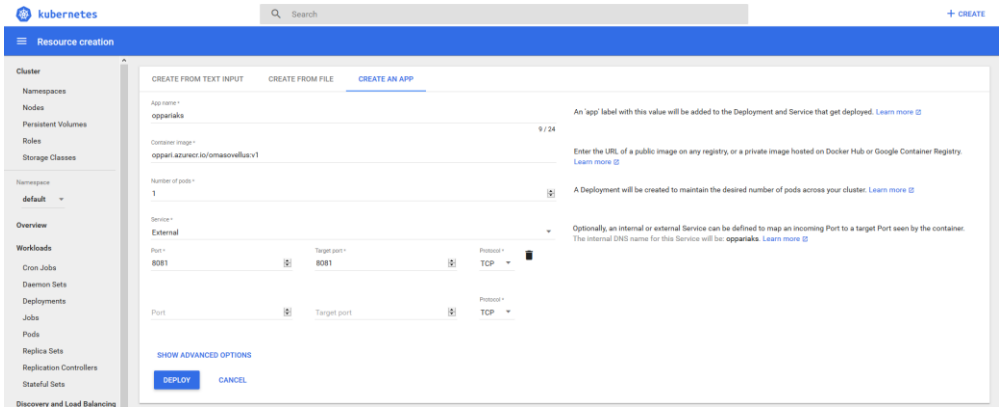
Kontin URL voidaan hakea Azure-konttirekisteristä.

Number of pods (säiliöiden lukumäärä): 1

Service (palvelu)**:** External > port 8081 > target port 8081 > protocol TCP

Palveluksi valitaan ulkoinen portti, koska sovellus toimii portissa 8081.

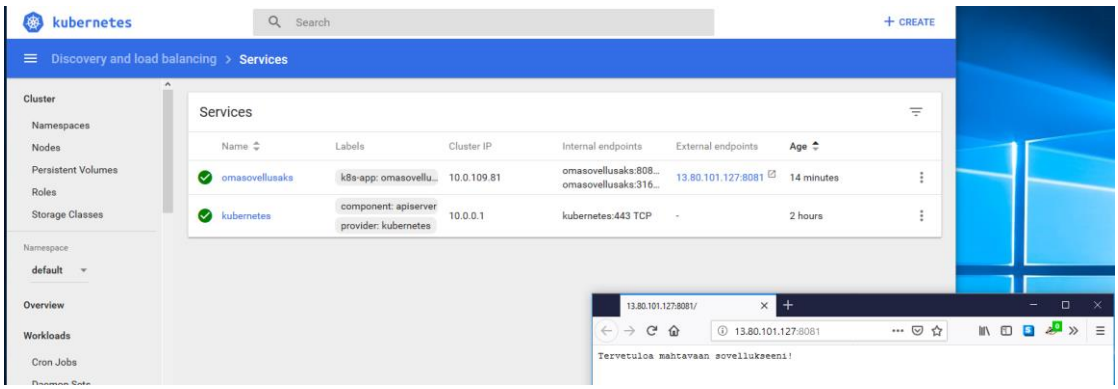
Lopuksi klikataan ”deploy”-näppäintä. Kuvassa 24 näytetään tehdyn Kubernetes-sovelluksen tiedot:



Kuva 24. Kubernetes-hallintapaneelinäkymässä luodun Kubernetes-sovelluksen tiedot.

9.3 Kubernetes-sovelluksen kutsuminen ja resurssien poistaminen

Kubernetes-sovellusta kutsutaan julkisesti klikkaamalla vasemmalla näkyvässä valikossa ”Services” ja katsomalla omasovellusaks external enpoints-kohdan. Tässä tapauksessa se on 13.80.101.127:8081 (Kuva 25).



Kuva 25. Kubernetes hallintapaneelin- ja selaimen näkymä, kun Kubernetes-sovellus kutsutaan julkisesti internetosoitteen välityksellä.

Azuressa palveluun liittyvät resurssit poistetaan poistamalla palveluun littyvä resurssiryhmä. Resurssiryhmä poistetaan klikkaamalla Azuren-hallintapaneelissa ”Resource Groups” ja poistamalla kaikki siellä näkyvät resurssiryhmät. Resurssiryhmät pitää poistaa yksitellen klikkaamalla haluttua resurssiryhmää. Resurssiryhmä poistetaan klikkaamalla ”Delete resource group”-näppäintä ja seuraamalla ohjeita. Resurssiryhmän poistaminen on onnistunut, kun resurssiryhmää ei näy ”resource group”-kategoriassa.

10 YHTEENVETO

Opinnäytetyön tavoitteena oli tehdä toimiva sovellus, joka laitettiin konttiin ja integroitiin Azure-pilvipalveluun ja Kubernetes-kontinhallinta-alustaan. Tavoitteessa onnistuttiin. Sovelluksen yksinkertaisuuden takia mikropalveluarkkitehtuurin hyviä puolia ei tullut laajasti ilmi. Nämä tulisivat paremmin ilmi monimutkaisemmassa sovelluksessa, joka käyttää useampia palveluja.

Sovelluksen pitäminen mahdollisimman yksinkertaisena oli aluksi haastavaa, sillä sovelluksen tekemistä ajateltiin liian monimutkaisesti. Haasteena oli toistuvuuden rajoittaminen ja tarpeeksi yksinkertaisen sovelluksen keksiminen. Teoriapuolen rajaaminen kaikista relevanteimpiin käsitteisiin ja niistä kertominen yleisellä tasolla menemättä liian yksityiskohtaisesti käsitteisiin oli myös haastavaa.

Opinnäytetyön tuloksena sovelluksen lisäksi saatiin kokonaiskuva mikropalveluarkkitehtuurin toiminnasta, käytettävistä teknologioista, huomioonotettavista seikoista ja eroista monoliittiseen arkkitehtuuriin verrattuna. Tuloksia voidaan hyödyntää mikropalveluarkkitehtuurin käyttöönoton aloittamisessa yrityksessä sovelluksen rakentamisesta konttiteknologialla pilvipalveluun. Mikropalveluarkkitehtuuriratkaisu on järkevä, koska sen avulla saadaan estettyä isojen vaikeasti hallittavien sovellusten syntymistä.

Tutkimuksen aikana oli muutamia ongelmia. Ensimmäinen ongelma oli Kubernetes-klusterin hallintapaneelin avaamisen vaikeus roolipohjaisen pääsynhallinnan takia. Toinen ongelmakohde oli Azure-konttirekisterin ja Kubernetes-klusterin kommunikointi keskenään. Tämä onneksi onnistui sopivan roolin avulla ja tämän jälkeen imagen käyttäminen oli helppoa, koska yhteys oli luotu konttirekisterin ja klusterin välille, jolloin pelkkä imagen nimen kirjoittaminen riitti käyttöönottoon. Näiden vaikeuksien jälkeen muu osuus tutkimuksesta sujui ongelmitta.

Tutkimuksen tuloksia voidaan hyödyntää liittämällä muita palveluja tai toimia sovelluskehitykseen. Palvelujen testaus tai sovelluksen julkaisemisen automatisointi palvelujen avulla voisivat olla esimerkkejä jatkotöistä. Monet yritykset tekevät sovelluksia edelleen monoliittisen arkkitehtuurilla, mikä helposti paisuttaa sovelluksen monimutkaisuutta. Mikropalveluarkkitehtuurissa voi olla vastaus tähän ongelmaan.

LÄHTEET

Anderson, P. 2018. Step-by-step guide: Creating and managing Azure Container Registry. Viitattu 23. huhtikuuta 2019 http://techgenix.com/azure-container-registry/.

Bakker, P. 2019. One year using Kubernetes in production: Lessons learned. Viitattu 24. maaliskuuta 2019 https://techbeacon.com/devops/one-year-using-kubernetes-production-lessons-learned.

Bauer, R. 2018. What's the Diff: VMs vs Containers. Viitattu 10. maaliskuuta 2019 https://www.backblaze.com/blog/vm-vs-containers/.

Corbasson, L. 2007. File:SOA Metamodel.svg. Viitattu 22. huhtikuuta 2019 https://en.wikipedia.org/wiki/File:SOA_Metamodel.svg.

Docker 2019a. Docker build. Viitattu 7. huhtikuuta 2019 https://docs.docker.com/engine/reference/commandline/build/.

Docker 2019b. Docker container ls. Viitattu 7. huhtikuuta 2019 https://docs.docker.com/engine/reference/commandline/container_ls/.

Docker 2019c. Docker overview. Viitattu 23. helmikuuta 2019 https://docs.docker.com/engine/docker-overview/.

Docker 2019d. Docker run. Viitattu 7. huhtikuuta 2019 https://docs.docker.com/engine/reference/commandline/run/.

Docker 2019e. Docker start. Viitattu 7. huhtikuuta 2019 https://docs.docker.com/engine/reference/commandline/start/.

Docker 2019f. Docker stop. Viitattu 7. huhtikuuta 2019 https://docs.docker.com/engine/reference/commandline/stop/.

Docker 2019g. Dockerfile reference. Viitattu 31. maaliskuuta 2019 https://docs.docker.com/engine/reference/builder/.

Docker 2019h. What is a container? Viitattu 23. helmikuuta 2019 https://www.docker.com/resources/what-container.

Foulds, L.; Benoit, M. & Peterson, N. 2018a. Access the Kubernetes web dashboard in Azure Kubernetes Service (AKS). Viitattu 23. huhtikuuta 2019 https://docs.microsoft.com/en-us/azure/aks/kubernetes-dashboard.

Foulds, L.; Rhoads, Z. & Polkovnikov, A. 2018b. Quickstart: Deploy an Azure Kubernetes Service (AKS) cluster using the Azure portal. Viitattu 23. huhtikuuta 2019 https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough-portal.

Fowler, M. & Lewis, J. 2014. Microservices. Viitattu 6. helmikuuta 2019 https://martinfowler.com/articles/microservices.html.

Google 2019a. Kubernetes features. Viitattu 24. maaliskuuta 2019 https://kubernetes.io/.

Google 2019b. What is Kubernetes? Viitattu 24. maaliskuuta 2019 https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.

Harvey, C. 2017. Microsoft Azure. Viitattu 21. maaliskuuta 2019 https://www.datamation.com/cloud-computing/microsoft-azure.html.

Hauger, D. 2010. Windows Azure General Availability. Viitattu 16. maaliskuuta 2019 https://blogs.microsoft.com/blog/2010/02/01/windows-azure-general-availability/.

Hirschauer, J. 2018. Problems solved and problems created by Kubernetes. Viitattu 24. maaliskuuta 2019 https://www.instana.com/blog/problems-solved-and-problems-created-by-kubernetes/.

Jayanandana, 2018. Benefits of Kubernetes. Viitattu 24. maaliskuuta 2019 https://medium.com/platformer-blog/benefits-of-kubernetes-e6d5de39bc48.

Khtan66 2016. File:Kubernetes.png. Viitattu 22. huhtikuuta 2019 https://commons.wikimedia.org/wiki/File:Kubernetes.png.

Kobos, J. 2018. When and Why to Use Docker? Viitattu 25. helmikuuta 2019 https://www.linode.com/docs/applications/containers/when-and-why-to-use-docker/.

Labrecque, M. 2018. Pros and Cons of Docker. Viitattu 25. helmikuuta 2019 https://affinitybridge.com/blog/pros-and-cons-docker.

Leoni, K. 2018. Docker Windows vs Linux Containers. Viitattu 23. helmikuuta 2019 https://blog.heroix.com/blog/docker-windows-vs-linux-containers.

Lepow, D.; Marsh, M. & Lyon, R. 2018. Authenticate with Azure Container Registry from Azure Kubernetes Service. Viitattu 23. huhtikuuta 2019 https://docs.microsoft.com/en-us/azure/container-registry/container-registry-auth-aks.

Lyon, R.; Schonning, N.; Wesley, D. & Casey, C. 2019. What is role-based access control (RBAC) for Azure resources? Viitattu 17. toukokuuta 2019 https://docs.microsoft.com/en-us/azure/role-based-access-control/overview.

Marquez, E. 2018. The History of Container Technology. Viitattu 10. maaliskuuta 2019 https://linuxacademy.com/blog/containers/history-of-container-technology/.

Martin, S. 2014. Upcoming Name Change for Windows Azure. Viitattu 16. maaliskuuta 2019 https://azure.microsoft.com/en-us/blog/upcoming-name-change-for-windows-azure/.

Mauersberger, L. 2017. Microservices: What They Are and Why Use Them. Viitattu 7. helmikuuta 2019 https://blog.leanix.net/en/a-brief-history-of-microservices.

mdnwebdocs-bot; HMVisuality; Noooshu1234 & chraiet 2019. What is JavaScript? Viitattu 31. maaliskuuta 2019 https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.

Microsoft 2019a. Analytics. Viitattu 17. maaliskuuta 2019 https://azure.microsoft.com/en-us/product-categories/analytics/.

Microsoft 2019b. Azure pricing. Viitattu 21. maaliskuuta 2019 https://azure.microsoft.com/en-us/pricing/.

Microsoft 2019c. Compute. Viitattu 17. maaliskuuta 2019 https://azure.microsoft.com/en-us/product-categories/compute/.

Microsoft 2019d. Container Registry. Viitattu 5. huhtikuuta 2019 https://azure.microsoft.com/en-us/services/container-registry/.

Microsoft 2019e. Containers. Viitattu 17. maaliskuuta 2019 https://azure.microsoft.com/en-us/overview/containers/.

Microsoft 2019f. Create your Azure free account today. Viitattu 29. huhtikuuta 2019 https://azure.microsoft.com/en-gb/free/.

Microsoft 2019g. Get started with Azure. Viitattu 16. maaliskuuta 2019 https://docs.microsoft.com/en-us/azure/#pivot=get-started&panel=get-started1.

Microsoft 2019h. Microservices in Azure. Viitattu 16. maaliskuuta 2019 https://azure.microsoft.com/en-in/solutions/microservice-applications/.

Microsoft 2019i. What is cloud computing? Viitattu 17. maaliskuuta 2019 https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/.

Opensource 2019. What is Docker? Viitattu 23. helmikuuta 2019 https://opensource.com/resources/what-docker.

Osnat, R. 2018. A Brief History of Containers: From the 1970s to 2017. Viitattu 10. maaliskuuta 2019 https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016.

Patel, P. 2018. What exactly is Node.js? Viitattu 31. maaliskuuta 2019 https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5.

Patel, P. 2018. What exactly is Node.js? Viitattu 29. huhtikuuta 2019 https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5.

Rouse, M. & Bigelow, S. 2018. Microsoft Azure (Windows Azure). Viitattu 21. maaliskuuta 2019 https://searchcloudcomputing.techtarget.com/definition/Windows-Azure.

Rouse, M.; Casey, K. & Earls, A. 2017b. Kubernetes. Viitattu 24. maaliskuuta 2019 https://searchitoperations.techtarget.com/definition/Google-Kubernetes.

Rouse, M.; Gillis, A. & Matturro, B. 2018c. Microservices. Viitattu 6. helmikuuta 2019 https://searchmicroservices.techtarget.com/definition/microservices.

Rubens, P. 2017. What are containers and why do you need them? Viitattu 10. maaliskuuta 2019 https://www.cio.com/article/2924995/what-are-containers-and-why-do-you-need-them.html.

Runnable 2019. Why use Docker? Viitattu 25. helmikuuta 2019 https://runnable.com/docker/why-use-docker.

Shortslef, N. 2019. Microsoft Azure Explained: What It Is and Why It Matters. Viitattu 21. maaliskuuta 2019 https://ccbtechnology.com/what-microsoft-azure-is-and-why-it-matters/.

SmartBear Software 2015. Why You Can't Talk About Microservices Without Mentioning Netflix. Viitattu 7. helmikuuta 2019 https://smartbear.com/blog/develop/why-you-cant-talk-about-microservices-without-ment/.

Swersky, D. 2018. What is Docker, and why is it so popular? Viitattu 10. maaliskuuta 2019 https://raygun.com/blog/what-is-docker/.

Toivanen, A. 2018. Mini vai Mikropalvelu? Viitattu 6. helmikuuta 2019 http://blogi.hiqfinland.fi/mini-vai-mikropalvelu.

Tozzi, C. 2016a. 3 Pros and 3 Cons of Working with Docker Containers. Viitattu 25. helmikuuta 2019 https://sweetcode.io/3-pros-3-cons-working-docker-containers/.

Tozzi, C. 2017b. Docker Downsides: Container Cons to Consider before Adopting Docker. Viitattu 25. helmikuuta 2019 https://www.channelfutures.com/open-source/docker-downsides-container-cons-to-consider-before-adopting-docker.

Tozzi, C. 2017c. Why Is Docker So Popular? Explaining the Rise of Containers and Docker. Viitattu 10. maaliskuuta 2019 https://www.channelfutures.com/open-source/why-is-docker-so-popular-explaining-the-rise-of-containers-and-docker.

Wasson, M.; Wilson, M.; Buck, A. & Celarier, S. 2018c. Microservices architecture style. Viitattu 6. helmikuuta 2019 https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices.

Wasson, M. ym. 2018b. Designing microservices: Logging and monitoring. Viitattu 16. maaliskuuta 2019 https://docs.microsoft.com/en-us/azure/architecture/microservices/logging-monitoring.

Wasson, M.; Wilson, M.; Buck, A. & Tam, B. 2018a. Designing microservices: Continuous integration. Viitattu 16. maaliskuuta 2019 https://docs.microsoft.com/en-us/azure/architecture/microservices/ci-cd.

Wright, E. 2019. A Brief History of Microservices. Viitattu 7. helmikuuta 2019 https://blog.turbonomic.com/blog/on-technology/a-brief-history-of-microservices.