Tampereen yliopisto

Tuomas Pekkanen

VERKKOKAUPAN ARKKITEHTUURI-SUUNNITELMA MIKROPALVELUNA

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Toukokuu 2020

TIIVISTELMÄ

Tuomas Pekkanen: Verkkokaupan arkkitehtuurisuunnitelma mikropalveluna
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Toukokuu 2020

Tutkielmassa esitellään mikropalveluarkkitehtuuri ja sen hyödyt ohjelmistokehityksessä. Mikropalvelut ovat itsenäisiä verkkopyynnöillä toisilleen keskustelevia ohjelmiston komponentteja, joiden päätarkoitus on jakaa ohjelmisto pienempiin osakokonaisuuksiin. Mikropalveluiden etuja on niiden skaalautuvuus, vapaus toteutustekniikoissa sekä oikein toteutettuna parempi vikasietoisuus.

Mikropalveluiden tueksi esitellään työkalut Docker ja Microsoft Azure Service Fabric, jotka toimivat mikropalveluiden ajoympäristönä. Docker on konttiteknologia, joka luo ajoympäristön sovelluksille, eli tässä tapauksessa mikropalveluille. Docker mahdollistaa ohjelman vaatimien riippuvuuksien hallitsemista asentamatta niitä isäntäkoneelle. Microsoft Azure Service Fabric on pilvipalvelualusta, joka tarjoaa palvelinresursseja mikropalveluiden ja Docker konttien ajamiseen. Service Fabric esitetään työssä tekijänä, joka lisää vikasietoisuutta automaattisella mikropalveluiden skaalaamisellaan sekä viallisen mikropalvelun korvaamisella toimivalla vikatilanteissa.

Mikropalveluarkkitehtuuria sovelletaan verkkokaupalle, joka haluaa toimia kustannustehokkaasti useassa eri maanosassa. Verkkokaupan vaatimuksiksi esitetään eri toiminnallisuuksia, joidenka perusteella luodaan mikropalvelut verkkokaupalle. Luodut mikropalvelut ovat tilauskäsittelijä, tuotevalikoima, tuotevarasto, mainospalvelu ja asiakasprofiilit. Arkkitehtuurin teknistä puolta ja sen vaikutuksia tutkitaan etenkin ajoympäristön ja tietokantojen osalta. Service Fabric todetaan hyväksi vaihtoehdoksi sen tarjoamien mikropalveluiden hallintatyökalujen, kattavan palvelinvalikoiman sekä jatkuvan päivitysominaisuutensa vuoksi. Jokaisella mikropalvelulla on oma tietokantansa yhden yhteisen mikropalveluiden välisen tietokannan sijaan. Yhteinen tietokanta loisi riippuvuuksia mikropalveluiden välille, joka vaikeuttaisi ohjelmiston skaalausta sekä hidastaisi sen kehittämistä. Mikropalveluilla on etuna lisäksi niiden teknologiariippumattomuus. Verkkokaupan mikropalveluista etenkin tuotevarasto on tehokriittinen, ja voidaan toteuttaa tehokkaalla ohjelmointikielellä kuten C++, vaikka muut mikropalvelut käyttäisivätkin toisia teknologiaratkaisuita.

Avainsanat: Mikropalvelu, arkkitehtuurisuunnitelma, verkkokauppa, ohjelmointi

SISÄLLYSLUETTELO

1. JOHDANTO

Nykyaikana entistä useampi palvelu siirtyy pilvipohjaisiin ratkaisuihin, esimerkiksi verkkokauppa Amazon ja videopalvelu Netflix mikropalveluarkkitehtuureillaan [1]. Pilvipalveluilta vaaditaan luotettavuutta, ketterää kehitystä sekä kustannustehokkuutta, jotta ne kykenevät vastaamaan niiden globaaliin kysyntään [2]. Perinteisesti ohjelmistot ovat rakennettu monoliitteina, eli valtavina kokonaisuuksina, joita on suuren kokonsa ja jäykän rakenteensa takia hankalaa jatkokehittää [3]. Tässä työssä esitetään ratkaisuna monoliiteille mikropalveluarkkitehtuuria. Mikropalvelut ovat tapa jakaa ohjelmisto pienempiin itsenäisiin osiin, jolloin niiden kehittäminenkin on ketterämpää [2].

Tutkielmassa luodaan verkkokaupalle mikropalveluarkkitehtuurisuunnitelma. Tavoitteena on luoda mikropalveluista kokonaisuus, joka kykenee kustannustehokkaasti palvelemaan suuriakin kävijämääriä ja varautuu virhetilanteisiin siten, että niiden seurauksena verkkokaupasta vain mahdollisimman pieni osa on poissa käytöstä. Tutkielman tutkimuskysymyksenä on skaalautuvan ja vikasietoisen arkkitehtuurin luominen verkkokaupalle.

Työn toisessa luvussa esitellään mikropalveluiden teoreettinen tausta. Kolmannessa luvussa kuvaillaan suunniteltavan verkkokaupan rakenne sekä jaetaan se mikropalveluihin. Neljännessä luvussa pohditaan verkkokaupan arkkitehtuurillisia ratkaisuita yksityiskohtaisemmin. Viimeisessä luvussa kootaan tutkimuksen tulokset sekä johtopäätökset niistä.

2. MIKROPALVELUT

Tässä luvussa esitellään ensin yleisesti mikropalvelut ja niiden käyttötarkoitus. Luvussa esitellään myös mikropalveluiden pystyttämisessä hyödyllisiä työkaluja. Ensimmäisenä työkaluna esitellään konttiteknologia Docker, jossa yksittäisiä mikropalveluita on mahdollista suorittaa [4]. Toisena työkaluna esitellään Microsoftin Azure Service Fabric, joka on Microsoftin tarjoama mikropalvelualusta [5].

Mikropalveluarkkitehtuuri perustuu siihen, että yhden valtavan palvelun sijasta luodaan pieniä, itsenäisiä palveluita eli mikropalveluita. Mikropalvelut toimivat yhdessä samoin kuin ne olisivat perinteisen arkkitehtuurin iso kokonaisuus, mutta niiden väliset riippuvuudet ovat löyhiä tai useimmiten niiden välillä ei ole ollenkaan suoria riippuvuuksia [3]. Mikropalveluiden edut perustuvatkin siihen, että ne ovat erittäin siirrettäviä. Niitä on vaivatonta lisätä, muokata ja poistaa.

Mikropalveluiden itsenäisyys perustuu niiden löyhään kommunikaatioon. Mikropalvelut keskustelevat toisillensa kuin ne olisivat omia prosessejansa nettipyyntöjen avulla [3]. Nettipyyntöjen avulla suoritetut pyynnöt tekevät mikropalveluista riippumattomia toisistaan, koska jo niitä suunniteltaessa on pääperiaatteena se, että mikropalvelu ei tiedä muiden mikropalveluiden olemassaolosta. Tällöin mikropalvelun julkinen rajapinta (API), joka tarjotaan nettipyynnöillä, luodaan kuin palvelu olisi tarkoitettu toimimaan yksinään.

Koska mikropalvelut ovat toisistaan riippumattomia, voidaan yksittäiset mikropalvelut toteuttaa radikaalisti eri tavalla. Ainoa mikropalveluita sitova asia on verkkopyyntöjen välillä tapahtuva kommunikointi, jolloin niiden toteutus on useista arkkitehtuurimalleista poiketen riippumaton ohjelmointikielistä ja tekniikoista [2]. Tehokriittinen osa voidaan toteuttaa käyttämällä matalan tason ohjelmointikieltä, kuten C++. Kun tehon sijasta on tarve luoda korkeamman tason asioita, voidaan kieli vaihtaa toisessa mikropalvelussa esimerkiksi JavaScriptiin. Mikropalveluissa on suhteellisen riskitöntä kokeilla uusia teknologioita, koska kokeiltava osa voidaan eristää eikä koko järjestelmää tarvitse muuttaa uuteen teknologiaan kerralla [3].

Mikropalvelut suunnitellaan sopeutumaan ympäristöönsä. Nykyään on useita eri alustoja, jotka vaativat samankaltaisia toiminnallisuuksia. Esimerkiksi musiikkipalvelun tarjoama musiikkitiedosto voidaan haluta wav-formaatissa tietokoneella kuunneltavaksi, mutta älypuhelimille tämä muoto voi olla tuntematon. Palveluun pitäisi siis lisätä uusi musiikkiformaatti älypuhelimille, mikä on mahdollisesti hankala muutos koko palvelun

perustuessa pelkästään wav-formaattiin. Mikropalvelu kykenee mukautumaan muutoksiin ilman tarvetta kirjoittaa suuria määriä koodia uudestaan [3].

Mikropalvelut ovat luonteeltaan skaalautuvia [6]. Koska mikropalvelut ovat siirrettäviä, niitä voidaan luoda lisää tarpeen mukaan. Joitain palvelun osia käytetään useammin kuin toisia, jolloin mikropalveluarkkitehtuurissa voidaan lisätä dynaamisesti kuormitetulle mikropalvelulle lisää ilmentymiä (instance). Mikropalvelu soveltuu kaikenkokoisille sovelluksille, koska sen laskentatehoa ja siten ylläpitokustannuksiakin pystyy säätämään ilmentymien lukumäärää muuttamalla.

Mikropalveluiden käyttöönotto (deployment) on mahdollista tehdä yksittäisen mikropalvelun tasolla. Perinteisessä palvelinarkkitehtuurissa voi syntyä tilanne, jossa muutoksia ei tehdä usein, koska päivittäminen vaatii koko järjestelmän uudelleenkäynnistyksen. Tällöin päivitysten koko yleensä kasvaa ja riski virheellisestä koodista kasvaa. Mikropalveluarkkitehtuurissa jokainen mikropalvelu voidaan päivittää muista mikropalveluista riippumatta, jolloin päivitykset voidaan jakaa pienempiin osiin. [5]

2.1 Vikasietoisuuden hallinta mikropalveluissa

Mikropalvelut tuovat mukanaan useita etuja, mutta ne sisältävät yhä verkkopalveluiden riskejä. Kun palvelut on sidottu internettiin, ovat ne myös riippuvaisempia siitä [7]. Mikropalveluiden on kyettävä reagoimaan verkkoyhteyden hetkelliseen ja pitkäaikaiseen katkeamiseen. Toinen riskitekijä ovat laiteviat, jotka ovat myös odottamaton tapahtuma [6].

Kaikki mikropalvelut eivät saa joutua vikatilaan, jos yksittäinen mikropalvelu kaatuu odottamattomasta syystä. Mikropalvelut, jotka tarvitsevat epäkunnossa olevaa mikropalvelua voivat aiheuttaa kaatumisten ketjureaktion, jos ne olettavat toisen palvelun toimivan aina. Mikropalveluiden on kyettävä reagoimaan toisen mikropalvelun osan kaatumiseen ja hidastumiseen ja sen seurauksena ottaa palvelun osia pois käytöstä niin, että järjestelmä kokonaisuudessaan säilyy mahdollisimman toimintakykyisenä. [6]

Mikropalveluiden teknologiariippumattomuus on etu, mutta se tuo myös uusia haavoittuvuuksia järjestelmään. Jos otetaan käyttöön useita eri ohjelmointikieliä ja teknologioita, on varauduttava myös siihen, että näistä jokaisella on omat tietoturvaongelmansa. Mahdolliset haavoittuvuuksien lähteet lisääntyvät, kun teknologioiden määrää kasvatetaan [3].

Käyttöönotetun koodin hallinta ja seuranta on tarpeellista, kun esimerkiksi uutta ominaisuutta kokeillaan tuotannossa. Vanhan version korvaaminen uudella on toteutettava pal-

velun käyttäjän kannalta huomaamattomasti siten että katkoja palvelussa ei synny. Käyttöönotettu ominaisuus täytyy olla mahdollista myös peruuttaa, mikäli sen huomataan olevan viallinen tuotannossa tapahtuvan testauksen aikana. [6]

2.2 Docker - Konttiteknologia

Docker on kontteihin (container) perustuva virtuaalinen ajoympäristö ohjelmistoille [8]. Sen tarkoitus on luoda ohjelmille eristetty ympäristö, mikä sisältää kaikki ohjelman vaatimat riippuvuudet ilman, että niitä täytyy asentaa Dockeria suorittavalle isäntäkoneelle [9]. Dockerin päätarkoitus on mahdollistaa virtuaalikoneen kaltainen suoritusympäristö ohjelmille minimoiden samalla ajettavan ohjelman kannalta ylimääräiset ominaisuudet pois. Näin saavutetaan huomattavasti pienemmät koot konteille verrattuna virtuaalikoneisiin, sekä nopeampi ympäristön pystytys konttiin. [10]

Docker on noussut suureen suosioon kehittäjien keskuudessa viime vuosina [8]. Se syntyi ratkaisuna ongelmaan jatkuvasti muuttuvista kehitysympäristöistä. Ohjelmiston kehitysvaiheessa kehittäjillä voi olla useita eri kehitysympäristöön perustuvia oletuksia, jotka tuottavat ongelmia ympäristön muuttuessa. Näitä ovat muun muassa käyttöjärjestelmän versio, ohjelmointikielen tai ohjelmistokehityspaketin (software development kit, SDK) versio sekä muut mahdollisesti käyttöjärjestelmään asennetut ja taustalla pyörivät ohjelmistot. Kun uusi kehittäjä tulee mukaan kehitystyöhön tai ympäristössä tapahtuu muutoksia esimerkiksi käyttöjärjestelmän päivityksen seurauksena, voi kehitettävän ohjelman toiminta muuttua tai se voi lakata toimimasta kokonaan johtuen näistä ulkoisista muutoksista. Docker pyrkii ratkaisemaan tämän ongelman. [10]

Dockerin toiminnan voi ymmärtää reaalimaailman esimerkillä. Dockerin toiminta perustuu kontteihin, jotka tarjoavat kehitysympäristön työkalut ja jossa esimerkiksi mikropalveluita voi ajaa [10]. Sama kontin käyttötarkoitus on olemassa nykypäivän tavarankuljetuksessa, jossa tuotteet kuljetetaan pakatuissa konteissa. Samantyyppiset kontit ovat käytössä muun muassa laiva-, lento- ja junarahdissa. Ennen konttien yleistymistä tuotteiden valmistajat joutuivat varautumaan erilaisiin kuljetusympäristöihin, sillä esimerkiksi sama pakkauksen koko ei ollut optimaalinen jokaiseen kuljetustapaan, vaan jokaiselle kuljetusympäristölle saatettiin suunnitella oma pakkaustapa [10]. Kyseinen vertaus pätee suoraan myös Dockerin kontteihin. Kontit tarjoavat standardin ympäristön ohjelmistolle, jolloin ympäristön muutokset tapahtuvat hallitusti ja ne voidaan tarvittaessa myös peruuttaa. Kontti sisältää sen suorittaman sovelluksen tarvitsemat riippuvuudet, kirjastot ja mahdolliset konfigurointitiedostot. Kontin kehittäjä määrittää näiden versiot, jolloin kaikki kontissa ohjelmistoa ajavat saavat tismalleen saman ympäristön [10]. Kuten rahtikontit,

myös Dockerin kontit ovat ”pakkausvaiheessa” eli ohjelman konttiin laitossa alussa tyhjiä [10]. Kontti siis alustetaan uudelleen, jolloin edellisen suorituksen aiheuttamat muutokset eivät vaikuta ohjelman seuraavaan suoritukseen.

Samankaltainen toiminnallisuus on saavutettavissa myös virtuaalikoneita käyttämällä. Miksi virtuaalikoneen sijasta kannattaisi sitten valita Docker? Virtuaalikone sisältää koko käyttöjärjestelmän ja ajaa taustalla halutun ohjelmiston lisäksi käyttöjärjestelmän muita ohjelmia, joista valtaosa ei liity kehitettyyn ohjelmistoon. Virtuaalikone käyttääkin huomattavasti enemmän resursseja, kuten CPU:ta ja RAM-muistia, kuin yksittäinen kontti [8]. Dockerin kontit perustuvat käyttöjärjestelmän virtualisointiin kuten virtuaalikoneetkin, mutta jakavat käyttöjärjestelmän ytimen (kernel) fyysisen koneen kanssa, tehden konteista huomattavasti kevyempiä kuin virtuaalikoneista. Jos halutaan suorittaa yhtä aikaa useampaa virtuaalikonetta, tarvitaan yhtä monta virtualisoitua käyttöjärjestelmää, mikä vaatii huomattavasti resursseja tietokoneelta. Dockerin kontit taas pyörivät yhden ja saman virtualisoidun ”kevyen” käyttöjärjestelmän päällä, jolloin yksi tietokone kykenee ajamaan useaa konttia. [10] Edellisen rahtivertauksen tavoin Dockerin ja virtuaalikoneiden eron voi havainnollistaa laivan ja sen rahtikonttien avulla. Yksi laiva, kykenee kuljettamaan useamman rahtikontin kerralla, tehden rahtikonteista tehokkaan tavan kuljettaa tavaroita. Useamman laivan käyttäminen taas olisi erittäin raskas vaihtoehto kuljettaa vastaavan konttien määrän tavaraa. Esimerkissä laiva vastaa virtuaalikonetta ja kontti Dockerin konttia.

Käydään läpi Dockerin eri vaiheet, joilla sovellus saadaan ajoon. Dockerin käyttöönotto vaatii Docker-ohjelmiston asennuksen. Ajettavalle ohjelmalle luodaan seuraavaksi Dockerfile. Dockerfile on asennusohje tai määrittely halutulle ajoympäristölle. Sillä ohjataan Docker asentamaan tarvittavat riippuvuudet ja kirjastot, määritellään ympäristömuuttujia sekä tehdään muita vaadittuja alustustoimenpiteitä ajoympäristölle. [11]

Kun Dockerfile on valmis, siitä voidaan rakentaa (build) kuvia (image) [11]. Dockerin kuvat ovat rinnastettavissa tiedostojärjestelmään, ne ovat muokkaamattomissa olevia ympäristöjä, jotka odottavat käynnistystä [10]. Virtuaalikoneisiin verrattuna niiden voi ajatella olevan virtuaalikoneen käyttöjärjestelmän näköistiedosto. Kuva täytyy rakentaa uudelleen, jos Dockerfilea muokataan, sillä muutokset eivät automaattisesti tule mukaan jo luotuihin kuviin.

Seuraava koodi on esimerkki Dockerfilestä, joka määrittelee NodeJS-alustalla pyörivän sovelluksen ympäristön.

Ohjelma 1. *Dockerfile NodeJS-sovellukselle*

```
# Kuva perii viimeisimmän NodeJS-kuvan pohjaksensa.
```

```
FROM node:latest

# Asetetaan sovelluksen alkusijainti kuvassa.
WORKDIR /hello_world_application

# Kopioidaan sovelluksen vaatimat riippuvuudet sisältävä tiedosto kuvaan.
COPY package.json .

# Ajetaan kuvan sisällä tiedostojärjestelmää muokkaava komento.
# Komento asentaa package.json määrittelemät riippuvuudet konttiin.
RUN npm install

# Määritellään kontin käynnistyksessä ajettava komento.
# Kun kontti ajetaan, se kutsuu "npm run start".
CMD [ "npm", "run", "start" ]

# Kopioidaan ohjelman varsinainen sisältö kuvaan isäntäkoneelta.
# Kuvaan siirtyy esimerkiksi lähdekoodi ja konfiguraatiotiedostot.
COPY . .
```

Ohjelman 1 alussa määritellään Dockerfilen luoman kuvan perivän uusimman version NodeJS-kuvasta. Kuvat perustuvat usein toiseen kuvaan, joka sisältää jo ympäristön tarvitsemat tärkeimmät ohjelmat [11]. Tässä tapauksessa perittävä kuva sisältää erityisesti NodeJS-ohjelmointialustan valmiiksi asennettuna. Seuraavaksi asetetaan kuvan tiedostojärjestelmään työskentelykansio, eli oletussijainti kuvaan tehtäville operaatioille. Työskentelykansion asetuksen jälkeen siihen kopioidaan package.json-tiedosto, joka sisältää kehitettävän sovelluksen vaatimat kirjastot ja niiden versiot. Tämä tiedosto määrittää esimerkissä sovelluksen vaatiman ympäristön. Ympäristö asennetaan RUN-komennolla kuvan tiedostojärjestelmään, kun sillä ohjataan tiedostojärjestelmässä ajettavaksi komento npm install. Määritellään myös kontin käynnistyessä suoritettava komento, joka aloittaa työskentelykansiossa olevan ohjelman. Esimerkin sovellus käynnistyy komennolla npm run start. Viimeisenä vaiheena siirretään kuvaa luovalta tietokoneelta lähdekoodi, konfiguraatiotiedostot ja muut isäntäkoneen sovelluskansiossa olevat tiedostot kuvan tiedostojärjestelmään. Tämän komennon jälkeen kuva on valmis.

Kun kuva on valmiina, siitä voidaan käynnistää kontteja. Kontti on esiintymä kuvasta, joka voidaan käynnistää tai sammuttaa. Yhdestä kuvasta on mahdollista luoda useita kappaleita kontteja. Kontin sammuessa se palautetaan täysin alkutilaansa, jolloin ajon aikana tehdyt muokkaukset eivät jää kontin seuraavaan käynnistykseen mukaan. [9]

Mikropalveluiden yksi päävaatimus on niiden löyhä sidonnaisuus toisiinsa, ja palveluita oletetaan olevan useampi kappale [3]. Dockerilla on mahdollista hallita useamman kontin käynnistystä eri kuvista, joka on yksi tapa käynnistää mikropalvelu sen ollessa kokonaan pois päältä. Tämä toiminnallisuus saadaan aikaan Dockerin Compose-työkalulla, joka

suorittaa Compose-tiedoston, kehittäjän määrittelemän käynnistysohjeen ja -järjestyksen sovelluskokonaisuuden konteille. [9, 11]

2.3 Microsoft Azure Service Fabric - Palvelualusta

Service Fabric on Microsoftin tarjoama palvelualusta mikropalveluiden ja konttien hallintaan ja käyttöönottoon [5]. Service Fabric on osa Azure-palvelukokonaisuutta, joka tarjoaa pilvipohjaisen palvelualustan. Azure mahdollistaa palveluiden siirtämisen pilveen, jolloin sovelluksien vaatima laskentateho voidaan siirtää yrityksen ulkopuolelle [5].

Mikropalveluiden kannalta Service Fabric tarjoaa skaalautuvuutta, vikasietoisuutta sekä hallintatyökaluja. Skaalautuvuus syntyy laajasta palvelinverkostosta, joka kattaa usean mantereen [12]. Palveluita voidaan skaalata antamalla niille lisää laskentatehoa palvelinverkostoista tarvittaessa. Skaalaus voidaan suorittaa automaattisesti kysynnän mukaan, jolloin sovellukset reagoivat käyttäjämäärien nousuihin ja laskuihin [6]. Vikasietoisuutta saadaan asteittaisista päivityksistä sekä mikropalveluiden reitityksestä niiden tilan mukaan. Asteittaisilla päivityksillä tarkoitetaan päivityksiä, jotka tuodaan palveluun mukaan hallitusti keskeyttämättä palvelun toimintaa päivityksen ajaksi [5]. Päivitys keskeytyy, jos sen huomataan olevan viallinen. Mikropalveluiden reititys tarkoittaa mikropalveluiden tilan varmistamista ja kuorman jakamista niiden välillä. Service Fabric seuraa palveluiden vastauksia jatkuvasti ja varmistaa niiden toiminnan. Jos Jonkin osan huomataan toimivan väärin tai lakkaavan vastaamasta lainkaan, vaihdetaan osa toiseen, jolloin keskeytykset käytössä minimoidaan [2]. Hallintatyökaluihin taas sisältyy esimerkiksi hälytykset katkoksista palvelussa, käyttöliittymä konttien ja mikropalveluiden käynnistämiseksi ja sammuttamiseksi sekä päivitysten määrittäminen [6].

3. VERKKOKAUPPA MIKROPALVELUNA

Tässä luvussa esitellään esimerkkiprojekti, joka on tarkoitus toteuttaa mikropalveluna. Esitellään ohjelmiston vaaditut toiminnallisuudet ja mahdolliset rajoitteet. Valitaan komponenttijako ohjelman eri osille mikropalveluarkkitehtuurin mukaisesti.

3.1 Verkkokauppa

Kuvitteellinen vaatteiden myyntiin erikoistunut liike haluaa laajentaa toimintaansa perinteisistä myymälöistä verkkokauppaan. Liike toimii useissa eri valtioissa ja omaa suuren kysynnän sekä asiakaskunnan, mutta ei tähän mennessä ole toteuttanut itsellensä verkkomyymiseen vaadittavaa ympäristöä. Yleisiä vaatimuksia verkkokaupalle ovat luotettavuus, skaalautuvuus ja tietoturvallisuus.

Luotettavuudella halutaan vikasietoista palvelua. Erilaisiin virhetilanteisiin pitää varautua ja järjestelmässä tapahtuvat virheet minimoida etenkin kuluttajien näkökulmasta. Kaupan toimintojen keskeytykset täytyy rajoittaa pakollisiin osiin. Jos esimerkiksi kaupan varastosta tietojen hakemisessa on ongelmia, ei tuotteiden katselu saa asiakkaan osalta keskeytyä sen vuoksi.

Skaalautuvuudella varaudutaan verkkokaupan myynnin vaihteluun. Liike on havainnut myyntiensä moninkertaistuvan alennusmyyntien aikana, mutta toisaalta on myös ajanjaksoja, jolloin kysyntä on selvästi normaalia alhaisempaa. Verkkokaupan tulee kyetä palvelemaan poikkeuksellisen suurta asiakasmäärää kysyntäpiikkien aikana, mutta kulujen säästämiseksi verkkokaupan kapasiteettia ei haluta pitää jatkuvasti maksimissaan. Kaupan tulee kyetä alentamaan kapasiteettiaan ja siten käytettyjä resursseja kysynnän mukaan.

Luotettavuuden ja skaalautuvuuden lisäksi verkkokaupan halutaan toimivan pienillä viiveillä ja suurella käyttöastetakuulla (uptime guarantee). Verkkokaupan resurssit halutaan ulkoistaa yrityksen ulkopuoliselle palveluntarjoajalle, joka pystyy takaamaan vaatimukset.

3.2 Verkkokaupan toiminnallisuudet

Verkkokauppa tarjoaa asiakkailleen käyttöliittymän, jota kautta suurinta osaa palveluista käytetään. Tässä suunnitelmassa keskitytään sovelluslogiikan suunnitelmaan itse käyttöliittymän sijaan, mutta käyttöliittymän on tarkoitus ohjata toiminnot suunnitelluille komponenteille.

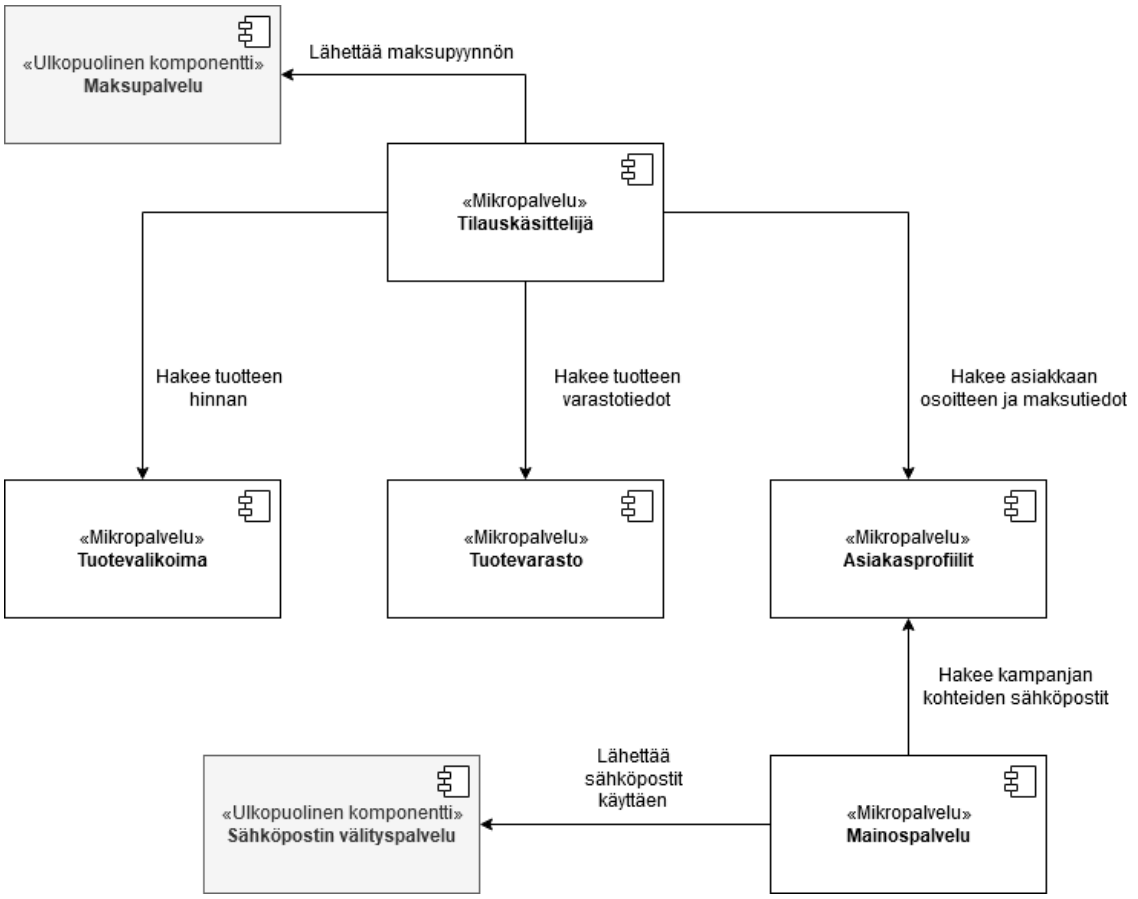
Verkkokaupassa on esiteltävä kaupan tuotevalikoimaa. Tuotevalikoimasta voi valita tuotteen, josta näkee tarkemman kuvauksen tuotteesta sekä sen hinnan. Tuotevalikoima on erillään kaupan tuotevarastosta, eli tuotetta voidaan esitellä, vaikka se ei juuri nyt olisi saatavilla varastossa. Tällöin tuotetta ei kuitenkaan voi tilata.

Asiakas voi luoda tilin (account) verkkokauppaan. Asiakas voi esimerkiksi syöttää tilille osoitetietonsa, jotka helpottavat tilauksen tekoa jatkossa. Asiakkaalle voidaan lähettää sähköpostilla viestejä erilaisista kampanjoista ja alennuksista. Kauppa haluaa ulkoistaa sähköpostien välityspalvelun.

Asiakas voi tilata verkkokaupasta tuotteen. Tilausta tehdessä kirjautunut asiakas saa tietonsa suoraan tilinsä tiedoista, mutta myös kirjautumaton asiakas voi tilata tuotteita täyttämällä vaadittavat tiedot ostoksen yhteydessä. Asiakas saa kuitin ostoksesta sähköpostiinsa. Maksu suoritetaan ulkoisen maksupalvelun kautta, joka maksun vahvistettuaan palauttaa tiedon kaupalle. Tilauksen toimittamiseksi työntekijät katsovat lopuksi tilaukset ja lähettävät ne tilauksessa mainittuun osoitteeseen.

3.3 Jako mikropalveluihin

Tehdään komponenttitason suunnitelma verkkokaupasta mikropalveluna. Mikropalveluiden jakamisen tärkeimpänä perusteena on loogisen kokonaisuuden muodostaminen, joka ei ole liian suuri, saati liian pieni. Osien on kyettävä toimimaan keskenään, mutta myös toisista riippumatta, mikäli esimerkiksi toinen mikropalvelu on kaatunut. Mikropalvelut valitaan osittamalla verkkokaupan toiminnallisuudet niitä vastaaviksi mikropalveluiksi. Kuvassa 1 esitellään mikropalvelun komponentit ja niiden suhteet toisiinsa.



Kuva 1. *Verkkokaupan mikropalvelut*

Verkkokauppa on jaettu seuraaviin mikropalveluihin: tilauskäsittelijä, tuotevarasto, tuotevalikoima, asiakasprofiilit sekä mainospalvelu. Kuvassa 1 on esitelty myös verkkokaupan ulkopuoliset komponentit, jotka integroidaan palveluun. Näitä ovat maksupalvelu ja sähköpostivälittäjä.

Tuotevalikoiman tehtävänä on säilöä tietoa tuotteista, jotka ovat kaupan valikoimissa. Säilöttyihin tietoihin kuuluu vähintään tuotteen nimi, hinta ja kuvaus tuotteesta. Palvelulta voidaan suorittaa kyselyitä tuotteista esimerkiksi aakkosjärjestyksessä, laskevan hinnan ja tuotekategorian mukaan. Tuotevalikoima ei tiedä tuotteen määrää varastossa, vain tiedot tuotteista, joita kauppa ylipäätään myy. Nämä tuotteiden metatiedot tallennetaan tuotevalikoiman omaan tietokantaan.

Tuotevarasto on vastaavasti vastuussa tuotteiden varastotilanteesta, eli siitä montako kappaletta tiettyä tuotetta on kaupan varastoissa. Tuotevarasto pitää kirjaa tuotteen nimestä tai vastaavasta uniikista numerotunnisteesta (ID), jonka avulla tuotteen tila voidaan hakea varastosta. Tunnisteen lisäksi ainoa tieto varastossa on kyseisen tuotteen määrä varastossa. Jokaisen tuotteen tunnisteen on oltava sama kuin se on tuotevalikoimassa. Tuotevarastoon voi tehdä kyselyjä tuotteen varastomäärästä. Tuotevarastolla on oma erillinen tietokanta kuten tuotevalikoimallakin, jonne se säilöö tietonsa.

Asiakasprofiilit hallitsee palveluna asiakkaiden tietoja. Nämä tiedot sisältävät tuotteiden toimitusta varten tarvittavat osoitetiedot, sekä mahdollisen tallennetun maksutavan. Tietojen tehtävänä on helpottaa tilausten tekoa, kun asiakkaan ei tarvitse täyttää tietojaan uudelleen. Asiakkaiden tiedot tallennetaan mikropalvelun omaan tietokantaan, erillään muista tietokannoista. Asiakasprofiilipalvelu vastaa kyselyihin tietyn asiakkaan tiedoista uniikin tunnisteen perusteella sekä massaoperaatioihin kaikista asiakkaista. Asiakkaan uniikki tunniste on sidoksissa tunnukseen, jolla asiakas kirjautuu verkkokaupan verkkosivuille, esimerkiksi sähköpostiin.

Tähän asti mainitut mikropalvelut ovat täysin riippumattomia muiden osien toiminnasta. Nämä mikropalvelut ovat siis erittäin skaalautuvia, ja samalla myös palvelun kriittisimmät osat. Jos esimerkiksi suuri joukko asiakkaita rekisteröityy sivuille yhtä aikaa, on asiakasprofiilikomponentin resurssit helposti kasvatettavissa. Jos taas verkkokaupan tuotteita katsotaan alennusmyynnin aikana moninkertaisesti normaaliin nähden, voidaan tuotevalikoiman resursseja kasvattaa. Sama pätee myös päinvastoin. Jos esimerkiksi pyhäpäivän takia verkkokaupassa on normaalia matalampi määrä kävijöitä, voidaan resursseja vähentää ja samalla säästää mikropalveluiden kuluissa.

Mainospalvelu lähettää kohdennetusti tietyille asiakasryhmille tai kaikille asiakkaille tietoa liikkeen kampanjoista sähköpostitse. Mainospalvelu saa tehtävänannon liikkeen työntekijöiltä joko toteutetusta käyttöliittymästä, joka kutsuu mainospalvelun rajapintaa. Mainospalvelu hakee kyselyä vastaavien asiakkaiden sähköpostit asiakasprofiileista massakyselyllä, jonka jälkeen se aloittaa sähköpostien välityksen. Mainospalvelu voi lähetyksen onnistumisen varmistumiseksi pitää kirjaa lähetettävistä sähköposteista omassa tietokannassaan, jolloin kaatumisen jälkeen sähköpostien välitystä voidaan jatkaa samasta kohdasta. Mainospalvelu lähettää sähköpostit ulkoiseen sähköpostien välityspalveluun, joka vastaa niiden perille toimittamisesta. Esimerkki tällaisesta palvelusta on Mailgun [13]. Jos suuren mainoskampanjan aikana sähköposteja on lähetettävänä suuri määrä kerralla, voidaan luoda uusia mainospalvelun esiintymiä ja osittaa sähköpostit niiden kesken, jolloin kuorman kasvuun voidaan reagoida.

Tilauskäsittelijä on muista palveluista riippuvaisin, mutta riippuvuudet ovat löyhiä. Se vastaa asiakkaan tilauksen tekemisestä verkkokaupassa, jolloin se tarvitsee tietoja muilta mikropalveluilta. Kun asiakas siirtyy tilaamaan tuotetta, varmistetaan ensin, onko kyseessä uusi asiakas vai onko asiakkaalla jo olemassa profiili. Jos asiakas on kirjautuneena palveluun, profiilin oletetaan olevan olemassa. Tällöin tilauskäsittelijä pyytää asiakasprofiililta kyseisen asiakkaan tiedot. Tilauskäsittelijä luodaan kuitenkin oletuksella, että jokainen muu mikropalvelu voi olla epäkunnossa. Jos asiakasprofiiliin ei saada yh-

teyttä kyselyllä, ohjataan asiakas täyttämään toimitustiedot ja maksutapa asiakasprofiilista riippumatta. Toimitustietojen täytön jälkeen voidaan verkkosivulla näyttää yhteenveto tuotteista. Käyttöliittymä hakee yhteenvetoa varten tuotteiden tiedot suoraan tuotevalikoimalta, jos se on toimintakykyinen. Tilauskäsittelijä hakee tuotteen hinnan tuotevalikoimalta. Seuraavaksi verkkokaupassa siirrytään maksuvaiheeseen. Jos asiakas valitsee maksutavaksi esimerkiksi pankkikortin, tilauskäsittelijä ohjaa pyynnön ulkoiselle maksupalvelulle. Maksupalvelu hyväksyy tai hylkää maksun, ja palauttaa tiedon maksusta tilauskäsittelijälle. Eräs potentiaalinen Suomessa toimiva maksupalvelu on Paytrail [14]. Tilauskäsittelijän tulee olettaa, että myös maksupalvelu voi olla kyvytön toimimaan. Jos maksupalvelu ei vastaa tilauskäsittelijän pyyntöihin, ehdotetaan asiakkaalle muita maksutapoja, kuten laskua. Näin asiakas voi tilata tuotteen myös maksupalvelusta riippumatta. Viimeisenä mikropalveluna tilauskäsittelijä tarvitsee tuotevaraston. Tuotevaraston täytyy olla saatavilla, jotta tilauskäsittelijä voi varmistaa siltä tuotteen saatavuuden. Tilauskäsittelijä pyytää tuotevarastoa vähentämään tuotevaraston tuotteita tilauksen mukaan maksun varmistuttua. Tuotteen saatavuus voidaan myös merkitä avoimeksi, jolloin tuotteita voi tilata vaikkei niitä olisikaan kyseisenä hetkenä varastossa. Tuotevarastosta riippuvuutta se ei siltikään poista, sillä tuotevarasto myös varmistaa tuotteen olemassaolon sekä sen, saako sitä tilata avoimena tilauksena. Tilauskäsittelijä pitää tapahtumistaan kirjaa omassa tietokannassaan. Tietokannan olennaisin tieto on maksetut, mutta toimittamattomat tilaukset. Liikkeen työntekijät voivat varmistaa tilauksen ja aloittaa toimituksen tilauksen päätteeksi.

4. TEKNINEN TOTEUTUS

Tässä luvussa perehdytään verkkokaupan mikropalveluiden tekniseen suunnitteluun tarkemmin. Luvussa valitaan mikropalveluille ajoalusta, analysoidaan arkkitehtuurin kelpoisuutta globaaliin toimintaan sekä perustellaan mikropalveluiden tietokantojen jako.

4.1 Palvelualusta

Verkkokaupan kaikki mikropalvelut käynnistyvät Docker Composen avulla. Jokaiselle mikropalvelulle luodaan oma kontti, jotka on mahdollista käynnistää myös toisistaan riippumatta. Verkkokauppa on mahdollista sijoittaa yrityksen omiin tiloihin, mutta ulkoistamalla mikropalvelut voidaan hyödyntää palvelualustojen ominaisuuksia.

Microsoftin Azure Service Fabric on potentiaalinen valinta verkkokaupan palvelualustaksi. Service Fabric tarjoaa ympäristön mikropalveluiden hallinnointiin. Sen kautta mikropalvelut saadaan skaalautumaan automaattisesti kysynnän mukaan ja päivitykset saadaan suoritettua jatkuvasti siten, että palvelu ei keskeydy niiden ajaksi. Service Fabric myös lupaa palveluiden toipuvan virheistä itsenäisesti. [5]

Palvelualustan käyttäminen sopii verkkokaupan globaaliin markkinaan. Esimerkiksi Azuren palvelinkeskuksia on useita kappaleita eri mantereilla, joka takaa lyhyet vasteajat verkkokaupalle liki mistä päin tahansa maailmaa [12].

Myös muut palvelualustat tarjoavat vastaavia ominaisuuksia erilaisilla hinnoittelumalleilla. Toteutuksen alkaessa on syytä tarkistaa sopivin palvelualusta kyseisenä ajanhetkenä. Azure ja sen Service Fabric ovat valittu työssä esimerkeiksi, joilta löytyivät kaikki halutut palvelut.

4.2 Tietokannat

Mikropalvelut suunniteltiin siten, että niillä on omat tietokantansa yhden yhteisen tietokannan sijaan. Näin vältetään tilanne, jossa mikropalveluiden välille syntyy riippuvuus yhteisen tietokannan kautta. Mikäli mikropalvelut käyttäisivät yhteistä tietokantaa, olisi todennäköisempää, että eri mikropalveluille syntyy oletuksia tietokannan rakenteesta, jonka perusteella mikropalvelua kehitetään. Mikropalveluilla on kuitenkin eri käyttötarkoitukset ja yhteisen tietokannan muuttamisen vaikutuksia täytyisi seurata jokaisessa sitä käyttävässä mikropalvelussa. Näin tietokanta itsessään loisi yhteyden mikropalveluiden välille, mikä estää mikropalveluiden itsenäistä toimintaa. Kun jokaisella mikropalvelulla on oma tietokantansa, ne saavat enemmän vapauksia tietokannan rakenteen suhteen.

Tietokantoja voidaan myös skaalata mikropalvelutasolla osiin, joihin sitä tarvitaan, koko järjestelmän yhteisen tietokannan skaalaamisen sijaan.

Tietokantojen jakaminen hyödyttää myös verkkokaupan palveluiden jakamisessa alueittain. Vaikka verkkokauppa toimiikin globaalisti, saattaa se haluta hajauttaa tuotevarastojaan ympäri maailmaa pienentääkseen toimitusaikoja ja mahdollisia tullimaksuja valtioiden välillä. Tuotevarastot voidaan jakaa alueittain, siten että ne hyödyntävät tietyn alueen paikallista tuotevarastoa globaalin varaston sijaan. Tietokannat voidaan siis jakaa myös alueittain, jolloin näytetään kyseisen alueen tuotteet.

4.3 Teknologiat

Koska mikropalvelut ovat vain verkkopyyntöjen välityksellä yhteydessä toisiinsa, voivat ne käyttää omia teknologiaratkaisuita vaikuttamatta muiden mikropalveluiden toimintaan [3]. Verkkokaupan tapauksessa tätä voidaan hyödyntää erityisesti tehokkuuden optimoinnissa.

Tuotevarasto on tärkeä osa verkkokauppaa, ja tulee luultavasti kokemaan suuria pyyntömääriä yhtä aikaa niin asiakkailta kuin varaston työntekijöiltä. Tuotevarasto ei ole asiakkaille suoraan nähtävissä oleva osa ja keskittyykin pelkästään toiminnan tehokkuuteen. Tuotevaraston teknologiaratkaisut kannattaa siis valita tehokkuutta priorisoiden. Koska tuotevaraston operaatiot ovat enimmäkseen tietokantaoperaatioita, on tietokannan sopivuudesta tehtävään syytä varmistua. Itse mikropalvelu on syytä toteuttaa alhaisen tason ohjelmointikielellä tehokkuuden varmistamiseksi. Esimerkiksi C ja C++ ovat tehokkaita ohjelmointikieliä tähän tarkoitukseen [15].

Kun mikropalvelu ei ole tehokkuuskriittinen, voidaan vastaavasti keskittyä korkeamman tason ratkaisuihin kehityksen nopeuttamiseksi. Esimerkiksi mainospalvelu ei ole aikakriittinen, sillä sähköpostien ei edes oleteta kaikkien saapuvan yhtä aikaa.

4.4 Johtopäätökset

Mikropalveluilla saatiin huomattavia etuja verkkokaupan palveluihin. Tavoitteena oli saada verkkokaupasta kävijämäärille skaalautuva ja vikasietoinen. Mikropalvelut olivat toimiva ratkaisu skaalautuvuuden saamiseksi. Verkkokaupan jakaminen osiin mahdollistaa skaalaamisen yksittäisten mikropalveluiden tasolla sen lisäksi, että jo itse arkkitehtuuri ohjaa kehittäjiä luomaan eri komponentit itsenäisiksi ja täten helposti skaalattaviksi. Tietokantojen eristäminen jokaisen mikropalvelun omaksi estää syntymästä tilannetta, että riippuvuudet yhteiseen tietokantaan hidastaisivat skaalausta.

Mikropalveluarkkitehtuuri ohjaa verkkopalvelua myös vikasietoiseen toimintaan. Valittu palvelualusta huomaa vialliset komponentit ja korvaa ne itse toimivalla vastineella. Suurin osa luoduista mikropalveluista toimii itsenäisesti, ja toimivat vaikka jossain toisessa mikropalvelussa olisi virhe. Muita mikropalveluita käyttävät mikropalvelut kuten tilauskäsittelijä ohjattiin luotavaksi oletuksella, että muut mikropalvelut voivat milloin vain olla epäkunnossa. Näin myös tilauskäsittelijä tarjoaa palveluaan rajatusti, jos sen käyttämä toinen mikropalvelu ei vastaa pyyntöihin.

Mikropalveluarkkitehtuurin edut ovat merkittäviä, ja uskon niiden hyötyjen ylittävän kustannukset, jotka mikropalveluarkkitehtuuriin siirtymisessä tai sen luomisessa syntyy. Poikkeuksena pidän hyvin pieniä ohjelmistoja, jotka hyötyisivät enemmän suoraviivaisemmasta kehitystavasta. Hyvin pieniä ohjelmistoja voi olla vaikeaa jakaa mikropalveluihin ja niiden suora kehittäminen mikropalveluihin jakamisen sijaan tuottaa mielestäni parempia tuloksia.

5. YHTEENVETO

Työssä esiteltiin mitä mikropalvelut ovat ja niiden toteutuksessa hyödyllisiä työkaluja. Esitelly työkalut olivat mikropalveluiden ajoalustana toimiva Docker sekä Microsoftin Azure Service Fabric, jolla hallinnoidaan ja skaalataan Docker konttien resursseja.

Mikropalveluille esitettiin soveltamiskohteeksi verkkokauppa. Verkkokauppa sisälsi useita eri vaatimuksia globaalina toimijana, etenkin kustannustehokkaita ratkaisuita alennusmyyntien kävijäpiikkien aiheuttaman kuormituksen ratkaisemiseen sekä vakaata toiminnallisuutta, jolloin verkkokauppa kykenee toimimaan myös jonkin sen osa-alueen ollessa epäkunnossa. Verkkokauppa jaettiin mikropalveluihin, jotka ovat löyhästi toisiinsa sidottuja. Omiksi mikropalveluikseen eriytettiin tilauskäsittelijä, tuotevarasto, tuotevalikoima, asiakasprofiilit ja mainospalvelu.

Skaalautuvuuden ja vikasietoisuuden takaamiseksi mikropalveluita suositeltiin ajettavaksi palvelualustassa, kuten Microsoft™ Azure Service Fabric. Jokaisen mikropalvelun todettiin tarvitsevan oma tietokantansa, jotta ei synny riippuvuutta yhteen tietokantaan.

LÄHTEET

[1] Villamizar, M. et al. (2017) Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. Service Oriented Computing and Applications

[2] S. Ahsan, A. Ahuja, J. Chen, M. Coskun, C. Daniel, M. Fussell, I. Gupta, A. Gupta, R. Hasha, G. Kakivaya, V. Kidambi, R. Kong, Y. Li, M. Lin, D. Mastrian, A. Mhatre, V. Modi, M. Mohsin, R. Nair, B. Narasimman, O. Platon, T. Pleiger, A. Ram, A. Rao, S. Shivaprakash, R. Sinha, M. Snider, P. Subbarayalu, M. Tarta, R. Wang, A. Warwick, A. Wun, L. Xun, Service fabric: a distributed platform for building microservices in the cloud, 2018. Saatavissa: https://dl-acm-org.libproxy.tuni.fi/citation.cfm?id=3190546

[3] S. Newman, Building Microservices: Designing Fine-Grained Systems, 2015, O'Reilly Media Inc, ISBN: 1491950315, 9781491950319

[4] Docker Use Cases. Saatavissa: https://www.docker.com/use-cases

[5] Microsoft Azure Service Fabric. Saatavissa: https://azure.microsoft.com/en-us/services/service-fabric/

[6] Chawla, H. & Kathuria, H. (2019) Building Microservices Applications on Microsoft Azure Designing, Developing, Deploying, and Monitoring. 1st ed. 2019. Berkeley, CA: Apress. ISBN: 9781484248287

[7] C. Esposito, A. Castiglione, K. R. Choo, Challenges in Delivering Software in the Cloud as Microservices, IEEE Cloud Computing, vol 3, no. 5, pp. 10-14, 2016. Saatavissa: http://ieeexplore.ieee.org.libproxy.tuni.fi/stamp/stamp.jsp?tp=&arnumber=7742281&isnumber=7742214

[8] D. Swersky, "What is Docker, and why is it so popular?", 2018, Raygun. Saatavissa: https://raygun.com/blog/what-is-docker/

[9] Docker overview, Docker v19.03. Saatavissa: https://docs.docker.com/get-started/overview/

[10] Jangla, K. (2018) Accelerating Development Velocity Using Docker Docker Across Microservices . 1st ed. 2018. Berkeley, CA: Apress. ISBN: 978-1-4842-3935-3

[11] Development workflow for Docker apps, Microsoft. Saatavissa: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/docker-application-development-process/docker-app-development-workflow

[12] Azure Global Infrastructure, Microsoft. Saatavissa: https://azure.microsoft.com/en-us/global-infrastructure/

[13] Sähköpostivälityspalvelu Mailgun. Saatavissa: https://www.mailgun.com/

[14] Maksupalvelu Paytrail. Saatavissa: https://www.paytrail.com/maksupalvelu

[15] Ceccon, F. et al. (2016) Momentum Strategies: Comparison of Programming Language Performance. Journal of Trading. Saatavissa: http://search.proquest.com/docview/1826404610/.