Perinteisistä monoliittisista palvelimista palvelimettomaan

Oulun Yliopisto
Tieto- ja sähkötekniikan tiedekunta
Kandidaatintutkielma
Joona Heiskanen
21.01.2022

Tiivistelmä

Yritykset ovat nykypäivänä yhä isomman paineen alla kasvavien datamäärien, sovellusten kompleksisuuden kasvamisen ja nousevan kilpailun takia. Palvelimeton arkkitehtuuri on luomassa uusia tapoja toteuttaa sovelluksia lyhyemmässä ajassa ja olemalla kustannustehokkaampi kuin perinteiset ratkaisut. Ala onkin kokemassa yhdenlaista murrosta ja yhä enenevissä määrin siirtymässä käyttämään palvelimetonta arkkitehtuuria. Se ei ole kuitenkaan aina paras ratkaisu ja sen edut ja haitat olisi hyvä tiedostaa etukäteen.

Tutkielmassa tarkasteltiin tutkimusongelmana mitä asioita palvelinarkkitehtuurista päättävän tahon tulisi ottaa huomioon ennen päätöstä palvelimettoman järjestelmän kehityksestä tai siihen siirtymisestä perinteisen monoliittisen palvelimen sijaan. Tutkielma on toteutettu kirjallisuuskatsauksena.

Tutkielmassa tuli esille useita eri asioita, joita tulee ottaa huomioon ja jotka voivat vaikuttaa valittaessa perinteisen monoliittisen palvelinmallin ja palvelimettoman arkkitehtuurin välillä. Samalla selvisi, ettei palvelimeton sovellu automaattisesti kaikkiin käyttötarkoituksiin. Lisää tutkimusta aiheesta kuitenkin kaivattaisiin, sillä se on yritysmaailman verrattuna huomattavasti jäljessä. Moni tähänastinen tutkimus keskittyy lähinnä taloudellisiin seikkoihin ja palvelimien skaalautumisen eroihin, joten tilaa tutkimuksille uudenlaisista näkökulmista riittäisi.

Avainsanat
monoliittinen, palvelimeton, FaaS, palvelin, pilvipalvelu

Ohjaaja
Dorina Rajanen

Sisällysluettelo

1. Johdanto

Pilvipalvelut ja pilvilaskenta ovat yleistyneet huimasti viimeisen vuosikymmenen aikana. Yritysmaailmassakin kehitystä on tapahtunut ja yhä useampi ostaakin IT infrastruktuuria pilvipalveluna. Pilvipalveluiden markkinat ovat kasvaneet vuosi vuodelta ja palvelimettoman suosio siinä samalla. Vuoden 2021 kolmannella kvartaalilla yritysten pilvipalveluihin kohdistetut menot ylittivät 45 miljardia dollaria, joka oli 37-prosenttia kasvua vuoden takaisesta (Synergy Research Group, 2021). Erityisesti tutkielman aiheena oleva palvelimeton arkkitehtuuri on nostanut suosiotaan viime vuosina sovelluskehittäjien keskuudessa.

Käyttäjämäärien ja datan noustessa, sekä sovellusten kompleksisuuden ja kilpailun kasvaessa yritykset ovat alkaneet pohtia vaihtoehtoja perinteisille palvelinratkaisuille. Palvelimeton arkkitehtuuri tarjoaa uusia mahdollisuuksia yrityksille toteuttaa ohjelmistoratkaisuja verrattuna siihen, että käytössä olisi perinteinen monoliittinen palvelinratkaisu. Se ei sovellu välttämättä jokaiseen käyttötarkoitukseen, mutta voi tarjota merkittäviä hyötyjä ja ratkaisuja ongelmiin, joihin törmättäisiin monoliittisten palvelimien kohdalla. Palvelimettoman arkkitehtuurin avulla voidaan saavuttaa esimerkiksi säästöjä taloudellisesti ja vähentää tarvetta yrityksen sisäiseen palvelininfrastruktuurin ylläpitämiseen. Kratzken (2018) mukaan yritykset nykypäivänä hyödyntävätkin palvelimetonta arkkitehtuuria, koska se mahdollistaa yrityksen luoda merkittävästi enemmän taloudellista arvoa pienemmälläkin työntekijämäärällä. Yksi esimerkki palvelimetonta hyödyntävästä yrityksestä on nykyisin kaikkien tuntema Netflix, joka käyttää palvelimetonta arkkitehtuuria perustoimintojensa toteuttamiseen (Kratzke, 2018).

Verrattuna monoliittiseen palvelinratkaisuun, palvelimettomassa ratkaisussa palvelun infrastruktuuri on siis pilvessä. Järjestelmä koostuu pienistä funktioista, joita voidaan suorittaa aina tarpeen vaatiessa itsenäisesti. Tämä on erityisen tärkeä tekijä siinä miksi palvelimeton voi olla paljon tehokkaampi ratkaisu. Se ei kuitenkaan ole automaattisesti parempi vaihtoehto ja sen käyttöä tulisi harkita kulloisenkin tarpeen mukaan, sen sijaan että valittaisiin automaattisesti palvelimeton arkkitehtuuri.

Tutkielman aihe on erityisesti kiinnostava esimerkiksi yrityksille, jotka pohtivat seuraavan sovelluksensa arkkitehtuurisia ratkaisuja tai jopa niille, jotka haluavat päivittää vanhan sovelluksen nykyaikaisempaan muotoon ja samalla muuttaa sen palvelinratkaisua. Vastaavasti tietynlaisia parannuksia, kuten parempaa skaalautuvuutta hakeville, palvelimeton lupaa paljon ja voi olla houkutteleva vaihtoehto.

Tässä tutkielmassa tutustutaan perinteiseen monoliittiseen palvelinmalliin ja palvelimettoman arkkitehtuurin perusideaan. Lisäksi tutustutaan olennaisiin palvelimettoman palveluntarjoajiin sekä siihen miksi ja milloin perinteisestä palvelinmallista on kannattavaa siirtyä palvelimettomaan ratkaisuun. Tutkielma on toteutettu kirjallisuuskatsauksena. Aihe on alati kehittyvä ja tutkimusta on vielä suhteellisen vähän tai se on hyvin tuoretta, joten lähteenä on jouduttu käyttämään myös muita kuin puhtaasti tieteellisiä papereita. Esimerkiksi palveluntarjoajien tarjoamien palveluiden ominaisuuksista on haettu tietoa suoraan heidän omista dokumentaatioistaan mahdollisimman ajantasaisen tiedon saamiseksi.

Tutkielmassa tarkasteltiin tutkimusongelmana mitä asioita palvelinarkkitehtuurista päättävän tahon tulisi ottaa huomioon ennen päätöstä palvelimettoman järjestelmän kehityksestä tai siihen siirtymisestä perinteisen monoliittisen palvelimen sijaan. Tutkimus

on toteutettu kirjallisuuskatsauksena aiheesta tehtyihin tutkimuksiin, tieteellisiin papereihin ja saatavilla olevaan aineistoon, eikä siis sisällä muutoin omaa tutkimusta. Tutkielma käsittelee palvelimettoman heikkouksia ja hyviä puolia usealta eri näkökannalta. Tämän pohjalta tutkimuksen lopuksi on toteutettu pohdintaa siitä, miksi palvelimettomaan arkkitehtuuriin siirtyminen voi olla kannattavaa tai milloin se on järkevää.

Tutkielman pääasiallisina löydöksinä oli, että palvelimeton soveltuu erityisesti suurempiin projekteihin. Sen avulla voidaan luoda säästöjä sekä skaalata järjestelmiä pidemmälle, kuin mihin monoliittiset järjestelmät kykenevät. Perinteiset palvelimet eivät kuitenkaan ole turhia, vaan voivat olla jopa parempi vaihtoehto. Palvelimetonta arkkitehtuuria pidettiin usein vaikeana tai työläänä toteuttaa. Samoin useat löydökset tukivat päätelmää siitä, että palvelimeton ei juurikaan sovellu käyttökohteisiin, jossa sillä suoritettavat prosessit ovat kookkaita, pitkäaikaisia tai dataintensiivisiä.

Rajoituksina tutkielmassa ei varsinaisesti keskitytä siihen, miten sovellukset tai palvelut käytännössä toteutettaisiin, koska toteutustapoja ja käyttökohteita on lukemattomia. Aihetta tarkastellaan sen sijaan paljon ylemmän tason näkökulmasta. Varsinaista omaa muuta tutkimusta ei myöskään suoriteta, vaan tulos pohjautuu kirjallisuuskatsauksena kerätyn tiedon varaan. Pieniltä osin tuloksia rajoittaa myös kohteeksi valitut palveluntarjoajat. Koska pilvipalveluita ja palvelimettoman palveluntarjoajia alkaa nykypäivänä olla hyvinkin monia, ei niitä kaikkia voitu tämän tutkielman puitteissa ottaa tarkasteltavaksi, minkä lisäksi kaikista alustoista ei ole tutkimusmateriaalia vielä saatavilla.

2. Palvelimetonta edeltäneet arkkitehtuurit ja pilvipalvelut

Tässä kappaleessa tutustutaan pääasiassa monoliittiseen palvelinarkkitehtuuriin. Lisäksi tutustutaan mikropalveluihin sekä palvelimetonta edeltäneisiin pilvipalveluihin, jotka siihen olennaisesti liittyvät.

2.1 Monoliittinen palvelinarkkitehtuuri

Monoliittinen palvelin on yleisesti käytetty ilmaus perinteisestä yksinkertaisesta palvelinrakenteesta, jossa järjestelmän palvelinpuolen toteutus (eng. backend) koostuu ja on sijoitettuna yhtenä kokonaisuutena palvelimelle. Sen vastuulla on käytännössä bisneslogiikka kokonaisuudessaan, tietokantapyynnöt, HTTP-pyyntöihin vastaaminen ja mahdollisesti muutakin sovelluskohtaisesta toteutuksesta ja tarpeista riippuen. Pienenkin osan muutos vaatii koko järjestelmän päivittämisen ja tuotantoon viennin, koska kyseessä on yksi looginen kokonaisuus, eli *monoliitti*. (Fowler & Lewis, 2014.)

Koska kuormitus monoliittisella palvelimella voi vaihdella merkittävästi, hyödynnetään niiden tapauksessa yleensä myös ylivarausta (eng. overprovisioning) palvelimien määrän tai resurssien tehon suhteen. Palvelimella on järkevämpää olla liikaa resursseja, kuin liian vähän. Tämä siksi, ettei palvelin ylikuormitu ja hidastu kuormituksen kasvaessa. Pahimmassa tapauksessa palvelin voi kaatua rasituspiikkien aikaan, mikäli käytössä ei ole vaadittua määrää resursseja. Tästä johtuva tarve ylivaraukselle nostaa erityisesti sen käyttökustannuksia, koska resursseja pitää olla huomattava määrä varalla (Roberts & Chapin, 2017.)

Perinteinen palvelinarkkitehtuuri ei ole missään nimessä turha. Se on luonnollinen toteutustapa ja varsinkin sovellusprojektien alussa yksinkertainen tapa luoda järjestelmän serveripuolen toteutus. Pienille järjestelmille se on lähestyttävämpi ja helpompi tapa aloittaa ohjelmiston kehitys. (Villamizar ja muut, 2017.) Tämä johtuu siitä, että kaikki logiikka on yhden loogisen kokonaisuuden alla. Hyvällä suunnitelulla se on mahdollista pitää järkevänä kokonaisuutena myös sen koon kasvaessa. Skaalausta voidaan hoitaa tiettyyn pisteeseen asti horisontaalisesti, lisäämällä resursseja ja ottamalla käyttöön kuormituksen tasapainottaja (eng. load balancer), joka tasaa kuormitusta palvelimien välillä (Fowler & Lewis, 2014.)

Fowlerin ja Lewisin (2014) mukaan suurimpiin ongelmiin tällaisten ratkaisujen kanssa aletaan kuitenkin törmätä järjestelmän koon ja monimutkaisuuden kasvaessa. Pienikin muutos vaatii koko järjestelmän uudelleenkoonnin ja erillisen tuotantoon viennin. Samalla kokonaisuuden hahmottamiskyky heikkenee järjestelmän kompleksisuuden kasvaessa. Skaalautuvuuden parantamiseksi tarvitaankin usein koko järjestelmän eri osien parantamista.

2.2 Mikropalveluarkkitehtuuri

Monoliittisten ohjelmistojen rajoituksien myötä syntyneen mikropalveluarkkitehtuurin voi nähdä yhdenlaisena edeltäjänä palvelimettomalle arkkitehtuurille. Kuten Newman (2015) toteaa, monesti monoliittisten järjestelmien koon kasvaessa myös niiden koodikanta alkaa pirstaloitua ja samankaltaisten funktioiden toiminnallisuus hajaantuu

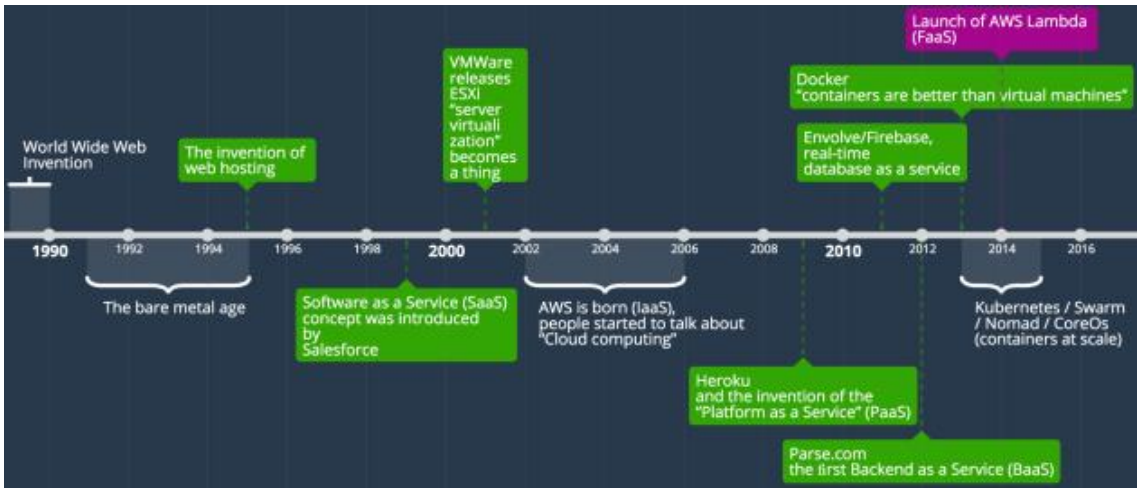
sitä myöten ympäri ohjelmiston eri osia. Tämän estämiseksi koodia yritetään yleensä abstrahoida ja jakaa omiin moduuleihin koheesion avulla. Tämän ajattelumallin pohjalta on lopulta syntynyt mikropalvelut, jossa ohjelmiston koodi jaetaan omiin loogisiin palveluihin ylläpidettävyyden parantamiseksi. Jokainen mikropalvelu on oma eristetysti tuotantoon vietävissä oleva entiteettinsä, ja ne kommunikoivat keskenään niiden omien rajapintojensa kautta.

Mikropalvelurakenteen etu verrattuna monoliittisiin järjestelmiin on, että järjestelmä on hajautettuna useampaan palaseen ja yhden vikaantuessa muut osat voivat vielä toimia itsenäisesti. Tuotantoon vienti myös helpottuu, kun koko järjestelmää ei tarvitse päivittää kerralla. Mikropalvelut helpottavat myös skaalautuvuutta järjestelemän koostuessa useammista palveluista, jotka voivat toimia itsenäisesti omilla alustoillaan. (Newman, 2015.)

Mikropalvelut eivät toki sovellu jokaiseen käyttötarkoitukseen ja joissain tapauksissa monoliittinen rakenne on järkevämpää kuin mikropalvelut. Lisäksi huonosti toteutetuissa mikropalveluratkaisuissa voidaan ajautua lopulta ongelmiin, jotka vastaavat monoliittisia palveluita, kuten mikropalveluiden koon ja kompleksisuuden kasvaminen liian isoksi, tai täysin uudenlaisiin mikropalveluita koskeviin ongelmiin. (Jamshidi, Pahl, Mendonça, Lewis & Tilkov, 2018.)

2.3 Pilvipalvelut

Useat as-a-service-tyyppiset pilvipalvelut tulivat markkinoille ennen palvelimetonta. Kuten Roberts ja Chapin (2017) mainitsevat, palvelimetonta voidaan pitää omalta osaltaan kaikkien näiden yhdistelmänä tai niiden evoluutiona. Yksi ensimmäisistä pilvipalveluista oli Amazonin AWS palveluun tullut EC2 IaaS-palvelu. IaaS keskittyy nimensä mukaisesti lähinnä siihen, ettei yritysten tarvitse huolehtia infrastruktuurista, jonka päällä palvelinohjelmisto toimii.



Kuva 1. Aikajanalla havainnollistettuna eri pilvipalveluiden syntyvaiheet (Kumar, 2019).

Seuraava askel pilvipohjaisissa palveluissa oli Platform-as-a-Service kuva 1:n mukaisesti. Erityisesti tämän teki tunnetuksi Heroku. PaaS rakentaa IaaS-mallin päälle tarjoten käyttöjärjestelmän ja siihen liittyvät päivitykset myös palveluntarjoajan puolelta, sekä esimerkiksi automaattisen monitoroinnin. Tällöin käyttäjä voi yksinkertaisesti viedä ohjelmiston tuotantoon PaaS:n avulla helpommin kuin IaaS:n tapauksessa. Myöhemmin

markkinoille on tullut vielä korkeamman tason abstrahointia tarjoava Container-as-a-service eli CaaS (Roberts & Chapin 2017.)

3. Palvelimeton arkkitehtuuri

Tässä luvussa käsitellään tutkielman kannalta kiinnostavinta osa-aluetta, eli palvelimetonta arkkitehtuuria. Alussa käydään läpi perustietoa liittyen palvelimettomaan, jonka jälkeen tutustutaan suurimpiin palveluntarjoajiin ja niiden eroihin pääpiirteissään. Tämän jälkeen käydään läpi aineistoa siitä, mitä etuja palvelimettomaan siirtymisestä on ja milloin sitä tulisi käyttää, sekä mihin käyttötarkoituksiin se ei sovellu ja mitä ongelmia sen käytössä voi esiintyä. Lopuksi on käytynä läpi tutkimuksia koskien migraatiota perinteisistä palvelimista palvelimettomaan.

3.1 Palvelimettoman synty

Alun perin perinteisistä palvelimista on siirrytty ensin mikropalvelu-tyyppiseen ratkaisuun, jossa monoliittinen palvelin jaetaan useampaan pienempään helpommin hallittavaan ja päivitettävään kokonaisuuteen, eli palveluun. Tällaisessa ratkaisussa on kuitenkin yleensä vielä kyse huomattavasti isommista palasista järjestelmää, kuin palvelimettoman tapauksessa. (Dragoni ja muut, 2017.) Tämän jälkeen markkinoille tulivat useat eri as-a-service-tyyppiset pilvipalvelut, kuten IaaS ja PaaS. Tästä on siirrytty taas palvelimettoman ja FaaS-mallin suuntaan ja nämä kolme eri vaihetta on erinäisiä havaittavissa olevia kehitysvaiheita matkalla palvelimettomaan (Kratzke, 2018).

Palvelimeton (eng. Serverless) nimi pilvipalvelukontekstissa juontaa juurensa vuoteen 2014, jolloin Amazon julkaisi AWS Lambda tuoteperheensä re:Invent tapahtumassa (viitattu lähteessä Castro, Ishakian, Muthusamy, & Slominski, 2019). Tämä oli ensimmäinen varsinainen FaaS-pohjainen palvelimeton palvelu (Castro ja muut, 2019). Google, Microsoft ja IBM julkaisivat omat vastineensa FaaS-palveluista vuonna 2016 (Eismann ja muut, 2020). Myös monet muut isot alan toimijat ovat julkaisseet omat vastineensa palvelimettomalle, kuten Oracle ja CloudFlare.

3.2 Palvelimeton arkkitehtuuri

Palvelimettomalla (eng. Serverless) arkkitehtuurilla tarkoitetaan pilvipalvelutyyppistä palvelua, jossa kehittäjien ei tarvitse huolehtia servereidensä infrastruktuurista tai skaalautuvuudesta. Palvelimettomassa arkkitehtuurimallissa serverit ovat kyllä edelleen olemassa, mutta niitä hallinnoi kolmas osapuoli, jolta palvelua ostetaan. Tämä kolmas osapuoli hallinnoi myös esimerkiksi palvelimien ylläpitoa. Palvelimettoman mallin mukaisesti resursseja skaalataan tarpeen mukaan automaattisesti. Tällä pystytään esimerkiksi eliminoimaan paljon kustannuksia, kun resursseja käytetään vain tarpeen vaatiessa. (Red Hat, 2020a.)

Tämä eroaa siis myös rakenteeltaan muista perinteisemmistä pilvipalveluista, kuten Infrastructure-as-a-Service (IaaS) -palvelumallista, jossa yritykset ostavat aina päällä olevia servereitä, joita palveluntarjoaja hallinnoi (Red Hat, 2020a.) Palvelimettomassa onkin yleensä kyseessä kokonaisvaltainen palvelu, joka ostetaan kolmannelta osapuolelta. Se onkin tavallaan yhdistelmä useita tunnettuja as-a-Service tyyppisiä ratkaisuja, kuten IaaS, PaaS tai BaaS, jotka se kokoaa yhdeksi kokonaisuudeksi (Roberts & Chapin, 2017).

3.2.1 Function-as-a-Service

Function-as-a-Service (FaaS) termiä käytetään useissa lähteissä melko huolettomasti lähes synonyyminä palvelimettomalle, vaikkakin se on todellisuudessa toteutusmalli palvelimettomalle. Tätä mallia käytetään yleisesti suurimpien palvelimettomien palvelujentarjoajien tuotteissa, kuten Amazon Web Services (AWS) Lambda, Microsoft Azure Functions ja Google Cloud Functions. (Red Hat, 2020b.) Nämä palvelut ovat myös yksinä suurimpina palveluntarjoajina tämän tutkielman tarkastelussa, joten FaaS on olennainen käsite tutkielman kannalta.

FaaS:n avulla voidaan kehittää tapahtumapohjaisia (eng. event-driven) sovelluksia, joissa luotuja funktioita käynnistetään ja suoritetaan tarpeen vaatiessa. Yleisesti FaaS pohjaisissa ratkaisuissa funktioiden tulee olla melko lyhyitä, jotta niitä voidaan käynnistää ja suorittaa riittävän nopeasti. (Red Hat, 2020b.) Perusideana on siis serveripuolen logiikkaa pyörittävien funktioiden luonti niitä kutsuttaessa ja tarpeen vaatiessa. Tämä auttaa erityisesti ylläpidettävyyttä, sekä skaalautuvuuden ja kustannusten hallintaa.

3.3 Katsaus palveluntarjoajiin

Palvelimettoman arkkitehtuurin palveluja tarjoavat useat yritykset. Näistä palveluista ehkä tunnetuimpia ovat Amazon AWS, Microsoft Azure ja Google Cloud. Muitakin palveluntarjoajia löytyy, kuten IBM, Oracle tai CloudFlare. Kaikkia näistä ei tämän tutkielman kontekstissa pystytä kuitenkaan käsittelemään. Tämän kappaleen tarkoituksena on tutustua siihen, miten eri tunnetuimmat palveluntarjoajat eroavat pääpiirteissään toisistaan.

Eri palvelut tukevat eri mahdollisuuksia kehittää palvelimettomia palveluita. Erityisesti tuetut ohjelmointikielet nousevat esille, koska ne rajoittavat osittain sitä, miten ja millaisia järjestelmiä voidaan näiden varaan kehittää. AWS Lambda tukee useita ohjelmointikieliä ja ajoympäristöjä, kuten Java, Go, PowerShell, Node.js (javascript), C#, Python, ja Ruby (Amazon, 2022). Microsoft tukee omassa Azure Functions palvelussaan ohjelmointikieliä hyvin samankaltaisesti muutamia poikkeuksia lukuun ottamatta, kuten tuki F#-kielelle (Microsoft, 2022). Googlen palvelun tuki on vähäisin, mutta viime aikoina noussut lähelle Amazonin ja Microsoftin vastaavaa. Tuettuna löytyy suosituimmat vaihtoehdot, eli Node.js, Python, Java ja .NET-kielet (C#, F#, VB). Palvelusta puuttuu kuitenkin Amazonin ja Azuren kaltaiset kustomoidut ajonaikaiset järjestelmät, joiden avulla voidaan käyttää tarvittaessa myös virallisesti tukemattomia ohjelmointikieliä ja ajoympäristöjä (Google, 2022).

Muita eroja palveluntarjoajien välillä voivat olla asiat, kuten tallennusratkaisut. Tallennusratkaisut ovat tärkeitä palvelimettomalle, koska ne ovat FaaS luonteensa takia yleisesti tilattomia (eng. stateless), jolloin pysyvä tieto täytyy tallentaa ja hakea tietokannasta tai muusta tallennusratkaisusta. Jokaisella kolmikosta Amazon, Microsoft ja Google on omanlaisensa ratkaisut tiedontallennuksen suhteen palvelimettomille alustoillensa. Esimerkiksi Amazon tarjoaa perustason oliotallennukseen S3-tallennusratkaisua ja tietokantapuolella käytössä on NoSQL tyyppinen DynamoDB, Microsoftilla vastaavat ratkaisut ovat Azure storage ja Azure Cosmos DB, ja Googlella Cloud Storage sekä Cloud Datastore. (Kumar, 2019; Eismann ja muut, 2021.)

Vastaavasti jokaisella palveluntarjoajalla on omat ratkaisunsa API:en luomiseen ja hallintaan, sekä lokitukseen, analytiikkaan ja monitorointiin. Erityisesti monitorointi ja

lokitus on usein hankalaa palvelimettoman tapauksessa, joten palveluntarjoajien tarjoamat palvelut tähän liittyen ovat tärkeässä roolissa kehittäjien kannalta. Esimerkiksi AWS:n tuote tähän tarkoitukseen on pääasiallisesti CloudWatch ja vastaavasti Microsoft Azuressa Application Insights (Kumar, 2019.)

Wangin, Lin Zhangin, Ristenpartin ja Swiftin (2018) tutkimuksessa vertailtiin jo aiemmin mainittuja kolmea palvelua Amazonilta, Microsoftilta ja Googlelta. Tutkimuksessa saatiin osviittaa siitä, miten ne suoriutuvat ja millainen niiden arkkitehtuuri on sisäisesti palvelimettoman osalta. Tuloksina AWS Lambda tarjosi parasta skaalautuvuutta ja kylmäkäynnistyslatenssia, jolla tarkoitetaan, kuinka nopeasti palvelu pystyi käynnistämään uuden instanssin tietystä funktiosta. Tämä johtunee osittain siitä, että AWS Lambda tukee funktioiden ”esilämmitystä” (eng. prewarming) (Eismann ja muut, 2021). Vastaavasti McGrathin ja Brennerin (2017) tutkimuksissa saatiin selville, että AWS Lambda näyttää skaalautuvan lineaarisesti ja parhaiten tästä kolmikosta.

Amazonin AWS tuoteperheen Lambda onkin yleisesti käytetyin alusta palvelimettomien sovellusten kehitykseen (Eismann ja muut, 2020). Tämä on johtanut myös siihen, että monissa palvelimettomaan liittyvissä tutkimuksissa AWS Lambda on yliedustettuna ja muihin alustoihin keskitytään vähemmän. Tämä on todennäköisesti suoraa seurausta siitä, että se oli ensimmäisenä markkinoilla. (Lynn., Rosati, Lejeune & Emeakaroha, 2017.)

3.4 Palvelimettoman etuja

Yleisesti suurimmat hyödyt, joita palvelimettomalla saavutetaan, koostuvat helpommasta kehitettävyydestä tietyin varauksin, pienemmistä ylläpito kustannuksista sekä helpommasta skaalautuvuudesta. Nämä kaikki ovat ongelmia, joihin voidaan törmätä perinteisissä palvelinratkaisuissa. Tässä kappaleessa käydään tarkemmin läpi aineistoa tähän liittyen.

Roberts & Chapin (2017) huomauttavat että nimenomaan vapaus keskittyä itse järjestelmän logiikan toimintaan, sen sijaan että resursseja kuluu serverin toiminnan ylläpitämiseen, on yksi houkuttelevimmista palvelimettoman eduista. Heidän mukaansa DevOps työn määrä vähenee samalla, joka osaltaan mahdollistaa palkkakustannusten laskun, sekä helpottaa riskienhallintaa.

Kun järjestelmän koodi on FaaS:n myötä jaettuna pienempiin itsenäisiin osiin, on sen ylläpitäminen helpompaa. Samalla tämä voi parantaa järjestelmien ja sovellusten kehityksen helppoutta ja nopeutta jo olemassa oleville järjestelmille, varsinkin pienillä ohjelmistokehitystiimeillä. (Roberts & Chapin, 2017). Monoliittisen palvelimen tapauksessa yleensä koko monoliitti tulee päivittää muutoksia tehdessä, siinä missä palvelimettomassa voidaan päivittää pienempiä osia kerralla (Fowler & Lewis, 2014; Roberts & Chapin, 2017.)

Palvelimettomat palvelut toimivat yleisesti sillä periaatteella, että käyttäjiä laskutetaan käytön mukaan (eng. pay-as-you-go) (Castro ja muut, 2019). Siinä missä aiemmat palvelinratkaisut ovat käyttäneet esimerkiksi pilvipalvelussa toimivia virtuaalikoneita, jotka vaativat käyttöönsä sille varatun laitteiston, palvelimeton vaadi tällaista ainakaan suoraan. Palvelimettomassa funktioita ajetaan FaaS-mallin mukaisesti vain tarvittaessa, jolloin laskutus perustuu enemmänkin käytettyyn muistiin ja käskyjen määrään. (Varghese ja Buyya 2017.) Tarkemmin ilmaistuna suurimmat palveluntarjoajat laskuttavat todellisuudessa siis funktioiden käynnistysten ja suoritusaikojen mukaan (eng. Pay-as-you-execute) (Stafford, Toosi & Mjeda, 2021). Tällöin palvelimettomalla

arkkitehtuurilla rakennettu palvelin ei käytä käytännössä mitään resursseja, kun sitä ei kutsuta. Turhaa palvelimen lepoajasta maksamista ei siis varsinaisesti tapahdu palvelimettoman tapauksessa, josta seurauksena voi olla huomattaviakin säästöjä.

Villamizar ja kumppanit (2017) selvittivät tutkimuksessaan palvelimettoman järjestelmän taloudellisia vaikutuksia ja vertasivat sitä perinteisiin monoliittisiin järjestelmiin sekä mikropalveluilla toteutettuun järjestelmään. Tutkimuksen perusteella yritykset voivat säästää palvelimettomalla ratkaisulla jopa 77,08 prosenttia IT-infrastruktuurin kustannuksissa. (Villamizar ja muut, 2017.) Golin ja muiden (2020) tapaustutkimuksessa saatiin pitkälti vastaavanlaisia tuloksia. Vertailukohtana palvelimettomalle järjestelmälle tutkimuksessa käytettiin kahta hyvin eritehoista virtuaalikonetta, jossa monoliittista järjestelmää pyöritettiin. Palvelimeton järjestelmä suoriutui ajetuista tehtävistä huomattavasti nopeammin ja erityisesti tehokkaampaa sekä kalliimpaa monoliittista järjestelmää halvemmin. Jos perinteisten palvelimien toimeton aika otetaan huomioon, oli palvelimeton ratkaisu molempia monoliittisia ratkaisuja halvempi (Gol ja muut, 2020.)

Gol ja muut (2020) osoittivat tutkimuksessaan hyvin myös sen, kuinka paljon paremmin skaalautuvuutta voidaan hallita palvelimettomilla sovelluksilla. Perinteinen monoliittinen palvelin ei juurikaan skaalautunut, vaikka sen prosessointikykyä nostettiin neljästä virtuaaliytimestä 96-virtuaaliytimeen ja sen muistimäärä yli 20-kertaistettiin 15 gigatavusta 360 gigatavuun. Suoritusaika samalle tehtävälle laski tällöin 6,27 tunnista 5,07 tuntiin, palvelimettoman ratkaisun selviytyessä tehtävästä reilussa neljässä minuutissa. (Gol ja muut, 2020). Palvelimettomat palvelut eivät tosin skaalaudu loputtomiin. Esimerkiksi AWS Lambda rajoittaa rinnakkaisia funktioiden suorituksia oletuksena tiettyyn raja-arvoon lieventääkseen mahdollisten palvelunestohyökkäysten vaikutuksia. Tätä rajoitusta on kuitenkin mahdollista nostaa ottamalla yhteyttä palveluntarjoajaan. (Roberts & Chapin, 2017.)

3.5 Palvelimettoman heikkouksia

Palvelimettomat toteutukset ovat käytännössä aina tilattomia (eng. stateless). Vaikka tilattomuus on skaalautuvuuden kannalta olennaista, tämä voi luoda ongelmia kehitysvaiheessa tai jopa estää sen käytön tietynlaisissa käyttökohteissa. Koska funktiot joutuvat kommunikoimaan tilallisten komponenttien kanssa tallentaakseen tietoa, aiheutuu tästä esimerkiksi latenssia kyseisiä funktioita suoritettaessa. Palveluntarjoajien tallennusratkaisujen erotessa toisistaan merkittävästi, tarvitaan tilan hallintaan myös joka alustalle eriävä ratkaisu, joka estää palveluntarjoajan nopean vaihtamisen toiseen tai sekaratkaisujen käytön (Roberts & Chapin, 2017.)

Villamizar ja kumppanit (2017) mainitsevat tapaustutkimuksessaan, että AWS Lambdan avulla kehitetty palvelimeton ratkaisu oli työläs toteutettava eritysesti projektin alkuvaiheessa. Tämä johtui siitä, että jokainen FaaS:n mukainen funktio täytyy kehittää, konfiguroida ja julkaista erikseen tuotantoon AWS alustalle. Lin ja Khazaei (2021) mainitsevat, että palvelimettoman järjestelmän rakennus voi olla monesti kompleksinen prosessi, johtuen suuresta määrästä tarvittavia funktioita ja konfigurointioptioita. Myös Yan, Castro, Cheng ja Ishakian (2016) mainitsevat, että erityisesti toisistaan riippuvaisten funktioiden luominen ja liittäminen toisiinsa järkevästi voi olla haastavaa.

Johtuen palvelimettoman arkkitehtuurin suhteellisen nuoresta iästä, on sille olemassa myös hyvin vähän valmiita suunnittelumalleja ja standardeja. Käytännössä tämä osa-alue on vielä täysin tutkimuksen alla. Joitakin suunnittelumalleja on jo alkanut ilmaantua, kuten Hong, Srivastava, Shambrook ja Dumitras (2018) julkaisussaan esittämät kuusi

erilaista tietoturvaan keskittyvää mallia. Käytännössä suunnittelumallien ja standardien puuttuminen voi vaikeuttaa kehitystä, kuten aiemmassakin kappaleessa mainittiin.

Useat kehitystyökalut ovat monesti palvelimettoman tapauksessa alustariippuvaisia. Esimerkiksi monitorointi ja lokitus hoidetaan pääasiassa pilvipalvelun omien alustojen palveluiden kautta, joka voi vaikeuttaa virheiden löytämistä (Yan ja muut, 2016.) Kehitystä vaikeuttaa myös se, että palvelimeton vaatii testauspuolelle omat ympäristönsä, siinä missä perinteisellä palvelimella riittää monesti se, että kehittäjällä on lokaalit versiot esimerkiksi tietokannasta. Palvelimetonta järjestelmää kehitettäessä ollaan yleensä palveluntarjoajan tarjoamien työkalujen varassa monessa muusakin asiassa, kuten analytiikkatyökaluissa. (Roberts & Chapin, 2017.)

Eismannin ja muiden (2021) tutkimuksen tuloksissa selvisi, että palvelimeton ei sovellu juurikaan pitkään käynnissä oleviin tehtäviin tai tehtäviin, joissa käytetään isoja datavolyymejä. Myös Varghese ja Buyya (2017) sanovat ettei palvelimeton ole kovinkaan sopiva tehtäviin, joissa suoritetaan pitkäaikaisia ja paljon dataa hyödyntäviä tehtäviä. Palveluntarjoajilla on yleisesti käytössä raja-arvot sille, kuinka kauan yhtä funktiota voidaan suorittaa maksimissaan yhtäjaksoisesti. Raja-arvon ylittyessä tapahtuu aikakatkaisu. AWS:n tapauksessa tämä raja on nykyisin 15 minuuttia (Amazon, 2018).

Yksi mahdollinen ongelma palvelimettomassa liittyy funktioiden kylmäkäynnistykseen. Tällä tarkoitetaan tilannetta, jossa tiettyä funktiota ei ole kutsuttu vähään aikaan, jolloin se puretaan palvelimien muistista ja sen uudelleenkäynnistämisestä aiheutuu huomattava viive ajettaessa funktiota ensimmäistä kertaa uudelleen. Tätä viivettä voidaan yrittää eliminoida tai sen vaikutusta pienentää erilaisilla tekniikoilla. Tekniikoiden toteutustapa riippuu alustasta ja alustan tarjoamista konfigurointimahdollisuuksista. (Manner, Endreß, Heckel & Wirtz, 2018; Roberts & Chapin, 2017.) Yleisesti tähän on kuitenkin käytetty niin sanottua ping-metodia, jossa palvelua yksinkertaisesti kutsutaan tietyin väliajoin sen pitämiseksi käynnissä. Tämä kuitenkin vähentää palvelimettomasta saatuja hyötyjä, koska resursseja hukataan pitämällä funktioita turhaan käynnissä. Muita tutkimuksissa havaittuja mahdollisia keinoja vaikuttaa on valita ohjelmistoa toteutettaessa käännettävän kielen sijaan tulkattava kieli, joka vähentää funktion suorittamisen yleiskustannuksia sitä uudelleen käynnistettäessä ja näin ollen nopeuttaa sen suorittamisen aloittamista (Manner ja muut, 2018.)

Eivy (2017) puolestaan varoittaa palvelimettoman taloudellisista seikoista toteamalla, että se ei tule kaikissa käyttökohteissa halvemmaksi kuin perinteiset palvelinratkaisut. Lyhyessä tutkimuksessa selvisi, että jos palvelimettomalla toteutettua sovellusratkaisua kuormitetaan erittäin raskaasti, voi sen käyttökustannukset nousta suuremmaksi kuin perinteisellä monoliittisella palvelinratkaisulla.

3.6 Migraatio monoliittisesta palvelimettomaan

Migraatio monoliittisesta palvelinrakenteesta palvelimettomaan voi ymmärrettävästi olla työmäärällisesti iso tehtävä, sillä FaaS vaatii omanlaisensa toteutuksen. Lisäksi Stafford ja muut (2021) toteavat tutkimuksessaan, että palvelimettomalle ei ole olemassa vielä vastaavia ja yhtä kypsiä migraatiotekniikoita, kuin vaikkapa mikropalveluarkkitehtuurin siirtymiselle on.

Yksi sovellusten migraatiosta perinteisestä monoliittisesta palvelinmallista palvelimettomaan on Golin ja muiden (2020) tapaustutkimus. Tutkimuksessa muutettiin olemassa oleva CPU- (central processing unit) ja dataintensiivinen sovellus

palvelimettomaan arkkitehtuuriin, jonka jälkeen näitä kahta toteutusta vertailtiin keskenään. Migraatiossa tuli ottaa huomioon asioita, kuten koodin muuttaminen tukemaan skaalautumista palvelimettomassa ympäristössä. Koodi tuli siis jakaa palasiin ja hyödyntää tilattomuutta. Kustannuksiltaan ja tehokkuudeltaan uusi palvelimeton toteutus päihitti monoliittisen toteutuksen selkeästi. Suurin vaikutus tässä oli sillä, että palvelimeton toteutus pystyi skaalautumaan lähes lineaarisesti suorituskyvyn osalta, siinä missä perinteinen toteutus ei juurikaan parantunut, vaikka sen prosessointitehoa moninkertaistettiin. (Gol ja muut, 2020.)

Staffordin ja muiden (2021) tekemässä tutkimuksessa pilkottiin (eng. Decomposition) monoliittista palvelinmallia käyttävä ohjelmisto palvelimettoman FaaS-malliin mukautuviksi funktioiksi. Tutkimuksen toteutuksessa käytettiin palveluntarjoajana Microsoft Azurea. Monoliittisesta ohejlmistosta pilkotut funktiot ajettiin ensin ilman tutkimuksessa aiemmin testattuja ja ilmi tulleita optimointeja vertailukohteen saamiseksi. Tämän jälkeen niihin hyödynnettiin aiempina löydöksinä identifioituja yleisiä toimintamalleja, joita on hyvä toteuttaa migraatiota tehdessä. Ensimmäinen näistä on, että jokaisen FaaS-funktion tulisi toteuttaa yhden vastuun periaatetta (eng. Single responsibility pattern). Toisena huomiona riippuvuusinjektio (eng. Dependency injection) kannattaa pyrkiä minimoimaan. Näiden ja muiden spesifisimpien optimointien jälkeen funktioiden suorittamisesta aiheutuneet kustannukset laskivat 40 prosentilla. Tämä johtui siitä, että funktioiden suoritusaikaa ja muistinkäyttöä saatiin laskettua, jotka molemmat ovat pääasiallinen kustannusten laskemiseen käytettävä metriikka suurimpien palveluntarjoajien kesken. (Stafford ja muut, 2021). Migraatiossa ei siis välttämättä riitä se, että funktiot pilkotaan suoraan olemassa olevasta monoliittisesta järjestelmästä palvelimettomaan ja FaaS mallin mukaiseksi. Ohjelmiston koodia voidaan joutua muokkaamaan ja optimoimaan varta vasten sopivammaksi FaaS ympäristöön, jotta siitä saadaan täysi hyöty irti esimerkiksi kustannusten osalta.

4. Pohdinta

Tämän tutkielman tarkoituksena oli löytää syitä, joiden perusteella palvelimeton toteutus on kannattavaa tai ei. Kirjallisuus aiheesta on vielä hieman rajallista, koska kyseessä on niin tuore aihealue, joka myös kehittyy erittäin nopealla tahdilla. Yleisesti kirjallisuuskatsauksella löydetyt tutkimukset käsittelivätkin aihetta melko suppeasti. Monet tällä hetkellä julkaistuista tutkimuksista käsittelevät nimenomaan enemmänkin kustannuksiin ja suorituskykyyn liittyvää puolta.

Palvelimeton voi yleisesti helpottaa kehitystä vähentämällä tarvetta huolehtia IT-infrastruktuurista ja DevOpsin tarvetta (Roberts & Chapin, 2017.). Sitä ei kuitenkaan tulisi valita automaattisesti. Erityisesti pienille projekteille, sekä monien projektien alkuvaiheessa, perinteisempi monoliittinen ratkaisu voi olla luontevampi vaihtoehto (Villamizar ja muut, 2017.) Useat tutkimukset ja mielipiteet puoltavat myös sitä, että palvelimettoman järjestelmän kehitys voi olla projektin alussa haastavaa (Lin & Khazaei 2021; Roberts & Chapin, 2017; Yan ja muut, 2016). Palvelimettomasta näyttääkin olevan hyötyä erityisesti suuremmille projekteille pienempien sijaan. Tämä osaltaan tukee myös sitä kuvaa, että palvelimeton on luotu alun perin lieventämään ongelmia, joihin törmätään perinteisissä monoliitissa järjestelmissä, kuten skaalautuvuuden hallinta.

Palveluntarjoajien työkalut ja palvelimettomaan liittyvät muut palvelut, kuten tiedontallennus ovat hyvin yksilöllisiä ratkaisuja. Vastaavasti palvelimettomien sovellusten monitorointi tapahtuu pääosin palveluntarjoajan oman alustan työkaluilla, joten niiden laadulla on merkittävä vaikutus kehittäjien työskentelyyn. (Kumar, 2019; Eismann ja muut, 2021.) Tämä tarkoittaa sitä, että alustan valinnassa tulee noudattaa harkintaa ja vaihtaminen toisesta palveluntarjoajasta toiseen ei ole suoraviivainen prosessi. Yhdenlainen standarditтomuus on havaittavissa ja tästä kirjoittaa myös Hong ja muut (2018) tutkimuksessaan, jossa esitettiin muutamia uusia palvelimettoman suunnittelumalleja.

Skaalautuvuus palvelimettoman osalta on merkittävä etu verrattuna perinteisiin monoliittisiin palvelimiin. FaaS-mallin mukainen sovellus pystyy teoriassa skaalautumaan loputtomasti, vaikkakaan käytännössä näin ei ole (Roberts & Chapin, 2017). Golin ja muiden (2020) tapaustutkimus oli hyvä esimerkki siitä, kuinka paljon tehokkaammin palvelimeton sovellus voi suoriutua verrattuna perinteiseen monoliittiseen järjestelmään.

Katsauksessa selvisi myös, että palvelimettomalla voidaan saavuttaa merkittäviä hyötyjä kustannustehokkuudessa, koska palvelimettomassa käyttäjät maksavat funktioiden käyttöajasta ja muistinkäytöstä, eikä turhaan palvelimien joutoajasta (Castro ja muut, 2019). Staffordin ja muiden (2021) tutkimuksessa löydettiin myös toimintamalleja, joilla migraatiotilanteessa voidaan FaaS-funktioita optimoida edelleen suurempien kustannussäästöjen aikaan saamiseksi. Kustannustehokkuus ei ole kuitenkaan täysin yksiselitteistä ja Eivy (2017) varoittaa myös osaltaan siitä, että palvelimeton ei ole välttämättä halvempaa kuin muut vaihtoehdot. Erityisesti raskaiden tehtävien kohdalla, joita kutsutaan lähes jatkuvasti, palvelimeton voi tulla jopa kalliimmaksi. Palvelimetonta harkitessa tuleekin perehtyä siihen, millaista järjestelmän käyttö on ja voidaanko se jakaa tarpeeksi pieniin suoritettaviin FaaS-mallin mukaisiin funktioihin, jotta palvelimettomasta saadaan sen hyöty irti kustannustehokkuuden osalta.

Eismannin ja muiden (2021) yksi huomio oli, että palvelimettomien soveltuvuus on parhain lyhyisiin, usein suoritettaviin tehtäviin. Samalla linjalla ovat Varghese ja Buyya

(2017). Jos tehtävää ei voida pilkkoa pienempiin osiin FaaS-malliin sopeutuvasti, ei palvelimeton ole tällöin siis sopiva ratkaisu. Vastaavasti suuret datamäärät nostavat funktioiden suoritettavaa aikaa ja tätä myöten kustannuksia. Paljon dataa vaativiin tai pitkään suoritettaviin tehtäviin vaaditaankin osittain kehitystä palvelimettomien palvelujen saralla. Mikäli palvelimetonta päädytään kyseisessä tilanteessa käyttämään, tulee funktiot pilkkoa ja niiden suoritus optimoida niin hyvin, ettei funktioiden suoritusajat tai datamäärät kasva liian suuriksi.

Vanhan järjestelmän päivitys palvelimettomaan ei ole mahdotonta ja sillä voidaan saavuttaa säästöjä sekä parannuksia itse järjestelmään. Golin ja muiden (2020) tapaustutkimuksessa muutettiin olemassa oleva perinteinen palvelinsovellus palvelimettomaan ja havaittiin merkittävästi parempaa skaalautuvuutta sekä kustannussäästöjä.

Kuten Stafford ja muut (2021) huomauttivat, palvelimettomaan liittyy vielä paljon tuntemattomia. Standardeja ei juurikaan ole ja monoliittisen järjestelmän migraatiosta palvelimettomaan on vielä vähän tutkimuksia. Lisätutkimusten pohjalta voitaisiin kehittää uusia toimintamalleja suorituskyvyn ja tehokkuuden parantamiseksi kyseisissä tilanteissa.

5. Yhteenveto

Tutkielmassa käsiteltiin palvelimetonta arkkitehtuuria ja mahdollisia syitä sen valitsemiseen perinteisen monoliittisen arkkitehtuurin sijaan. Lopputuloksena löydettiin muutamia tärkeimpiä syitä miksi palvelimeton voi olla hyvä ratkaisu ja myös tilanteita, joihin sen käyttö ei sovellu.

Palvelimeton arkkitehtuuri voi helpottaa kehitystyötä erityisesti suuremmissa projekteissa, koska monoliittisten palvelinratkaisujen päivitettävyys ja ylläpidettävyys laskee projektin koon kasvaessa huuomattavasti (Roberts & Chapin, 2017). Sen avulla voidaan saavuttaa säästöjä yrityksen kustannuksissa ja parantaa järjestelmän suorituskykyä paremmalla skaalautuvuudella (Gol ja muut, 2020; Villamizar ja muut, 2017).

Palvelimeton ei kuitenkaan ole aina paras vaihtoehto ja voikin tulla tietyissä tapauksissa kalliimmaksi kuin perinteinen monoliittinen ratkaisu (Eivy, 2017). Se ei sovellu juurikaan paljon dataa käyttäviin tai pitkän prosessointiajan tehtäviin (Eismann ja muut, 2021; Varghese & Buyya, 2017). Palvelimettomaan arkkitehtuuriin perustuvan palvelun kehityksen on todettu olevan myös työläämpää etenkin uuden projektin aloitusvaiheessa (Lin & Khazaei 2021; Roberts & Chapin, 2017; Yan ja muut, 2016).

Pilvipalveluiden suosio on yleisesti kasvanut merkittävää vuosittaista vauhtia yritysmaailmassa (Synergy Research Group, 2021). Toisaalta Eismann ja kumppanit (2020) totesivat, että palvelimettoman tutkimukseen liittyy tällä hetkellä paljon harmaata kirjallisuutta. Aiheesta onkin saatavilla vain rajallisesti oikeita vertaisarvioituja tutkimuksia. Uutta tutkimusta kaivattaisiinkin kipeästi yritysmaailman kiinni saamiseksi. Erityisesti tutkimuksia kaivattaisiin muistakin näkökulmista kuin palvelimettoman taloudellisista vaikutuksista tai skaalautuvuudesta.

Tämän tutkielman tuloksia rajoittaa erityisesti se, että tutkimuksia aiheesta on vielä erityisen vähän. Aihe on suhteellisen uusi, joten se myös kehittyy jatkuvasti niin palveluntarjoajien kuin tutkimustenkin osalta. Muutaman vuoden takainenkin tutkimus tai kirjallisuus voi alkaa olla jo osittain vanhentunutta. Lähitulevaisuudessa voi olettaa olevan tiedossa yhä enemmän aiheeseen viittaavaa tutkimusta, sillä yritykset käyttävät yhtä enenevissä määrin pilvipalveluita ja sen markkinaosuus palvelinratkaisuista sekä siinä liikkuvat rahamäärät ovat kasvamassa (Synergy Research Group, 2021). Palvelimetonta tullaan todennäköisesti hyödyntämään jatkossa yhä enemmän sen positiivisten vaikutusten myötä skaalautuvuuteen ja kustannustehokkuuteen, jotka molemmat ovat tärkeitä seikkoja isommille yrityksille.

Tutkielmassa tarkasteltujen tutkimuksien menetelmät vaikuttivat olevan myös hyvin samankaltaisia, esimerkiksi suorituskykymittauksia tai tapaustutkimuksia. Kvalitatiivisia menetelmiä ei juurikaan havaittu. Tällaisia tutkimuksia voitaisiin jatkossa tehdä esimerkiksi siitä, miten kehittäjät kokevat palvelimettoman muuttaneen heidän työskentelyään tai millaisina he kokevat eri palveluntarjoajien tarjoamien työkalujen käytön päivittäisessä työskentelyssä. Muita mahdollisia tutkimuskohteita voisi olla olemassa olevien isompien järjestelmien muuntaminen palvelimettomaan ja laajempien ohjelmistoprojektien tutkiminen kvantitatiivisin menetelmin.

Lähteet

Amazon (2018). AWS Lambda enables functions that can run up to 15 minutes. Lainattu 17.1.2022, saatavilla: https://aws.amazon.com/about-aws/whats-new/2018/10/aws-lambda-supports-functions-that-can-run-up-to-15-minutes/

Amazon (2022). Lambda runtimes. Lainattu 5.1.2022, saatavilla: https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtimes.html

Castro, P., Ishakian, V., Muthusamy, V. & Slominski, A. (2019). The server is dead, long live the server: Rise of Serverless Computing. *Communications of the ACM,* 62(12), 44-54

Dragoni, N., Giallorenzo, S., Lluch Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R. et al. (2017). Microservices: yesterday, today, and tomorrow. In M. Mazzara & B. Meyer (Eds.), *Present and Ulterior Software Engineering* (pp. 195–216). Zurich: Springer.

Eivy, A. (2017). Be Wary of the Economics of "Serverless" Cloud Computing. *IEEE Cloud Computing, 4(2),* 6–12.

Eismann, S., Scheuner, J., van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N. et al. (2020). *A Review of Serverless Use Cases and their Characteristics.* Spec RG Cloud.

Eismann, S., Scheuner, J., van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N. et al. (2021). Serverless Applications: Why, When and How*? IEEE Software*, 38(1), 32–39

Fowler, M. & Lewis, J. (2014). Microservices. Lainattu 5.12.2020, saatavilla: https://martinfowler.com/articles/microservices.html

Goli, A., Hajihassani, O., Khazaei, H., Ardakanian, O., Rashidi M. & Dauphinee, T. (2020). Migrating from Monolithic to Serverless: A FinTech Case Study. *Proceedings of the 20th International Conference on Performance Engineering*, 20–25.

Google (2022). Writing Cloud Functions. Lainattu 5.1.2022, saatavilla: https://cloud.google.com/functions/docs/writing

Hong, S., Srivastava, A., Shambrook, W. & Dumitras, T. (2018). Go Serverless: Securing Cloud via Serverless Design Patterns. *Proceedings of the 10th USENIX*

Workshop on Hot Topics in Cloud Computing. Saatavilla: https://www.usenix.org/system/files/conference/hotcloud18/hotcloud18-paper-hong.pdf

Jamshidi P., Pahl C., Mendonça N. C., Lewis J. & Tilkov S. (2018). Microservices: The journey so far and challenges ahead. *IEEE Software,* 35(3), 24–35

Kumar, M. (2019). Serverless Architectures Review, Future Trend and the Solutions to Open Problems. *American Journal of Software Engineering*, 6(1), 1–10

Kratzke, N. (2018). A Brief History of Cloud Application Architectures. *Applied Sciences (Switzerland), 8(8)*

Lin, C. & Khazaei, H. (2021). Modeling and Optimization of Performance and Cost of Serverless Applications. *IEEE Transactions on Parallel and Distributed Systems, 32(3),* 615–632.

Lynn, T., Rosati, P., Lejeune, A. & Emeakaroha, V. (2017). A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms. *Proceedings of the 2017 IEEE 9th International Conference on Cloud Computing Technology and Science,* 162–169.

Manner, J., Endreß, M., Heckel, T. & Wirtz, G. (2018). Cold Start Influencing Factors in Function as a Service. *Proceedings of the 11th IEEE/ACM International Conference on Utility and Cloud Computing,* 181–188.

McGrath, G. & Brenner, P. (2017). Serverless Computing: Design, Implementation and Performance. *Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems Workshop*, 405–410.

Microsoft (2020). Supported languages in Azure Functions. Lainattu 5.1.2022, saatavilla: https://docs.microsoft.com/en-us/azure/azure-functions/supported-languages

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems.* Sebastopol: O'Reilly Media, Inc.

Red Hat (2020-a). What is serverless? Lainattu 25.11.2020, saatavilla: https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless

Red Hat (2020-b). What is Function-as-a-Service (FaaS)? Lainattu 25.11.2020, saatavilla: https://www.redhat.com/en/topics/cloud-native-apps/what-is-faas

Roberts, M. & Chapin, J. (2017). *What is Serverless?* Sebastopol: O'Reilly Media, Inc.

Stafford, A., Toosi, F. G. & Mjeda, A. (2021). Cost-Aware Migration to Functions-as-a-Service Architecture. *Proceedings of the 15th European Conference on Software Architecture, vol. 2978.* Saatavilla: http://ceur-ws.org/Vol-2978/saerocon-paper3.pdf

Synergy Research Group (2021). Amazon, Microsoft & Google Grab the Big Numbers – But Rest of Cloud Market Still Grows by 27 %. Lainattu 8.1.2022, saatavilla: https://www.srgresearch.com/articles/amazon-microsoft-google-grab-the-big-numbers-but-rest-of-cloud-market-still-grows-by-27

Varghese, B. & Buyya, R. (2017). Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems,79(3),* 849–861

Villamizar, M., Garces, O., Ochoa, L., Castro, H., Salamanca, L., Verano, 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, 11(6), 233–247*

Wang, L., Li, M., Zhang, Y., Ristenpart, T. & Swift M. (2018). Peeking Behind the Curtains of Serverless Platforms. *Proceedings of the 2018 USENIX Annual Technical Conference,* 133–145.

Yan, M., Castro, P., Cheng, P. & Ishakian, V. (2016). Building a Chatbot with Serverless Computing. *Proceedings of the 1st International Workshop on Mashups of Things and APIs,* 1–4.