Tampereen yliopisto

Akseli Kolari

PALVELITTOMAN LASKENNAN KÄYTTÖKOHTEET

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
Tarkastaja: Maarit Harsu
Toukokuu 2020

TIIVISTELMÄ

Akseli Kolari: Palvelittoman laskennan käyttökohteet
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Toukokuu 2020

Tämä työ esitellee palvelitonta laskentaa (engl. serverless computing) ja sen käyttökohteita. Edellä mainittujen lisäksi työssä esitellään kevyesti palvelitonta laskentaa tukevia palveluntarjoajia ja heidän palveluidensa ominaisuuksia. Työn tutkimuskysymyksenä on se, mihin palvelitonta laskentaa tulisi käyttää sekä missä tilanteissa jokin toinen ratkaisu voi olla parempi vaihtoehto. Työn aineisto on kerätty alan tietokirjallisuudesta, tutkimuksista, tieteellisistä artikkeleista sekä konferenssijulkaisuista. Palveluntarjoajia koskeva aineisto on haettu tarjoajien omista dokumentaatioista.

Palveliton laskenta on tapa toteuttaa pilvilaskentaa (engl. cloud computing) tarjolla olevia resursseja tehokkaasti hyödyntäen. Palvelittoman laskennan erityinen piirre on, että sovellusta tai sen osia ei pidetä ajoympäristössään koko ajan, vaan ohjelma tuodaan ajoon vain silloin kun sitä tarvitaan. Palvelitonta laskentaa hyödyntäen rakennetut sovellukset ovat siis tapahtumapohjaisia; ne reagoivat erilaisiin tapahtumiin.

Työssä huomattiin, että palveliton laskenta sopii varsin hyvin tapahtumapohjaisiin sovelluksiin, jotka ovat myös tilattomia. Tälläisia sovelluksia voivat esimerkiksi olla kuvia muokkaavat ohjelmat, jotka toimivat, kun palvelimelle lähetetään uusi kuva. Työtä tehdessä havaittiin myös, että palveliton soveltuu hyvin osaksi mikropalveluarkkitehtuuria, toteuttaen jonkin pienen osan isommasta kokonaisuudesta. Automaattisen skaalautumiskykynsä vuoksi palveliton on myös hyvä valinta tilanteisiin, joissa sovelluksen käyttäjämäärä vaihtelee paljon.

Käyttökohteita tutkittaessa selvisi myös tilanteita, joissa palvelittoman sijaan tulisi valita jokin muu ratkaisu. Palvelittoman laskennan sisältäessä enemmän viivettä, ei sitä tule käyttää tilanteissa, joissa mahdollisimman alhainen viive on erityisen tärkeää. Myös tilanteissa, joissa palvelun käyttömäärät ovat hyvin ennakoitavissa, voi jokin toinen ratkaisu tulla halvemmaksi ja helpommaksi. Ongelmaksi voi myös muodostua palveluntarjoajaan lukittautuminen, sillä palvelittomaan laskentaan tarjotut alustat eivät välttämättä ole yhtenäisiä ja voivat tehdä palveluntarjoajan vaihdosta haastavaa. Palveluntarjoajaa valittaessa onkin tehtävä tarkka taustoitus ja palveluntarjoajan tarjoamien ominaisuuksien kartoitus.

Avainsanat: mikropalvelut, palveliton laskenta, palveliton-arkkitehtuuri, web-tekniikat, FaaS, Function as a Service, esineiden internet, reunalaskenta

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

AR	AR eli Augmented Reality lisää virtuaalisia elementtejä maailmaan
AWS Lambda	Amazonin tarjoama pilvipalvelu, jossa voidaan ajaa FaaS-funktioita
FaaS	FaaS eli Function as a Service on nimitys palvelittomalle (serverless) arkkitehtuurille
FCC	FCC eli Federal Communications Commission on Yhdysvaltojen liittovaltion komissio, joka huolehtii tietoliikenteeseen liittyvistä asioista liittovaltion tasolla. Jakaa esimerkiksi taajuusalueita teleoperaattoreille
GB-sekuntti	GB-sekuntti on palvelittomassa laskennassa usein käytetty hinnan yksikkö. Sillä ilmaistaan sitä hintaa, mikä laskutetaan kun palvelitonta funktiota ajetaan yhden sekunnin ajan ajoympäristössä, jossa on käytettävissä 1 GB muistia
GCF	GCF eli Google Cloud Functions on Googlen tarjoama pilvipalvelu, jossa voidaan ajaa FaaS-funktioita
GHz-sekuntti	GHz-sekuntti on palvelittomassa laskennassa käytetty hinnan yksikkö. Sillä ilmaistaan sitä hintaa, joka laskutetaan kun palvelitonta funktiota ajetaan yhden sekunnin ajan ajoympäristössä, jossa on käytössä 1 GHz prosessoritaajuus
IBM Cloud Functions	IBM:n tarjoama pilvipalvelu, jossa voidaan ajaa FaaS-funktioita
IoT	IoT eli Internet of Things. Suomeksi esineiden internet
JWT	JWT eli JSON Web Token on todentamisessa käytetty tekniikka
Microsoft Azure	Microsoftin tarjoama pilvipalvelu, jossa voidaan ajaa FaaS-funktioita
VR	VR eli Virtual Reality tarkoittaa virtuaalista todellisuutta, joka voidaan toteuttaa esimerkiksi lasien avulla

1 JOHDANTO

Aikanaan yrityksillä oli ongelmana kalliiden palvelimien hankinta ja hintava ylläpito. Pelastajaksi ongelmaan keksittiin virtuaalitietokoneet, joita kyettiin vuokraamaan suurilta palveluntarjoajilta ympäri maailman. Ongelmaksi muodostui kuitenkin se, että virtuaalikoneille tuli hankkia kiinteä määrä CPU-ajoaikaa sekä muistia. Huonosti lasketut vaatimukset saattoivat alimitoitettuna aiheuttaa palvelun nykimisen tai jopa kaatumisen, kun taas liian yläkanttiin arvioidut tehovaatimukset aiheuttivat ylimääräisiä kuluja palvelinten maatessa tyhjänpanttina käyttäjien nukkuessa. Kyseiseen ongelmaan on kuitenkin keksitty ratkaisu, nimittäin palveliton laskenta (engl. serverless computing).

Tämä kandidaatintutkielma käsittelee palvelitonta laskentaa ja FaaS (engl. Function as a Service) -palveluita, joka hyödyntää palvelittoman laskennan perusajatuksia vahvasti. Työ esittelee lukijalleen mahdollisia käyttökohteita palvelittomalle laskennalle sekä samalla sen eroavaisuuksia muihin tapoihin tehdä sovelluksia. Työssä havainnollistetaan myös palvelittoman heikkouksia sekä vahvuuksia. Työn tutkimuskysymyksenä on se, mihin palvelitonta laskentaa voi käyttää sekä missä tilanteissa jokin toinen ratkaisu voi olla parempi vaihtoehto.

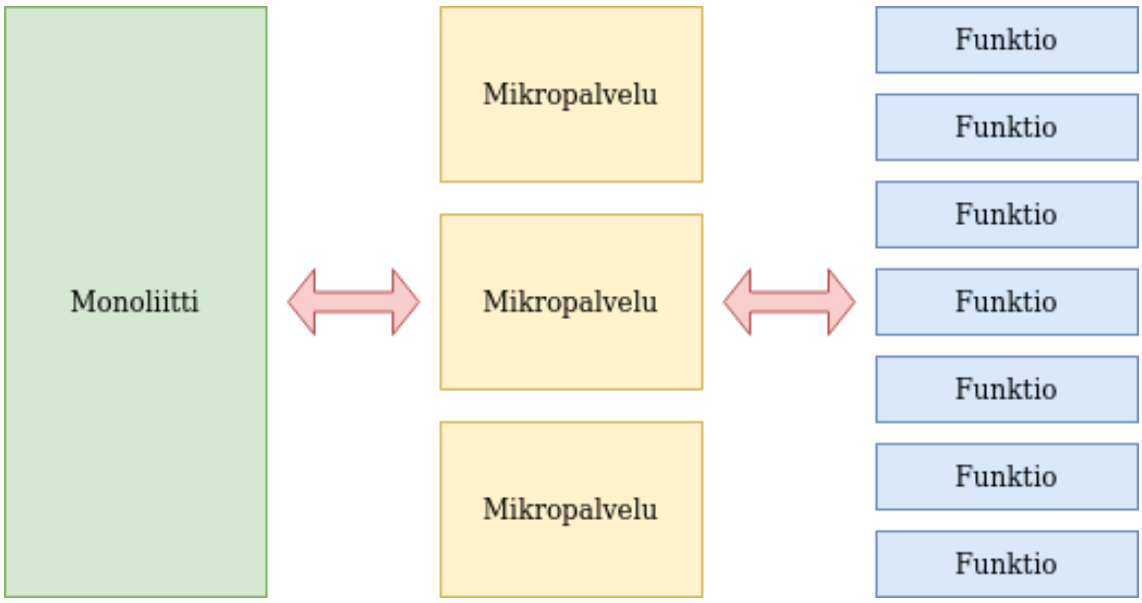
Luvussa 2 esitellään erilaisia sovellusarkkitehtuureja sekä myös palvelittoman laskennan perusteet. Luku 3 esittelee käyttökohteita palvelittomalle laskennalle ja esiin tuodaan palvelittoman laskennan vahvuuksia sekä heikkouksia. Luku 4 esittelee palvelitonta laskentaa tukevia palveluntarjoajia. Luvussa 5 käsitellään työtä tehdessä tehtyjä havaintoja ja luvussa 6 suoritetaan työn yhteenveto.

2 SOVELLUSARKKITEHTUURIT

Tämä luku esittelee tarkemmin erilaisia arkkitehtuureja, joiden avulla sovelluksia voidaan kehittää. Erityisesti luvussa tutkitaan monoliittia, mikropalveluarkkitehtuuria sekä palvelitonta laskentaa omana kokonaisuutenaan. Palvelitonta laskentaa käsitellään myös liitännäisenä mikropalveluarkkitehtuuriin.

2.1 Monoliitti

Monoliitti on palvelu, jonka yksittäisiä komponentteja ei voida ajaa itsenäisesti. Kaikki komponentit myös jakavat palvelimen samat resurssit keskenään. [1] Mikäli sama monoliittinen palvelu toteuttaa kaksi näennäisesti erillistä palvelua, ei toisen palvelun resursseja voida ruuhkatilanteessa nostaa samalla nostamatta mahdollisesti vähemmällä rasituksella olevan palvelun tehoja. Tämä aiheuttaa resurssien tuhlausta ja turhia kustannuksia palvelimen vuokraajalle.



Kuva 2.1. *Monoliittinen palvelu, mikropalvelu sekä FaaS-palvelu.*

Kuvassa 2.1 havainnollistetaan eroa monoliitin, mikropalvelun sekä FaaS-palvelun välil-

lä. Monoliittia havainnollistetaan yhtenä isona kokonaisuutena, kun taas mikropalveluissa kokonainen toteutus on pilkottu jo kolmeen omaan palveluunsa. FaaS-palvelu on taas esitetty kasana funktioita, jotka yhdessä muodostavat monoliittia sekä mikropalvelua vastaavan kokonaisuuden. Jos jokaista kuvassa esitettyä suorakulmiota ajatellaan omana kokonaisuutenaan, voidaan nopeasti ymmärtää, miksi esimerkiksi teknologiavaihdokset ovat huomattavasti haastavampia monoliittisessa palvelussa kuin mikropalveluissa.

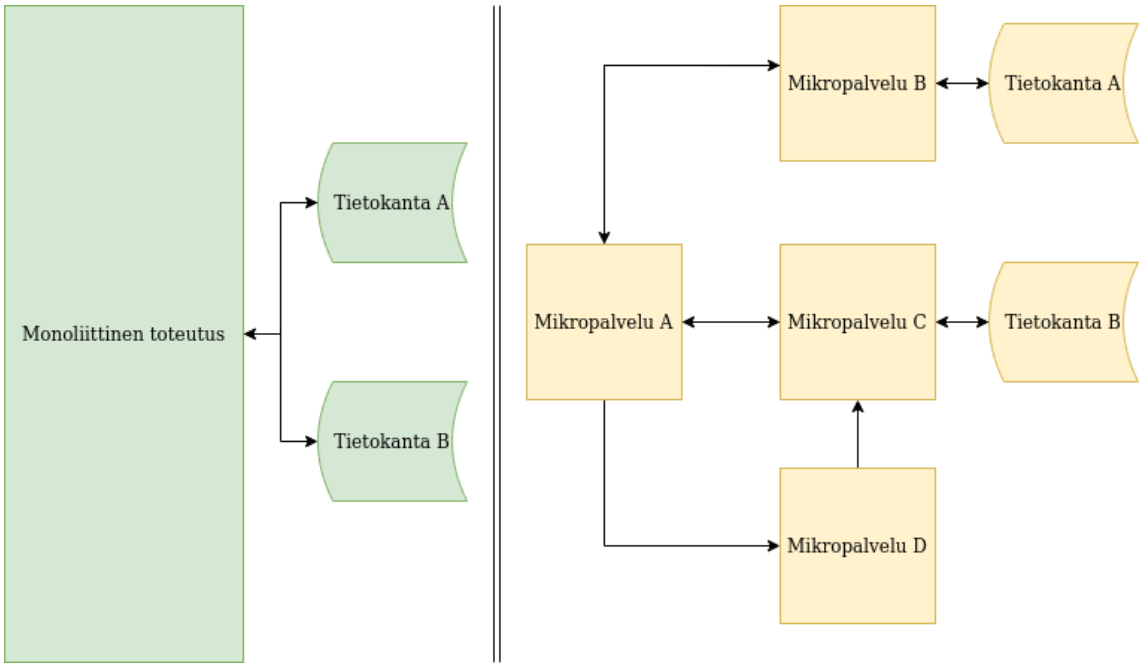
Isoissa monoliiteissa ongelmiksi muodostuvat myös ylläpito sekä jatkuvan kehittämisen haastavuus. Yhden komponentin päivitys vaatii koko palvelun alasajamisen ja uudelleen käynnistämisen, joka hidastaa kehitystä ja saattaa aiheuttaa käyttäjille häiriöaikaa (engl. downtime). Myös bugien metsästys voi olla haastavaa ohjelmistokoodin ollessa hyvin laaja. [1]

Monoliittien skaalaminen on myös hyvin haastavaa. Skaalautuminen joudutaan yleensä hoitamaan niin, että jos palvelulle tulee enemmän pyyntöjä, kuin se kykenee hallitsemaan, luodaan palvelun rinnalle kopio samasta palvelusta. Tämän jälkeen käyttäjien pyynnöt jaetaan tasaisesti kopioiden kesken parantaen vasteaikoja. Usein pyyntöjen kasvu kuitenkin keskittyy vain tiettyyn komponenttiin koko palvelusta, jolloin pelkästään tämän komponentin skaalaminen riittäisi käyttäjien palvelemiseen. Monoliittisessa palvelussa tämä ei kuitenkaan ole mahdollista, sillä komponentteja ei voida ajaa omana kokonaisuutenaan.

2.2 Mikropalveluarkkitehtuuri

Mikropalveluarkkitehtuuri liittyy olennaisesti palvelittomaan laskentaan ja erityisesti FaaS-palveluihin, sillä sitä käytetään usein osana isompaa kokonaisuutta. Näin ollen kokonaisia palveluita ei useinkaan toteuteta vain palvelitonta laskentaa hyödyntäen, vaan ne tukevat toisiaan. Usein tämä isompi kokonaisuus on toteutettu mikropalveluarkkitehtuuria hyödyntäen. [2, 3].

Mikropalveluarkkitehtuurissa jokin suurempi järjestelmä on koostettu pienemmistä palasista, niin sanotuista mikropalveluista (engl. microservice). Mikropalvelut kommunikoivat keskenään tarkasti määriteltyjen rajapintojen kautta, eikä niillä ole käytössään yhteistä välimuistia. Mikropalvelut pyrkivät toteuttamaan tarkasti määritellyn tehtävän palvelussaan, ja näin koodimäärä ei yhdessä palvelussa yleensä paisu liian suureksi. Kun palveluiden tehtävä on tarkasti määritelty, on palvelun hajotessa viallisen palvelun löytäminen usein nopeaa ja helppoa. [2, 4]



Kuva 2.2. *Monoliitti ja mikropalvelutoteutus.*

Kuvassa 2.2 on esitetty kuvitteellisen sovelluksen esimerkkitoteutus monoliittina sekä mikropalveluina. Esimerkiksi mikropalvelutoteutuksessa palvelun D päivittäminen uuteen on melkko helppoa, sillä sen tehtävä ja kommunikointi rajapintojen kautta palveluihin A ja C on määritelty tarkasti. Monoliitissa tämän saman asian toteuttavan palasen löytäminen voi olla huomattavasti hankalampaa ellei jopa mahdotonta.

Koska mikropalvelut eivät oikeastaan liity toisiinsa, voidaan niiden toteutuksissa käyttää jopa eri ohjelmointikieliä. Näin ollen jokainen osa suuremmasta kokonaisuudesta voidaan toteuttaa kaikista optimaalisimmalla tekniikalla, juuri haluttuun tehtävään. [4]. Kaikista parhaimman tekniikan käyttö tietysti tehostaa palvelun toimintaa, ja tämä näkyy käyttäjälle nopeampana ja responsiivisempana palveluna.

Teknologioiden vaihto voidaan myös tehdä helposti mikropalveluarkkitehtuurissa. Koska mikropalvelut ovat omia kokonaisuuksiaan, voidaan niitä myös korvata helposti uusilla. [4]. Kun kommunikointi on tarkasti määriteltyä tiettyjä rajapintoja pitkin, voidaan vanha palvelu helposti korvata uusia teknologioita käyttävällä mikropalvelulla, johon on vähintäänkin toteutettu korvattavan palvelun vastaava rajapinta.

Helpon korvattavuuden lisäksi mikropalveluarkkitehtuurin etuna on järkevästi tapahtuva skaalaus. Mikäli tietty osa palvelusta alkaa saamaan paljon enemmän pyyntöjä käyttäjiltä, voidaan vain tätä yksittäistä palvelua skaalata ylöspäin ja näin vastata kasvavaan kysyntään. Monoliittisessa palvelussa hukataan paljon resursseja yhden palvelun kysyntään

vastattessa, koska myös kaikki muut palvelut skaalautuvat mukana. [2, 4]

2.3 Palveliton laskenta

Palvelittoman laskennan perusidea on luoda mahdollisuus ajaa ohjelmakoodia vain silloin, kun sitä tarvitaan johonkin. Palvelitonta käyttävää ohjelmaa ei siis ajeta pilvipalvelun tarjoajan palvelimilla koko ajan, vaan resursseja voidaan vapauttaa muuhun käyttöön palvelun käytön ollessa olematonta.[2]. Tämä mahdollistaa palveluntarjoajalle olemassa olevien resurssien paremman hyödyntämisen ja tätä kautta taloudellisia säästöjä [5].

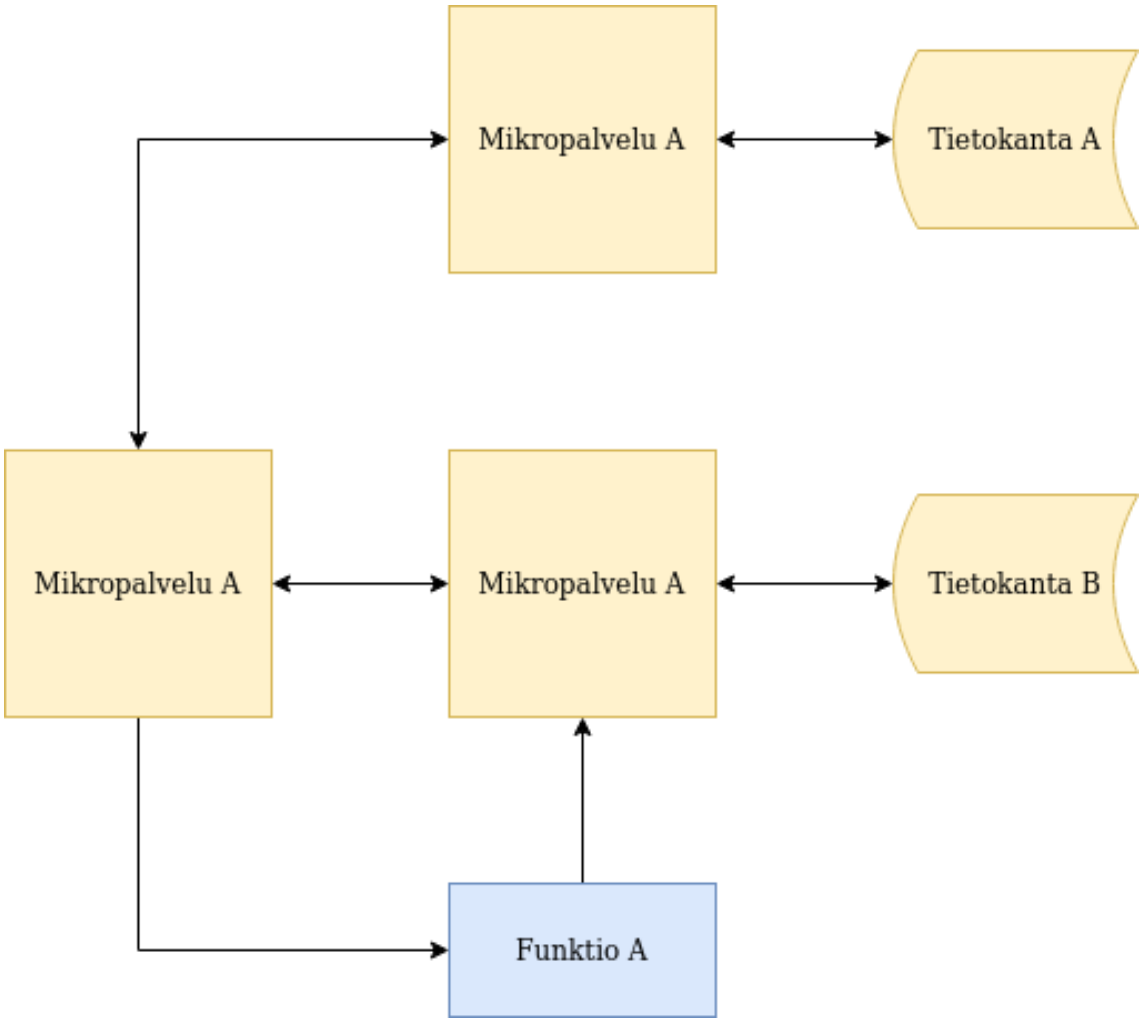
Palvelittomassa laskennassa sovellus on siis tapahtumapohjainen. Se reagoi käyttäjältä tai muualta saapuviin erilaisiin herätteisiin ajamalla esimerkiksi funktioita (FaaS). Kun tapahtuma on suoritettu, katoaa funktio pois ajoympäristöstä viemästä resursseja. [6]

Tyypillisesti palveliton laskenta tuo sovelluskehittäjälle helpomman tavan luoda ja toimittaa ohjelma käyttäjilleen. Ohjelman luoja ei joudu huolehtimaan ajoympäristön (engl. runtime environment) luomisesta tai hallinnasta, vaan tämä osa on ulkoistettu palveluntarjoajalle. Sovelluksen kehityksessä voidaankin siis keskittyä vain itse sovelluksen tekemiseen, mikä tehostaa ohjelmoijien ajankäyttöä [3]. Haittana on toki se, että ohjelmiston kehittäjä joutuu sitoutumaan palveluntarjoajan antamaan ympäristöön ja sen tarjoamiin paketteihin [5].

FaaS-funktiot ovat pieniä funktioita, jotka toteuttavat hyvin pienen osan jostain suuremmasta kokonaisuudesta. Funkioita ajetaan yleensä hyvin lyhyitä aikoja, ja funktiot ovat yleensä tilattomia (engl. stateless). [5] Tilattomuudella tarkoitetaan tässä tapauksessa sitä, että peräkkäin tai rinnakkain ajettavilla funktioilla ei ole käsitystä siitä, mitä aiemmin tai tällä hetkellä tapahtuu. On kuitenkin mahdollista toteuttaa myös tilallisia FaaS-funktioita, jotka ovat yhteydessä esimerkiksi tietokantoihin. [2]

Kuvassa 2.3 on toteutettu muutos kuvassa 2.2 esiteltyyn kuvitteelliseen sovellukseen, jossa mikropalvelu D on korvattu funktiolla A. Näin FaaS-funktio on integroitu osaksi isompaa kokonaisuutta. Funktion vastuulla voi esimerkiksi olla kuvien käsittely ja pakkaaminen sopivampiin kokoihin eri alustoille.

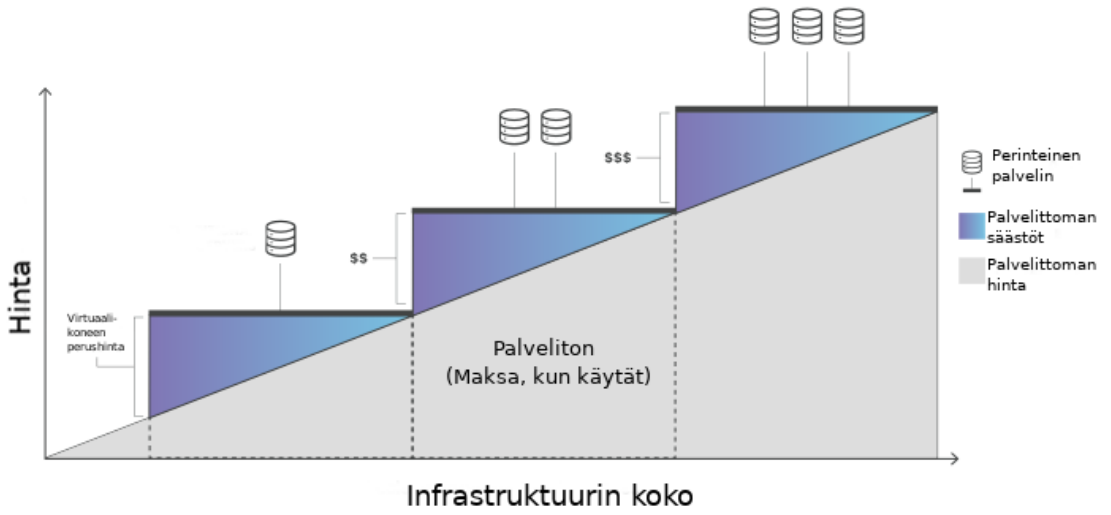
Palveluntarjoajille hyötyä tuo myös se, että palveliton arkkitehtuuri antaa heille mahdollisuuden tutkia ja ymmärtää asiakkaidensa tarpeita aivan uudella tavalla. Koneoppimisen avulla voi olla mahdollista ennustaa FaaS-funktioiden ajamista tutkimalla funktion tuottamia lokeja. Tämä voi mahdollistaa vieläkin paremman palvelinkäytön optimoinnin, kun



Kuva 2.3. *Mikropalveluarkkitehtuurin toteutettu FaaS-funktio.*

palveluntarjoaja osaa ennustaa asiakkaidensa tarpeita ja tuoda funktioita ajoon oikeaan aikaan. [7, s. 8]

Säästöjä palvelinkustannuksissa voi saada myös asiakas. Koska koodia ajetaan vain tarvittaessa, laskutetaan siitä myös vain silloin, kun sitä käytetään. Näin ollen asiakas ei maksa palvelimista turhaan silloin, kun niitä ei kukaan käytä. [2, 5]. Kuitenkin jossain tilanteissa asiakas saattaa joutua maksamaan odottelusta, esimerkiksi tilanteissa, joissa tietoliikenneyhteyden eivät toimi optimaalisesti ja paketteja joudutaan lähettämään uudestaan tai yhteys on kokonaan poikki [7, s. 8]. Tulee myös muistaa, etteivät säästöt ole automaattisia, vaan tietyissä tilanteissa halvemmaksi saattaa tulla toisenlainen arkkitehtuuri. Mikäli itse palvelu on suuri, ja käyttäjämäärät ovat helposti ennustettavia ja tasaisia, voi halvemmaksi tulla virtuaalikoneen vuokraus. Kuvassa 2.4 on pyritty esittelemään palvelittoman arkkitehtuurin hinnoittelua verrattuna perinteiseen virtuaalikoneen vuokraamiseen. [8]



Kuva 2.4. *Palvelittoman arkkitehtuurin tuottamia säästöjä esittelevä kuvaaja. Muokattu lähteestä [8].*

Palveluntarjoajan varmistaessa, että palvelimia on riittävästi, ei asiakkaan tarvitse myöskään huolehtia palvelunsa skaalautumisesta. Esimerkiksi lippukauppa, joka ei normaalisti saa kovinkaan paljon liikennettä sivustolleen, voi hyötyä huomattavasti palvelittomasta laskennasta. Jos lippukauppa saa myytäväkseen erittäin suositun artistin viimeisen keikan lippuja, voi liikenne sivuille moninkertaistua lippujen myynnin ajaksi. Palvelittomassa palveluntarjoaja alkaa välittömästi ruuhkan sattuessa ajamaan lippukaupan palvelufunktioita useita kappaleita rinnakkain, jolloin palvelu skaalautuu automaattisesti käyttäjämäärään. [5] Näin lippukauppa kykenee palvelemaan kaikkia asiakkaitaan kohtuullisella viiveellä.

Todellinen esimerkki palvelun kaatumisesta huonosta skaalautumisesta johtuen on vuonna 2017 tapahtunut Yhdysvaltojen liittovaltion Federal Communications Comission eli FCC:n nettisivujen kaatuminen massiivisen liikenteenlisäyksen vuoksi. FCC:n nettisivut kaatuivat, koska ne eivät kyenneet skaalatumaan tarpeeksi sivuston liikenteen moninkertaistuttua hetkellisesti. Kyseinen kaatuminen oltaisiin voitu mahdollisesti estää, mikäli FCC olisi käyttänyt palvelussaan palvelitonta arkkitehtuuria, joka olisi kyennyt nopeasti ja helposti skaalautumaan lisääntyneeseen liikenteeseen. [7, s. 7]

Palveliton arkkitehtuuri sisältää myös omat akilleen kantapäänsä. Koska koodia ei ajeta koko ajan, on sen käynnistymisessä pieni viive johtuen ohjelmakoodin tuomisesta ajoympäristöön. Täten palvelun vastaamisessa on hieman enemmän viivettä verrattuna perinteiseen koko ajan ajossa olevaan koodiin. Viive johtuu koodin varastoinnista massamuis-

tiin ja sen siirtämisestä sieltä ajoympäristöön. Tätä tapahtumaa kutsutaan kylmäkäynnistymiseksi (engl. cold start). [6, 9]

Kylmäkäynnityksen viivettä voidaan minimoida muutamalla kikalla. Ohjelmalle voidaan esimerkiksi allokoida hieman enemmän muistia, joka nopeuttaa käynnistymistä. Koodi voidaan myös asettaa säilymään välimuistiin ajonsa jälkeen hetkeksi, jolloin useat peräkkäiset kutsut vastaavat nopeammin kylmäkäynnistyksen tapahtuessa vain kerran. [6]

Ohjelmakoodin testaaminen ja bugien metsästäminen voi myös olla haastavaa. Yksikkötestaus on melko helppoa toteuttaa palvelittoman funktioluonteen vuoksi, mutta integraatiotestaaminen voi olla hyvinkin haastavaa, koska palveluntarjoaja huolehtii ajoympäristöstä. Vastaavan ajoympäristön luominen lokaalisti voi olla jopa mahdotonta, koska ajoympäristöt eivät aina ole avointa lähdekoodia, vaan tarkoin varjeltuja liikesalaisuuksia. [6]

Myös tietoturvan puolella riittää pohdittavaa. Eräs ongelma on esimerksi sen varmistaminen, että funktioita käyttävät vain sellaiset käyttäjät, joilla siihen on oikeus. Eräs mahdollisuus käyttäjän todentamiseen ovat tilattomat JWT-poletit (engl. JSON Web Tokens), mutta mikäli palveluun otetaan yhteys ilman salausta, voi potentiaalinen väärinkäyttäjä kopioida todentuneen käyttäjän poletit ja hyödyntää niitä itse. Ongelmaksi voi kuitenkin muodostua laitteet, joiden laiteiston tehot eivät yksinkertaisesti riitä salauksien tai polettien käsittelyyn. Tälläisiä laitteita voivat olla esimerkiksi esineiden internetissä olevat pienet mittalaitteet, joissa pieni energiankulutus on pakollista. [9]

Käyttäjien yksityisyyteen on myös kiinnitettävä huomiota. Vaikka hyökkääjä ei pääsisikään ajamaan järjestelmän erilaisia funktioita, voi hän hyödyntää konteksuaalista dataa johtopäätösten tekemiseen. Jos hyökkääjä onnistuu saamaan tietoonsa, minkä nimisiä funktioita ajetaan minäkin hetkenä, voi päätelmiä pystyä tekemään melkoisesti. Esimerkiksi jos käyttäjällä on älykoti, josta ensin ajetaan funktio "sammuta valot"ja heti perään "avaa portti", voidaan konteksista päätellä, että käyttäjä on mahdollisesti poistunut kotoaan. Tälläisen datan salaaminen on ehdottoman tärkeää, jotta käyttäjien yksityisyys säilyy. [9]

3 PALVELITTOMAN ARKKITEHTUURIN KÄYTTÖKOHTEITA

Tässä luvussa käydään läpi joitain mahdollisia käytännön käyttökohteita palvelittomalle laskennalle.

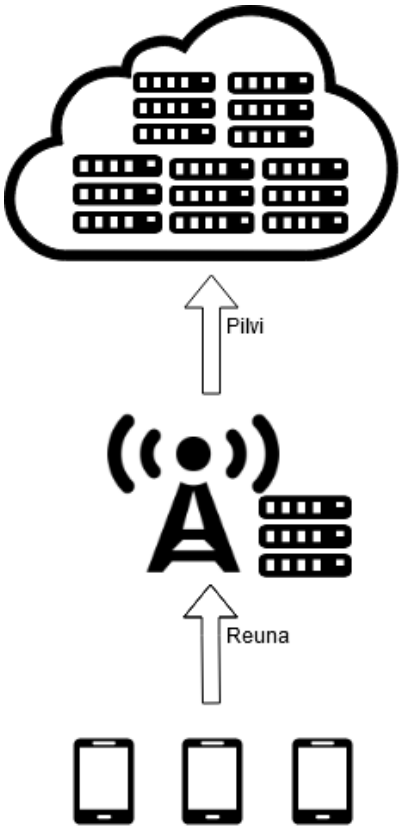
3.1 Reunalaskenta

Reunalaskenta (engl. edge computing) tarkoittaa laskentatehon ja tallennustilan siirtämistä internetin reunoille, lähemmäs loppukäyttäjää. Esimerkiksi mobiiliverkossa reunalaskija voi sijaita tukiaseman välittömässä läheisyydessä, palvellen yhteydessä olevia mobiilikäyttäjiä. Reunalaskenta pyrkii tehostamaan olemassaolevan verkkoinfrastruktuurin käyttöä vähentämällä datan kulkemista loppukäyttäjältä pilveen eli palvelinkeskukselle asti, estäen pullonkaulojen muodostumisen keskitetyissä palveluissa. [10, 11]

Kuvassa 3.1 esitellään reunalaskennan sijainti pilven ja käyttäjien välissä. Reunalaskenta sijaitsee fyysisesti mahdollisimman lähellä käyttäjiä. Reunalaskennalla on vähemmän resursseja käytössään kuin itse pilvellä, ja siksi niiden käytössä tulee olla tarkka.

Reunalaskentaa hyödyntämällä loppukäyttäjille voidaan siis tarjota entistä responsiivisempi palvelu pienemmällä viiveellä, koska reunalaskentaa toteuttavat palvelimet sijaitsevat lähempänä käyttäjää. Pienemmästä viiveestä hyötyvät erityisesti AR- ja VR-palvelut, jotka luonteensa vuoksi vaativat hyvin matalia viiveitä. Myös pilven päätyessä virhetilaan, voidaan sen häiriötilaa peittää osittain reunalaskennan avulla sen suorittaessa myös hetkellisesti pilven tehtäviä. [10]

Koska reunoille ei kuitenkaan voida sijoittaa kovinkaan paljon resursseja, ei perinteistä pilveä voida sijoittaa suoraan sinne. Ratkaisuna resurssipulaan on esitetty palvelitonta laskentaa, sen paljon tehokkaamman resurssinkäytön vuoksi. Esimerkkinä voidaan käyttää sovellusta, joka analysoi kameran ottamaa videokuvaa ja esittää käyttäjälle kiinnostavia faktoja kuvassa näkyvistä rakennuksista. Jotta sovellus on mahdollisimman toden-



Kuva 3.1. *Reunalaskenta sijaitsee käyttäjien ja pilven välissä.*

tuntuinen ja käyttäjälleen hyödyllinen, tulee faktojen ilmestyä nopeasti ja luotettavasti. Palvelitonta laskentaa voidaan reunalla hyödyntää kuvien nopeaan analysointiin ja sen jälkeen välittää vain metatieto analyysin tuloksista pilveen tarvittavien faktojen hakemiseksi. Näin pilven ei tarvitse analysoida kuvia, ja kaistaa säästetään, kun kaikkia kuvia ei välitetä pilveen asti. [11]

3.2 Esineiden internet

Esineiden internet eli IoT (engl. Internet of Things) on konsepti, jossa erilaisille esineille tai asioille annetaan IP-osoite, mahdollisuus luoda dataa erilaisten sensorien avulla ja sitten lähettää sitä internetin ylitse. Konsepti on melko laaja, eikä kovinkaan tarkasti määritely. Termin on alunperin luonut Kevin Ashton vuonna 1999. [12]

Esineiden internetillä on kuitenkin kolme tunnusmerkkiä. Siinä tulee olla ainakin yksi sensori, joka luo dataa. Sensori voi esimerkiksi olla lämpötilamittari, joka mittaa ilman tai esineen lämpötilaa minuutin välein. Esineiden internetissä sensoreilla tulee olla myös yhteys internettiin, jotta sen luomaa dataa voidaan välittää eteenpäin, tai jotta sensori voi saada uusia toimintaohjeita. [12] Yhteyden sensoriin ei tarvitse olla reaaliaikainen, vaan

yhteys voi olla päällä vain hetkellisesti säästäen virtaa. Näiden lisäksi esineiden internetissä tulee olla laskennallista tehoa, jotta sensorien tuottamaa dataa voidaan analysoida ja täten tuottaa hyödyllistä informaatiota [12].

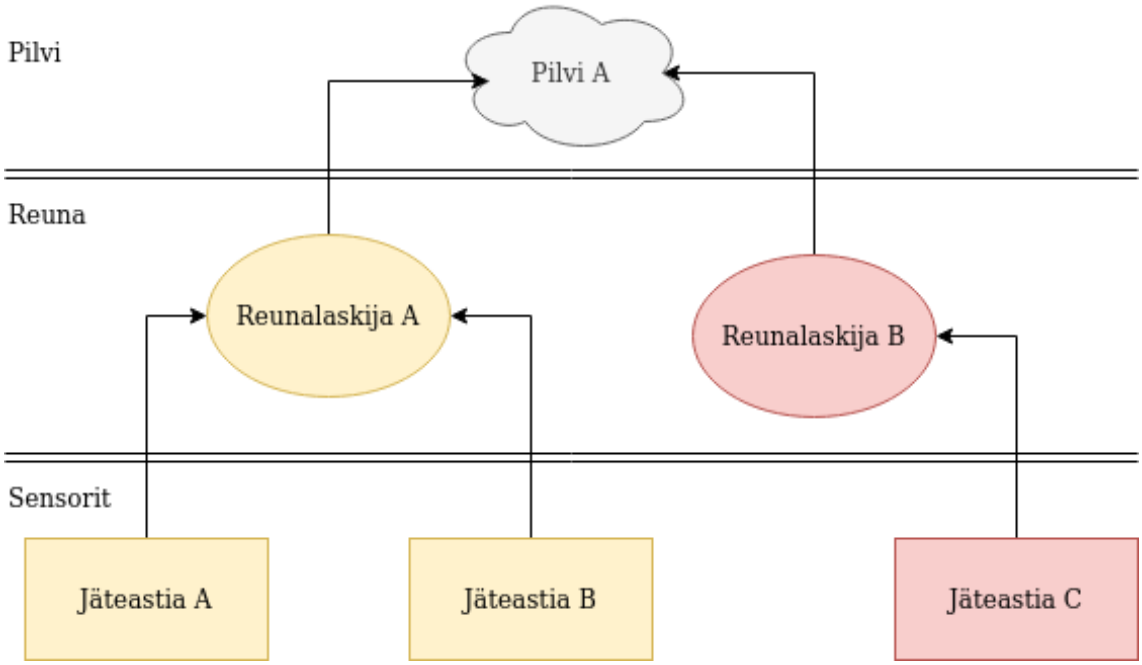
Palveliton arkkitehtuuri sopii esineiden internetin kanssa hyvin yhteen, sillä se soveltuu hyvin analysoimaan sensorien joskus satunnaisestikin välittämää dataa. Analysoinnissa käytettävät funktiot voivat hyvinkin toimia herätteillä, jolloin niitä ajetaan vain tarvittaessa (eli kun dataa saapuu analysoitavaksi). Mikäli sensori välittää dataa vain harvoin ja satunnaisesti saadessaan yhteyden internettiin, ei ole ideaalista pyörittää analysointipalvelua päällä perinteiseen tapaan koko ajan.

Myös nopea skaalautuminen on hyödyksi datan analysoinnissa. Esimerkiksi kuvitellaan valtamerellä seilaava konttilaivan kuljettavan konteissaan paljon esineitä, jotka välittävät dataa sijainnistaan internetin ylitse analysoitavaksi. Kun laiva saapuu satamaan, saavat kaikki laitteet näennäisesti samaan aikaan yhteyden internettiin. Tämä voi johtaa hetkelliseen tietoryöppyyn, joka voi kaataa analysointipalvelun. Palveliton arkkitehtuuri mahdollistaa kuitenkin analysointifunktioiden nopean luomisen ja rinnakkain ajamisen, eikä palvelu pääse kaatumaan. Perinteisemmässä järjestelmässä dataa saattaisi jopa mennä hukkaan, mikäli mahdollisia piikkejä ei ole otettu suunnittelussa huomioon. Toisaalta, jos piikit on otettu huomioon, voi olla, että palvelimien käyttötehokkuus on usein hyvin pientä ja resursseja hukataan paljon.

Eräs oikea esimerkki palvelittomasta arkkitehtuurista yhdistettynä reunalaskentaan on recycle.io. Se pyrkii vastamaan jätteiden lajittelussa kohdattaviin ongelmiin, joissa kierrätysastoihin päätyy sinne kuulumatonta jätettä. Vääriin astioihin päätynyt jäte hankaloittaa jätteiden kierrätystä ja kuluttaa turhaan resursseja, kun jätettä joudutaan jälkikäteen lajittelemaan, tai esimerkiksi poltettavaksi päätyy ainetta, joka ei pala. [13]

Järjestelmä analysoi kahdella sensorilla astiaan laitettavaa jätettä ja lähettää saamansa sensoridatan reunalaskentaan analysoitavaksi. Reunalaskenta on järjestelmässä toteutettu palvelittomana, ja se pyrkii analysoimaan saamastaan sensoridatasta, onko jäte astiaan kuuluvaa vai ei. Kun reunalaskenta on saanut analyysinsa valmiiksi, se välittää luomansa informaation eteenpäin pilveen, joka taas hoitaa datan esittämisen loppukäyttäjälle. [13] Järjestelmän arkkitehtuuri on esitetty kuvassa 3.2.

Kun järjestelmästä on kerätty tietoa tarpeeksi kauan, voidaan sen avulla kohdentaa hyvinkin tarkasti kierrätysohjeita ongelmakohteiden läheisyyteen sekä mahdollisesti antaa sakkoja haitallisen toiminnan estämiseksi. Näin jätehuoltoa pyörittävät yritykset kykene-

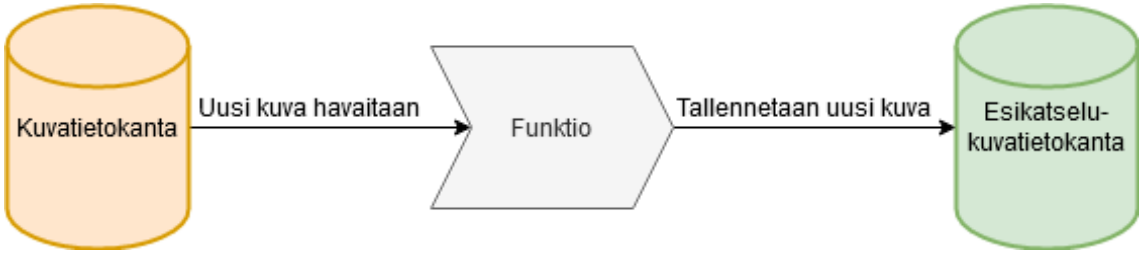


Kuva 3.2. *Periaatekuva recycle.io -palvelun arkkitehtuurista.*

vät keskittämään resurssinsa paremmin ja saamaan säästöjä toimintaansa. [13]

3.3 Kuvien ja videoiden manipulointi

Eräs hyvin suosittu käyttökohde palvelittomalle on kuvien käsittely ja manipulointi. Kuvia voidaan käsitellä esimerkiksi tapahtumapohjaisesti, eli voidaan lähettää tapahtumailmoitus, kun kuva on ladattu palvelimelle. Tämä tapahtuma voidaan liittää herätteeksi palvelittomalle funktiolle, joka voi esimerkiksi automaattisesti muokata kuvan kokoa tai tiedostotyyppiä. [14]



Kuva 3.3. *Yksinkertainen kuvan prosessointi funktion avulla. Muokattu lähteestä [14].*

Kuvassa 3.3 on esitetty yksinkertainen tapaus kuvan prosessoinnista. Palvelimelle ladataan uusi kuva, josta halutaan tuottaa esikatselukuva (ns. thumbnail-kuva). Uuden kuvan saapuessa palvelimelle, lähetetään heräte funktiolle, joka muokkaa kuvasta sopivan kokoisen ja tallentaa sen uuteen esikatselukuvatietokantaan.

Esimerkiksi Yhdysvalloissa The Seattle Times käyttää palvelittomia funktioita kuviensa kokojen muuttamiseen. Tämä helpottaa sivuston ylläpitämistä, sillä kuvia lähetettäessä ei tarvitse valmiiksi ajatella, mihin kaikkiin kokoihin kuvaa täytyy muokata. Palvelittomat funktiot tekevät muokkaukset automaattisesti ottaen huomioon myös sen, katsotaanko kuvaa kännykän pienellä näytöllä vai älytelevision suurelta näytöltä. Isojen uutisten saapuessa ja sivuston käyttäjämäärän räjähtäessä, hoituu myös skaalautuminen helposti palvelittoman arkkitehtuurin ansiosta. [15]

3.4 Nettisivut

Erityisesti hetkeksi aikaa luotavat nettisivut ovat erinomainen mahdollisuus hyödyntää palvelitonta arkkitehtuuria. Esimerkiksi konfrenssin nettisivujen toteuttaminen palvelittomalla arkkitehtuurilla voi olla hyvinkin kustannustehokas ratkaisu. Konfrenssin lähestyessä nettisivulla vierailijoiden määrä voi olla hyvinkin suurta, mutta sen jälkeen kävijämäärä voi laskea hyvinkin nopeasti hyvin vähäiseksi. Palvelittoman laskennan avulla voidaan pyyntöjen määrän vaihtelut hoitaa kustannustehokkaasti. [6]

Myös staattiset nettisivut toimivat palvelittomassa arkkitehtuurissa. Luonteeltaan ne ovat tilattomia, eivätkä siksi tarvitse tietoa siitä, mitä aiemmin on tapahtunut. Näin ollen niiden tarjoilu käyttäjälle on helppoa ja kustannustehokasta palvelittomasti. [6]

3.5 Viestibotit

Viestibotit (engl. chatbot) ovat ohjelmia, joiden pyrkimys on simuloida keskustelua ihmisen kanssa. Keskustelu niiden kanssa muistuttaa kuitenkin usein tennispeliä, sillä yleensä keskustelu pallottelee vuorotellen puhujan ja botin välillä. [16]

Viestibottien rakentaminen on melko haastavaa, mikäli toimintaan halutaan saada aitoutta. Koska botin kanssa viestivä ihminen voi ilmaista saman asian hyvin monella tavalla, vaaditaan botilta kykyä tunnistaa tekstiä tai puhetta hyvinkin monipuolisesti. Tästä muodostuu myös ongelma, sillä botin pitäisi kyetä keskustelemaan useiden eri palvelujen kanssa, jotka hoitavat omia osa-alueitaan, esimerkiksi juuri tekstin analysointia [17].

Palvelittoman arkkitehtuurin funktioita voidaankin hyödyntää eri palvelujen välissä välittäjinä [17]. Palveliton arkkitehtuuri on toimiva ratkaisu viestiboteille, sillä ne ovat enimmäkseen tilattomia eikä niiden tarvitse olla koko ajan päällä. Näin saadaan rahallisia säästöjä sekä myös lyhyt kehitysaika, kun taustalla olevista palvelimista ei tarvitse niinkään huo-

lehtia. [18]

3.6 Koneoppiminen

Koneoppiminen (engl. machine learning) tarkoittaa sellaisia ohjelmia, jotka osaavat oppia kokemastaan ja oppimansa avulla muuttamaan toimintaansa ilman, että niitä tarkalleen ottaen käsketään niin tekemään [19, 20]. Koneiden opettaminen vaatii usein paljon aikaa ja sitä varten hankittua laitteistoa, kuten esimerkiksi näytönohjaimia. Kuitenkin opettamisvaiheen jälkeen, koneoppimisen käyttö päättelemiseen (engl. inferencing, serving) ei enää vaadi niin paljon tehoa laitteistoltaan. [21]

Koneoppimisen päätelmien esittäminen palvelittoman avulla on hyvinkin mahdollista. Herätteinä funktioille voi toimia esimerkiksi palvelimelle lähetetty kuva, josta halutaan tunnistaa kaikki kasvot, joista halutaan kenties poistaa punaiset silmät. Kuitenkin mikäli funktiot joutuvat tekemään kylmäkäynnistyksen, hidastuu palvelu tuntuvasti, koska funktion sisältävän kontin pystyttäminen ajoympäristöön vie hetken aikaa [21]. Mikäli kontti kuitenkin on jo ajossa, ovat vastausnopeudet kohtuullisia, koska sitä ei jouduta pystyttämään muistista [21].

Jotta koneoppimista voitaisiin toteuttaa enemmänkin palvelitonta laskentaa hyödyntäen, tulee palveluntarjoajien tarjota mahdollisuus näytönohjaimien käyttöön laskennassa. Näytönohjaimien käyttö lisää koneoppimisen tehokkuutta ja nopeutta, jolloin myös mahdollisuudet hyödyntää palvelitonta koneoppimisessa kasvavat. Täyden potentiaalin saavuttaminen vaatii myös nopeita tietokantapalveluita. [21, 22]

3.7 Tilausvideo

Tilausvideo (engl. video-on-demand) on nimitys palveluille, joista käyttäjän on mahdollista katsoa ennalta tallennettuja videoita silloin kun käyttäjä niin haluaa tehdä [23]. Esimerkkejä tilausvideopalveluista ovat esimerksi Yle Areena, Youtube ja Netflix.

Palveliton soveltuu videoiden tarjoamiseen käyttäjille hyvin, sillä niiden tarjoilu on loppujen lopuksi hyvin tapahtumapohjaista. Kun käyttäjä haluaa katsoa jonkin videon, hän lähettää pyynnön siitä palvelimelle, joka alkaa lähettämään videota verkon ylitse käyttäjälle. Skaalautuminen tapahtuu palvelittoman avulla helposti suositun videon kohdalla, kun taas hiljaisempana hetkenä resursseja voidaan ohjata muuhun käyttöön.

Koska videot ovat loppujen lopuksi vain jono peräkkäin nopeasti toistettavia kuvia, sovel-

tuvat FaaS-funktiot myös niiden muokkaamiseen. Palveluntarjoajat voivat hyödyntää tätä optimoidessaan videota eri laitteille. Asiaa on käsitelty tarkemmin kappaleessa 3.3.

Esimerkiksi Netflix käyttää FaaS-funktioita oman tilausvideopalvelunsa toteuttamiseen. Eräitä syitä Netflixin valinnalle ovat kehitysympäristöjen helppous kehittäjille, sillä heidän ei tarvitse keskittyä kuin itse koodin kirjoittamiseen sen sijaan, että huolehtisivat monimutkaisen ajoympäristön luomisesta. Järjestelmä toi säästöjä myös palvelinkäytössä [24]. Netflix onnistui myös parantamaan virheidensä raportointia sekä kehittämään tietoturvaansa hyödyntäen FaaS-funktioita [25].

4 PALVELITONTA ARKKITEHTUURIA TUKEVIA PALVELUNTARJOAJIA

Tässä luvussa suoritetaan hyvin kevyt yleiskatsaus palvelitonta laskentaa tukeviin alustoihin. Palveluntarjoajista esiteltäväksi valikoitui suurimmat ja tunnetuimmat, niiden olessa todennäköisin valinta palveluntarjoajaa etsittäessä. Mittausyksikkönä hinnassa käytetään GB-sekunttia, joka tarkoittaa sitä, että funktiota ajetaan sekunnin ajan ympäristössä, jossa on 1 GB muistia.

4.1 AWS Lambda

Amazon Web Servicen Lambda -palvelu saapui markkinoille vuonna 2014 ja olikin ensimmäinen isompi tarjoaja alalla. AWS Lambdalla on monia suuria asiakkaita, joista tunnetuimpia ovat Netflix, Thomson Reuters ja The Coca Cola Company. [26]

AWS Lambda tukee ympäristössään monia kieliä. Tuettuja ovat Java, Go, Node.js, C, Python ja Ruby. AWS Lambda tukee monipuolisesti myös erilaisia herätteitä funktioihinsa, ja integraatio muiden AWS palvelujen kanssa on lähes saumatonta, minkä vuoksi esimerkiksi koneoppimismallien käyttö helpottuu. [26]

Palvelun hinnasto vaihtelee hieman maittain, mutta Frankfurtissa hinta on GB-sekunnilta on $0.0000166667. Pyynnöistä laskutetaan myös $0.20 per miljoona pyyntöä, joskin ensimmäiset miljoona pyyntöä ovat ilmaisia. [26]

4.2 Microsoft Azure Functions

Microsoftin Azure pilvialusta tukee palvelittomia funktioita ja sisältää monia valmiiksi räätälöityjä palveluja asiakkailleen. Palvelu tukee mm. integraatiota Azuren kognitiivisiin palveluihin, joita voidaan hyödyntää puheen tai tekstin ymmärtämisessä. Azuren palveliton laskenta tukee myös Azuren omia koneoppimismalleja ja niiden käyttöä palvelittoman

rinnalla. [27]

Tuettuja kieliä Azuren palvelussa on seitsemän kappaletta. Ne ovat C, Javascript, F, Java, Powershell, Python ja TypeScript. [27]

Myös Azurella hinnasto vaihtelee hieman kohteittain, mutta kohteessa "Germany West Central"hinta GB-sekunnilta on $0.000016. Pyynnöistä laskutetaan sama hinta kuin AWS Lambdallakin, eli $0.20 per miljoona pyyntöä ensimmäisen miljoonan pyynnön jälkeen.[27]

4.3 Google Cloud Functions

Myös Googlen Cloud Functionsista eli GCF:sta löytyy kattava ja laaja tuki Google Cloudin muille palveluille. Esimerkiksi Googlelta löytyy valmiiksi koulutettu koneäly Vision API, jota voi hyödyntää kuvien analysointiin palvelittomassa ympäristössä. Cloud Functions tukee myös avoimen lähdekoodin palvelittoman laskennan viitekehyksiä (engl. framework), joita tukemalla voidaan estää palveluntarjoajaan lukkiutuminen (engl. vendor lock-in). [28]

GCF:n kielivalikoima on hieman heikompi kuin muiden. Se tukee tällä hetkellä kolmea kieltä, jotka ovat Node.js, Python ja Go.[28]

Hinnastoltaan Google Cloud on jakanut hintansa kahteen tasoon alueiden mukaan. Tason yksi hinta on $0.0000025 GB-sekuntilta, kun taas tason kaksi hinta on $0.0000035. Kaikki muut alueet paitsi Frankfurt kuuluvat tason yksi hinnastoon. Pyynnöissä ensimmäiset kaksi miljoonaa ovat ilmaisia, mistä eteenpäin hinta on $0.40 per miljoona pyyntöä. Huomioitavaa hinnastossa on, että vaikka GB-sekuntiltaan GCF on hyvinkin halpa, laskuttaa se myös GHz-sekunteista $0.0000100. [28]

4.4 IBM Cloud Functions

IBM Cloud Functions hyödyntää taustallaan Apachae OpenWhisk -tekniikkaa. OpenWhisk itsessään on avoimen lähdekoodin palveliton viitekehys, jota kuka tahansa voi halutessaan hyödyntää palvelittomien sovellusten kehityksessä. Myös IBM tarjoaa palvelittoman kylkeen omia koneoppimismallejaan, kuten IBM Watson®API:n, jonka avulla voidaan toteuttaa kognitiviista analyysia. [29]

IBM Cloud Functionsin tukee myös vertailusta useimpia kieliä, sillä sen tukemia kieliä ovat Node.js, Python, Swift, PHP, Ruby, Java ja .NET Core. IBM ilmoittaa myös tukevansa muita kieliä käyttäen Docker-tekniikkaa. [29]

Hinnastoltaan IBM Cloud Functions on myös kilpailukykyinen. Sen hinta GB-sekunteissa on $0.000017 ja pyynnöillä ei ole erikseen hintaa. Suuremmilla määrillä pyyntöjä hinta on hyvinkin kilpailukykyinen. [29]

5 JOHTOPÄÄTÖKSET

Palveliton laskenta toimii hyvin tilanteissa, joissa koodin ajamiseen johtaa jonkinlainen tapahtuma. Palveliton laskenta on luonteeltaan hyvin tapahtumapohjaista, joten tulos ei sinänsä yllätä. Erityisen hyvin palveliton laskenta soveltuu tilanteisiin, jossa palvelimen vasteaika ei ole kriittinen tai jos palvelimen pitää sopeutua hyvin nopeisiin käyttöasteiden muutoksiin.

Palvelittoman laskennan kyky skaalautua nopeasti käyttäjämäärän kasvamiseen on erittäin hyvä ominaisuus tilanteissa, jossa käyttäjämäärää ei osata tarkasti ennustaa. Esimerkiksi uutissivustot voivat hyötyä tästä nopeasta skaalautumisesta, sillä niiden ei tarvitse ostaa liikaa laskentatehoa suurien uutisten varalle, jotka voivat moninkertaistaa sivuston käyttäjämäärän. Tämä ominaisuus tuo palvelittoman käyttäjälle säästöjä ja mielenrauhaa, sillä skaalautumisen ratkaisut jäävät palvelitonta laskentaa tarjoavan palveluntarjoajan pohdittavaksi.

Nopeita markkinavaltauksia on myös mahdollista tehdä helpommin palvelittoman avulla. Kehitystiimi voi palvelitonta käyttäessään keskittyä vain ohjelman logiikan luontiin palvelimien ylläpidon ja ajoympäristöjen kasaamisen jäädessä valitun palveluntarjoajan hoidettavaksi. Tämä nopeuttaa markkinoille pääsyä ja täten markkinaosuuksien valtausta.

Palveliton soveltuu hyvin resurssikriittiseen reunalaskentaan ja esineiden internetin massiivisen datamäärän analysointiin ja käsittelyyn. Varsinkin reunalaskennan vähäiset laskuresurssit hyötyvät suuresti siitä, että palvelittonta laskentaa hyödynnyttäessä resursseja voidaan vapauttaa palveluilta, jotka eivät niitä aktiivisesti käytä.

Palvelittoman pahin heikkous on sen viive kylmäkäynnistyksen yhteydessä. Tämä lisäviive rajoittaa sen käyttöä tilanteissa, joissa nopea vasteaika on ehdoton edellytys. Näin ollen palvelitonta laskentaa ei voida hyödyntää ainakaan suoraan esimerkiksi itsestään ajavien autojen kriittisen sensoridatan analysoinnissa, jos kyseistä dataa käytetään muiden autojen varoittamiseen tai muuhun hyvin kriittiseen. Mikään ei toki estä palvelittoman hyödyntämistä jonkin ei kriittisen analysointiin, kuten vaikkapa autojen ulkolämpötilamit-

tausten hyödyntämiseen sääpalveluissa.

Myös mikäli palvelun käyttäjämäärä tiedetään hyvin tarkasti, eikä riskiä käyttäjäpiikeistä ole, voi perinteinen ratkaisu tulla jopa halvemmaksi kuin palvelittoman maksa kun käytät -ratkaisu. Palvelinarkkitehtuuria pohdittaessa on otettava monia asioita huomioon, eikä mikään ratkaisu yksiselitteisesti voita jokaisessa tilanteessa. Palveliton laskenta ei ole tähän sääntöön poikkeus, vaan se jopa korostaa suunnittelun tärkeyttä ohjelmistoprojektissa. Mikäli suunnittelu tapahtuu ilman kunnollista käsitystä vaihtoehdoista, voi palvelittomankin edut jäädä käyttämättä.

6 YHTEENVETO

Tämä kandidaatin tutkielma käsitteli palvelitonta laskentaa ja FaaS-funktioita. Tutkielma esitteli palvelittoman laskennan perusperiaatteet sekä palvelittoman osana mikropalveluarkkitehtuuria kuten myöskin sen suuremmat eroavaisuudet verrattuna monoliittiin.

Tutkielmassa perehdyttiin tutkimuskysymyksenä myös siihen, mihin palveliton laskenta oikein soveltuu. Tuloksena saatiin, että palveliton soveltuu monelaisiin erilaisiin tilanteisiin varsinkin, kun se voidaan ottaa yhdeksi palveluksi isompaan mikropalveluarkkitehtuuriin. Erityisen hyvin havaittiin palvelittoman soveltuvan tilanteisiin, jossa asiat tapahtuvat tapahtumajohteisesti, esimerkiksi kun palvelimelle tuodaan uusi kuva tai kun mittauslaite lähettää mittaustuloksensa analysoitavaksi.

Ohessa havaittiin myös, että palvelittoman heikkoudet tulee ottaa huomioon jo suunnitteluvaiheissa, jotta esimerkiksi sen hitaus kylmäkäynnistyksen yhteydessä osataan kompensoida. Palveliton ei myöskään sovellu kaikkiin tilanteisiin, ja esimerkiksi hyvin nopeaa vasteaikaa vaativat palvelut eivät vielä tällä hetkellä sovellu palvelittomaan laskentaan juurikin kylmäkäynnistyksen hitauden vuoksi.

Näiden lisäksi tutkielmassa perehdyttiin nopeasti palvelitonta laskentaa tukeviin palveluntarjoajiin ja vertailtiin hieman näiden käyttöä. Palveluntarjoajien omat alustat ovat kehittyneet hyvinkin monipuolisiksi, eikä selkeää voittajaa voida sanoa olevan olemassa. Palveluntarjoajaa valittaessa tulee kuitenkin huomioida mahdollinen lukittautuminen palveluntarjoajaan, sillä palveluntarjoajien totetukset eroavat eivätkä palvelut ole välttämättä suoraan liikuteltavissa palveluntarjoajalta toiselle. Onkin tärkeää tehdä tutkimusta ennen alustan valintaa, jotta palveluntarjoaja vastaa omia tarpeita.

LÄHTEET

[1] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R. ja Safina, L. Microservices: Yesterday, Today, and Tomorrow. *Present and Ulterior Software Engineering*. Toim. M. Mazzara ja B. Meyer. Cham: Springer International Publishing, 2017, 195–216. ISBN: 978-3-319-67425-4. DOI: 10.1007/978-3-319-67425-4_12.

[2] Ruutiainen, O. Diplomityö. *Serverless-arkkitehtuurin hyödyntäminen ohjelmistoprojektissa*. Tampereen Teknillinen Yliopisto, 2017.

[3] Arora, C. *Function As A Service*. 2018. URL: https://zenodo.org/record/1487739 (viitattu 09. 03. 2020).

[4] Newman, S. Building microservices : designing fine-grained systems. *Building microservices : designing fine-grained systems*. Sebastopol, California: O'Reilly, 2015. ISBN: 1-4919-5033-1.

[5] Savage, N. Going serverless. *Communications of the ACM* 61.2 (2018), 15, 16. ISSN: 00010782.

[6] Zanon, D. *Building serverless web applications : build scalable web apps using Serverless Framework on AWS*. eng. Birmingham, England: Packt. ISBN: 1-78712-647-1.

[7] Fox, G. C., Ishakian, V., Muthusamy, V. ja Slominski, A. *Status of Serverless Computing and Function-as-a-Service(FaaS) in Industry and Research*. 2017.

[8] *What Is Serverless Computing?* URL: https://www.cloudflare.com/learning/serverless/ (viitattu 02. 03. 2020).

[9] Shafiei, H., Khonsari, A. ja Mousavi, P. *Serverless Computing: A Survey of Opportunities, Challenges and Applications*. 2019. URL: http://arxiv.org/abs/1911.01296.

[10] Satyanarayanan, M. Edge Computing. *Computer* 50.10 (2017), 36–38.

[11] Baresi, L., Filgueira Mendonça, D. ja Garriga, M. Empowering Low-Latency Applications Through a Serverless Edge Computing Architecture. *Service-Oriented and Cloud Computing*. Toim. F. De Paoli, S. Schulte ja E. Broch Johnsen. Cham: Springer International Publishing, 2017, 196–210. ISBN: 978-3-319-67262-5.

[12] Doyle, C. Internet of Things. *A Dictionary of Marketing*. Oxford University Press, 2016. ISBN: 9780198736424.

[13] Al-Masri, E., Diabate, I., Jain, R., Lam, M. H. L. ja Nathala, S. R. A Serverless IoT Architecture for Smart Waste Management Systems. *2018 IEEE International Conference on Industrial Internet (ICII)*. 2018, 179–180.

[14] Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A. ja Suter, P. *Serverless Computing: Current Trends and Open Problems*. 2017. URL: http://arxiv.org/abs/1706.03178 (viitattu 09. 04. 2020).

[15] *AWS Case Study: The Seattle Times*. URL: https://aws.amazon.com/solutions/case-studies/the-seattle-times/ (viitattu 09. 04. 2020).

[16] Stevenson, A. ja Lindberg, C. A. chatbot. *New Oxford American Dictionary*. Oxford University Press, 2015. ISBN: 9780195392883.

[17] Yan, M., Castro, P., Cheng, P. ja Ishakian, V. Building a Chatbot with Serverless Computing. *Proceedings of the 1st International Workshop on Mashups of Things and APIs*. MOTA '16. Trento, Italy: Association for Computing Machinery, 2016. ISBN: 9781450346696. DOI: 10.1145/3007203.3007217. URL: https://doi.org/10.1145/3007203.3007217.

[18] Lehvä, J., Mäkitalo, N. ja Mikkonen, T. Case Study: Building a Serverless Messenger Chatbot. English. Vol. 10544. Springer Verlag, 2018, 75–86. ISBN: 9783319744322.

[19] Butterfield, A., Ngondi, G. E. ja Kerr, A. Machine Learning. *A Dictionary of Computer Science*. Oxford University Press, 2016. ISBN: 9780199688975.

[20] Gorse, C., Johnston, D. ja Pritchard, M. Machine Learning. *A Dictionary of Construction, Surveying and Civil Engineering*. Oxford University Press, 2020. ISBN: 9780198832485.

[21] Ishakian, V., Muthusamy, V. ja Slominski, A. *Serving deep learning models in a serverless platform*. 2018. URL: http://arxiv.org/abs/1710.08460.

[22] Carreira, J., Fonseca, P., Tumanov, A., Zhang, A. ja Katz, R. *A Case for Serverless-Machine Learning*. 2018.

[23] Chandler, D. ja Munday, R. Video-on-Demand (VOD). *A Dictionary of Media and Communication*. Oxford University Press, 2020. ISBN: 9780198841838.

[24] Xiao, Y. *Going FaaSter: Function as a Service at Netflix*. Coding Tech. 2018. URL: https://www.youtube.com/watch?v=66PxX3oGVCA (viitattu 20. 05. 2020).

[25] Demian, J. *Serverless Case Study - Netflix*. dashbird. 2018. URL: https://dashbird.io/blog/serverless-case-study-netflix/ (viitattu 20. 05. 2020).

[26] *AWS Lambda*. URL: https://aws.amazon.com/lambda/ (viitattu 09. 04. 2020).

[27] Azure, M. *Azure Serverless dokumentaatio*. URL: https://azure.microsoft.com/en-us/solutions/serverless/ (viitattu 21. 05. 2020).

[28] Cloud, G. *Google Cloud Functions dokumentaatio*. URL: https://cloud.google.com/functions (viitattu 21. 05. 2020).

[29] IBM. *IBM Cloud Functions dokumentaatio*. URL: https://www.ibm.com/se-en/cloud/functions (viitattu 21. 05. 2020).