Joonas Raja

Serverless-teknologian hyödyntäminen koneoppimissovellusten tuotannossa

Tietotekniikan pro gradu -tutkielma

2. lokakuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Joonas Aleksi Raja

Yhteystiedot: joonas.a.raja@student.jyu.fi, 0503734920

Ohjaajat: Ari Viinikainen

Työn nimi: Serverless-teknologian hyödyntäminen koneoppimissovellusten tuotannossa

Title in English: Utilization of serverless technology on machine learning applications

Työ: Pro gradu -tutkielma

Opintosuunta: Ohjelmisto- ja tietoliikennetekniikka

Sivumäärä: 65 + 0

Tiivistelmä: Tutkielmassa perehdyttiin siihen, mitä serverless-teknologia tarjoaa pilvipohjaisille koneoppimissovelluksille ja miten koneoppimissovellus toteutetaan hyödyntäen serverless teknologiaa. Toteutuksessa tehtiin kaksi idealtaan vastaavanlaista koneoppimissovellusta, joista ensimmäinen hyödynsi Microsoft Azure Machine Learning-alustaa ja toinen toteutettiin puhtaasti serverless-funktioiden avulla. Toteutusten vertailun ja aiemman tutkimuksen avulla tehtiin suunnittelutieteellinen analyysi, jonka tuloksena syntyi arvio serverless-teknologian hyödyistä ja optimaalisesta käytöstä koneoppimissovellusten arkkitehtuurissa.

Avainsanat: Serverless, koneoppiminen, pilvilaskenta, Azure, SaaS, FaaS, PaaS

Abstract: This thesis focuses on what serverless technology offers for machine learning cloud applications and how they are built using the serverless technology. In the implementation part of the study, two similar machine learning applications were developed. The first one was done using Azure Machine Learning platform and the second one purely using serverless functions. A design scientific analysis was conducted based on the comparison of the implementations and previous research. The outcome of the study provided us an evaluation of the benefits and optimal use of serverless technology in the architecture of machine learning applications.

Keywords: Serverless, machine learning, cloud computing, Azure, SaaS, FaaS, PaaS

Termiluettelo

Serverless	Palveliton (paradigma, teknologia, liiketoimintamalli)
E2E	Alusta-loppuun (engl. End to End)
GPU	Grafiikkasuoritin (engl. Graphics Processing Unit)
API	Rajapinta (engl. Application Programming Interface)
IDE	Ohjelmointiympäristö (engl. Integrated Development Environment)
SDK	Ohjelmistokehityspaketti (engl. Software Development Kit)
Client	Asiakasohjelma
HTTP	Tietoliikenneprotokolla (Hyper Text Transfer Protocol)
AML	Azure Machine Learning
ML	Koneoppiminen (engl. Machine Learning)
DB	Tietokanta (engl. Database)
JSON	Dataformaatti (JavaScript Object Notation)
FaaS	Funktio palveluna (engl. Function as a Service)
SaaS	Sovellus palveluna (engl. Software as a Service)
PaaS	Alusta palveluna (engl. Platform as a Service)
IaaS	Infrastruktuuri palveluna (engl. Infrastructure as a Service)

Kuviot

Kuvio 1. Kehittäjän vastuualueet pilvimalleissa (Wolf 2020) 9
Kuvio 2. Serverlessin resurssitehokkuus (Van Eyk et al. 2018) 10
Kuvio 3. Serverless-teknologian kehittyminen (Van Eyk et al. 2018) 11
Kuvio 4. Valvottu koneoppimisprosessi (Oladipupo 2010) 15
Kuvio 5. Serverless ML kehitysmalli (Osipov 2020) 17
Kuvio 6. Serverless-instanssit (Feng et al. 2018) 18
Kuvio 7. AML työnkulku ("Azure Machine Learning Documentation | Microsoft Docs" n.d.) 26
Kuvio 8. Azure Resource Manager ("Azure Documentation | Microsoft Docs" n.d.) 27
Kuvio 9. AML Studion aloitusnäkymä 28
Kuvio 10. Mallin koulutusputki designerissa 29
Kuvio 11. Mallin arvioidut virheet 31
Kuvio 12. Päättelyputki designerissa 31
Kuvio 13. Testaustulos 32
Kuvio 14. API funktion ajaminen lokaalisti Azure Function Core Toolsin komentorivillä 35
Kuvio 15. API-funktion julkaisu .zip tiedostona Visual Studion käyttöliittymässä 36
Kuvio 16. Resurssiryhmän alle luodut resurssit Azuressa 36
Kuvio 17. Esimerkki API:lle annetusta syötteestä 37
Kuvio 18. Esimerkki API:lta saadusta vastauksesta 38
Kuvio 19. API-funktion käyttämä muistin määrä kutsujen aikana 38

Sisältö

1 JOHDANTO 1
1.1 Tutkimuskysymykset 2
1.2 Tutkimusstrategia 2
1.3 Tutkimuksen rakenne 3

2 PILVILASKENTA 4
2.1 Historia ja määrittely 4
2.1.1 Pilvilaskennan taloudelliset hyödyt 5
2.1.2 Haasteita ja niiden ratkaisuja 6
2.2 Käsitteiden määrittelyä 8
2.3 Serverless-paradigma 9

3 KONEOPPIMINEN 13
3.1 Teoriaa 13
3.1.1 Oppimistyyppejä 14

4 SERVERLESS KONEOPPIMISEN NÄKÖKULMASTA 17
4.1 Haasteet verkon koulutuksessa 18
4.2 Mallin koulutus 18
4.3 Päättely 20

5 TOTEUTUS 22
5.1 Käytetyt työkalut 22
5.1.1 Azure Functions 23
5.1.2 Azure Machine Learning 23
5.1.3 Muita toteutuksessa käytettyjä työkaluja 23
5.2 Perusteet toteutukselle 24
5.2.1 Azure Machine Learning työnkulku 26
5.3 Kooditon toteutus AML Designerin avulla 27
5.4 Toteutus serverless-funktioiden avulla 32

6 ANALYYSI 39
6.1 Tutkimukseen liittyvät haasteet 39
6.2 Valmiit ratkaisut ja koneoppimispalvelut 40
6.3 Toteutusten arviointi 42
6.3.1 Toteutusten erot 43
6.3.2 Miksi serverless 46
6.3.3 Miksi ei serverless 46
6.3.4 Yhteenveto 47
6.4 Erilaisia arkkitetuuri- ja optimointiratkaisuja 48
6.4.1 Serverless viitekehyksiä koneoppimissovelluksille 49
6.4.2 Hyvän serverless-viitekehyksen vaatimuksia 50
6.5 Jatkotutkimus 51

7 YHTEENVETO 53

LÄHTEET 55

1 Johdanto

Serverless on pilviteknologioiden ammattilaisten keksimä lempinimi kuvaamaan tietynlaista lähestymistapaa pilvilaskentaan, jossa kehittäjien ei tarvitse huolehtia palvelintason asioista, vaan he voivat keskittyä sovelluskehityksen olennaisuuksiin ja optimoida kustannuksiaan paremmin serverlessin käyttöön perustuvan laskutuksen avulla. Serverless ei tarkoita sitä, etteikö palvelimia olisi, vaan sitä että kehittäjät voivat olla välittämättä niiden olemassaolosta. Serverless teknologia tekee pilvipohjaisesta sovelluskehityksestä huomattavan lähestyttävämpää ja nopeampaa, mikä mahdollistaa esimerkiksi kehittäjien ketteryyden nopeasti muuttuvassa markkinatilanteessa, missä uusia tuotteita ja palveluita on saatava nopeasti markkinoille automaattisesti skaalautuvina.

Koneoppiminen on tekoälyn ja data-analytiikan osa-alue, jossa algoritmi saadaan tekemään ratkaisuja tai päättelyitä perustuen suureen määrään dataa, jonka se on aiemmin käsitellyt ja luonut dataan perustuvan mallin. Koneoppimista käytetään tänä päivänä lukuisissa sovelluksissa tekemään data-analytiikan automatisointia ja ennusteita.

Serverless- ja koneoppimisteknologiat sijaitsevat kahden informaatioteknologian alan risteyksessä, jossa suuret palveluntarjoajat käyvät kiivasta kamppailua asiakkaista. Koneoppiminen avaa uusia mahdollisuuksia yrityksille luoda uusia tuotteita ja palveluita samaan aikaan kun serverless tekee sovelluskehittämisestä koko ajan lähestyttävämpää ja nopeampaa. Kyseisten teknologioiden sujuva yhteistoiminta ja yhteinen kehityssuunta on näin ollen hyvin tärkeää. Näiden teknologioiden kehitystahti on tällä hetkellä erittäin nopeaa ja on vaikea arvioida mihin suuntaan esimerkiksi serverless kehittyy lähitulevaisuudessa palveluntarjoajien osalta.

Tutkimuksen ideana on selvittää mitä hyötyjä ja rajoitteita liittyy koneoppimissovelluksen toteuttamiseen serverless-arkkitehtuurin mukaisesti ja miten tällainen sovellus suunnitellaan ja toteutetaan pilvialustalle. Tutkimuksen aikana kartoitetaan tietämystä toteutusta varten ja selvitetään vastausta kysymyksiin: “Miten serverless arkkitehtuuri eroaa tavallisesta palvelinsovelluksen arkkitehtuurista ja mitä etuja ja rajoitteita se tuo mukanaan”, “Miten koneoppimissovellus toteutetaan serverlessinä pilvialustalle”, sekä “Miten serverless teknologiaa

voidaan ja kannattaa nykyisin hyödyntää koneoppimissovellusten arkkitehtuurissa?”. Tutkimuksen aineistona toimii kaksi kaupalliselle pilvialustalle suunniteltua ja toteutettua koneoppimissovellusta, jotka koulutetaan ja testataan valmiilla datasetillä. Tutkimuksen tuloksena toimii syntyneistä sovelluksista ja kirjallisuudesta tehty suunnittelututkimuksen mukainen analyysi.

1.1 Tutkimuskysymykset

Alustavat tutkimuskysymykset ovat:

1. Miten serverless-arkkitehtuuri eroaa tavallisesta palvelinsovelluksen arkkitehtuurista ja mitä etuja ja rajoitteita se tuo mukanaan koneoppimissovellukselle?
2. Miten koneoppimissovellus toteutetaan serverlessinä pilvialustalle?
3. Miten serverless teknologiaa voidaan ja kannattaa nykyisin hyödyntää koneoppimissovellusten arkkitehtuurissa?

Ensimmäiseen kysymykseen vastataan pääosin kirjallisuusosuudessa ja toiseen ja kolmanteen kysymykseen perehdytään toteutuksessa ja analyysissa.

1.2 Tutkimusstrategia

Tutkimus tullaan toteuttamaan suunnittelutieteellisenä (engl. design science) tutkimuksena, sillä tutkimuksen kiinnostuksen kohteena on erityisesti tapa, jolla toteutus tehdään ja se miten se eroaa muista toteutustavoista. Lopullinen konstruktio, eli toteutettava sovellus, ei ole tässä tapauksessa huomion keskipisteenä.

Tarkoituksena on kehittää yksinkertainen koneoppimissovellus siten, että se voidaan toteuttaa serverless teknologiaa hyödyntämällä kaupalliselle pilvialustalle, sekä alustan omaa koneoppimispalvelua hyödyntävä vaihtoehto vertailun vuoksi. Tutkimuksen aineistona on siis sovelluksen kehityskaari sekä lopputulos. Tutkittavaksi alustaksi valikoitui Microsoft Azure.

Koneoppimissovelluksen käyttämä datasetti ei ole olennainen tutkimuksen kannalta. Tutkimuksen tuloksena tulisi löytyä vastauksia tutkimuskysymyksiin saaden saamalla käsitys siitä, miten koneoppimissovellus suunnitellaan ja toteutetaan pilveen ja erityisesti miten serverless teknologiaa hyödynnetään optimaalisesti tällaisen sovelluksen arkkitehtuurissa.

1.3 Tutkimuksen rakenne

Tutkimuksen alussa perehdytään aihealueisiin, kuten pilvilaskentaan, koneoppimiseen ja serverlessiin liittyvään kirjallisuuteen ja aikaisempiin tutkimuksiin. Tarkoituksena on saada kattava ymmärrys aiheeseen, jotta tutkimuksen toteutusta voidaan lähestyä oikein. Kirjallisuuskatsauksessa pyritään myös selvittämään serverlessin ja koneoppimisen suhdetta ja löytämään vastaus siihen, mitä etuja ja rajoituksia serverless teknologia tarjoaa koneoppimisen näkökulmasta. Tutkimuksen toteutusvaiheessa käydään läpi toteutukseen liittyvää suunnittelua, sekä dokumentoidaan toteutuksen vaiheet ja testaus. Analyysikappaleessa pyritään löytämään vastauksia tutkimuskysymyksiin toteutukseen ja kirjallisuuteen perustuen.

2 Pilvilaskenta

Pilvilaskenta ja tietojenkäsittelyn tarjoaminen palveluna ovat muokanneet informaatioteknologian alaa huomattavassa määrin erityisesti viimeisen vuosikymmenen aikana. Ohjelmistoista palveluina on tullut sekä niitä tarjoaville yrityksille, että niitä käyttävälle ihmisille alati houkuttelevampi vaihtoehto. Sovelluskehittäjät pystyvät toteuttamaan ratkaisujaan nopeasti ilman omaa kallista palvelinlaitteistoa, kantamatta huolta esimerkiksi skaalautuvuusongelmista.

Tässä kappaleessa määritellään ja avataan pilvilaskentaan liittyviä käsitteitä, sekä perehdytään pilvessä tapahtuvaan laskentaan, pilviarkkitehtuureihin sekä erilaisiin tarjolla oleviin pilvipalveluihin. Erityisesti tarkastelemme niin kutsuttua palvelitonta (engl. serverless) arkkitehtuurimallia, joka on tämän tutkimuksen keskiössä. Mitä se tarkoittaa teknisestä- ja liiketoiminnallisesta näkökulmasta ja mitkä ovat sen tuomat edut?

2.1 Historia ja määrittely

Pilvilaskenta (engl. cloud computing) on 2000-luvulla syntynyt käsite, joka viittaa henkilökohtaisen tietokoneen ulkopuolella hajautetusti tapahtuvaan laskentaan internetin välityksellä. Yksinkertaisimmillaan sillä voidaan tarkoittaa ulkoiselle palvelimelle tallennetun datan käyttöä, mutta useimmiten käsitettä käytetään silloin, kun puhutaan prosessoinnin ulkoistamisesta.

Prosessoinnin ulkoistaminen on mahdollistanut sovelluskehittäjien irtautumisen palvelinarkkitehtuurin rakentamisesta ja ylläpidosta, sekä vapauttanut sovelluskehityksen käyttöjärjestelmien ja sovelluksen välisistä yhteensopivuusongelmista. Sovelluskehittäjät eivät joudu enää suunnittelemaan ja päivittämään sovelluksiaan loppukäyttäjien käyttöjärjestelmien ja laitteiston vaatimusten mukaan vaan voivat kehittää, testata ja suorittaa sovelluksen valitsemassaan ympäristössä ja tarjota sen palveluna verkon yli. (Hayes 2008)

Hajautettu laskenta tapahtuu useissa hyvin tehokkaissa laitteissa, jotka ovat yhteydessä toisiinsa nopeilla tietoliikenneyhteyksillä. Näitä laitteita hallitaan ja ylläpidetään keskuksissa,

jotka ovat myös yhteydessä toisiinsa verkon välityksellä. Tämä kokonaisuus on pilvilaskennan ydin, joka vastaanottaa ulkopuolelta tulevia käyttäjien kutsuja ja suorittaa tarjottavan palvelun mukaista prosessointia. (Hayes 2008)

Pilvelle on erilaisia määritelmiä riippuen kontekstista, mutta siitä puhuttaessa voidaan esimerkiksi viitata datakeskuksen laitteistoon sekä ohjelmistoon. (Armbrust et al. 2010) Pilvi voi olla julkinen, eli kaikkien vapaasti saatavissa oleva, tai sisäinen, eli jonkin organisaation omaan käyttöön tarkoitettu. Organisaatioiden sisäiset pilvet eivät kuitenkaan useimmiten pysty tarjoamaan kaikkia julkipilven etuja, kuten lähes loputonta skaalautuvuutta.(Armbrust et al. 2010) Julkisen ja sisäisen pilven lisäksi on olemassa myös useiden organisaatioiden ylläpitämiä yhteisöpilviä sekä useasta pilvimallista koostuvia hybridipilviä, jotka yhdistävät useampia pilviä standardoiduilla protokollilla. (Sajid and Raza 2013)

Pilven määritteleviä erityispiirteitä ovat: (Sajid and Raza 2013):

- Välitön saatavuus itsepalveluna
- Saavutettavuus internetyhteydellä tavanomaisin protokollin
- Resurssien yhdistäminen (engl. pooling) eri käyttäjille
- Resurssien nopea skaalautuvuus
- Palvelun mitattavuus

2.1.1 Pilvilaskennan taloudelliset hyödyt

Kehittäjien näkökulmasta pilvilaskennan välitön skaalautuvuusmahdollisuus sekä prosessointiin perustuva laskutus tuovat liiketoiminnallisia etuja. Pilvilaskennan perimmäisenä ajatuksena liiketoiminnan näkökulmasta on siirtää pääomakustannuksia operatiivisiksi kustannuksiksi. (Armbrust et al. 2010) Tällöin prosessointiin käytettävistä resursseista maksetaan tarpeen mukaan ilman alkupääomaa, jolloin erityisesti riskin osuus liiketoiminnassa pienenenee merkittävästi. Pilvilaskennan taloudellisista hyödyistä puhuttaessa voidaan seuraavat kolme asiaa nostaa erityisesti pinnalle:

- Laskentatehon nopea skaalautuvuus mahdollistaa ajan mukaan vaihteleviin kuormituspiikkeihin mukautumisen ilman suurta tyhjäkäyntiä. Tavallisten datakeskusten

palvelinkoneiden käyttöaste on keskimäärin vain noin 5 % - 20 %, sillä kuormituspiikit ovat yleensä noin 2–10 kertaisia keskimääräiseen kuormitukseen nähden. (Armbrust et al. 2010) Valtaosa laitteiden laskentatehosta jää siis normaalisti hyödyntämättä pelkästään kuormituspiikkeihin varautumisen vuoksi.

- Skaalautuvuus mahdollistaa liiketoiminnan kasvattamisen tai pienentämisen kysynnän mukaan ilman ennalta varautumista. Lisäkoneiden käyttöönottoon menee viikkojen sijasta minuutteja.
- Tuhannen laskentakoneen prosessointi tunnin maksaa saman verran kuin yhden vastaavan koneen prosessointi tuhat tuntia. Yritykset voivat prosesseja analysoimalla vähentää niin kutsuttuja pullonkauloja suorittamalla hitaimpia prosesseja nopeammin.

2.1.2 Haasteita ja niiden ratkaisuja

Pilvilaskentaan liittyy erilaisia teknisiä ja liiketoiminnallisia haasteita pilvipalveluita käyttävien organisaatioiden näkökulmasta. Seuraavaksi on lueteltu yleisimpiä haasteita, sekä niiden ratkaisuja. (Armbrust et al. 2010) (Sajid and Raza 2013)

Liiketoiminnan jatkuvuus ja palvelun saatavuus.

Teknisesti suurten toimittajien pilvipalvelut ovat olleet tähän mennessä hyvin luotettavia ja häiriöitä on ollut todella harvoin. Kuitenkaan täysin varmaa ratkaisua ei ole olemassa ja ainoa keino riskien hävittämiseksi on hajauttaa palvelu tarvittaessa useammalle palveluntarjoajalle.

Datan saatavuus

Dataa on vaikea siirtää palvelusta toiselle, jolloin asiakkaat eivät uskalla sitoutua palveluntarjoajiin, jotka saattavat kadottaa dataa tai lopettaa toimintansa. Ongelman ratkaisuksi voidaan toteuttaa standardoituja rajapintoja, joiden avulla sama palvelu voidaan toteuttaa useammalla alustalla. Tämä mahdollistaisi myös palvelun toteuttamisen ns. puuskalaskentana (engl. Surge computing), jossa ainoastaan suurin kuormitus käsiteltäisiin pilvessä sisäisen kapasiteetin tullessa rajoitteeksi.

Datan luottamuksellisuus

Dataa käsittelevän tahon on kyettävä takaamaan sen turvallisuus asiakkailleen. Pilven ulkoiset uhat ovat pitkälti samoja, kuin suurilla datakeskuksilla, jolloin palvelinarkkitehtuuri on kyettävä suojaamaan ulkoapäin tulevilta hyökkäyksiltä. Sisäiset uhat liittyvät käyttäjien toimintaan. Käyttäjien data, sekä pilviarkkitehtuuri on kyettävä suojaamaan käyttäjiltä itseltään. Ensisijaisena keinona tähän on käytetty virtualisointia, jonka avulla käyttäjän ja infrastruktuurin väliin luodaan abstraktiokerros. On myös tärkeä huomioida käyttäjien suojaaminen palveluntarjoajalta itseltään.

Pullonkaulat datan siirrossa

Käsiteltävän datan määrä ohjelmistoissa kasvaa jatkuvasti ja tätä myöten datan siirto tulee yrityksille yhä kalliimmaksi. Pilvipalveluiden käyttäjien ja tarjoajien on otettava datan siirrosta syntyvä pullonkaula huomioon kaikilla järjestelmän tasoilla.

Suorituskyvyn ennalta-arvaamattomuus

Virtuaalilaitteiden keskusmuistin ja levymuistin kirjoitusnopeuden keskihajonnassa on suuria eroja ja tämä aiheuttaa useiden laitteiden klustereiden toimintaan häiriötä. Tällaista häiriötä voidaan vähentää palvelinarkkitehtuurisin ratkaisuin, kuten käyttämällä Flash-muistia kovalevyjen sijaan.

Bugit suuren mittakaavan hajautetuissa järjestelmissä

Suurten hajautettujen järjestelmien bugeja ei useinkaan saada replikoitua pienessä mittakaavassa, jolloin ne tulee kyetä ratkaisemaan tuotannossa. Virtuaalikoneiden käyttö on yksi keino ongelman ratkaisemiseksi.

Maineen jakaminen

Samassa pilvessä toimivien yritysten toiminnasta voi aiheutua hankaluuksia muille esimerkiksi IP osoitteiden mustalistauksen myötä.

2.2 Käsitteiden määrittelyä

*SaaS (*engl. Software-as-a-Service*)* viittaa useimmiten liiketoimintamalliin, jossa ohjelmisto tarjotaan palveluna, poiketen perinteisestä mallista, jossa käyttäjä suorittaa ohjelmaa omalla koneellaan.

*PaaS (*engl. Platform-as-a-Service*)* tarkoittaa ohjelmistoalustan tarjoamista palveluna. Alustan käyttäjä pystyy luomaan omaa ohjelmakoodiaan alustalle tietämättä mitään palvelinympäristöstä, jossa ohjelmakoodi suoritetaan.

*IaaS (*engl. Infrastructure-as-a-Service*)* tarkoittaa infrastruktuurin tarjoamista palveluna. IaaS-palvelun tarjoaja tarjoaa palvelimia asiakkaidensa käyttöön, mutta jättää näiden konfiguroinnin asiakkaan haltuun.

*FaaS (*engl. Function-as-a-Service*)* tarkoittaa tilattomien funktioiden tarjoamista palveluna. Kehittäjien luomat funktiot suorittavat asiakkaiden tekemiä kutsuja ja niiden ylläpidosta vastaa palveluntarjoaja.

*Virtualisointi (*engl. virtualization*).* Virtuaalikone on tietokoneen emulaatio tietokoneen sisällä. Virtualisoinnilla voidaan abstrahoida laitteisto sovellustasosta ja jakaa fyysisiä resursseja useiden eri sovellusten ja käyttäjien välillä.

Säiliöinti (engl. containerization). Säiliöt ovat yksinkertaistettuja versioita virtuaalikoneista. Ne ovat koneen sisälle luotuja ympäristöjä, jotka voivat pyörittää sovelluksia siten, etteivät sovellukset pääse käsiksi itse laitteeseen. Säiliöistä on myös käytetty nimitystä virtuaaliympäristö.

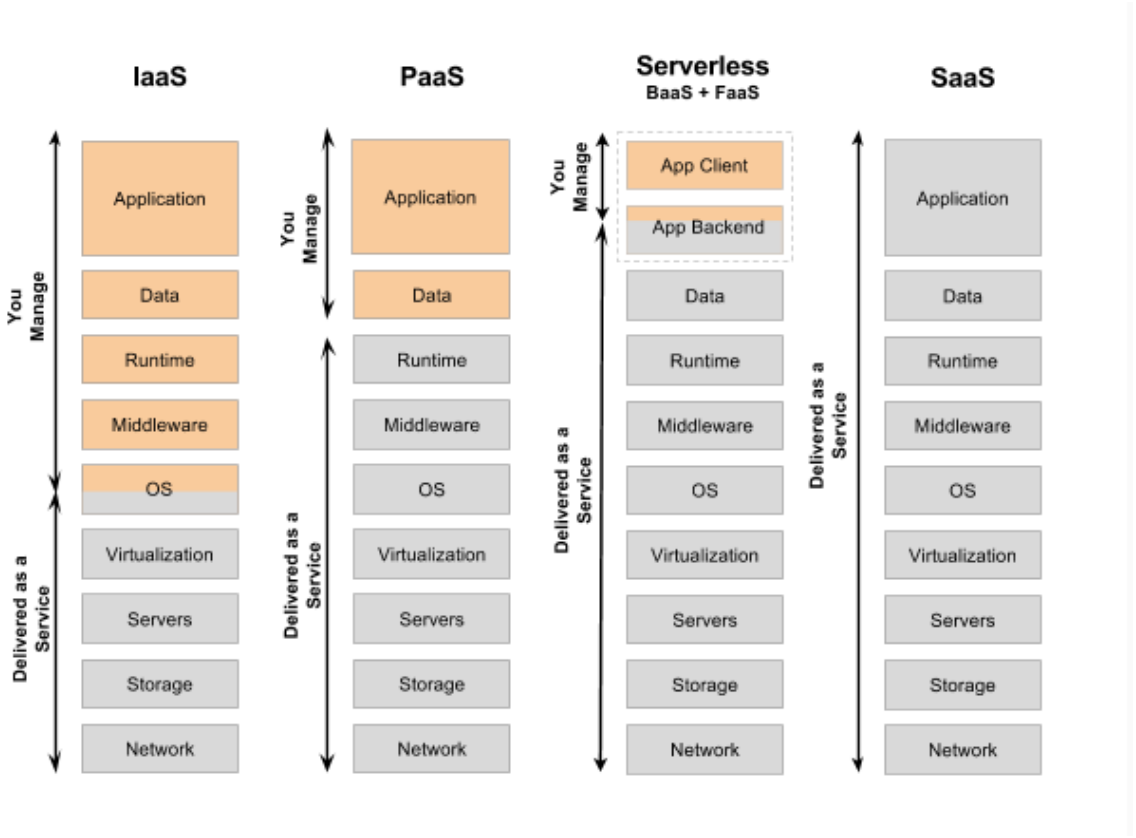
Micropalvelu (engl. microservice). Itsenäinen sovellus, joka tarjoaa tiettyä tarkoitusta palvelevia funktioita tarkasti määritellyn protokollan mukaan.

Tilattomuus (engl. statelessness). Tilattomuus viittaa tietoliikenneprotokollaan, jossa vastaanottaja ei säilytä istuntotietoja (engl. session information). Jokainen lähetetty paketti on siis aiempien lähetettyjen pakettien kontekstista riippumaton.

Tapahtumaohjattu (engl. event driven). Tapahtumaohjattu ohjelmointi on yleisin graafisten käyttöliittymien ohjelmointiparadigma, jossa ohjelman suorituksen kulun määrää käyttäjän tekemät tapahtumat, joita pääsilmukka jatkuvasti kuuntelee.

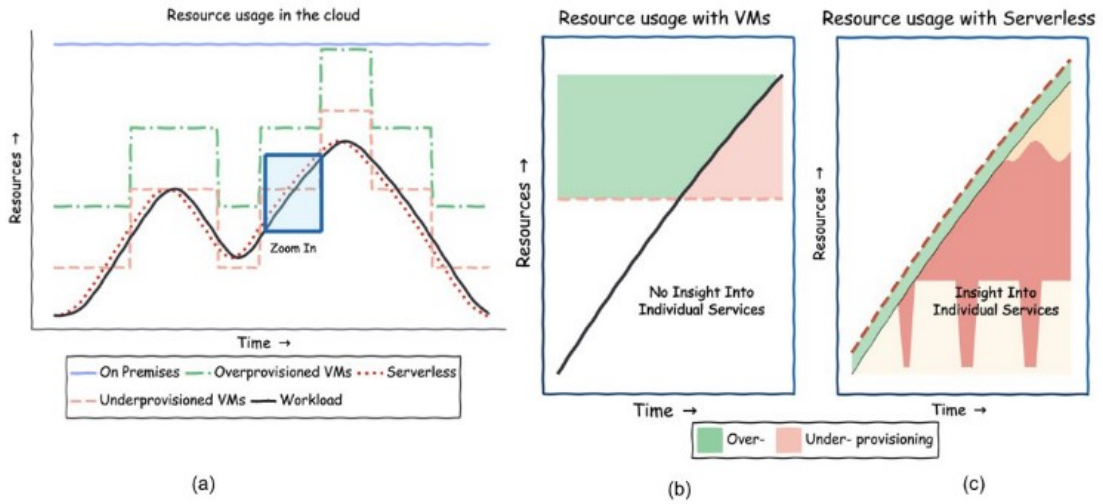
2.3 Serverless-paradigma

Palveliton laskenta (engl. Serverless Computing) on pilvilaskennan muoto, jossa abstraktiotaso kehittäjän näkökulmasta on viety vielä virtuaalikonetta pidemmälle. Virtuaalikoneen konfiguroinnin sijaan kehittäjän tarvitsee enää huolehtia ainoastaan funktioiden tuottamisesta pilvialustalle jättäen palvelinarkkitehtuurin epäolennaiseksi. Korkea abstraktiotaso tarkoittaa kuitenkin kompromisseja kehitettäviin sovelluksiin erityisesti yksityiskohtien osalta. Kehittäjät joutuvat hyväksymään sen tosiasian, että aivan kaikki ei ole tällaisessa mallissa mahdollista, vaan heidän täytyy toimia alustan tarjoaman kirjaston puitteissa. Kehittäjän hyöty tämänkaltaisessa mallissa on ohjelmiston ulos saamisen nopeus ja helppous.



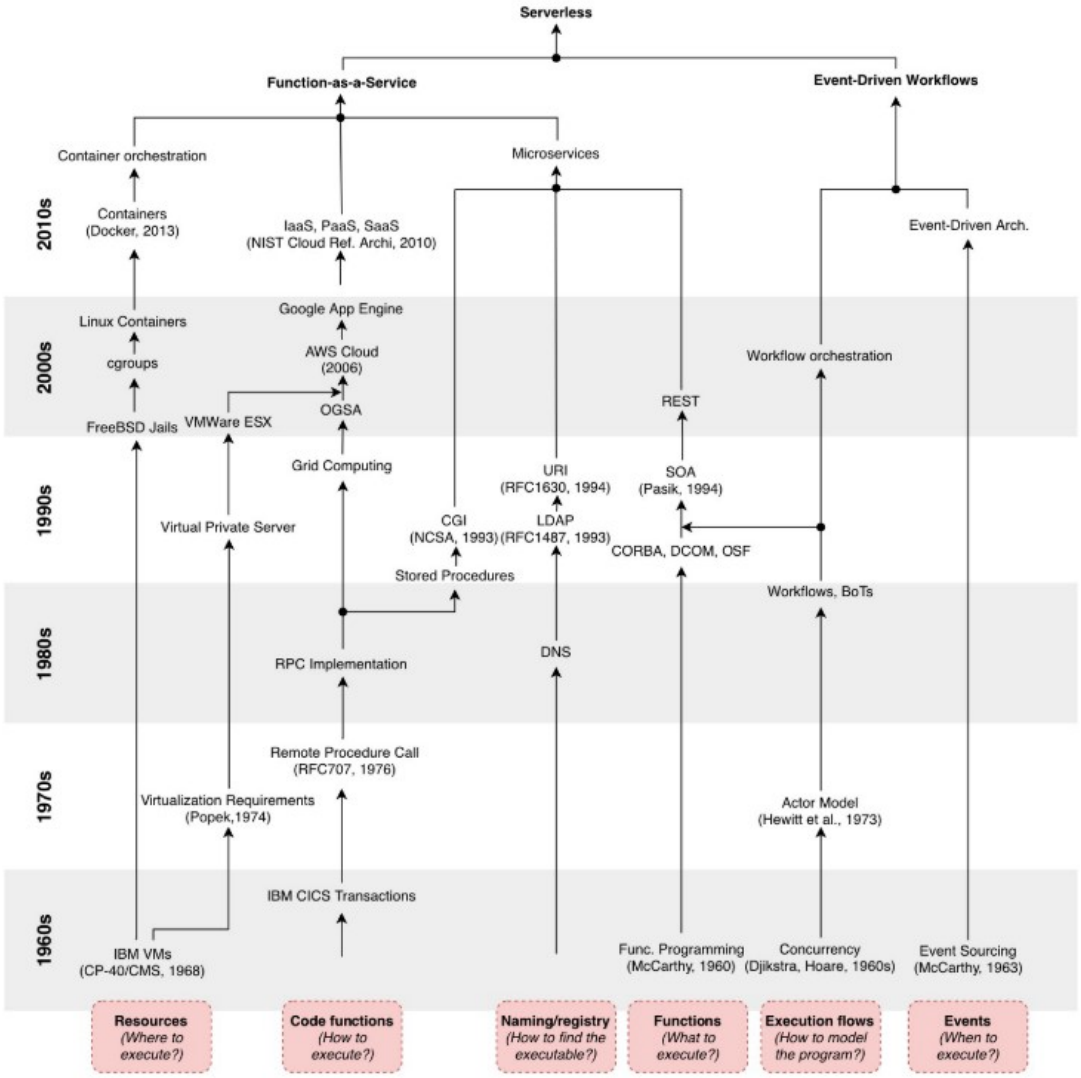
Kuvio 1. Kehittäjän vastuualueet pilvimalleissa (Wolf 2020)

Serverless-funktiot ajetaan ainoastaan pyydettäessä. Tämä tukee palveluntarjoajien liiketoimintamallia, jossa kehittäjät maksavat ainoastaan funktioihin tehdyistä kutsuista ja niiden ajamiseen käytetystä ajasta. Kutsujen määrän kasvaessa funktiosta ajetaan vain useampia versioita, jolloin palvelu skaalautuu kehittäjän näkökulmasta automaattisesti kysynnän mukaan parantaen vielä entisestään pilvilaskennan resurssitehokkuutta. (Savage 2018)



Kuvio 2. Serverlessin resurssitehokkuus (Van Eyk et al. 2018)

Serverless-malli perustuu vahvasti 2000-luvun alussa kehitettyyn säiliöintiteknologiaan. (Savage 2018) Joka kerta kun funktiota kutsutaan, järjestelmä luo uuden säiliön (engl. container), jossa se ajaa kutsutun funktion. Yksi merkittävimpiä ongelmia serverless-teknologian kehittämisessä tällä hetkellä on säiliön käynnistämiseen kuluvan ajan lyhentäminen, joka on noin sekunnin luokkaa. (Savage 2018) Käynnistysaikojen saamiseksi millisekunteihin, on tehtävä muutoksia palvelinten käyttämään käyttöjärjestelmään. Prosessorin välimuistin hyödyntäminen käynnistysaikojen lyhentämiseksi serverless-mallissa on myös yksi mahdollisuus, mutta tutkimukset siihen liittyen ovat vielä kesken. (Savage 2018)



Kuvio 3. Serverless-teknologian kehittyminen (Van Eyk et al. 2018)

Nykyään useiden kaupallisten toimijoiden rinnalle on alkanut viime aikoina ilmestyä myös avoimen lähdekoodin FaaS-alustoja, kuten Apache OpenWhisk, Fission ja Open Lambda. Kehittäjien avuksi on myös tullut eri alustojen rajapintojen standardisointiongelmia ratkaisevia viitekehyksiä, kuten Apex ja Serverless Framework, jotka tarjoavat alustasta riippumattomia malleja. (Van Eyk et al. 2018)

Serverless-sovellus voi olla joko ulkoisen tapahtuman kutsuma joukko funktioita, tai funktioista koostuva ohjelmisto. Serverlessiin liittyviä ominaisuuksia ovat tapahtumaohjautu-

vuus, tilattomuus, lyhyet suoritusajat, nopea skaalautuvuus, sekä kustannustehokkuus. Valitettavasti nämä ominaisuudet eivät kuitenkaan ole aivan kaikkeen käyttöön optimaalisia, vaan suosivat tietyntyyppisiä sovelluksia. (Feng et al. 2018)

Yleensä tilattomia funktiota hyödyntävä serverless-malli ei ole optimaalinen koneoppimis- tai muita suuria datamääriä käsitteleviä sovelluksia varten. (Savage 2018) (Van Eyk et al. 2018) Teknologia on kuitenkin vielä kehittyvässä vaiheessa ja asia saattaa hyvinkin muuttua parempaan suuntaan tulevaisuudessa. Ratkaisuna voisi olla erilaisten hienojakoisten ja datakeskeisten kehitysmallien syntyminen. Yksi lupaava tutkimuskohde aiheeseen liittyen on hajautetut lupaukset (engl. distributed promises) serverless-ympäristössä. (Van Eyk et al. 2018)

Suurimmat tahot, jotka vastaavat serverless-teknologian kehittämisestä ovat suuret pilvipalveluntarjoajat Google, Amazon, Microsoft ja IBM. Teknologian kehittymisen edellytyksenä on näiden suurien yritysten motivaatio, joka lähtökohtaisesti riippuu pitkälti tällaisten teknologioiden kysynnästä.

Joka tapauksessa serverless on äärimmäisen lupaava teknologia, joka liittyy olennaisesti nykyisenkaltaiseen kehitykseen, jossa ohjelmistokehityksestä tehdään koko ajan lähestyttävämpää ja jossa sovellusten tuotantoon saaminen on koko ajan saumattomampaa.

3 Koneoppiminen

Tässä kappaleessa perehdytään koneoppimiseen teknologiana. Koneoppiminen on tilastotieteen ja tietotekniikan väliin sijoittuva tieteenala, jonka tarkoituksena on kehittää kokemuksen avulla kehittyviä algoritmeja. Koneoppimisalgoritmit luovat matemaattisen mallin koulutusdatan avulla, jonka avulla ne kykenevät tekemään päätöksiä, joihin kyseisiä algoritmeja ei ole suoranaisesti ohjelmoitu. Koneoppimisalgoritmeja käytetään erityisesti tekemään päätöksiä tilanteissa, joissa mahdollisia tilanteita on niin paljon, että niitä kaikkia ei voida, tai ei kannata, etukäteen ohjelmoida. Algoritmit toimivat myös hyvin datan perusteella ennustamiseen ja koneoppimisesta käytetäänkin tietyissä yhteyksissä nimitystä ”ennustava analytiikka”.

Koneoppimista hyödynnetään tänä päivänä käytännön tekoälyratkaisuna esimerkiksi konenäön, puheen- ja tekstintunnistuksen parissa, sekä erityisesti erilaisissa dataintensiivisissä ongelmissa, kuten kuluttajapalveluissa tai vianmäärityksessä. (Jordan and Mitchell 2015)

3.1 Teoriaa

Koneoppimisessa on pohjimmiltaan kyse algoritmien suunnittelusta, jotka mahdollistavat tietokoneen oppimisen. Tietokoneen oppiminen ei ole tietoista kuten ihmisellä, vaan kyseessä on vain tilastollisten toistuvuuksien tai kaavamaisuuksien löytämistä datasta. (Oladipupo 2010) Merkittävä käsite koneoppimisessa on yleistäminen. Koneoppimisalgoritmin tarkoituksena on kyetä tekemään yleistyksiä, jotka yltävät sille annetun datan ulkopuolelle. (Domingos 2012) Lähtökohtaisesti algoritmin on kyettävä käsittelemään tilanne, jota koulutusdatassa ei tullut vastaan, tekemällä yleistyksiä koulutusdatan perusteella.

Koneoppimisprosessi koostuu datan valmistelusta, mallin koulutuksesta, mallin evaluoinnista ja päättelystä (tai ennustamisesta), eli mallin hyödyntämisestä koulutettuun tarkoitukseensa.

Data on koneoppimisalgoritmin voimanlähde, jonka avulla algoritmi kehittyy tarkemmaksi ja tarkemmaksi. Vaikka datan tapauksessa yleensä enempi on parempi, sen kanssa on kui-

tenkin oltava tarkkana, sillä sen laatu merkitsee myös paljon. Huonolaatuinen data, joka sisältää esimerkiksi paljon vääriä tai tyhjiä arvoja voi vinouttaa mallia merkittävästi. Tästä syystä data on kyettävä käsittelemään laadukkaaksi ennen sen koulutusta. Sen on oltava sopivassa muodossa ja se ei saa sisältää turhaa ”melua”, joka ei ole päättelyn kannalta olennaista vaan aiheuttaa virhettä. Data on myös pilkottava koulutukseen käytettävään dataan, sekä testaukseen käytettävään dataan, jota malli ei ole aiemmin vielä käsitellyt ja jonka perusteella mallin suorituskyky arvioidaan.

Erilaisia algoritmityyppejä on useita ja ne kaikki sopivat erilaisiin tarkoituksiin. Tämän tutkimuksen toteutuksessa käytettäväksi algoritmityypiksi valittiin lineaariregressio, jossa malli muodostaa matemaattisen kuvaajan numeerisen datan perusteella, jolle se tekee sovituksia. Sovituksen perusteella malli päättelee sen, kuinka hyvin päättelydata sopii kuvaajaan. Malli tekee datalle aina myös jonkinlaisen pisteytyksen, jonka perusteella pääteltävä arvo määräytyy, kuten esimerkiksi käytetyn auton hinta auton teknisten tietojen perusteella.

3.1.1 Oppimistyyppejä

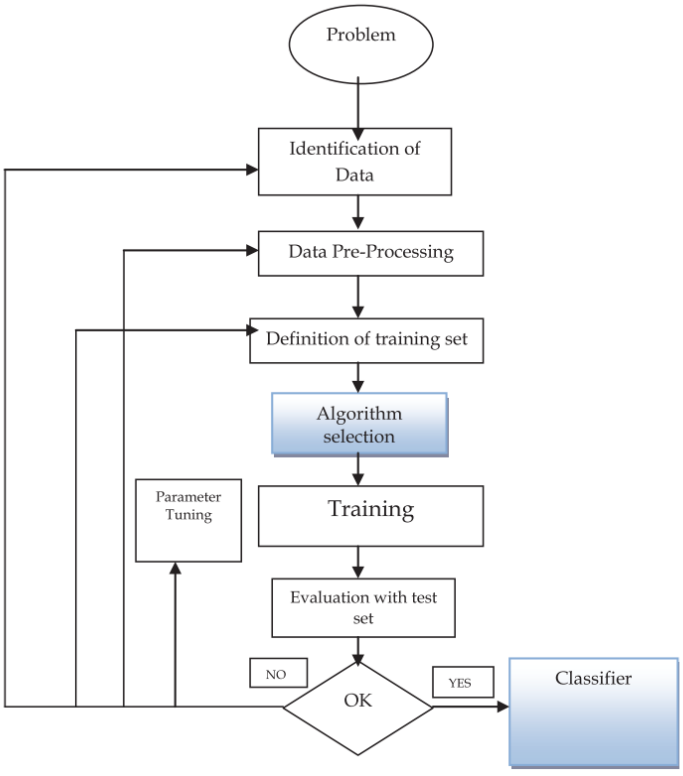
Koneoppimisalgoritmeja kategorisoidaan niiden halutun lopputuloksen mukaan. Yleisimpiä tekniikoita toteuttaa koneoppimista ovat valvottu, valvomaton, sekä vahvistusoppiminen. (Oladipupo 2010):

Valvottu oppiminen

Valvottu oppiminen on yleisesti käytetty algoritmityyppi luokittelutehtävien yhteydessä, sillä luokittelu perustuu useimmiten ihmisen luomaan luokittelujärjestelmään. Yleisesti valvottu oppiminen sopii hyvin ongelmiin, joissa luokittelun päättely on olennaista ja helposti tehtävissä. (Oladipupo 2010) Valvotussa oppimisessa algoritmi luo funktion, joka kartoittaa syötteet halutuille tulosteille. Valvottu oppiminen on yleisimmin käytetty oppimistyyppi neuroverkkojen ja päätöspuiden (engl. decision tree) kouluttamiseen, sillä tämänkaltaisten mallien toiminta riippuu niille annettujen luokittelujen kautta saadusta informaatiosta. (Oladipupo 2010)

Sovellettaessa valvottua oppimista johonkin oikean elämän ongelmaan täytyy ensin valmistella data valitsemalla eniten informaatiota tarjoavat ominaisuudet eri vaihtoehdoista. Mikäli näitä ei pystytä valitsemaan on mahdollista myös mitata kaikkia ominaisuuksia (engl. brute-force) ja toivoa että tärkeimmät ominaisuudet tulisivat näin esille. Useimmiten tällä tavalla valmisteltu datasetti ei kuitenkaan sovellu suoraan sellaisenaan käyttöön, vaan vaatii vielä merkittävästi ylimääräistä käsittelyä virheiden karsimiseksi. (Oladipupo 2010; Zhang, Zhang, and Yang 2003)

Seuraava vaihe on datan esikäsittely, johon voidaan soveltaa useita erilaisia tekniikoita, kuten tapausvalinta (engl. instance selection) ja ominaisuuksien osajoukkovalinta (engl. feature subset selection). Tarkoituksena on tehostaa datan käsittelyä yksinkertaistamalla sitä esimerkiksi hävittämällä virheellisiä arvoja tai vähentämällä mitattavien ominaisuuksien määrää. (Oladipupo 2010) Datan käsittelyn tarkoituksena valvotussa oppimisessa on saada aikaiseksi koulutussetti, jonka syötteet on luokiteltu koulutusta varten. Tämän jälkeen valitaan kyseiseen tilanteeseen sopiva algoritmi, jolle edellä käsitelty setti koulutetaan.



Kuvio 4. Valvottu koneoppimisprosessi (Oladipupo 2010)

Valvomaton oppiminen

Valvomattomassa oppimisessa tarkoituksena on opettaa algoritmi tekemään jotakin, mitä ei ole määritelty sille. (Oladipupo 2010) Tämä voidaan toteuttaa kahdella eri tavalla, joista ensimmäisessä algoritmille annetaan luokittelujen sijaan vain jonkinlainen palkitsemissysteemi, joka ilmaisee onnistumisen tason. Tämän kaltainen toteutus sopii hyvin reaalimaailman tilanteisiin, jossa algoritmia ikään kuin palkitaan tai rankaistaan sen suoritusten perusteella. (Oladipupo 2010) Toinen menetelmä toteuttaa valvomatonta oppimista on klusterointi, jonka tarkoituksena on löytää datasta samankaltaisuuksia ja muodostaa näiden perusteella joukkoja. Oletuksena tässä on se, että muodostetut joukot vastaavat yleensä melko hyvin ihmisen intuitiivisesti muodostamaa luokittelua. (Oladipupo 2010)

Vahvistusoppiminen

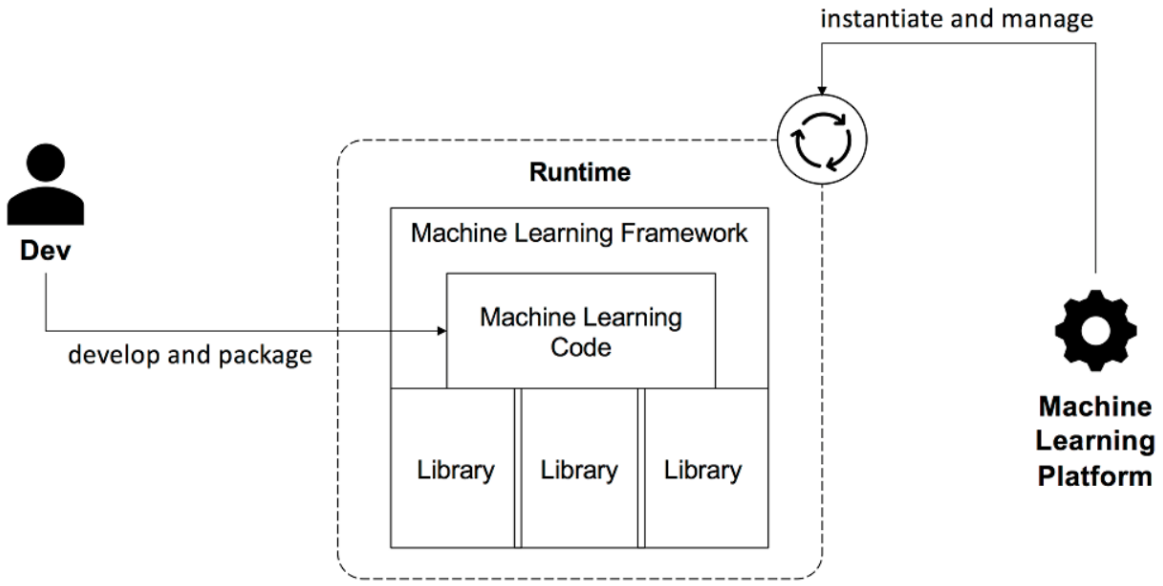
Vahvistusoppimisessa koulutukseen käytettävä data on valvottuun ja valvomattomaan oppimiseen käytetyn datan välimalli, jossa oikeiden esimerkkien kouluttamisen sijaan data antaa vain viitteitä siihen, että onko mallin suorittama toimenpide oikein vai väärin. (Jordan and Mitchell 2015) Syötteiden sarjoissa yksittäisten toimenpiteiden palkitsemisen sijaan palkitaan kokonaisia sarjoja, jonka perusteella malli pyrkii mahdollisimman palkitseviin suorituksiin. Tarkoituksena mallille on useimmiten löytää oikeanlainen toimintamalli dynaamisessa ja arvaamattomassa ympäristössä lukuisten kokeilujen avulla. Kaikista koneoppimisen oppimistyypeistä vahvistusoppiminen on lähimpänä ihmisten ja eläinten oppimista. Monet vahvistusoppimisen algoritmeista ovatkin saaneet inspiraatiota biologisista systeemeistä. (Sutton and Barto 2015)

4 Serverless koneoppimisen näkökulmasta

Serverless-arkkitehtuuriin soveltuvat sovellukset ovat useimmiten tilattomia, tapahtumaohjattuja ja suoritusajaltaan nopeita. Sovellusten skaala on kuitenkin kasvussa erilaisten tietovarastojen ja palveluiden yhdistelmien tarjoaman pysyvyyden ansiosta. Esimerkiksi pidempiä suoritusaikoja voidaan saada aikaiseksi ketjuttamalla useita serverless-instansseja erillisen tietokannan säilyttäessä tiedon muutoksista. (Feng et al. 2018)

Koneoppimisen tapauksessa serverless-teknologia on tuottanut tähän mennessä vaihtelevia tuloksia. (Feng et al. 2018) Se on soveltunut hyvin ennustamiseen pilviympäristössä, mutta ei kovin hyvin mallien kouluttamiseen, joka vaatii paljon muistia ja laskentatehoa. (Feng et al. 2018) Hajautetun laskennan hyödyntäminen lienee parhaimpia keinoja parantaa mallien kouluttamisen tehokkuutta, mutta asiasta on tehty vasta varsin vähän tutkimusta. (Feng et al. 2018) Tässä kappaleessa perehdytään koneoppimissovellusten mahdollisuuksiin ja haasteisiin serverless-ympäristössä.

Serverless teknologian suurimpia etuja koneoppimissovelluksille on se, miten se vähentää kehittäjille normaalisti kuuluvaa palvelintason operatiivista työtä.

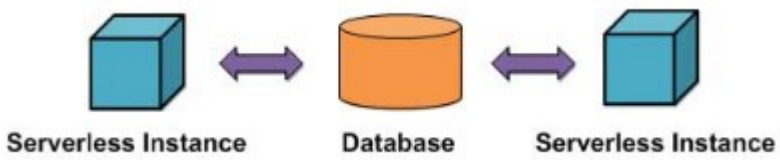


Kuvio 5. Serverless ML kehitysmalli (Osipov 2020)

4.1 Haasteet verkon koulutuksessa

Verkon kouluttamisen suurimpana haasteena on se, että sitä ei voida toteuttaa yhdellä instanssilla johtuen serverless-funktioiden muisti- ja suoritusaikarajoitteista. Tällöin eri instanssit on toteutettava käyttäen hyödyksi rinnakkaisuutta (engl. parallelization).

Suurin ero tyypillisten hajautettujen systeemien ja rinnakkaisten serverless-instanssien välillä tulee jälkimmäisten tilattomasta luonteesta ja rajallisuudesta ajan suhteen. (Feng et al. 2018) Tällä hetkellä serverless-instanssien välinen tiedonsiirto ei ole vielä mahdollista, joten näiden avuksi tarvitaan erillinen tietovarasto säilyttämään eri instansseja yhdistävän tilan. Rinnakkaisuus tuo kuitenkin tässä tapauksessa mukanaan ylimääräisiä kustannuksia johtuen muun muassa instanssien ja tietovaraston välillä siirtyvän datan viiveestä ja datan lataamisesta useita kertoja. (Feng et al. 2018)



Kuvio 6. Serverless-instanssit (Feng et al. 2018)

Suorituskyvyn tehostamiseksi koneoppimisverkon kouluttamisessa on tärkeää kyetä minimoimaan serverless-instanssien välillä tapahtuva datan siirto. (Feng et al. 2018) Yksi keino parantaa instanssien välistä kommunikaatiota olisi se, että ne voisivat sijaita kollektiivisesti samassa fyysisessä lokaatiossa ja jakaa resursseja.

4.2 Mallin koulutus

Koulutuksen ideana on muokata kyseisen mallin parametreja annetun datan perusteella siten, että sen päättelytulokset vastaavat koulutukseen käytettyä dataa. (Feng et al. 2018) Usein algoritmit laskevat datan perusteella erilaisia liukuarvoja, eli gradientteja, joiden avulla mallin parametrit päivitetään.

Rinnakkaisissa systeemeissä valittu datasetti jaetaan osiin, jotka jokainen koulutetaan erikseen aina yhdelle neuroverkkoinstanssille. Itsenäisiä instansseja pyörittäviä koneita kutsutaan työskentelijöiksi (engl. worker). (Feng et al. 2018) Jokaisen työskentelijän laskemat gradientit viedään koulutuksen jälkeen erilliselle parametrikäsittelijälle (engl. parameter server), joka vastaa verkon parametrien päivityksestä. Serverless-mallissa kaikki edellä mainitut koneet toteutetaan serverless-instansseina.

Rinnakkaisissa systeemeissä verkon parametrien päivitys voidaan toteuttaa joko synkronoidusti, odottamalla kaikkien gradienttien valmistumista, tai asynkronisesti yksi kerrallaan. (Feng et al. 2018)

Yksi keino vähentää rinnakkaisten systeemien viivettä verkon koulutuksessa on muodostaa useita parametrikäsittelijöitä useammalle tasolle, jolloin parametrikäsittelijät voivat tehdä verkon parametrien päivityksen rinnastetusti. (Feng et al. 2018) Mahdollisten datansiirrossa tapahtuvien paketinmenetysten ja uudelleenlähetysten vuoksi rakenne on hyvin epävarma vielä käytännön tasolla yli kahden käsittelijätason systeemeille.

Toinen, varsin tehokkaaksi osoittautunut, keino mallien koulutuksen tehostamiseksi serverless-mallissa on hyperparametrien rinnakkainen viritys. Johtuen kyseisen menetelmän rajoitteista, se soveltuu kuitenkin ainoastaan kevyiden mallien käsittelyyn. Serverless-mallin rinnakkainen hyperparametrietsintä on osoittautunut huomattavasti PC-versiota paremmin skaalautuvaksi. (Feng et al. 2018)

Koneoppimissovellusten kehittäjät joutuvat toteuttamaan useita erilaisia tehtäviä mallin koulutuksessa. Tavallinen työnkulku koostuu useimmiten datan esikäsittelystä, mallin koulutuksesta sekä hyperparametrien virittämisestä. Perinteinen virtuaalikoneiden käyttö näiden tehtävien toteuttamiseen sisältää kuitenkin omat ongelmansa. Virtuaalikoneiden resursseja on hallittava hyvin yksityiskohtaisesti, joka lisää kehittäjän työtaakkaa huomattavasti. Resurssien ylimitoittaminen eri tehtäville on myös varsin tavanomaista tämänkaltaiselle työnkululle. (Carreira et al. 2018)

Serverless tarjoaa ratkaisuja resurssien jakamiseen liittyviin ongelmiin, kuten valmiita mekanismeja ylimitoittamisen välttämiseksi. Erityisesti serverlessin skaalautuvuuskyky laskennan ja levytilan suhteen nostaa sen houkuttelevuutta koneoppimismallien alustaksi. (Carreira et al. 2018)

Serverlessin rakenteelliset periaatteet aiheuttavat kuitenkin erilaisia yhteensopivuusongelmia olemassa olevien koneoppimiskehysten kanssa, mistä seuraa omia haasteitaan, kuten: (Carreira et al. 2018)

- Funktioiden muistikapasiteetti on hyvin rajallinen.
- Funktioiden kaistan nopeudet ovat huomattavasti virtuaalikoneita alhaisempia.
- Funktioiden suoritusajat ovat lyhyitä ja käynnistysajat suhteellisen pitkiä. Sovelluksen on myös kyettävä sietämään näiden ennalta-arvaamattomuutta.
- Funktiot eivät kykene kommunikoimaan keskenään ja tarvitsevat nopean välikäden. Datavarastot ovat tällä hetkellä vielä varsin hitaita funktioiden tarpeisiin.

Hyvän serverless-koneoppimissovelluksen vaatimukset E2E työnkulkuihin (Carreira et al. 2018):

- Pyhton frontend, joka tarjoaisi rajapinnan kaikille koneoppimissovellusten eri vaiheille.
- Tilallinen backend, joka ohjaa lambdafunktioiden kutsuja, sekä ylläpitää eri työvaiheiden tilannetta.
- Worker runtime, joka tarjoaa rajapinnan iteraattorin datasettien kouluttamiseen, sekä hajautettuun datavarastoon.
- Hajautettu datavarasto, joka on alustettuna virtuaalikoneelle alhaisen viiveen takaamiseksi.

4.3 Päättely

Päättelyn toteutus täysin serverlessinä voi olla hyvin kustannustehokasta. Rajoitetulla suoritusajalla, sekä oikean kokoisella aineistolla serverless ratkaisu on todettu olevan edullisempi vaihtoehto jatkuvasti käynnissä olevaan palveluun verrattuna. (Seiler 2019)

Serverless funktiot eivät kuitenkaan ole osoittautuneet optimaaliseksi syväoppimismallien tapauksessa. Syväoppimismallit ovat kooltaan sen verran suuria, että funktioiden rajoitukset tulevat nopeasti vastaan. Funktiot eivät myöskään tue GPU-prosessointia, joka vähentäisi viivettä merkittävästi. Kuten mallien koulutuksessa, niin myös päättelyn suhteen funktioiden rinnakkainen toiminta tehostaisi suorituskykyä, johon ne eivät kuitenkaan kykene. (Kaiser 2020)

5 Toteutus

Tässä kappaleessa käydään läpi pilvialustalla toimivan koneoppimissovelluksen vaatimuksia ja vaihtoehtoja toteutukselle. Lisäksi dokumentoidaan sovelluksen suunnittelua ja toteutusta, sekä kuvataan lyhyesti toteutuksen aikana käytetyt työkalut.

Pilvialustaksi toteutukselle valikoitui Microsoft Azure, sillä sen käyttöliittymä oli suurimpien palveluntarjoajien alustoista entuudestaan tutuin. Pohdinnassa olleet vaihtoehdot olivat Google Cloud Platform ja Amazon Web Services. Pienten palveluntarjoajien, tai avoimen lähdekoodin alustoja ei otettu vaihtoehdoksi, sillä niiden tarjoamat ominaisuudet olisivat saattaneet osoittautua toteutusvaiheessa puutteellisiksi.

Lähtökohtana sovellukselle oli valmiilla datasetillä koulutettava koneoppimismalli, jonka koulutus ja päättely ovat toteutettu pilvessä. Tähän hyödynnettiin Microsoftin tarjoamia työkaluja Azure-pilvialustalla. Tietämys toteutusta varten saatiin Azuren dokumentaatiosta (“Azure Documentation | Microsoft Docs” n.d.). Koska toteutuksen tavoitteena oli kartoittaa tietämystä aiheeseen ja tutkimuskysymyksien vastauksiin, ei sovelluksen datasetillä tai koneoppimissovelluksen ratkaisemalla ongelmalla ollut suurta merkitystä. Kriteerinä oli saada toteutettua toimiva sovellus, jonka käyttämää koneoppimismallia oli mahdollista testata onnistuneesti syötteillä saaden oikeanlaisia vastauksia.

Toteutuksen tarkoituksena oli kartuttaa tietämystä koneoppimissovellusten kehittämisestä pilvialustoilla saaden näin ymmärrystä eri toteutusvaihtoehdoista ja serverless-ratkaisun hyödyistä ja haasteista virtuaalikoneella pyörivään toteutukseen verrattuna kehittäjän näkökulmasta. Toteutuksen tuloksena syntyi dedikoitua laskentakohdetta hyödyntävä, täysin kooditon ratkaisu, sekä serverless funktioiden avulla julkaistu koodattu versio. Kummassakin tapauksessa kyseessä on lineearista regressioalgoritmia käyttävä malli.

5.1 Käytetyt työkalut

Useimmat tässä kappaleessa esitellyistä työkaluista ovat Azuren omia sovelluksia ja palveluita, joita luodaan ja käytetään työtilassa. Tämän lisäksi toteutukseen tarvittiin ohjelmointiympäristö tarvittavine kirjastoineen, sekä testauksessa hyödynnetty API Client -sovellus.

5.1.1 Azure Functions

Azure Function on laskentapalvelu, joka mahdollistaa koodin ajamisen ilman palvelimen määritystä tai hallintaa. Koodi ajetaan vain käskystä ja se skaalautuu automaattisesti. Palvelun hinta määräytyy käytetyn laskenta-ajan mukaan, eikä tyhjäkäynnistä veloiteta. Koodi itsessään voi olla käytännössä mitä tahansa sovellus- tai taustajärjestelmäkoodia. Funktioita voidaan alustaa Azuren käyttöliittymän konsolista, sekä IDE, SDK ja komentorivityökalujen avulla. Azuren konsolista löytyy koodieditori koodin muokkausta ja testausta varten.

Funktio laukaistaan, eli käynnistetään, aina tietyntyyppisen tapahtuman avulla. Tässä tapauksessa käytetty laukaisin oli HTTP pyyntö. Muita mahdollisia laukaisimia voivat olla esimerkiksi reagointi datassa tapahtuviin muutoksiin tai viesteihin, sekä ajoitettu käynnistyminen. (“Azure Functions Overview | Microsoft Docs” n.d.)

5.1.2 Azure Machine Learning

Azure Machine Learning (AML) on pilvipohjainen ympäristö, jonka avulla voidaan kouluttaa, toteuttaa, automatisoida, hallita ja seurata itsensä tai muiden tekemiä koneoppimismalleja. Alustalla voi toteuttaa Python ja R-kielten lisäksi malleja graafisen työkalun avulla. Alusta toimii hyvin yhteen yleisimpien koneoppimiskehysten kuten PyTorch, TensorFlow, Scikit-learn ja Ray RLlib kanssa.

AML:n työkalut mallien toteuttamiseen ovat:

- AML Designer. Graafinen työkalu, jolla voi suunnitella ja toteuttaa kokeiluja, joiden perusteella luoda putkia.
- Jupyter Notebooks. Alusta, jonka avulla voi koodata malleja Python ja R kielillä suoraan koodieditorissa.
- Automated ML. Palvelu, jossa sovellus kouluttaa ja etsii sopivimman mallin datasetin perusteella automaattisesti.

5.1.3 Muita toteutuksessa käytettyjä työkaluja

- Visual Studio / Visual Studio Code – IDE

- Microsoft ML.Net – Ohjelmointikirjasto
- Azure Functions Core Tools
- Postman – API Client
- Azure Blob Storage – Pilvidatavarasto
- Azure Cosmos DB – Pilvitietokanta

5.2 Perusteet toteutukselle

Koneoppimissovelluksen toteutuksen pilvessä voi tehdä useilla eri tavoilla riippuen sovelluksen käyttötarkoituksesta. Erilaiset valmiit palvelut eri alustoilla mahdollistavat nykyään jopa täysin koodittomien ratkaisujen toteuttamisen, joissa koko ratkaisun voi hoitaa datan prosessoinnista aina mallin koulutukseen ja tuotantoon vientiin asti kirjoittamatta riviäkään varsinaista koodia. Aina on myös vaihtoehtona toteuttaa eri työvaiheet puhtaasti koodaten, mikä tarjoaa kehittäjälle vapauksia valmiisiin ratkaisuihin nähden nostaen kuitenkin työtaakkaa huomattavasti.

Kehitettävän sovelluksen tarkoituksena on yksinkertaisesti kyetä tekemään päättelyitä sille annetun datan perusteella. Päättelystä vastaa koneoppimismalli, joka on koulutettu koulutusdatan avulla. Malli sijoitetaan pilveen, jossa se voi toteuttaa päättelyä sille tehtävien kutsujen laukaisemana.

Koneoppimissovelluksen suunnittelun keskiössä on ongelma, jota sovelluksella halutaan kyetä ratkaisemaan. Toteutuksessa käytetty data (“UCI Machine Learning Repository: Automobile Data Set” n.d.) sisälsi autojen teknisiä tietoja, joiden avulla oli tarkoitus kyetä ennustamaan käytetyn auton hinta. Datasetti on tuotantokäyttöön aivan liian vanha ja pieni, mutta tässä tutkimuksessa toteutetuissa ratkaisuissa datalla ei kuitenkaan ole merkitystä toteutuksen ollessa täysin kokeellinen ja konstruktion tuottaessa ainoastaan tietämystä tutkimuskysymyksiä varten.

Datasetit harvoin ovat suoraan valmiita koneoppimismallin kouluttamiseen. Osa sarakkeista voi olla tarpeettomia tai jopa suoranaisesti haitallisia lopputuloksen tarkkuutta ajatellen. Li-

säksi tyhjiä tai virheellisiä arvoja sisältävät rivit useimmiten vain aiheuttavat virhettä tarkkuuteen. Ennen koulutusta on tärkeää valmistella datasetit siivoamalla virhettä aiheuttavat sarakkeet ja rivit, sekä muuntaa jollakin keinolla ei-numeeristen sarakkeiden arvot numeerisiksi. Data jaetaan vielä lopuksi koulutusdataan ja testausdataan, jonka avulla mallin koulutuksen onnistuminen, eli mallin tarkkuus, voidaan arvioida.

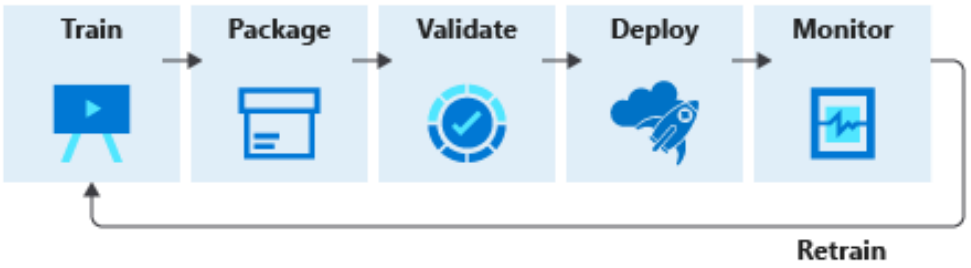
Valmisteltu koulutusdata voidaan kouluttaa mallille. Koneoppimismallien kouluttamiseen on kehitetty useita erilaisia algoritmeja, jotka sopivat kukin hieman erilaisiin käyttötarkoituksiin ja erilaisille dataseteille. Tässä toteutuksessa käytetyille dataseteille valittu algoritmi oli lineaarinen regressio, joka sopii erityisesti numeraalisten arvojen, kuten tässä tapauksessa hinnan, ennustamiseen.

Onnistuneen koulutuksen jälkeen mallille ajetaan vielä testausdata, jonka arvojen mukaan malli voidaan hylätä tai hyväksyä. Mikäli tulos ei ole riittävän tarkka, voidaan mallille suorittaa hyperparametrien muokkausta tai yrittää parantaa datan laatua muuten.

Koulutettu malli sijoitetaan pilveen ajettavaksi. Sijoituksen voi myös toteuttaa usein eri tavoin riippuen hieman mallin käyttötarkoituksesta ja raskaudesta, sekä päättelyiden määrästä. Tämän tutkimuksen toteutuksessa käytetyt vaihtoehdot koneoppimismallin ajamiseen pilvessä ovat virtuaalikone sekä serverless-funktio. Serverless-funktion tapauksessa malli tallennetaan datavarastoon, josta ajettava funktio noutaa sen.

Pilveen sijoitettua ja koulutettua koneoppimismallia voidaan hyödyntää erilaisiin koneoppimissovelluksiin erilaisin keinoin, joista yksi on tehdä päättelyitä rajapintakutsuilla. Azure Machine Learning –palvelu toimii ikään kuin itsenäisenä koneoppimissovelluksena, jonka sisällä kaikki toiminta koulutuksesta päättelyyn tapahtuu graafisessa ympäristössä.

5.2.1 Azure Machine Learning työnkulku



Kuvio 7. AML työnkulku (“Azure Machine Learning Documentation | Microsoft Docs” n.d.)

Azure Machine Learning noudattaa hyvin selkeää ja helposti lähestyttävää työnkulkua mallien koulutuksesta sijoitukseen. Palvelu ei kuitenkaan mahdollista merkittävää poikkeamista sen tarjoamista toteutustavoista, joten sen avulla kehitetyn mallin toteuttamisessa on noudatettava ainakin toistaiseksi tiettyä kaavaa. Koneoppimismallin toteutuksen työnkulku AML:ssä sisältää seuraavat vaiheet:

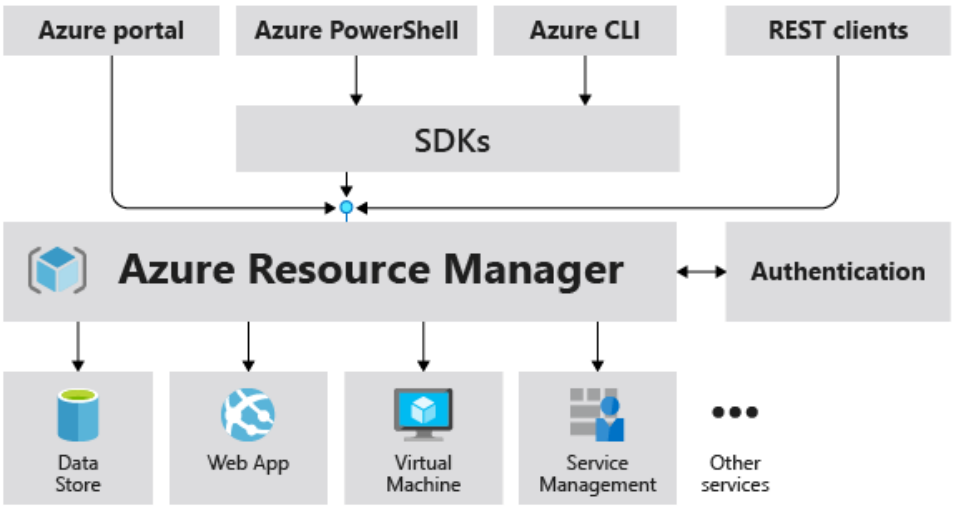
1. Koulutus
    a. Koneoppimisskriptin kehittäminen Pythonilla, R:llä tai designerilla.
    b. Laskentakohteen eli laskentaresurssin luonti.
    c. Skriptin ajaminen laskentakohteella. Koulutus synnyttää ajoja (engl. run) työtilaan (engl. workspace), jotka ryhmitellään kokeilujen (engl. experiments) alle.
2. Paketointi – Sopivan ajon löydyttyä malli rekisteröidään mallirekisteriin (engl. model registry).
3. Validointi – Selvitetään ajon onnistuminen toteuttamalla kokeilun kysely. Mikäli mittaristo ei osoita toivottua arvoa, niin palataan takaisin kohtaan 1 ja muokataan skriptiä.
4. Sijoitus (engl. deployment) – Kehitetään mallia käyttävä pisteytysskripti ja viedään malli web palveluna Azureen.
5. Seuranta - Seurataan mallissa tapahtuvia muutoksia koulutusdatan ja päättelydatan välillä. Mikäli muutosta tapahtuu huomattavasti, on syytä palata taas kohtaan 1.

5.3 Kooditon toteutus AML Designerin avulla

Koneoppimismallin koulutus- ja päättelyputkien luonti AML Studio Designerin avulla sisälsi seuraavat työvaiheet:

Uuden resurssiryhmän luonti

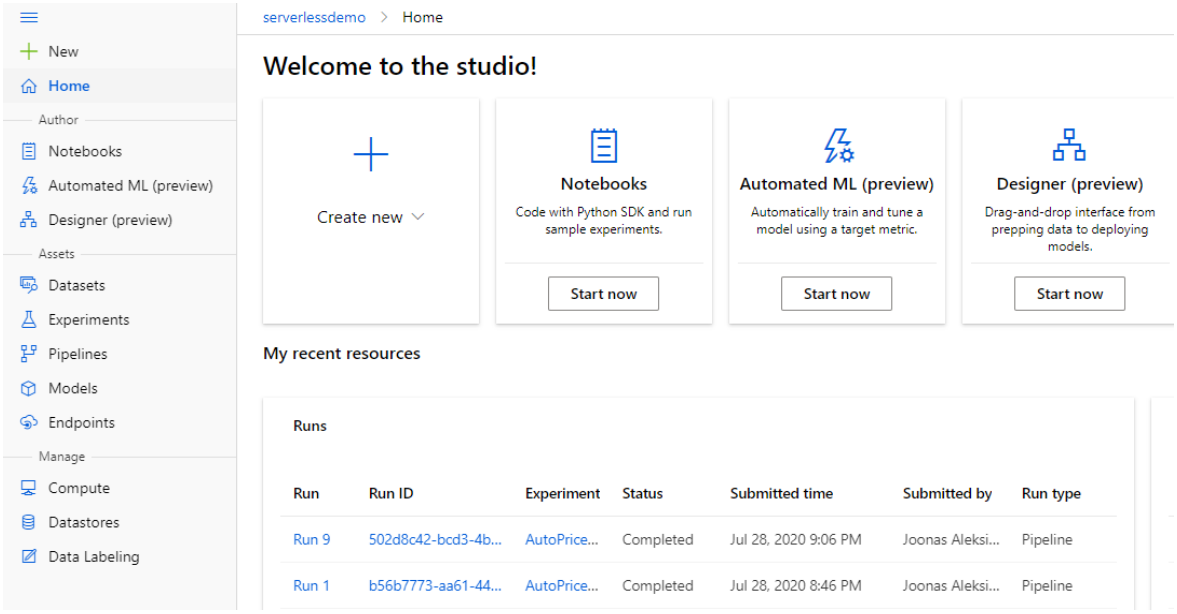
Ensimmäisenä Azureen täytyi luoda uusi resurssiryhmä, joka niputtaa kaikki käytetyt resurssit helpommin hallittavaksi kokonaisuudeksi. Resursseja ovat kaikki Azuren työkalut ja palvelut kuten AML, datavarastot ja virtuaalikoneet. Resurssiryhmiä hallinnoidaan Resource Managerin kautta, joka luo hallintakerroksen käytettävien resurssien organisoimiseksi ja suojaamiseksi. Kaikkien resurssiryhmän alla olevien resurssien tulee jakaa sama elinkaari, eli ne viedään ja poistetaan tuotannosta samanaikaisesti. Uusi resurssiryhmä luotiin Azuren portaalin käyttöliittymän avulla.



Kuvio 8. Azure Resource Manager (“Azure Documentation | Microsoft Docs” n.d.)

Uuden AML-työtilan luonti

Resurssiryhmän alle luotiin uusi AML-työtila (engl. workspace). Kaikki AML:n luomat ja käyttämät resurssit tallentuvat kyseisen työtilan alle. Työtilasta voidaan laukaista uusi instanssi AML Studiosta.



Kuvio 9. AML Studion aloitusnäkymä

Putken luonti

Mallin luonti ja koulutus toteutettiin käyttäen AML Studion Designer työkalua, joka tarjoaa hyvin helpon lähestymistavan mallien toteuttamiseen ilman koodia. Graafisessa käyttöliittymässä luotiin uusi putki (engl. pipeline), jossa varsinaiset mallin toteuttamiseen liittyvät toimenpiteet määritellään. Putken avulla mallin toteutuksen prosessi saatiin automatisoitua datasetin valmistelusta koulutettuun malliin asti.

Putkien luonti onnistuu designerissa täysin graafisesti pudottelemalla eri tehtäviä kehykseen ja yhdistelemällä näiden sisään- ja ulostuloja saaden aikaan automatisoidun prosessin datan käsittelyyn ja mallin kouluttamiseen.

Kuvio 10. Mallin koulutusputki designerissa

Laskentakohteen määrittely

AML vaatii ainakin toistaiseksi erikseen määritellyn oletuslaskentaresurssin. Käytännössä tämä tarkoittaa virtuaalikoneen määrittelyä, jonka konfiguroinnista kehittäjän ei tarvitse kuitenkaan huolehtia. Oletuslaskentakohde määriteltiin AML Studion käyttöliittymän kautta.

Datan tuonti

Datasetti valittiin designerin oppimistarkoitukseen tehtyjen valmiiden datasettien valikoimasta. Datan voi myös tuoda mistä tahansa ja yleisin vaihtoehto tälle on jokin .csv -muotoinen, tarpeeksi paljon rivejä sisältävä tiedosto.

Datan valmistelu

Putken ensimmäisissä vaiheissa toteutettiin datan valmistelu koulutusta varten. Datasetistä poistettiin ensin sarake, joka sisälsi suurelta osin tyhjiä arvoja. Seuraavaksi datasetistä poistettiin kaikki loput tyhjiä arvoja sisältävät rivit.

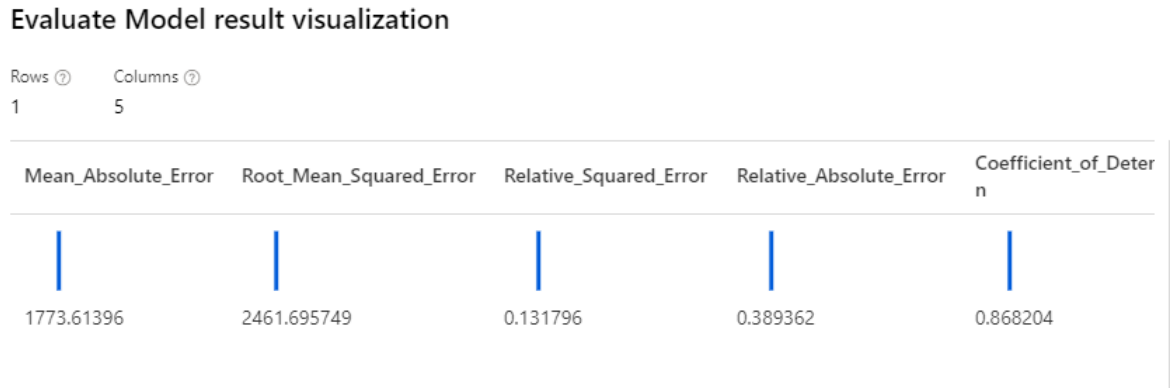
Mallin koulutus

Ennen mallin koulutusta datasetti jaettiin kahteen osaan, joista suurempi (70 %) käytettiin mallin kouluttamiseen ja loput (30 %) kyseisen koulutusajon testaukseen. Tämän jälkeen koulutusdata ajettiin regressioalgoritmille, joka koulutti mallin.

Mallin evaluointi

Mallin koulutuksen jälkeen toteutettiin vielä mallin evaluointi testidatalla. Putkeen tehtiin koulutuksen jälkeen vielä testausdatan pisteytys, sekä evaluointimoduuli, joka arvostelee koulutuksen onnistumisen testausdatan perusteella. Evaluointimoduuli tarjoaa käyttäjälle seuraavia arvoja:

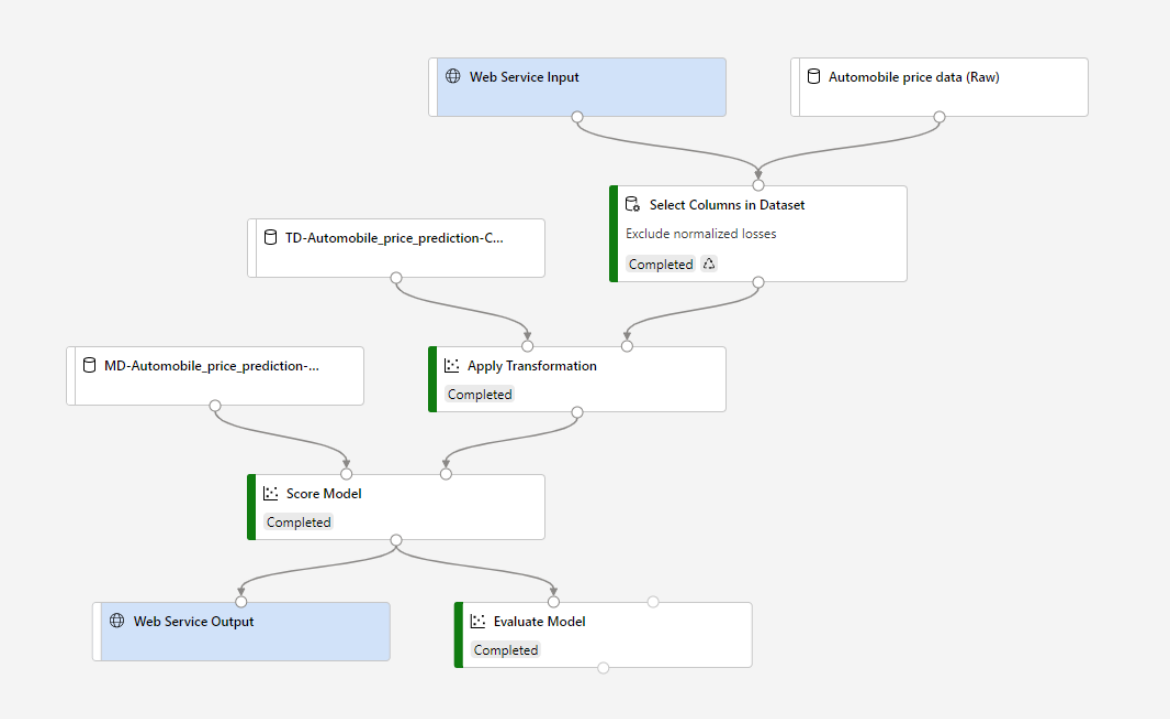
- *Mean Absolute Error* – Todellisten ja pääteltyjen arvojen erotusten, eli virheen, keskiarvo.
- *Root Mean Squared Error* – Keskineliövirheen neliöjuuri.
- *Relative Abosulute Error* – Virheen keskiarvo suhteutettuna todellisten arvojen ja kaikkien arvojen keskiarvon erotukseen.
- *Relative Squared Error* – Keskineliövirhe suhteutettuna todellisten arvojen ja kaikkien arvojen keskiarvon erotukseen.
- *Coefficient of Determination* – Selitysaste, joka kertoo sen, kuinka hyvin malli sopii dataan.



Kuvio 11. Mallin arvioidut virheet

Reaaliaikaisen päättelyputken luonti

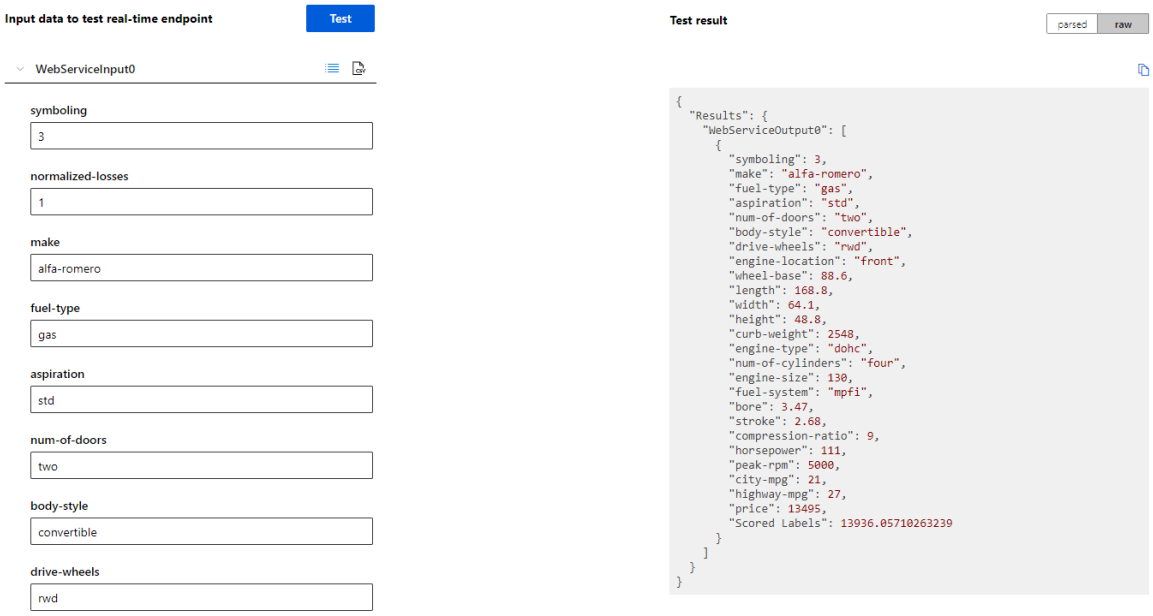
Hyväksytty malli toimitettiin työtilan resurssiksi designerin käyttöliittymän kautta. Designeriin on luotu valmis toiminnallisuus päättelyputken luomiseksi koulutusputken perusteella. Putki on muuten sama, kuin koulutuksessa, mutta ”Train Model” ja ”Data Split” moduulit poistuivat ja putken alkuun ja loppuun sijoittui ”Web Service Input ja Output”, jotka toimivat datan vastaanottamisessa ja tarjoamisessa sovelluksen käyttäjälle.



Kuvio 12. Päättelyputki designerissa

Päättelyputken vieminen tuotantoon

Putken vieminen tuotantoon designerissa on myös automatisoitu lähes kokonaan. Kehittäjän tehtäväksi jää oikeastaan vain laskentakohteen määrittely. Onnistuneen tuotantoon viennin jälkeen päättelyputkea ja mallin toimintaa voitiin testata antamalla päätepisteelle syötteitä. Päätepistettä pystyttiin testaamaan esimerkiksi AML Studion kautta.



Kuvio 13. Testaustulos

5.4 Toteutus serverless-funktioiden avulla

Putken luonti serverlessinä toteutettiin hyödyntäen kahta Azure Functionia, datavarastoa ja tietokantaa. Ensimmäinen serverless-funktio (CarPriceModelTrainer) toteutti mallin kouluttamisen ja evaluoinnin, jonka jälkeen se tallensi mallin zip-tiedostona datavarastoon. Funktion laukaisu toteutettiin manuaalisesti. Toinen serverless-funktio (CarPriceAPI) tarjoaa rajapinnan päättelyiden tekemiseen. Laukaisu toimii http-kutsulla, jonka sisällössä on määriteltynä päättelyparametrit. API-funktio noutaa koulutetun mallin datavarastosta ja suorittaa päättelyn, jonka jälkeen se palauttaa päättelyn tuloksen vastauksessa. Toteutus pohjautuu Towards Data Science -sivustolla julkaistussa blogipostauksessa (Velida 2020) esiteltyyn ratkaisuun.

Serverless-funktioiden käyttämä koodi toteutettiin .Net ohjelmointikehyksen avulla käyttäen Microsoftin ML.NET, Azure.Storage ja SDK.Functions kirjastoja. Funktioiden kehitys toteutettiin Visual Studio 2019 IDE:ssä. Putken luonti serverlessinä piti sisällään seuraavat työvaiheet:

Datan valmistelu

Serverless-toteutuksessa käytetty datasetti on sama, kuin AML:ssä tehdyssä versiossa. Datasettiin tehtiin siis myös tällä kertaa samat toimenpiteet, eli liikaa tyhjiä arvoja sisältävän sarakkeen poisto, sekä tyhjiä arvoja sisältävien yksittäisten rivien siivous. Koska datan käsittelyputki on tismalleen sama molemmissa toteutuksissa, niin se tehtiin tällä kertaa manuaalisesti yksinkertaisuuden vuoksi.

Varasto- ja tietokantaresurssien luonti

Ennen varsinaista ohjelmointia Azuren käyttöliittymässä luotiin projektille oma resurssiryhmä, jonka alle funktiot ja muut tarvittavat resurssit sijoitettiin. Resurssiryhmän alle luotiin Azure Storage-datavarasto, jolle luotiin uusi säiliö (engl. container) koulutetun mallin sijoittamista varten. Tämän lisäksi luotiin uusi Azure Cosmos Database-tietokanta, johon päättelyssä syntyvä JSON-data voitiin tallentaa.

Koulutusfunktion luonti

Koulutusfunktion projektiin luotiin seuraavat tiedostot:

- Kaksi malliluokkaa, joista ensimmäinen määrittelee koulutuksessa käytettävän datan parametrit ja niiden tyypin. Toinen määrittelee päättelymallin, jossa on pääteltävä arvo eli tässä tapauksessa auton hinta ja sen tyyppi.
- Kaksi avustajaluokkaa, joista ensimmäinen vastaa funktion käynnistymisestä ja alustaa tarvittavat konfiguraatiot. Toinen vastaa Azuren tietokantayhteyksistä ja mallin tallentamisesta.
- Itse funktioluokka, jossa mallin koulutus tapahtuu.
- Koulutus- ja testausdata.

Ensin funktiossa toteutettiin Azureen kirjautuminen. Tämän jälkeen koulutukseen ja evaluointiin käytettävä data luetaan tiedostoista. Kaikille ei numeerisille arvoille tehtiin ennen mallin koulutusta vielä transformaatio, jossa arvot kategorisoitiin saaden näille numeeriset arvot regressiota varten. Tämän jälkeen suoritettiin mallin sovitus (engl. fit), sekä mallin evaluointi. Lopuksi funktio tallensi koulutetun mallin datavarastoon .zip tiedostona.

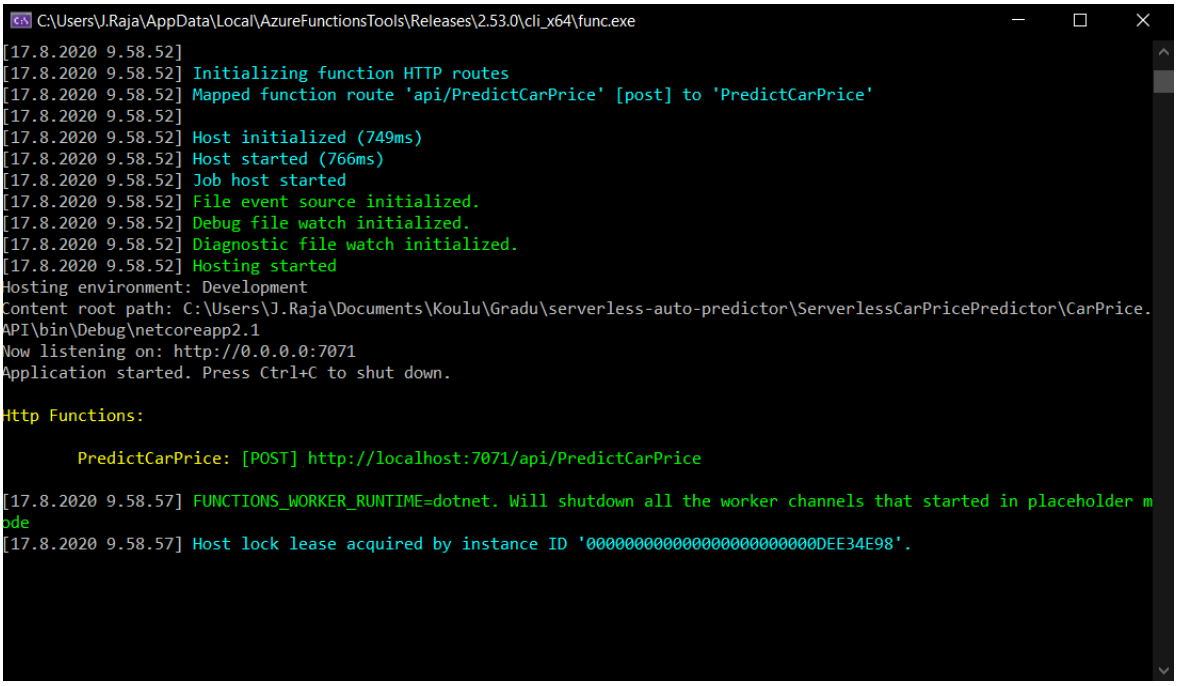
Päättelyfunktion luonti

Päättelyfunktion projektiin luotiin vastaavat malliluokat, käynnistysavustaja, sekä itse funktio. Käynnistysavustajassa toteutettiin alustus API päätepisteelle. Koulutusfunktiota vastaavien malliluokkien lisäksi päättelyfunktio tarvitsi vielä datamallin funktion muodostamasta datasta, joka piti sisällään kaikki syötteenä käytetyt attribuutit sekä ennustetun hinnan.

Päättelyfunktio suoritti ensin mallin hakemisen datavarastosta sekä http-syötteen parsimisen JSON-objektiksi. Tämän jälkeen funktio suoritti päättelyn syötteen datalla, minkä jälkeen se muodosti palautettavan dataobjektin syötedatasta ja ennusteesta.

Funktioiden testaus lokaalisti

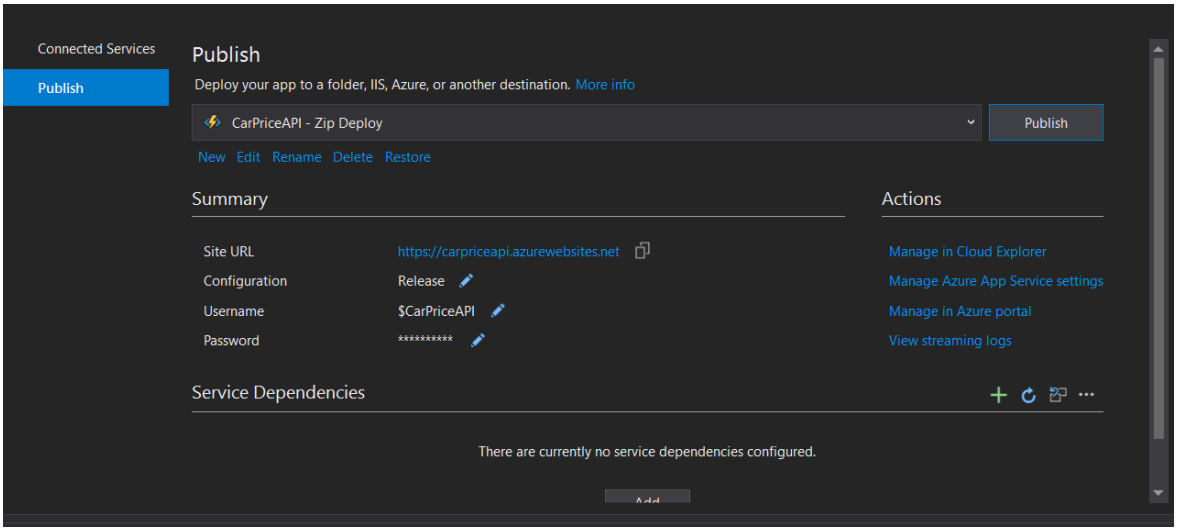
Funktioiden debuggaus ja testaus suoritettiin lokaalisti käyttäen avuksi Azure Functions Core Tools-työkalua. Koulutusfunktio suoritti mallin koulutuksen lokaalisti, mutta tallensi mallin pilveen, jolloin kantayhteyksien toimivuus saatiin varmistettua ennen julkaisua. Kun mallin koulutus oli testattu onnistuneesti, voitiin seuraavaksi testata päättely-API lokaalisti. Azure Function Core Toolsin avulla voitiin alustaa lokaali päätepiste API:lle, jolle antaa syötteitä. Syötteitä lähetettiin sovellukselle Postman-ohjelmiston avulla saaden onnistuneita vastauksia.



Kuvio 14. API funktion ajaminen lokaalisti Azure Function Core Toolsin komentorivillä

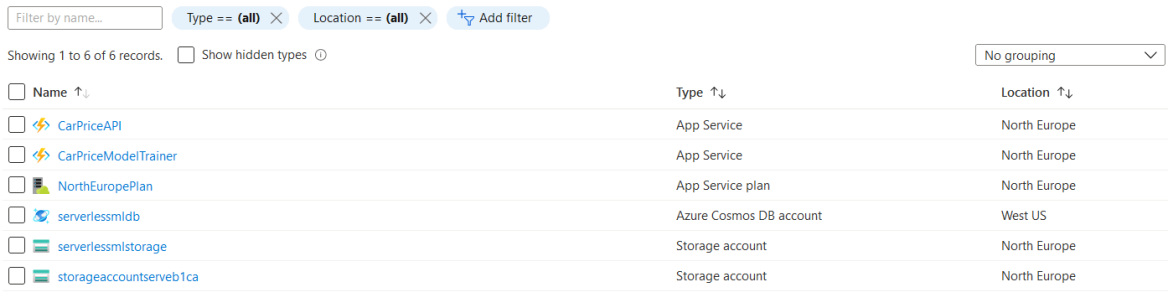
Funktioiden julkaisu Azureen

Funktioiden onnistuneen lokaalitestauksen jälkeen ne julkaistiin Azureen. Tämä tapahtui Azure Functions Core Tools-työkalun avulla Visual Studion käyttöliittymän kautta. Koodin julkaisu suoritettiin määrittelemällä ensin uudet Azure Function-instanssit Visual Studion kautta. Määritykseen tarvittiin kirjautuminen omilla Azure-tunnuksilla, käytettävän resurssiryhmän ja datavaraston nimi, sekä käytettävä App Service Plan, joka hallinnoi palvelun kerryttämää laskutusta.



Kuvio 15. API-funktion julkaisu .zip tiedostona Visual Studion käyttöliittymässä

Kun Azure Function-instanssit oli luotu, voitiin koodi julkaista. Käytetty menetelmä oli rakentamisessa (engl. build) syntyneen .zip tiedoston vienti pilveen kyseisen Azure Function-instanssin juureen, minkä jälkeen Azure huolehtii automaattisesti sen purkamisesta ja oikean kansiorakenteen luomisesta.

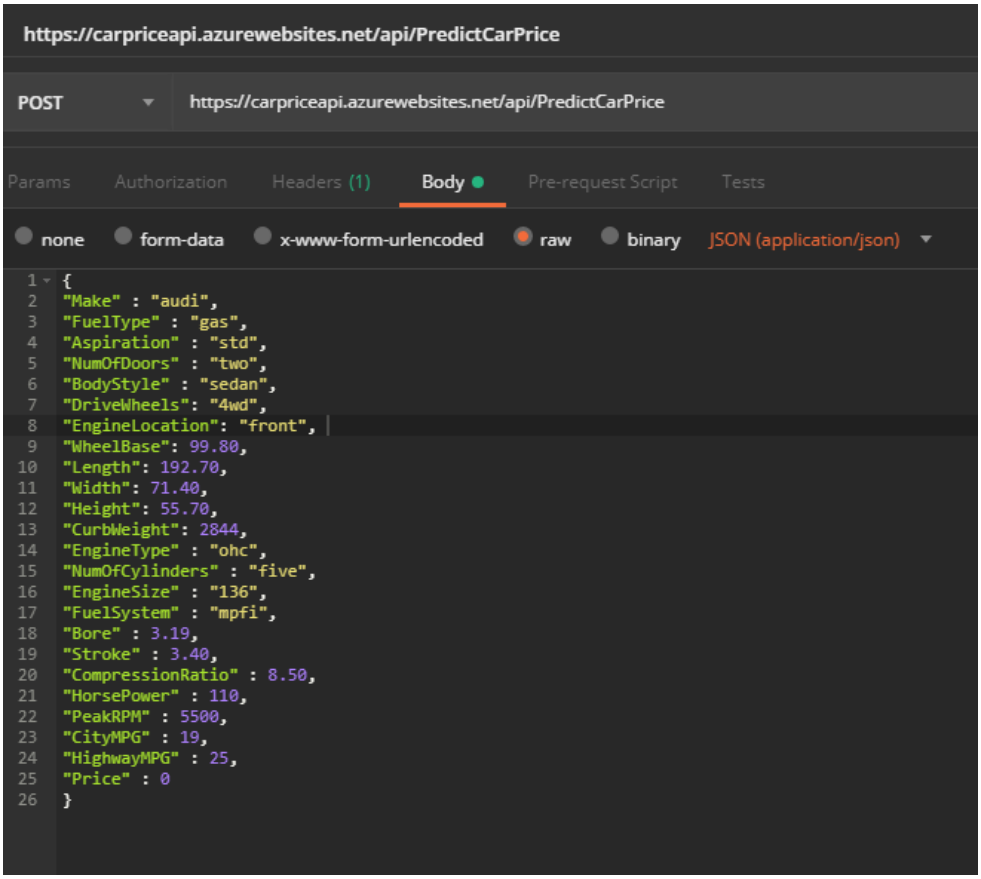


Kuvio 16. Resurssiryhmän alle luodut resurssit Azuressa

Sovellus tuotannossa

Pilveen julkaisun yhteydessä Azure luo funktioille päätepisteen. Koulutusfunktion käynnistykseen ei toteutettu erillistä laukaisinta tai automaatiota, vaan se ajettiin manuaalisesti, jonka jälkeen datavarastoon tallentui koulutettu malli. Tämän jälkeen API-funktion pääte-

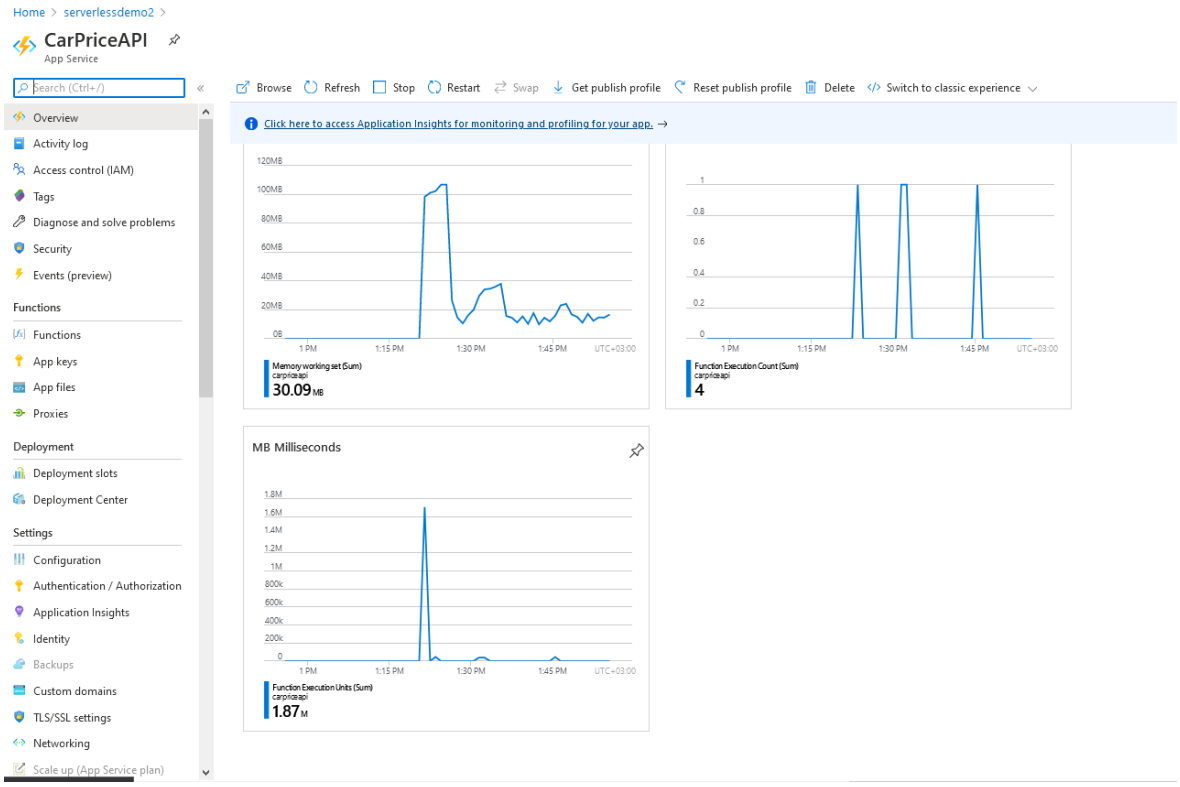
pisteeseen voitiin tehdä http-kyselyitä lähettämällä syötteen arvot JSON-objektina. Vastauksena saatiin viesti, jonka sisällöstä löytyi koneoppimismallin päättelemä hinta. Kaikki lähetetyt vastaukset myös tallennettiin automaattisesti tietokantaan.



Kuvio 17. Esimerkki API:lle annetusta syötteestä

```
"resource": {
    "id": "258529bb-8ef1-49ec-a0ce-cb1a281c42a3",
    "make": "audi",
    "fuelType": "gas",
    "aspiration": "std",
    "numOfDoors": "two",
    "bodyStyle": "sedan",
    "driveWheels": "4wd",
    "engineLocation": "front",
    "wheelBase": 99.8,
    "length": 192.7,
    "width": 71.4,
    "height": 55.7,
    "curbWeight": 2844,
    "engineType": "ohc",
    "numOfCylinders": "five",
    "engineSize": "136",
    "fuelSystem": "mpfi",
    "bore": 3.19,
    "stroke": 3.4,
    "compressionRatio": 8.5,
    "horsePower": 110,
    "peakRPM": 5500,
    "cityMPG": 19,
    "highwayMPG": 25,
    "price": 0,
    "predictedPrice": 19275.0215
},
```


Kuvio 18. Esimerkki API:lta saadusta vastauksesta



Kuvio 19. API-funktion käyttämä muistin määrä kutsujen aikana

6 Analyysi

Tässä tutkimuksessa tutkittiin serverless-teknologian tarjoamia mahdollisuuksia ja haasteita koneoppimissovellusten kehitykseen. Tutkimuksen aineisto kerättiin kirjallisuuden lisäksi toteuttamalla Microsoftin Azure-pilvialustalle kaksi toisiaan vastaavaa koneoppimissovellusta, joista ensimmäinen toteutettiin käyttäen virtuaalikonetta ja toinen käyttäen serverless-funktioita. Toteutettujen sovellusten tarkoituksena oli kartuttaa tietämystä koneoppimissovellusten kehityksestä ja tuoda esiin eroja niiden välillä.

Tutkimuskysymyksiin etsittiin vastauksia kirjallisuuden, sekä toteutuksessa syntyneiden konstruktioiden avulla. Ensimmäiseen tutkimuskysymykseen: ”Miten serverless arkkitehtuuri eroaa tavallisesta palvelinsovelluksen arkkitehtuurista ja mitä etuja ja rajoitteita se tuo mukanaan koneoppimissovellukselle?” vastattiin neljännessä kappaleessa lähdekirjallisuuden avulla. Toiseen tutkimuskysymykseen: ”Miten koneoppimissovellus toteutetaan serverlessinä pilvialustalle?” saatiin hyvä esimerkki toteutuksessa. Viimeiseen tutkimuskysymykseen: ”Miten serverless teknologiaa voidaan ja kannattaa nykyisin hyödyntää koneoppimissovellusten toteuttamisessa?” etsitään vastauksia analysoimalla yhdessä toteutusta ja erilaisia kirjallisuudesta löytyviä ratkaisuja ja ideoita.

Analyysissa perehdytään tutkimukseen liittyviin haasteellisuuksiin, erilaisiin vaihtoehtoisiin optimointiratkaisuihin ja viitekehyksiin serverless-toteutuksille, koneoppimispalveluiden ja valmiiden mallien hyötyihin, sekä siihen, mitä serverless lopulta tarjoaa koneoppimissovelluksille ja niiden kehittäjille.

6.1 Tutkimukseen liittyvät haasteet

Tutkimukseen liittyvät teknologiat, kuten koneoppiminen ja pilvilaskenta, eivät ole varsinaisesti teknologisina innovaatioina kovinkaan uusia sillä niiden teoreettiset lähtökohdat juontavat juurensa hyvin pitkälle. Nykyisenkaltaiset koneoppimisen ja pilvilaskennan PaaS ja SaaS ratkaisut, sekä niihin liittyvät viitekehykset ja ohjelmointikirjastot ovat sen sijaan hyvin tuoreita ja niiden kehitystahti nopeaa. Esimerkiksi ML.Net -kirjaston ensimmäinen versio julkaistiin vasta vuonna 2018.

Tämä osoittautui toteutusvaiheessa konkreettiseksi ongelmaksi, sillä toteutukseen perehtyessä törmäsi usein erityisesti koneoppimissovelluksiin liittyviin ohjeisiin, esimerkkeihin ja ideoihin, jotka olivat saattaneet vanhentua jopa vuodessa niin pahasti, ettei niitä olisi voinut enää toteuttaa sellaisenaan joidenkin kirjastojen tai viitekehysten uusimmilla versioilla. Tietysti uusien kirjastoversioiden pohjautuessa vanhempiin, kaikki samat toiminnallisuudet olivat täysin toteutettavissa niilläkin. Joitakin tapauksia tuli kuitenkin vastaan, joissa jonkin tietyn viitekehyksen tai kirjaston jotkin ominaisuudet eivät olleet yhteensopivia esimerkiksi uusimman Azure Function-version kanssa.

Eri palveluntarjoajien välinen kilpailu alalla kiihdyttää uusien teknologioiden kehittymistä, mutta se myös samalla luo haasteita työkalujen ja palveluiden yhteensopivuuteen. (ECIS 2014) Teknologioiden nopea kehittyminen ja palveluiden laaja tarjonta saa kuitenkin kehittäjät innovoimaan erilaisia ratkaisuja vanhoihin ongelmiin, kuten serverless-teknologian haasteisiin koneoppimissovellusten suhteen.

6.2 Valmiit ratkaisut ja koneoppimispalvelut

Koneoppimismallien kouluttamisen sijaan on myös usein mahdollista hyödyntää valmiiksi koulutettuja malleja. Valmiiksi koulutetuista malleista on tullut alalla usein käytetty vaihtoehto valikoiman kasvaessa jatkuvasti. Erityisen merkittävänä hyötynä näissä on helppous ja nopeus kehittäjille, joiden ei tarvitse kuluttaa aikaa omien mallien tekemiseen. Täytyy kuitenkin muistaa, että valmiiden mallien tarkkuus ei ole koskaan täysin sitä mitä niiden suorituskykytestit lupaavat ja niiden kanssa on otettava useita asioita huomioon, kuten: (Shao 2019)

- Kuinka hyvin päättelydata vastaa mallin koulutukseen käytettyä dataa ja kuinka se on esikäsitelty?
- Kuinka hyvin käytettävä back-end ja laitteisto toimivat mallin kanssa?
- Kuinka paljon jatkokoulutus vääristää mallia?
- Onko mallin koulutus ja päättely optimoitu yhtenäisesti?

Nämä kysymykset huomioituna valmiiksi koulutettujen mallien käyttö voi tuoda huomattavia hyötyjä erityisesti sovellusten tuotantonopeuteen.

Valmiita malleja on tarjolla esimerkiksi suurimpien pilvialustojen valmiissa koneoppimispalveluissa kuten toteutuksessa käytetty AML. Koneoppimispalveluiden tarjoamat työkalut mahdollistavat kokonaisten putkien luomisen alusta loppuun todella nopeasti ja hyvin helposti lähestyttävällä tavalla. Etuna kehittäjille on myös kaikkien toteutukseen liittyvien resurssien löytyminen samasta paikasta. (Xello 2020) Koneoppimispalveluiden lähestyttävyyttä korostaa erityisesti niiden käyttöliittymien graafisuus. Esimerkiksi AML:n avulla kyettiin toteutuksessa luomaan täysin serverless-ratkaisua vastaavanlainen koneoppimisputki alusta loppuun ilman koodia käyttäen apuna AML:n graafista työkalua putkien rakentamiseen.

Graafisten käyttöliittymien suurimpana etuna verrattuna ohjelmointiin on nopeus, joka mahdollistaa useiden kokeilujen toteuttamisen, joita ei tarvitse aina erikseen rakentaa alusta asti. (Brownlee 2018) Tämä lisää innovatiivisuutta kehittäjän etsiessä erilaisia mahdollisia ratkaisuja ongelmiin, jolloin on mahdollista tehdä löytöjä, joita ei muuten olisi osannut ajatella. Graafiset käyttöliittymät ovat myös erityisen hyödyllisiä aloittelijoille, koska vaativaa teknistä osaamista ei välttämättä tarvitse, jolloin ydinasian oppiminen nopeutuu huomattavasti. (Brownlee 2018)

Koneoppimispalvelut eivät ole kuitenkaan vielä saaneet kovin suurta suosiota alan ammattilaisten keskuudessa ja palveluiden tekemä liikevaihto on toistaiseksi ollut vielä heikkoa. (Li 2019) Koneoppimispalvelut ovat pilviyritysten tarjonnassa hyvin tuoreita ja ne kärsivät vielä lukuisista ongelmista, joita ei yksinkertaisesti ole ehditty korjata. Lisäksi niiden ominaisuudet ovat hyvin rajattuja ainakin toistaiseksi. Tällaisten palveluiden suurin ongelma onkin juuri niiden lähestyttävyydestä ja suoraviivaisuudesta johtuva kankeus, joka näkyy juuri ominaisuuksien ja erityisesti mahdollisuuksien rajoitteisuutena. Kaikkea mikä olisi ohjelmoitavissa ei yksinkertaisesti pysty valmiita koneoppimistyökaluja käyttämällä toteuttamaan. Myöskään täysin uusien ideoiden kokeilulle ja innovoinnille on hyvin vähän mahdollisuuksia tällaisilla työkaluilla, sillä ne toteuttavat vain ainoastaan niitä asioita, joihin ne on suunniteltu.

Hyvä esimerkki kankeudesta oli toteutuksessa ilmennyt ongelma, jossa AML:ssä toteutettuja malleja ei pysty mitenkään tuomaan ulos palvelusta, kuten lataamaan omalle koneelle. Tämä on klassinen esimerkki myyjälukosta (engl. vendor-lock), jossa tuotteen tai palvelun tarjoaja tarkoituksellisesti tekee asiakkaan riippuvaiseksi omasta palvelustaan. Koneoppimispalveluiden käyttäjän täytyykin muistaa, että mitä enemmän tällaisiin tuotteistettuihin ja suoraviivaisiin ratkaisuihin sitoutuu, niin sitä vaikeampi niistä on siirtyä pois menettämättä dataa. (Opara-Martins, Sahandi, and Tian 2016)

6.3 Toteutusten arviointi

Tutkimuksen toteutusvaiheessa toteutetut koneoppimissovellukset olivat idealtaan ja datasetiltään samat, mutta toteutuksiltaan erilaiset. Tutkimuksen lähtökohtana oli selvittää, minkälaisia vaihtoehtoja pilvipohjaiselle koneoppimissovellukselle on, ja miten serverless-teknologiaa hyödyttää tällaisia sovelluksia. Ensimmäinen toteutus tehtiin koneoppimispalvelu AML:n avulla ja toinen puhtaasti hyödyntäen serverless-funktioita.

Nykyiset pilvessä toimivat koneoppimispalvelut ja viitekehykset kuten AML ratkaisevat joitain serverless-teknologian haasteita, kuten esimerkiksi suurten datamäärien prosessointia tehokkaasti. (Wang, Niu, and Li 2019) Joissain yhteyksissä AML:stä puhutaan serverless alustana, vaikka se ei serverlessin määritelmää täytäkään. Vaikka tämänkaltaiset palvelut omaavat serverlessin lähestyttävyyteen, tuotantonopeuteen ja palvelintason abstrahointiin liittyviä etuja, ne eivät kuitenkaan ole ainakaan tällä hetkellä serverless-idean mukaisia sanan varsinaisessa merkityksessä. Monet serverlessin suurimmista hyödyistä jäävät saavuttamatta, sillä nykyiset hajautetut koneoppimispalvelut tarvitsevat edelleen dedikoidun laskentaklusterin suorittamaan operaatioitaan.

Dedikoidun laskentaklusterin valinta on jo itsessään palvelinkonfigurointia, josta serverless-paradigma pyrkii pääsemään eroon. Prosessointitehokkuudeltaan ja muistinmäärältään optimaalisen klusterin valinta on työläs prosessi, jossa laskentaresurssien tarve on kyettävä selvittämään aina erikseen jokaista eri toteutusta varten. Esimerkiksi AML:n laskentakohteita on kahdeksaa erilaista tyyppiä ja eri kokoisia virtuaalikoneita useita kymmeniä. Klusterin ylläpitäminen on myös kallista, erityisesti jos käyttöaste on pieni. Klusterin hinnoittelu on

juokseva ja jokainen sekunti sen käynnissä pitämisestä on laskutettavaa, vaikka mitään laskentaa ei tapahtuisi.

Serverless-toteutus mahdollistaa huomattavasti paremmin uudenlaisten ratkaisujen innovoinnin, koska funktiot eivät ole rajoitettu toteuttamaan sovelluksia pelkästään valmiiksi määritellyllä tavalla, toisin kuin valmiissa palveluissa. Tällöin erilaisia haasteita ja rajoituksia pystytään kiertämään huomattavasti paremmin.

6.3.1 Toteutusten erot

Toteutuksia ja niiden eroja voidaan analysoida suunnittelutieteellisessä tutkimuksessa deskriptiivisesti peilaten löytyneitä huomioita aiempaan tutkimukseen. (Hevner et al. 2004) Huomioitavaa on se, että toteutuksissa käytetty datasetti oli todella pieni ja näin ollen analyysissä arvioidaan myös suuremman datasetin ja tuotantokäytön vaikutuksia. Toteutusten kiinnostavimmat ominaisuudet serverlessin piirteitä ajatellen olivat seuraavia.

Toteutuksen helppous

Koneoppimissovelluksen toteuttaminen AML:n avulla on tehty äärimmäisen helpoksi ja se on tällaisen palvelun suurin etu. Se ei ole pelkästään helppo käyttää, vaan erittäin helposti lähestyttävä selkeine valikoineen ja työnkulkuineen, jotka ohjaavat käyttäjän askeleittain eteenpäin. Alusta tuntuu siltä, että se on kohdennettu koneoppimisen parissa vähän, tai ei ollenkaan, työskennelleille käyttäjille. AML:n avulla koneoppimissovellusten toteuttaminen onnistuu myös käyttäjiltä, joilla ei ole ollenkaan ohjelmointiosaamista, sillä missään vaiheessa toteutusta ei tarvinnut kirjoittaa riviäkään koodia.

Koska serverless-funktioille ei tarvitse määritellä laskentakohdetta, poistuu laskentaresurssien optimointiin liittyvä vastuu täysin, joka hieman helpottaa kehitystyötä. Serverless-toteutuksen tekeminen puolestaan vaati yllättävän paljon ohjelmointiin, tietoliikenteeseen ja Azuren resurssien tekniseen puoleen liittyvää tietämystä. Ennen toteutusta täytyi perehtyä huomattavissa määrin Azuren dokumentaatioon erityisesti Azure Functionsin ja Azure Functions core toolsin osalta. Tämän lisäksi tuli tutustua tarvittavaan ohjelmointikirjastoon

(ML.Net), sekä omata lähtökohtaisesti hyvät ohjelmointitaidot ja ymmärrys esimerkiksi kirjastojen ja viitekehysten asennuksesta. Funktioiden koodin määrä on kuitenkin melko vähäistä ja osaavalle kehittäjälle funktioiden tekeminen ei ole kovin työlästä. Aloituskynnys on vain huomattavan korkea.

Toteutuksen rajallisuus

Yksi merkittävimpiä eroja toteutusten välillä on niiden joustavuus. AML on valitettavan rajoittunut tekemään ainoastaan sellaisia asioita, joihin se on suunniteltu. Toteutuksen aikana selvitettiin mahdollisuutta tuoda AML:ssä koulutettu koneoppimismalli ulos alustalta, mutta se ei ollut mitenkään mahdollista. Tällöin koulutuksen lisäksi päättelyputki oli toteutettava AML:n avulla palvelun itse määräämällä tavalla. Koodittomassa toteutuksessa on vielä täysi rajoittuneisuus esimerkiksi valittavissa olevien algoritmien suhteen. AML tarjoaa kylläkin myös mahdollisuuden toteuttaa putken koodaamalla Python SDK:n avulla, mikä tarjoaa jonkin verran enemmän mahdollisuuksia.

Serverless-toteutuksessa funktioiden koodaaminen itse haluamallaan kielellä ja kirjastolla mahdollistaa huomattavia vapauksia verrattuna AML-toteutukseen. Ainoat merkittävät rajoitukset polveutuvat serverless-teknologian fundamentaalisista rajoitteista, kuten tilattomuudesta tai funktion ajoaikarajoituksista, mutta muuten kehittäjällä on hyvin vapaat kädet toteutuksen suhteen. Serverless-funktiot mahdollistavat sovelluksen laajentamisen huomattavasti useammilla eri tavoilla, sillä esimerkiksi koulutetun mallin voi helposti sijoittaa muuallekin kuin pelkästään kyseisten funktioiden omaan käyttöön. Erilaisia ongelmia pystyy tällöin lähestymään useista eri näkökulmista, sekä löytämään uusia innovatiivisia ratkaisuja.

Kustannukset

Toteutusten todellinen hintaero riippuu niiden käytöstä. AML-toteutuksessa käytetyn laskentaklusterin tuntilaskutus on valitulla prosessorilla 0,112 €/h ja kaikkein pienimmällä prosessorilla 0,056 €/h huomioimatta mahdollisia alennuksia. Kuukausitasolla tämä tarkoittaa näinkin kevyelle koneoppimissovellukselle noin 40–80 euron laskutusta, huolimatta sovelluksen käyttöasteesta. (“Pricing - Machine Learning | Microsoft Azure” n.d.)

Puhtaassa serverless-toteutuksessa jatkuvasti juoksevaa laskutusta ei ole, vaan kustannukset määräytyvät ainoastaan funktioiden suorituksen perusteella siten, että funktion ajossa käytettävä muistin määrä kerrotaan prosessointiin käytetyllä ajalla (gigatavu-sekunti). Azuren hinnasto serverless-funktioille on 0,000014 €/Gt-s, jonka lisäksi veloitetaan 0,169 € jokaisesta miljoonasta funktion suorituksesta. Käytettävä muisti pyöristetään aina lähimpään 128 megatavuun ja ajoaika lähimpään millisekuntiin. Vähimmäismuisti yksittäiseen funktion suoritukseen on 128 megatavua ja vähimmäisaika suoritukselle 100 millisekuntia.

Kevyiden serverless-funktioiden ajaminen muutamilla kutsuilla kuukaudessa on käytännössä ilmaista, sillä Azure tarjoaa ensimmäiset 400 000 Gt-s prosessointia ja miljoona suoritusta ilmaiseksi kuukausittain. Mikäli sovellus olisi oikeassa tuotantokäytössä sellaisenaan, tulisi funktion suorituksia olla kuukaudessa karkeasti arvioituna noin 25 miljoonaa, ennen kuin kustannukset ylittäisivät halvimman laskentaklusterin kuukausikustannukset. (“Pricing - Functions | Microsoft Azure” n.d.)

Suorituskyky ja responsiivisuus

Suorituskyvyn vertailu näin pienellä datasetillä ei ole mielekästä, sillä se ei tuottaisi kiinnostavia tuloksia. Erot suoritusajoissa ovat niin pieniä, että on vaikea selvittää mistä ne edes johtuvat, sillä suurin ero syntyisi todennäköisesti latenssista prosessointitehokkuuden sijaan. Responsiivisuudessa on kuitenkin merkittävä ero, joka johtuu serverless-funktioiden käynnistykseen kuluvasta ajasta. Erityisesti kylmäkäynnistyksessä voi kulua pahimmillaan useita sekunteja.

Suurella datasetillä AML-toteutus olisi ollut todennäköisesti ylivoimainen, sillä pienenkin laskentaklusterin prosessointitehokkuus on serverless-funktiota parempi. Tämän lisäksi puhtaalla serverless-versiolla olisi todennäköisesti saattanut tulla suuren datasetin koulutuksessa omat aika- ja muistirajoituksensa vastaan, minkä takia tämä toteutus ei olisi tällaisenaan toiminut gigatavujen kokoisilla dataseteillä.

6.3.2 Miksi serverless

Suurimmat hyödyt serverlessin valintaan pätevät yleisesti erilaisille pilvisovelluksille. Julkinen pilvi-infrastruktuuri tarjoaa nopean ja edullisen tavan kehittää sovelluksia, jotka skaalautuvat automaattisesti tarpeen mukaan. Tämä sama pätee myös koneoppimissovelluksille, jotka ovat luonteeltaan sellaisia, että ne hyötyvät erityisen paljon skaalautuvuudesta tehdessään paljon päättelyitä useille käyttäjille. (Osipov 2020)

Aikaisempien koneoppimissovellusten koodista arvioilta vain noin 5 % sisältää varsinaista koneoppimiseen liittyvää logiikkaa. (Sculley et al. 2015) Loput 95 % on yleensä erilaista alustakoodia, jota serverless-paradigma pyrkii siirtämään pois kehittäjän vastuulta. Näin ollen serverless auttaa kehittäjiä keskittymään varsinaiseen sovelluslogiikkaan paremmin.

Suuret, esimerkiksi satojen gigatavujen kokoiset datasetit, voivat helposti aiheuttaa muistin loppumiseen liittyviä ongelmia yksittäisillä palvelimilla ajettuna, joten hajautettujen järjestelmien rakentaminen on joka tapauksessa välttämätöntä suurille koneoppimissovelluksille.

Serverless-funktioita voidaan käyttää myös koneoppimista hyödyntävien suurempien sovellusten arkkitehtuurissa. Esimerkiksi sovelluksen useat koneoppimista hyödyntävät itsenäiset toiminnallisuudet voidaan toteuttaa omina serverless-mikropalveluina ja kutsua näitä tarvittaessa. (Yan et al. 2016)

6.3.3 Miksi ei serverless

Serverless-funktiot tarjoavat toistaiseksi vielä melko rajallisen valikoiman instanssityyppejä ja konfiguraatiomahdollisuuksia, kuin mitä koneoppimissovellusten kehittäjät toivoisivat. (Kaiser 2020) Tämän lisäksi funktioinstanssien välinen kommunikaatiomahdollisuuksien puute rajoittavat niiden toimintaa merkittävästi.

Klusteri on huomattavasti responsiivisempi tehden siitä paremman vaihtoehdon reaaliaikaiselle päättelylle, jossa vastaus halutaan mahdollisimman nopeasti. (Feyzkhanov 2020) Suurin ongelma on serverless-funktioiden alhainen kaistanleveys, joka esimerkiksi AWS

Lambdalla on vain noin 40 Mt/s verrattuna esimerkiksi keskikokoisen virtuaalikoneen vastaavaan yli 1 Gt/s. Tämän lisäksi serverlessin kylmäkäynnistyksen vaatima aika voi olla monille sovelluksille merkittävä haitta.

Ilman virtuaalikoneiden GPU resurssien tuomaa tehokkuutta serverless-funktioiden suorituskyky on toistaiseksi vielä liian heikkoa raskaiden syväoppimismallien koulutukseen. (Ishakian, Muthusamy, and Slominski 2018) FaaS-malli toimii hyvin kevyille funktioille, jotka vaativat vain vähän tilaa ja muistia ja joilla ei ole tarvetta suurille siirtonopeuksille tai keskinäiselle kommunikaatiolle. Viime aikoina on kuitenkin tutkittu erilaisia optimointiratkaisuja sekä kehitetty viitekehyksiä näiden rajoitteiden rikkomiseksi.

6.3.4 Yhteenveto

Serverless malli on todettu erittäin kustannustehokkaaksi ja helposti toteutettavaksi ratkaisuksi koneoppimissovellusten päättelyn toteuttamiseen erityisesti silloin kun tuotantokäyttö ei ole erityisen raskasta tai kysyntä liian suurta. Syväoppimismalleille nykyiset serverless-funktiot ovat toistaiseksi kuitenkin vielä liian kevyitä. Suurimmat edut päättelyn toteuttamiseksi serverlessinä ovat kustannusten hallinta, automaattinen skaalautuvuus, helppo ja vapaa integraatio suurempiin järjestelmäkokonaisuuksiin, sekä sovellusten tuotantonopeus.

Serverlessin rakenteellisten rajoitteiden takia serverless-funktiot eivät sellaisenaan sovellu mallien kouluttamiseen suurilla datamäärillä ilman optimoitua viitekehystä tai arkkitehtuuria, joka hyödyntää useita funktiota kerralla. Tämä on kuitenkin kehittäjän näkökulmasta huomattavan työläs ratkaisu ja on kyettävä pohtimaan tarkasti, että mitä konkreettista etua tällainen toteutus silloin tarjoaa. Mikäli mallin koulutusta halutaan tehdä jatkuvasti ja kerralla käytettävät datamäärät ovat tarpeeksi pieniä voi mallin jatkuvaan koulutukseen tarkoitettu serverless-funktio ajaa asiansa oikein hyvin. Esimerkiksi jos koulutusdataa tulee jostakin lähteestä aika-ajoin ja satunnaisesti, jolloin tarkoitus on täsmentää mallia voi tällainen ratkaisu toimia.

6.4 Erilaisia arkkitetuuri- ja optimointiratkaisuja

Suurimmat tekniset rajoitteet serverlessin suhteen ovat koodipaketin koko, funktion käytössä olevan muistin määrä sekä rajoitettu ajoaika. Optimointi-ideoita näihin teknisiin rajoitteisiin ovat esimerkiksi: (Christidis, Davies, and Moschoyiannis 2019)

- Käytettävien koodikirjastojen keventäminen. Mikäli kokorajoitukset tulevat vastaan, on syytä karsia turhia osuuksia käytetyistä kirjastoista.
- Ennalta koulutettujen mallien lataaminen dynaamisesti pysyvistä datavarastoista väliaikaiseen paikallisvarastoon funktion alustuksessa. Esimerkiksi AWS Lambda funktioiden temp-kansio tarjoaa väliaikaista muistia jopa 512 megatavua. Tätä on mahdollista hyödyntää mallin väliaikaiseen tallennukseen.
- Kaksi vaiheisen prosessin hyödyntäminen, jossa koulutukselle ja päättelylle käytetään omia kummallekin paremmin optimoituja viitekehyksiä.
- Tietokantadatan haun ja tallennuksen prosessien tehostaminen ajallisesti paremmilla tietokantaoperaatioilla.

Esimerkiksi AWS Serverless Lambda funktioiden suoritusaikarajoitteiden kiertäminen onnistuu AWS Step Functions-työnkulunohjaustyökalun avulla. (Correa 2019) Työnkulunohjaus mahdollistaa pitkään käynnissä olevien tehtävien koordinoinnin ja seurannan. AWS Step Functions tai muut vastaavat FaaS-orkestrointipalvelut tarjoavat serverless-funktioiden koordinointia kuitenkin tietyin rajoittein. Tällaisilla palveluilla ei esimerkiksi ole mahdollista synkronoida useita funktioinstansseja datan yhdenmukaisuuden takaamiseksi tai suorituksen etenemisen yhtenäistämiseksi. Tällaisen koordinoinnin tulisi myös olla latenssiltaan paljon alhaisempaa, kuin mitä nykyiset palvelut tarjoavat. Toimivien koordinointipalveluiden puutteessa serverless-viitekehysten on toteutettava nämä toiminnallisuudet itse. (Barcelona-Pons et al. 2019)

Serverless-funktioita tarjoavat palveluntarjoajat ovat alkaneet nykyisin tarjota vähemmän rajoitettuja versioita funktioista korkeammalla hinnalla. Esimerkiksi Azure tarjoaa funktioilleen ”Premium” laskutusluokkaa, jossa esimerkiksi funktion kylmäkäynnistystä ei enää tarvitse tehdä ja ajoaikarajoituksen voi konfiguroida rajattomaksi. Tällaiset premium-funktiot ratkovat joitain serverlessin haasteita kyllä, mutta ovat ensinnäkin kalliimpia sekä tuovat

mukanaan omia ongelmia, kuten esimerkiksi Azuren tapauksessa sen, että premium-funktioilla on jatkuva kuukausilaskutus johtuen vähintään yhden funktioinstanssin ”lämpimänä” pitämisestä.

6.4.1 Serverless viitekehyksiä koneoppimissovelluksille

SerFer

SerFer on viitekehys, joka on suunniteltu parantamaan Lambda funktioiden prosessointitehokkuutta reaaliaikaista päättelyä vaativille, suurikokoisille malleille, orkestroimalla useita funktioita toimimaan yhdessä. (Rao Divate Kodandarama, Danish Shaikh, and Patnaik 2020)

SIREN

SIREN hyödyntää suurta ja muokattavissa olevaa joukkoa serverless-funktioita mallin koulutuksessa. Viitekehyksen yksi erikoisuuksia on HSP (Hybrid Synchronous Parallel) operaatiomoodi, jossa funktiot päivittävät mallin parametreja asynkronoidusti yhteiseen pilvivarastoon. Jokaisen koulutuserän päässä on rajoitin, joka varmistaa aikataulutuksen optimoinnin. Pääosassa toteutusta on syvään vahvistusoppimiseen perustuva aikatauluttaja, joka on suunniteltu dynaamisesti säätämään serverless-funktioiden määrää ja niiden käyttämää muistia minimoidakseen koulutukseen käytetyn ajan. Aikatauluttaja hyödyntää syväoppimista optimoidakseen mallin koulutuksen laadun ja koulutuksen kustannukset aikaisempien samankaltaisten mallien perusteella. SIREN:in on todettu vähentävän koulutukseen kuluvaa aikaa jopa 44 % verrattuna AWS EC2 klustereiden avulla tehtyihin koulutuksiin. (Wang, Niu, and Li 2019)

CIRRUS

Cirrus on koneoppimisviitekehys, joka automatisoi datakeskusresurssien hallinnan työnkuluille hyödyntäen serverless-teknologiaa. Cirrus yhdistää serverless-ratkaisuiden lähestyttävyyden sekä skaalautuvuuden. Cirrus koostuu kevyestä 80 megatavun työskentelijäinstanssista, joka tarjoaa kehittäjille konfiguraatiomahdollisuudet löytää optimaalisin hinta – aika suhde prosesseille. Cirrus säästää huomattavan määrän koneoppimissovellusten yleisesti varaamaa muistia tehokkaiksi suunnitelluilla koulutusalgoritmeilla sekä striimaamalla pieniä

osia koulutukseen käytettävästä datasta erillisestä datavarastosta. Cirrus hyödyntää tilatonta arkkitehtuuria, jossa työskentelijöiden lisäys ja poisto on toteutettu tehokkaasti. Cirruksen on todettu olevan 100 kertaa tavallista serverless-ratkaisua nopeampi ja 3,75 kertaa nopeampi kuin koneoppimiseen erikoistuneet viitekehykset yleisillä infrastruktuureilla. (Carreira et al. 2019)

6.4.2 Hyvän serverless-viitekehyksen vaatimuksia

Alhainen latenssi

Eri funktioiden välisen viestinnän ja tilatiedon ylläpitämisestä vastaavan datavaraston tulisi omata mahdollisimman alhainen latenssi tarvittaville operaatioille. Tästä syntyy muuten helposti yksi suurimpia pullonkauloja toteutukselle.

Kevyt integraatio muihin järjestelmän osiin, joka ei kuitenkaan sido sovellusta

Sovelluksen API:n olisi mielellään tuettava useanlaisia komentoja liittyen datasetin esikäsittelyyn, koulutukseen sekä hyperparametrien virittämiseen. Kyseinen API on syytä toteuttaa ainakin toistaiseksi jollakin korkean tason kielellä kuten Pythonilla varmistaakseen nykyisten koneoppimissysteemien yhteensopivuuden sen kanssa.

Skaalautuvuus

Kuten esimerkkiviitekehyksien toteutuksissa huomattiin. Serverless-funktioiden määrä on mahdollista skaalata käsiteltävän datan määrän mukaan. Tämä vaatii kuitenkin huomattavaa arkkitehtuurista suunnittelua sen suhteen, että miten funktiot saadaan toimimaan optimaalisesti yhteen. Mitä tulee kyselyiden määrään yksittäisille funktioille, niin sen skaalaamisesta vastaa serverless-funktioilla luonnollisesti palveluntarjoaja.

Viestiohjaus asynkronoiduin viestiketjuin

Tämä on fundamentaalinen ominaisuus serverless-teknologioilla ja koko sovelluksen arkkitehtuuri on perustuttava asynkronoituihin viestiketjuin ohjattuun toimintaan.

Laskennallinen työtaakka on hallittu ja minimoitu

Koodin varaama muistinmäärä on kyettävä minimoimaan hyvin rajallisissa serverless-funktioissa, sekä tehtävä funktioista mahdollisimman optimoituja prosessin tehokkuuden suhteen. Kirjastoja karsimalla tai luomalla niille oman kerroksen voi optimoida koodin määrää huomattavasti.

Korkea virheensietokyky, sekä palautuminen itsenäisesti virhetiloista

Virhetilanteiden käsittely on automatisoitava. Järjestelmän on kyettävä palautumaan vähintäänkin alkutilanteeseen itsestään. Useimmiten palveluntarjoajien alustassa tällainen on huomioitu hyvin.

Reaaliaikainen datankäsittely

Päättelyputken on oltava reaaliaikainen, jolloin on syytä toteuttaa jonkinlainen rajapinta datan syöttämiselle.

6.5 Jatkotutkimus

Serverlessin rajoitusten rikkomisesta tai kiertämisestä on tehty tutkimusta, mutta toistaiseksi melko vähän. Tässä on kuitenkin myös hyvä muistaa, että suurin osa rajoituksista on periaatteessa palveluntarjoajien itse päättämiä ja mikäli kysyntä ns. raskaampaa käyttöä vaativille serverless-resursseille kasvaa, niin palveluntarjoajat varmasti kykenevät tällaisia tarjoamaan. Kuitenkin serverlessin fundamentaalisissa rakenteissa on varmasti tutkittavaa erityisesti siinä, miten teknologiaa voitaisiin kehittää siten, että se voisi tarjota samoja hyötyjä erilaisille operaatioille, tai miten se voisi palvella suurempaa käyttäjäkuntaa. Yksi tärkeimpiä kehityskohteita serverlessille olisi varmasti GPU-prosessoinnin hyödyntämisen mahdollisuus, sekä funktioiden klusterointi siten, että ne kykenisivät kommunikoimaan keskenään tai että yksittäinen funktio voisi hyödyntää prosessoinnissaan useita ytimiä. Näiden asioiden toteutus lienee kuitenkin kiinni yksinomaan palveluntarjoajien motivaatiosta, mutta niiden potentiaalia voisi tutkia.

Mitä tulee koneoppimissovelluksien kehittämiseen serverlessin rajoitusten puitteissa, niin yksi mahdollisia tutkimuskohteita voisi olla sovellusten optimointi erityisesti käytettävän muistin määrän osalta. Tämä tehostaisi koko toteutusta ja sopisi muutenkin hyvin serverlessin ideaan, jossa funktion on tarkoitus vain ajaa hyvin kevyttä koodia. Mahdollisia tutkimusaiheita voisivat olla esimerkiksi jonkinlaiset kompressioalgoritmit, sekä kerrostus. Nykyiset palveluntarjoajat itseasiassa tarjoavat jo palveluita, joilla toteuttaa kerrostusta mutta näiden käytöstä on vasta hyvin vähän tutkimusta.

Datavarastojen potentiaalin maksimoinnista ei myöskään ole hirveästi tutkimusta vielä. Yksi syy tähän on varmasti jälleen se, että näiden kehittämisestä vastaavat palveluntarjoajat itse. Nopea serverless-datavarasto olisi kuitenkin erittäin hyödyllinen, ei pelkästään koneoppimissovelluksille, vaan kaikenlaisille serverless-teknologiaa hyödyntäville sovelluksille ja tällaista odotetaan palveluntarjoajilta tulevaisuudessa. (Carreira et al. 2018)

Palveluntarjoajat ovat suurimmalta osin hyvin identtisiä mitä niiden tarjoamiin palveluihin tulee. Niiden resurssivalikoimassa ja eri resurssien teknisissä yksityiskohdissa on kuitenkin joitain eroja. Internetistä löytyviä kirjoituksia ja aiempia tutkimuksia läpi käydessä AWS osoittautui selkeästi suosituimmaksi alustaksi kehittäjien ja tutkijoiden keskuudessa koneoppimiseen liittyen. Yksi potentiaalinen jatkotutkimuskohde olisikin tehdä palveluntarjoajien välistä vertailua tähän aihepiiriin liittyen.

7 Yhteenveto

Tässä tutkimuksessa selvitettiin miten koneoppimissovellukset hyötyvät serverless-teknologiasta ja mitä haasteita asiaan liittyy. Toteutuksessa tehtiin puhdas serverless-ratkaisu, sekä Azuren koneoppimispalvelua hyödyntävä toteutus vertailun vuoksi. Tutkimuskysymyksiin etsittiin vastauksia hyödyntäen toteutuksen tuomaa tietämystä, sekä aiempia aiheeseen liittyviä tutkimuksia ja toteutuksia.

Heti kirjallisuuskatsauksen aikana selvisi, että serverless-teknologia ei ole tähän mennessä ollut optimaalinen vaihtoehto E2E serverless koneoppimistoteutusten tekemiseen. Serverless-funktioita ei yksinkertaisesti ole suunniteltu niin suurten datamäärien ja raskaiden operaatioiden käsittelyyn, joita erityisesti mallien koulutuksessa tarvittaisiin. Kuitenkin asiaan enemmän perehdyttyä löytyi tuoreita tutkimuksia, joissa serverlessin rajoitteita on onnistuttu kiertämään erilaisissa viitekehystoteutuksissa. Näin ollen rajoitukset voidaan nähdä pikemminkin haasteina.

Tutkimuksen aikana selkeytyi hyvin se, miten serverless-teknologiaa voidaan hyödyntää koneoppimissovellusten toteutuksessa optimaalisesti. Koneoppimissovellusta kehitettäessä on osattava ymmärtää, että mikä on puhtaan E2E serverless ratkaisun etu verrattuna siihen, että serverless-funktioita hyödynnettäisiin esimerkiksi vain päättelyn tekemisessä ja raskas mallin koulutus tehtäisiin jollakin muulla tavalla, tai että milloin päättelyn responsiivisuus on niin merkittävä tekijä, että serverless-funktion käyttämisen sijaan on kannattavampaa ylläpitää virtuaalikonetta.

Yksi asia mitä tutkimuksen toteutuksen kannalta olisi kannattanut tehdä, olisi ollut valita Amazon toteutuksen palveluntarjoajaksi, sillä suurin osa nykyisestä tutkimuksesta on tehty AWS:n avulla ja tämä olisi tarjonnut huomattavasti enemmän yhteistä kosketuspintaa aiempaan tutkimukseen. Toisaalta Azuren käyttö alustana oli hyvä vaihtoehto tutkimuksen kannalta, koska tällöin saatiin erilainen näkökulman verrattuna muihin tutkimuksiin.

Tutkimuksen lähtökohdat ja toteutuksen onnistuminen olivat lopulta hyvät, vaikka haasteita matkan varrella olikin. Suurimmat haasteet liittyivät teknologioiden tuoreuteen, mistä johtui

viitekehysten ja kirjastojen valtavan nopea kehitystahti, kirjallisuuden ja aiempien tutkimusten vähäisyys, sekä se, että jotkut alan käsitteet eivät ehkä vielä ole aivan täysin vakiintuneita.

Toteutusvaihtoehtojen vertailun haasteiksi muodostui lopulta se, että optimaalisuus on suhteellinen käsite, joka riippuu paljon kontekstista. Kokeen vuoksi tehty sovellus, jossa tarkoituksena oli saada tietämystä aiheeseen, ei tietenkään vastaa oikeaa tuotantoon toteutettua sovellusta. Tästä syystä optimaalisuus jäi huonosti määritellyksi ja vaikka eri toteutusvaihtoehdot käyttivät samaa datasettiä, ei niiden vertailu lopulta ollut kovin mielekästä esimerkiksi kustannusten tai prosessointinopeuden näkökulmasta. Näistä löytyi hyvää tutkimusdataa jo valmiiksi.

Tämän tutkimuksen tärkeimpänä lopputuloksena on nimenomaan ymmärrys tarkoituksesta, jota kehitettävä sovellus palvelee. Tällöin voidaan tehdä oikeat arkkitehtuuriset ratkaisut. Tutkimuksen tuloksena syntyi hyvää pohdintaa ja arvioita siitä, mitä asioita serverless koneoppimissovellusta kehitettäessä on otettava huomioon ja miten optimaalinen arkkitehtuuri syntyy näistä huomioista.

Lähteet

Armbrust, Michael, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, and Ariel Rabkin. 2010. "A View of Cloud Computing." *Communications of the ACM* 4 (1): 54.

"Azure Documentation | Microsoft Docs." n.d. Accessed July 21, 2020. https://docs.microsoft.com/en-us/azure/?product=featured.

"Azure Functions Overview | Microsoft Docs." n.d. Accessed July 16, 2020. https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview.

"Azure Machine Learning Documentation | Microsoft Docs." n.d. Accessed July 21, 2020. https://docs.microsoft.com/en-us/azure/machine-learning/.

Barcelona-Pons, Daniel, Gerard París, Pedro García-López, Pierre Sutra, and Marc Sanchez-Artigas. 2019. "On the FaaS Track: Building Stateful Distributed Applications with Serverless Architectures." *Middleware '19: Proceedings of the 20th International Middleware Conference*. https://doi.org/10.1145/3361525.3361535.

Brownlee, Jason. 2018. "What If I'm Not a Good Programmer." 2018. https://machinelearningmastery.com/what-if-im-not-a-good-programmer/.

Carreira, Joao, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. 2018. "A Case for Serverless Machine Learning." *Systems for ML*.

———. 2019. "Cirrus: A Serverless Framework for End-to-End ML Workflows." In *SoCC '19: Proceedings of the ACM Symposium on Cloud Computing*. ACM. https://doi.org/10.1145/3357223.3362711.

Christidis, Angelos, Roy Davies, and Sotiris Moschoyiannis. 2019. "Serving Machine Learning Workloads in Resource Constrained Environments: A Serverless Deployment Example." In *IEEE 12th Conference on Service-Oriented Computing and Applications (SOCA)*, 55–63. IEEE.

Correa, Rafael Felix. 2019. "Building an AWS Serverless ML Pipeline with Step Functions."

2019. https://tech.olx.com/building-an-aws-serverless-ml-pipeline-with-step-functions-b39feed12bab.

Domingos, Pedro. 2012. “A Few Useful Things to Know about Machine Learning.” *Communications of the ACM* 55 (10).

ECIS. 2014. “Cloud Computing Standards, Compatibility and Interoperability: Ensuring a Thriving Competitive Market.” ECIS. http://www.ecis.eu/wp-content/uploads/2014/10/Cloud-Computing-Standards-Compatibility-and-Interoperability_final-version1.pdf.

Eyk, Erwin Van, Lucian Toader, Sacheendra Talluri, Laurens Versluis, Alexandru Uta, and Alexandru Iosup. 2018. “Serverless Is More: From PaaS to Present Cloud Computing.” *IEEE Internet Computing* 22 (5): 8–17. https://doi.org/10.1109/MIC.2018.053681358.

Feng, Lang, Prabhakar Kudva, Da Silva, and Jiang Hu. 2018. “Exploring Serverless Computing for Neural Network Training.” In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. https://doi.org/10.1109/CLOUD.2018.00049.

Feyzkhanov, Rustem. 2020. “Using TensorFlow and the Serverless Framework for Deep Learning and Image Recognition.” 2020. https://www.serverless.com/blog/using-tensorflow-serverless-framework-deep-learning-image-recognition.

Hayes, Brian. 2008. “Cloud Computing.” *Communications of the ACM* 51 (7): 9–11. https://doi.org/10.1145/1364782.1364786.

Hevner, Alan R., Salvatore T. March, Jinsoo Park, and Sudha Ram. 2004. “Design Science in Information Systems Research.” *MIS Quarterly: Management Information Systems* 28 (1): 75–105. https://doi.org/10.2307/25148625.

Ishakian, Vatche, Vinod Muthusamy, and Aleksander Slominski. 2018. “Serving Deep Learning Models in a Serverless Platform.” In *2018 IEEE International Conference on Cloud Engineering (IC2E)*.

Jordan, M I, and T M Mitchell. 2015. “Machine Learning: Trends, Perspectives, and Prospects.” *Science 17*, 2015.

Kaiser, Caleb. 2020. “Why We Don’t Use Lambda for Serverless Machine Learning.” 2020. https://towardsdatascience.com/why-we-dont-use-lambda-for-serverless-machine-learning-d00038e1b242.

Li, Michael. 2019. “AWS SageMaker’s New Machine Learning IDE Isn’t Ready to Win over Data Scientists | VentureBeat.” 2019. https://venturebeat.com/2019/12/08/aws-sagemakers-new-machine-learning-ide-isnt-ready-to-win-over-data-scientists/.

Oladipupo, Taiwo. 2010. “Types of Machine Learning Algorithms.” In *New Advances in Machine Learning*. InTech. https://doi.org/10.5772/9385.

Opara-Martins, Justice, Reza Sahandi, and Feng Tian. 2016. “Critical Analysis of Vendor Lock-in and Its Impact on Cloud Computing Migration: A Business Perspective.” *Journal of Cloud Computing* 5 (1): 4. https://doi.org/10.1186/s13677-016-0054-z.

Osipov, Carl. 2020. *Serverless Machine Learning in Action*. https://livebook.manning.com/book/serverless-machine-learning-in-action/chapter-1/v-2/.

“Pricing - Functions | Microsoft Azure.” n.d. Accessed September 19, 2020. https://azure.microsoft.com/en-us/pricing/details/functions/.

“Pricing - Machine Learning | Microsoft Azure.” n.d. Accessed September 19, 2020. https://azure.microsoft.com/en-us/pricing/details/machine-learning/.

Rao Divate Kodandarama, Mohan, Mohammed Danish Shaikh, and Shreeshrita Patnaik. 2020. “SerFer: Serverless Inference of Machine Learning Models.” University of Wisconsin-Madison. https://divatekodand.github.io/files/serfer.pdf.

Sajid, Mohammad, and Zahid Raza. 2013. “Cloud Computing: Issues & Challenges.” In *International Conference on Cloud, Big Data and Trust 2013*. https://www.researchgate.net/publication/278117154.

Savage, Neil. 2018. “Going Serverless.” *Communications of the ACM* 61 (2): 15–16. https://doi.org/10.1145/3171583.

Sculley, D., Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean François Crespo, and Dan Dennison. 2015. "Hidden Technical Debt in Machine Learning Systems." *Advances in Neural Information Processing Systems* 2015-Janua: 2503–11.

Seiler, Klaus. 2019. "Pure Serverless Machine Learning Inference with AWS Lambda and Layers." 2019. https://medium.com/merapar/pure-serverless-machine-learning-inference-with-aws-lambda-and-layers-979702d9ae49.

Shao, Cecilia. 2019. "Approach Pre-Trained Deep Learning Models with Caution | by Cecelia Shao | Comet.Ml | Medium." 2019. https://medium.com/comet-ml/approach-pre-trained-deep-learning-models-with-caution-9f0ff739010c.

Sutton, Richard S, and Andrew G Barto. 2015. *Reinforcement Learning: An Introduction Second Edition, in Progress*. The MIT Press.

"UCI Machine Learning Repository: Automobile Data Set." n.d. Accessed August 17, 2020. https://archive.ics.uci.edu/ml/datasets/Automobile.

Velida, Will. 2020. "Building a Serverless Machine Learning API Using ML.NET and Azure Functions | by Will Velida | Towards Data Science." 2020. https://towardsdatascience.com/building-a-serverless-machine-learning-api-using-ml-net-and-azure-functions-ad3b24751106.

Wang, Hao, Di Niu, and Baochun Li. 2019. "Distributed Machine Learning with a Serverless Architecture." In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*.

Wolf, Oliver. 2020. "Purposes of the Serverless Architecture Style." 2020. https://specify.io/concepts/serverless-baas-faas.

Xello. 2020. "Amazon SageMaker: 3 Reasons To Use for Machine Learning." 2020. https://xo.xello.com.au/blog/amazon-sagemaker-3-reasons-to-use-for-machine-learning.

Yan, Mengting, Paul Castro, Perry Cheng, and Vatche Ishakian. 2016. "Building a Chatbot

with Serverless Computing." In *Conference: The 1st International Workshop*. https://doi.org/10.1145/3007203.3007217.

Zhang, Shichao, Chengqi Zhang, and Qiang Yang. 2003. "Data Preparation for Data Mining." *Applied Artificial Intelligence* 17 (5–6). https://doi.org/10.1080/08839510390219264.