Aapo Peiponen

Konttiteknologian hyödyntäminen koneoppimissovelluksissa

Tietotekniikan pro gradu -tutkielma

20. maaliskuuta 2024

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Aapo Peiponen

Yhteystiedot: aaalalpe@jyu.fi

Ohjaaja: Mikkonen Tommi

Työn nimi: Konttiteknologian hyödyntäminen koneoppimissovelluksissa

Title in English: Utilizing containers in machinelearning-applications

Työ: Pro gradu -tutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 52+0

Tiivistelmä: Tutkielmassa kartoitetaan mikropalveluarkkitehtuurin ja konttiteknologian käytön hyötyjä ja haittoja koneoppimista hyödyntävissä sovelluksissa. Tutkimus suoritetaan tapaustutkimuksena, jonka kohteena on Urheilun tietovarannon kehitysprojekti. UTV-projektin tavoitteena on kehittää alusta urheiludatan keräämiselle, hallinnoinnille, ja analyysille. Projektissa kehitettävä UTV-alusta koostuu mikropalveluista ja hyödyntää konttiteknologiaa toteutuksessaan. Alustalle kehitettiin erillinen koneoppimista hyödyntävä artefakti urheiludatan visualisointia varten. Artefaktia ja sen kehitysprosessia verrattiin kirjallisuudesta esiin nouseviin väitteisiin koneoppimisesta mikropalveluarkkitehtuurissa, ja havaittiin että väitteet eivät ole ristiriidassa artefaktin ja sen kehitysprosessin kanssa.

Avainsanat: konttiteknologia, kontainerisointi, tekoäly, koneoppiminen, tekoälysovellus, dataanalyysi, datan esikäsittely, virtualisointi, koneäly, urheiludata, umap

Abstract: The goal of this study is to map out the pros and cons of utilizing microservices and containers in machinelearning applications. The research is conducted as a case study, the target of which is the Sports Data Repository project (shortened as UTV). The goal of UTV-project is to develop a platform for the collection, control and analysis of sports data. The UTV-platform consists of numerous microservices implemented in containers. A separate artefact was implemented into the platform that uses machinelearning to visualize sports data. The artefact and its development process was compared to claims found in literature

in order to answer the predetermined research questions. It was observed that the claims in literature are not inconsistent with the artefact and its development process.

Keywords: containerization, artificial intelligence, machine learning, AI/ML, Ai application, data-analysis, data preprocessing, virtualization, machine intelligence, sports data, umap

Termiluettelo

AI/ML	Lyhenne sanoista Artificial Intelligence / Machine Learning (suom. tekoäly / koneoppiminen)
API	Lyhenne sanoista Application Programming Interface (suom. ohjelmointirajapinta), on ennaltamääritelty tapa jolla kaksi ohjelmaa voivat keskustella keskenään. APIn toteutustapa on täysin ohjelmoijan päätettävissä, mutta on olemassa erilaisia tyylejä toteuttaa API, kuten REST.
data-allas	Data-allas on datajärveä pienempi kokonaisuus, joka keskittyy kaiken datan tallettamisen sijaan yhteen kokonaisuuteen liittyvän datan tallentamiseen. Useampi data-allas voi siten muodostaa datajärven. Data-allas on datajärveä rakenteisempi kokonaisuus.
datajärvi	Datajärvi on tiedon tallennuspaikka, jossa tietoa ei jäsennellä tai muunneta tallennushetkellä. Tiedon tarkka rakenne ei välttämättä ole selvillä, koska se ei ole relevanttia tiedon säilömisen kannalta. Datajärvi voi sisältää mitä vain dataa, oli se sitten käsiteltyä dataa tai raakadataa.(Amazon Web Services 2021)
JSON	Lyhenne sanoista JavaScript Object Notation. Yleisesti käytetty rakenteellinen tekstipohjainen tiedostomuoto REST rajapintojen yhteydessä.
kontti	Kontti on muusta järjestelmästä eristetty nimiavaruus, joka sisältää esimerkiksi jotain ohjelmistoa ja sen vaatimat kirjastot. Kontti käyttää isäntäkäyttöjärjestelmän ydintä olematta vuorovaikutuksessa käyttöjärjestelmän muiden nimiavaruuksien (ja siten muiden ohjelmistojen tai konttien tms) kanssa. Kontit ovat merkittävästi kevyempiä kuin virtuaalikoneet, koska kontit eivät sisällä omaa kokonaista käyttöjärjestelmäänsä (Google 2022).
koneoppiminen	Koneoppimisella viitataan tekoälyn osa-alueeseen, joka keskittyy datan ja algoritmien hyödyntämiseen oppimisprosessin

	imitoimisessa. Koneoppimisalgoritmit hyödyntävät esikäsiteltyä dataa oppiakseen esimerkiksi luokittelemaan tai ennustamaan annettuun dataan liittyviä ilmiöitä. (IBM 2020)
LTH	Lyhenne tulee sanoista Liikunta, Terveys ja Hyvinvointi. LTH hanke on jatkoa aiemmalle UTV-hankkeelle, ja sen tavoitteena on mahdollistaa sovelluskehitys UTV-alustan päällä LTH-alan yrityksille. (University of Jyväskylä 2024a)
MSA	Tarkoittaa mikropalveluarkkitehtuuria (engl. MicroServices Architecture). Sovelluskehityksen arkkitehtuurityyli jossa sovellus on jaettu vielä pienempiin yksittäisiin palveluihin kuin SOAssa.
REST	Lyhenne sanoista REpresentational State Transfer. Toteutustyyli ohjelmointirajapinnoille.
SOA	Tulee englanninkielen sanoista Service Oriented Architecture, eli palveluarkkitehtuuri. Viittaa sovelluskehityksen arkkitehtuurityyliin jossa sovellukset jaetaan pienempiin osakokonaisuuksiin jotka ovat tulkittavissa yksittäisinä palveluina.
UTV	Lyhenne nimestä Urheilun tietovaranto (engl. Sports Data Repository). Alusta jolle tutkimuksessa tarkasteltava artefakti on toteutettu.
virtuaalikone	Virtuaalikone on ohjelmisto joka virtualisoi laitetason niin, että virtuaalikoneen sisällä ajettavalla ohjelmistolla ei ole suoraa pääsyä laitetasoon. Tämä mahdollistaa useiden eri käyttöjärjestelmien ajamisen samalla laitteella yhtä aikaa, koska virtuaalikoneiden sisällä olevat käyttöjärjestelmät eivät suoraan vaikuta toisiinsa eivätkä isäntäkäyttöjärjestelmään. (VMware 2017)
virtualisaatio	Virtualisaatio tarkoittaa useimmin laitetason resurssien jakamista hypervisorin avulla siten että se mahdollistaa useiden käyttöjärjestelmien ajamisen samalla laitteella yhtä aikaa. Virtualisointia on useaa eri tyyppiä, kuten esimerkiksi verkon vir-

tualisointi ja muistin virtualisointi (Kate Brush, Brian Kirsch 2021).

Kuviot

Taulukot

Sisällys

1 Johdanto

Tämän tutkimuksen tavoitteena on selvittää millä tavoilla mikropalveluarkkitehtuuri ja konttiteknologia helpottaa tai vaikeuttaa koneoppimispohjaisten sovellusten kehittämistä. Tutkimus suoritetaan tapaustutkimuksena kehittämällä UTV-alustalle erillinen artefakti urheiludatan visualisointia varten. Tätä artefaktia ja sen kehitysprosessia verrataan kirjallisuudesta löytyviin väitteisiin ja aiempiin havaintoihin vastaavista kehityskohteista. Artefakti tulee hyödyntämään sekä koneoppimista että konttitekniikkaa toteutuksessaan.

Konttitekniikalla tarkoitetaan käyttöjärjestelmän virtualisointia, minkä suurin ero perinteisiin virtuaalikoneisiin verrattuna on merkittävästi pienempi resurssitarve. Konttitekniikan parissa tunnetuimpiin työkaluihin lukeutuvat tällä hetkellä Kubernetes ja Docker, joista jälkimmäistä on käytetty UTV-alustan toteutuksessa. UTV-alusta on arkkitehtuuriltaan mikropalveluarkkitehtuuri, ja jokainen erillinen mikropalvelu on sijoitettu omaan konttiinsa.

Konttitekniikan lisäksi tutkimuksessa kehitettävä artefakti tulee hyödyntämään koneoppimista toteutuksessaan. Koneoppiminen sisältyy laajempaan tekoälyn määritelmään, johon kuuluvat koneoppimisen lisäksi esimerkiksi erilaiset älylliset agentit kuten tietokonepelien tekoälyhahmot (B.J. Copeland 2022). Tekoäly on käsitteenä jaettavissa kolmeen kategoriaan, jotka ovat heikko tekoäly, vahva tekoäly ja superäly (Kaplan ja Haenlein 2019). Heikko tekoäly on tekoäly, joka ei kykene toimimaan sille ennaltamäärätyn kapean sovellusalueen ulkopuolella, kun taas vahva tekoäly on yhtä kyvykäs ja älykäs kuin keskiverto ihminen ja kykenee siten toimimaan myös uusilla sovellusalueilla kuten ihminenkin. Vahva tekoäly puolestaan on ohittanut ihmisen kaikilla älykkyyden ja kognitiivisen kyvyn osa-alueilla. Kaikki koneoppiminen, mukaanlukien uusimmat generatiiviset tekoälyt, kuuluvat heikon tekoälyn kategoriaan (Pietikäinen ja Silven 2019).

Koneoppiminen vaatii dataa syötteeksi. Data voi olla ennalta luokiteltua tai luokittelematonta. Ensimmäisessä tapauksessa puhutaan ohjatusta oppimisesta, kun taas jälkimmäisessä ohjaamattomasta oppimisesta. Ohjatussa oppimisessa pyritään koneoppimisalgoritmin avulla löytämään malli, joka pääsee mahdollisimman lähelle ennaltamäärättyjä oikeita vastauksia. Tyypillinen virhe ohjatussa oppimisessa on ylioppiminen, jossa malli oppii annetun datan

täydellisesti, mutta ei toimi enää kovin hyvin uudella datalla. Ohjaamattomassa oppimisessa puolestaan ei ole määritelty oikeita vastauksia ennakkoon, vaan algoritmi itse pyrkii löytämään ne. Esimerkkinä ohjaamattomasta oppimisesta voisi olla klusterointialgoritmi, joka automaattisesti löytää sille syötetystä datasta klusterit. Tyypillinen virhe tällaisessa tapauksessa on joko yli- tai aliklusterointi, missä algoritmi tunnistaa liian paljon tai liian vähän klustereita. (Pietikäinen ja Silven 2019)

Ohjaamattoman oppimisen kategoriaan kuuluu myös artefaktissa hyödynnettävä UMAP-algoritmi. UMAP soveltuu erityisen hyvin moniulotteisen datan visualisointiin ja on siten sopiva artefaktin osaksi. Käytännössä UMAP on algoritmi moniulotteisen datan ulottuvuuksien vähentämiseen ja on siten sopiva jonkin suuremman koneoppimisputken osaksi. UMAPin tuottamaa visualisaatiota voisi hyödyntää esimerkiksi datan luokittelussa tai klusteroinnissa jollakin toisella siihen sopivalla algoritmilla, kuten HDBSCAN (McInnes 2018). Artefaktissa näin ei ole kuitenkaan tehty, vaan sen toiminnallisuus on rajattu pelkästään visualisaation tuottamiseen.

Tutkimus jakautuu kuuteen eri lukuun, joista ensimmäinen on johdanto. Toinen luku on taustaa, jossa pyritään tarkastelemaan tutkimuksen kannalta olennaista taustateoriaa, ja pyritään samalla selvittämään, mitkä mikropalvelujen ja konttiteknologian hyödyt ja haitat koneoppimiselle ovat jo tiedossa. Kolmannessa luvussa esitellään UTV-alusta tarkemmin, esitetään tutkimusongelma sekä tutustutaan käytettävään DSR-tutkimusmenetelmään. Neljännessä luvussa on artefaktille laadittu erillinen vaatimusmäärittely, tehty suunnitelma sen toteuttamiseksi sekä kuvattu lyhyesti sen toteutuksen vaiheet. Luvun lopussa on lyhyt arvio artefaktin toiminnallisuudesta sen vaatimusmäärittelyn pohjalta. Viides luku sisältää pohdinnan, jossa käydään läpi tutkimuksen tulokset ja mietitään, millä tavalla tuloksia olisi voinut parantaa. Kuudes ja viimeinen luku on yhteenveto, joka sisältää lyhyen tiivistelmän tutkimuksen kulusta ja sisällöstä.

2 Taustaa

Tutkimuksen yhteydessä toteutettu artefakti on kehitetty toimimaan mikropalveluympäristössä konttiteknologian varassa ja tuottamaan urheiludatasta visualisaatioita koneoppimisen avulla. Siten on tärkeää tutustua lyhyesti kuhunkin edellämainitusta kolmesta osa-alueesta sekä käytettävään algoritmiin. Luku jakautuu neljään alilukuun. Luvussa 2.1 tutustutaan virtualisointiin ja eri konttiteknologioihin, mikropalveluarkkitehtuuriin tutustutaan luvussa 2.2, AI/ML-sovelluksiin mikropalveluarkkitehtuurissa luvussa 2.3, sekä artefaktissa käytettyyn UMAP-algoritmiin luvussa 2.4.

2.1 Konttitekniikka

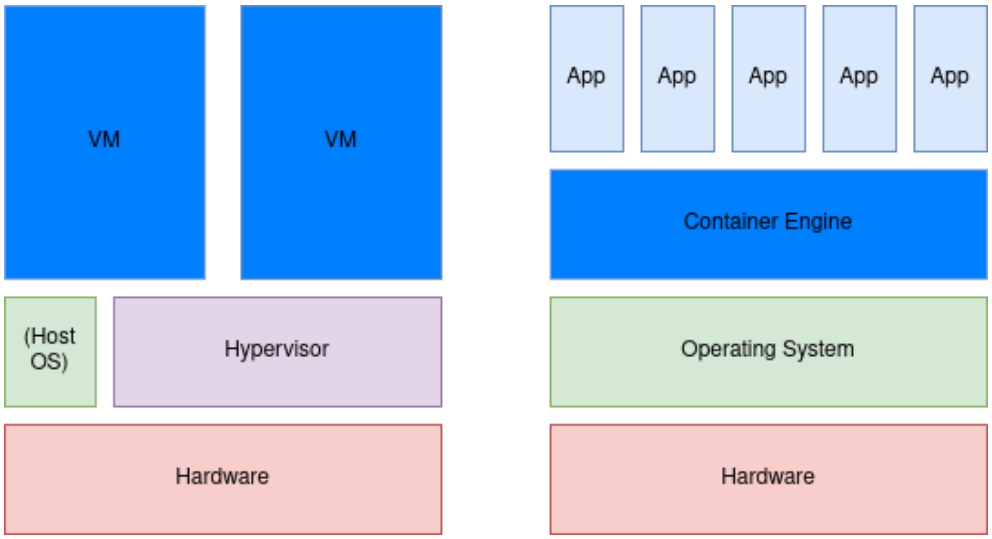
Konttitekniikka kuuluu terminä laajempaan virtualisaation käsitteeseen. Virtualisaatio on ollut olemassa käsitteenä jo 1960-luvulta lähtien, jolloin sillä viitattiin pääasiassa tietokoneen osituskäyttöön (Oracle 2011). Sen merkitys on ajan myötä laajentunut tarkoittamaan useita erilaisia tekniikoita joissa tietokoneen resursseja ositetaan tai yhdistetään. Eri virtualisointitekniikoiden erona on millä tasolla resurssien ositus tai yhdistäminen tapahtuu, ja mikä on resurssi jota virtualisoidaan. Virtualisointitekniikoita on monia, kuten muistin tai verkon virtualisointi, mutta tutkimuksen kannalta oleelliset virtualisointityypit ovat laitetason- ja käyttöjärjestelmätason virtualisoinnit.

Laitetason virtualisoinnissa tietokoneen fyysinen laitteisto on abstrahoitu sovelluskerroksen alle. Tämä mahdollistaa sen, että samalla tietokoneella voi olla yhtä aikaa useita eri käyttöjärjestelmiä käynnissä ilman keskinäistä konfliktia. Sovelluskerrosta, joka abstrahoi tietokoneen fyysisiä laitteita, kutsutaan hypervisoriksi. Hypervisorin päälle luotavia kokonaisuuksia puolestaan kutsutaan virtuaalikoneiksi (VMware 2017) tai järjestelmäkoneiksi (Marinescu 2018).

Virtualisointi voidaan laitetason lisäksi suorittaa myös käyttöjärjestelmätasolla. Tällöin virtualisointi perustuu käyttöjärjestelmän ytimen jakamiseen useiden eri instanssien välillä, jolloin jokainen instanssi voi käyttää ydintä olematta vuorovaikutuksessa isäntäkäyttöjärjestelmän tai muiden instanssien kanssa (Lingeswaran R. 2017). Tätä menetelmää kutsutaan

säilömiseksi (engl. containerization), ja säilöttyjä kokonaisuuksia kutsutaan konteiksi (engl. container).

Kontit omaavat monia merkittäviä hyötyjä virtuaalikoneisiin verrattuna. Näihin lukeutuvat niiden siirrettävyys, julkaisunopeus, helppo skaalautuminen ja parempi resurssien hyödyntäminen. Lisäksi kontit pystyvät käsittelemään pyyntöjä merkittävästi nopeammin kuin virtuaalikoneet ja skaalautumaan tarpeen tullen muutamissa sekunneissa siinä missä virtuaalikoneiden skaalautumiseen menee muutama minuutti (Joy 2015). Nopeus ja skaalautuvuusetujen lisäksi kontit pystyvät suorittamaan samat toiminnot käyttämällä vähemmän muistia kuin virtuaalikoneet (Chae, Lee ja Lee 2019). Tarkemmin, kontit käyttävät CPUta, kovalevyä ja keskusmuistia tehokkaammin kuin virtuaalikoneet. Tämä johtuu pitkälti siitä että, virtuaalikone saattaa kuluttaa järjestelmän resursseja vaikkei virtuaalikoneen tarjoaman palvelun näkökulmasta mitään olisi meneillään. Virtuaalikoneet ovat kuitenkin toistaiseksi tietoturvallisempia kuin kontit johtuen siitä, että virtuaalikoneet eivät jaa käyttöjärjestelmätasoa toisten virtuaalikoneiden kanssa toisin kuin kontit. Alla on kuvio 1, joka havainnollistaa sitä, kuinka virtuaalikoneet ja kontit eroavat toisistaan yleisellä tasolla.



Kuvio 1. Havainnollistava kuva virtuaalikoneiden ja konttien eroista.

Erilaisiin konttialustoihin lukeutuvat esimerkiksi Containerd, LXC, Podman, ja Docker Engine. Näistä suosituin kirjoitusaikaan on Docker, joka on käytössä myös UTV-alustalla. Docker julkaistiin vuonna 2013 ja se on tasaisesti kasvattanut suosiotaan siitä asti (Rani Osnat 2020). Dockerin lisäksi tunnetuimpiin konttitekniikoihin kuuluu myös Googlen vuonna 2015

julkaisema Kubernetes. Siinä missä Docker on ratkaisu yksittäisten konttien hallitsemiseen, Kubernetes mahdollistaa useiden konttien ja kokonaisuuksien hallitsemisen. Siten Kubernetes ei ole pelkästään konttityökalu, vaan laajempaan käyttöön tarkoitettu orkestraatiotyökalu (CNCF 2022). Konttien hallinnointiin tarkoitettu Kubernetes ja samaa toiminnallisuutta tarjoavat orkestraatiotyökalut kuten Docker Swarm on tarkoitettu suurten konttikokonaisuuksien hallitsemiseen, dynaamiseen skaalaamiseen ja ylläpitoon. Mikäli useista konteista koostuvassa sovelluskokonaisuudessa jokainen kontti sisältää jonkin määriteltävissä olevan palvelun, voidaan sanoa kyseisen sovelluksen noudattavan mikropalveluarkkitehtuuria.

2.2 Mikropalveluarkkitehtuuri

Sovelluskehityksessä on olemassa useita eri arkkitehtuureja, joita käytetään kehitettävän sovelluksen rakenteen määrittelyssä. Tämän tutkimuksen kannalta olennaisimmat arkkitehtuurityypit ovat monoliittinen, palvelu-, ja mikropalveluarkkitehtuuri. Näitä kolmea arkkitehtuuria yhdistävät niiden erilaiset lähestymistavat sovelluksen osittamiseen. Siinä missä palveluarkkitehtuuri (SOA) ja mikropalveluarkkitehtuuri (MSA) osittavat sovelluksen useisiin eri palveluihin, monoliittinen arkkitehtuuri ei osita sovellusta ollenkaan.

Monoliittista arkkitehtuuria noudattava sovellus sisältää kaiken sovelluksen tarvitseman toiminnallisuuden yhdessä kokonaisuudessa. Tämän arkkitehtuurin etuihin lukeutuvat esimerkiksi sen yksinkertaisuus ja nopeus. Monoliittinen sovellus onkin yleensä mikropalveluja ja palveluarkkitehtuureja nopeampi tilanteessa, jossa sovellus on sijoitettuna vain yhdelle laitteelle. Heikkoutena monoliittisissa sovelluksissa on niiden ylläpidon vaikeus sovelluksen koon ja kompleksisuuden kasvaessa. (Blinowski, Ojdowska ja Przybyłek 2022)

Palveluarkkitehtuurissa sovelluskokonaisuus on jaettu pienempiin itsenäisiin komponentteihin, joista kunkin sisältämä toiminnallisuus on tulkittavissa yhtenä palveluna. Jokainen sovelluksen komponenteista tarjoaa jonkin tietyn toiminnallisuuden tai palvelun sovelluskokonaisuudessa, jota sovelluksen muut komponentit, palvelut tai sovelluksen käyttäjät voivat hyödyntää. Tämä lähestymistapa mahdollistaa yksittäisten palvelujen hyödyntämisen sovelluksen eri osissa ja helpottaa yksittäisten palvelujen ylläpitoa ja päivittämistä. (Amazon 2024). Koska palveluarkkitehtuurissa sovelluskehitys koostuu yksittäisten palvelujen kehit-

tämisestä yhden suuren kokonaisuuden sijaan, saadaan muiden etujen ohella myös monoliittisia palveluita paremmin horisontaalisesti skaalautuvat palvelut. (Blinowski, Ojdowska ja Przybyłek 2022). Sovelluksen toimivuuden takaamiseksi yksittäiset palvelut kommunikoivat yleensä keskenään jollakin ennaltamäärätyllä protokollalla. Yleisin kommunikointitapa on HTTP-protokollan avulla kommunikointi palvelujen ohjelmointirajapintojen välillä. Tämä mahdollistaa yksittäisen palvelun kompleksisuuden piilottamisen rajapinnan taakse, mikä taas puolestaan tekee palveluista agnostisia sen suhteen, millä teknologioilla toiset palvelut on toteutettu. Tällöin sovelluskokonaisuuden ylläpito ja mahdollinen päivittäminen hyödyntämään uusia teknologioita helpottuu, kun yhden osa-alueen päivittäminen ei riko muita osa-alueita.

Mikropalveluarkkitehtuurissa ja palveluarkkitehtuurissa on sekä samankaltaisuuksia että eroja. MSA voidaan nähdä joko kokonaan uutena arkkitehtuurina tai sitten SOAn yhtenä toteutuksena. Joissakin lähteissä MSA on määritelty parannelluksi versioksi SOAsta (Amazon 2024), kun taas toisissa ne nähdään kokonaan eri arkkitehtuureina (Li ym. 2021). Joka tapauksessa hyödyt, joita palveluarkkitehtuurilla on verrattuna monoliittiseen arkkitehtuuriin, korostuvat entisestään mikropalveluarkkitehtuurissa. Mikropalvelut muodostavat kokonaisuuden, joka samaan tapaan palveluarkkitehtuurin kanssa muodostuu yhden itsenäisen osan sijaan useista itsenäisistä osista, jotka kykenevät toimimaan eristyksissä toisistaan. Kuten palveluarkkitehtuurissa, myös mikropalveluarkkitehtuurissa erilliset palvelut kommunikoivat keskenään verkon yli erilaisilla tilanteeseen sopivilla protokollilla. Mikropalveluarkkitehtuurissa palveluiden välinen kommunikaatio on kuitenkin yleensä toteutettu kevyemmillä tavoilla kuten REST rajapinnoilla, jotka käyttävät JSON tiedostoja datan välittämiseen, kun taas SOAssa palvelujen välisessä kommunikaatiossa hyödynnetään yleensä raskaampaa XML-tiedostomuotoa. Myös palvelujen kokoluokka on mikropalveluarkkitehtuurissa palveluarkkitehtuuria pienempi ja selkeämmin rajattu yhteen spesifiin toiminnallisuuteen. Palveluarkkitehtuurissa taas palvelu voi sisältää suuremmankin toimintokokonaisuuden tietyn palvelun tuottamiseksi. Ero on siis se, että MSA on hienojakoisempi ja kevyempi arkkitehtuuri kuin SOA.

Edellämainituilla kolmella arkkitehtuurilla on kullakin omat etunsa käyttötapauksen mukaan. Esimerkiksi sovelluksen osittaminen mikropalveluiksi tuo kompleksisuutta, joka ei

välttämättä ole tarpeen pienemmissä käyttötapauksissa, kun taas suuremmissa skaalautuvissa käyttötapauksissa se on välttämätöntä. UTV alustalla arkkitehtuurina on käytetty mikropalveluarkkitehtuuria johtuen sen suunnitellusta kompleksisuudesta ja skaalautuvuudesta.

Mikropalveluarkkitehtuuri ja konttiteknologia liittyvät hyvin vahvasti toisiinsa huolimatta siitä, että ne ovat täysin eri asioita. Mikropalveluarkkitehtuuri on tapa lähestyä sovelluskehitystä, kun taas konttiteknologiassa on kyse virtualisoinnista. Nämä kaksi konseptia liittyvät yhteen siinä, kun mikropalvelut sijoitetaan kontteihin. Tällöin jokaisen mikropalvelun toiminnallisuus on eristetty omaan konttiinsa ja ne voivat vaikuttaa toisiinsa vain ennaltamäärättyjen ohjelmointirajapintojen kautta. Useimmat konttiteknologiat, mukaanlukien Docker, jolla UTV-alusta on toteutettu, mahdollistavat virtuaaliverkkojen luomisen kontteja varten. Tällöin mikropalvelujen välinen kommunikointi helpottuu ja muuttuu turvallisemmaksi. Mikropalveluarkkitehtuurin toteuttaminen ilman kontteja on mahdollista, mutta hankalaa. Esimerkiksi sovelluksen kehittäminen vaikeutuisi, koska tällöin konttien tarjoama kehitysympäristöjen samankaltaisuus ja eri palvelujen vaatimien riippuvuuksien eristäminen toisistaan puuttuisi. Myös sovelluksen automaattinen testaus ja hyväksyntä (CI/CD) hankaloituisi. Konttiteknologia on yleisin ja helpoin tapa toteuttaa mikropalveluarkkitehtuuriin pohjautuva sovellus.

2.3 AI/ML-sovellukset mikropalveluarkkitehtuurissa

Mikropalvelujen ja konttien käyttämisestä koneoppimissovelluksissa voidaan löytää useita hyötyjä ja muutamia haittoja. Hyötyihin mikropalveluarkkitehtuurista koneoppimissovellusten toteutuksessa lukeutuvat yksittäisten komponenttien uudellenkäytettävyys koneoppimissovelluksen työnkulun (engl. workflow) eri vaiheissa. Eri vaiheita ovat esimerkiksi datan esikäsittely, piirreirrotus ja luokittelu, joista jokainen olisi eristetty omiin mikropalveluihinsa. Tämä mahdollistaa yksittäisten prosessin vaiheiden helpomman hallinnan ja päivittämisen (Urias ja Rossi 2023).

Koneoppimisprosessin rakentaminen mikropalveluarkkitehtuurin varaan mahdollistaa eri kielten, kirjastojen käyttämisen koneoppimisprosessin eri vaiheissa, ja näiden välisen kommunikaation toteuttamisen HTTP-protokollan yli API-kutsuilla. MSA tuo myös lisää skaalautu-

vuutta AI/ML sovelluksille (Torvekar ja Game 2019). Haittana tässä on tietenkin se, että koneoppimisprosessin jakaminen mikropalveluihin saattaa joissakin käyttötapauksissa hidastaa oppimisprosessia, kun seuraavan vaiheen alkamista edeltää aina tiedonsiirto yhdeltä mikropalvelulta toiselle. Koneoppimissovellukset monoliittisessa arkkitehtuurissa voivat siis olla nopeampia yksittäisten laskutehtävien suorittamisessa, koska niissä ei ole mikropalvelujen välisestä kommunikoinnista aiheutuvaa ylimääräistä kuormaa.

Koneoppimissovellusten toteuttaminen nopeutuu merkittävästi, jos sovellus on pakattu helposti konfiguroitavissa oleviin mikropalveluihin valmiiksi (Pahl ja Loipfinger 2018). Tähän liittyy vahvasti käsite koneoppimisesta palveluna (MLaaS). Tällaisen järjestelmän hyödyt korostuvat entisestään tapauksissa, joissa halutaan toteuttaa jokin ennalta ratkaistu koneoppimistehtävä nopeasti ilman tarvetta opetella koneoppimiskirjastojen yksityiskohtia. Yleinen este koneoppimisen käyttöönotolle onkin sen riittävän ymmärtämisen hankaluus. Vastaava ongelma koskee myös konttitekniikkaa ja mikropalveluarkkitehtuuria.

Mikropalveluissa etuna aiemmin mainittujen lisäksi on se, että niiden avulla voi rajoittaa kehitettävän palvelun kokoa ja siten lisätä tietoturvaa, esimerkiksi rajaamalla pääsy käyttäjädataan vain niille kehittäjille, jotka kehittävät koneoppimiseen liittyviä mikropalveluja (Wang, Kadiyala ja Rubin 2021). Vastaavasti konttien tuoma tietoturva tulee pitkälti sen eristyneisyydestä muista järjestelmän osista, mutta virtuaalikoneet tarjoavat edelleen vahvemman tietoturvan kuin kontit.

Alla on koottuna yleisimmät hyödyt ja haitat mikropalveluarkkitehtuurista ja konttitekniikasta koneoppimissovelluksissa kirjallisuuden pohjalta:

Tunnettuja hyötyjä	
Hyöty	Selite
Soveltuvuus	Kontit toimivat millä tahansa alustalla jolla kyseinen konttiteknologia (esimerkiksi Docker Engine) toimii. Tästä on hyötyä erityisesti reunalaskentaan suuntautuvissa AI/ML sovelluksissa johtuen kohdelaitteiden monimuotoisuudesta.

Skaalautuvuus	Konttien määrää voidaan dynaamisesti skaalata ylös tai alas tarpeen mukaan. Tällöin esimerkiksi esikäsittelyä suorittavien konttien määrää voidaan lisätä jos havaitaan sen olevan hidasteena koneoppimissovelluksessa.
Tietoturva	Kontit tarjoavat paremman tietoturvan kuin palvelujen suorittaminen suoraan isäntäkäyttöjärjestelmän päällä. Tällä on merkitystä, jos koneoppimisalgoritmiin syötettävä data on arkaluonteista.
Jaettavuus ja toistettavuus	Valmiita kontteja voi jakaa ja ajaa samalla tavalla kuin virtuaalikoneita mahdollistaen esimerkiksi simulaatioiden tulosten helpomman toistettavuuden. Tästä esimerkkinä on MLaaS, eli koneoppiminen palveluna.
Vaatimusten hallinta ja ylläpito	Jokainen palvelu on eristetty toisistaan ja kontit sisältävät kaikki tarvittavat kirjastot, joten eri sovellusten välisten vaatimusten kesken ei synny konflikteja. Lisäksi päivittäminen on helpompaa, koska päivitys voi kohdistua vain yhteen palveluun ilman, että muihin palveluihin tarvitsee koskea ollenkaan. Erityisesti koneoppimisen parissa jotkin sovellukset ovat riippuvaisia siitä, että käytössä on jokin tietty versio sen käyttämistä apukirjastoista.

Viansieto	Konttiteknologiat kuten Kubernetes ja Docker Compose tarjoavat mahdollisuuden vikatilanteissa käynnistää kontin takaisin päälle. Lisäksi yksittäisen kontin / palvelun kaatuminen ei vaaranna koko järjestelmää, jos se on hyvin suunniteltu.
Tehokkuus	Kontit käyttävät vähemmän resursseja kuin vastaavat sovellukset virtuaalikoneiden avulla toteutettuna. Ero johtuu siitä ettei konttien sisällä ole erillistä aktiivista käyttöjärjestelmää toisin kuin virtuaalikoneissa. Tämä on etu virtuaalikoneisiin verrattuna.

Tunnettuja haittoja	
Haitta	Selite
Kompleksisuus	Sekä kontit että mikropalveluarkkitehtuuri tuovat sovellukseen lisää kompleksisuutta, mikä on pienissä sovelluksissa tarpeetonta.
Tarpeettomuus	Pienissä sovelluksissa on parempi suosia monoliittista arkkitehtuuria MSAn sijaan.

Tehottomuus	Monoliittinen arkkitehtuuri mahdollistaa yksittäisen tehtävän suorituksen yleensä tehokkaammin kuin MSA.
Tietoturva	Erityisen arkaluontoisen datan käsittelyssä virtuaalikoneet ovat luotettavampia tietoturvan kannalta katsottuna kuin kontit.

Monet sovellusalueet hyödyntävät sekä mikropalveluarkkitehtuuria että konttiteknologiaa koneoppimisen tukena. Hyvin yleisiä esimerkkejä tällaisista sovellusalueista löytyy sekä reunalaskennan että pilvilaskennan alueilta. Pilvi- ja reunalaskennan käsitteet liittyvät vahvasti toisiinsa. Pilvilaskennalla tarkoitetaan laskentamallia, jossa laskentatehtävän vaatima data siirretään pilvipalveluun laskutoimituksen suorittamista varten. Reunalaskennassa puolestaan dataa ei siirretä yhtä kauas, vaan se pysyy lähellä datan lähdettä eli verkon reunalla. Reunalaskenta tarkoittaa siis sovelluksen toiminnan kannalta olennaisten laskutoimitusten suorittamista verkon reunalla ja/tai datalähteen lähellä. Verkon reunalla tarkoitetaan esimerkiksi autonomisten autojen konenäön vaatiman laskennan suorittamista paikallisesti sen sijaan, että tarvittava laskenta suoritettaisiin pilvessä. Reunalaskenta ei kuitenkaan korvaa pilvilaskentaa, vaan täydentää sitä. Tehtävät, jotka eivät vaadi suurta laskentatehoa tai joille on hyvin olennaista saada vastaus nopeasti, voidaan suorittaa paikallisesti. Tehtävät, jotka vaativat suurta laskentatehoa, mutta eivät ole kiireisiä, voidaan suorittaa keskitetysti pilvessä. (Cao ym. 2020).

Aiemmin listattuja etuja voidaan havaita useissa reaalimaailman esimerkeissä. Esimerkiksi älykaupungit, joissa kertyy runsaasti sensoridataa, hyötyvät reunalaskennasta. MSA takaa helpomman hallittavuuden yksittäisille palveluille, kun taas konttiteknologia tarjoaa vakaan suoritusympäristön. Toisena esimerkkinä on MLaaS, eli pilvipalvelut, jotka tarjoavat koneoppimista palveluna. Näissä pilvipalveluissa yksittäiset koneoppimiskomponentit on eritelty omiksi mikropalveluikseen, joita kysynnän mukaan voidaan skaalata lisää asiakkaiden tarpeiden täyttämiseksi. Tutkimuksessa toteutettava artefakti voisi hyvin olla osa MLaaS pil-

vipalvelua. Se tarjoaa oman toiminnallisuutensa ja se olisi voitu toteuttaa olemaan osa suurempaa koneoppimisputkea, jolla tarjotaan alustan käyttäjille mahdollisuus kouluttaa erilaisia malleja urheiludatan pohjalta. Artefaktissa käytetty UMAP-algoritmi sopisi koneoppimisputken alkuosaan datan esikäsittelyn jälkeen johtuen sen kyvykkyydestä moniulotteisen datan ulottuvuuksien vähentämisessä.

2.4 UMAP

Artefaktiin toteutettiin komponentti, joka hyödyntää UMAP-algoritmia visualisaation tuottamiseksi urheiludatasta. Algoritmi itsessään ei tuota visualisaatiota, vaan pudottaa moniulotteisesta datasta halutun määrän ulottuvuuksia pyrkien samalla säilytämään datan moniulotteisen rakenteen. UMAP on lyhenne sanoista Uniform Manifold Approximate Projection, ja se perustuu monistoissa oppimiseen ja topologiseen data-analyysiin. Algoritmiin sisältyy kolme oletusta sille syötetystä datasta:

1. Data on tasaisesti jakautunutta Riemannin monistossa. Tässä kontekstissa monistolla tarkoitetaan topologista avaruutta, joka on paikallisesti euklidinen, eli jokaisella avaruuden pisteellä on naapurusto, joka on topologisesti sama kuin avoin yksikköpallo. Vastaavasti Riemannin monisto on tasainen monisto, jossa on metriikka, mikä tarkoittaa että Riemannin monistossa kahden pisteen välinen etäisyys on laskettavissa (Weisstein 2023). Moniston tasaisuudella taas tarkoitetaan että datapisteet ovat tasaisesti jakautuneita tutkittavassa monistossa. Käytännössä tämä ei tapahdu, koska dataa on rajallinen määrä, mutta silti on mahdollista löytää sellainen metriikka, jolla data saadaan vaikuttamaan tasaisesti jakautuneelta kyseisen metriikan suhteen (McInnes, Healy ja Melville 2020).
2. Riemannin metriikka on paikallisesti vakio (tai approksimoitavissa sellaisena). Tällä tarkoitetaan sitä, että jokaiselle datapisteelle X_i voidaan määritellä eri metriikka, jolloin voidaan saavuttaa vaikutelma datan tasaisesta jakautumisesta, vaikka data ei todellisuudessa olisikaan tasaisesti jakautunutta.
3. Monisto on paikallisesti yhdistetty. Monisto on yhdistetty, jos mitkä vain kaksi pistettä, jotka kuuluvat monistoon, ovat yhdistettävissä jollakin käyrällä, jonka kaikki pisteet sijaitsevat myös monistossa. Vastaavasti monisto on paikallisesti yhdistetty, jos jokai-

sella pisteellä on olemassa naapurusto, joka on yhdistetty. Eli paikallisesti yhdistetyssä monistossa kaikkien pisteiden ei tarvitse olla yhdistettävissä toisiinsa, kunhan piste on yhdistettävissä sen lähimpiin pisteisiin.

UMAP on varsin samankaltainen kuin tSNE-algoritmi, mutta merkittävimpänä erona algoritmien välillä on se, että UMAP säilyttää datan globaalin rakenteen paremmin. Käytännössä tämä tarkoittaa sitä, että tSNE algoritmin tuottamassa visuaalisaatiossa pisteiden muodostamien klustereiden sisäisillä etäisyyksillä on merkitys, mutta klustereiden välisillä etäisyyksillä ei. UMAP algoritmin tuottamassa visuaalisaatiossa molemmilla etäisyyksillä on merkitystä.

UMAP-algoritmissa useat eri parametrit vaikuttavat lopputulokseen. Ei ole vain yhtä oikeaa tai hyödyllisintä tapaa asettaa parametrit projisointia varten. Se mihin arvoihin parametrit kannattaa asettaa riippuu siitä, millä tavalla dataa halutaan tarkastella. Esimerkiksi naapureiden määrä on parametri, joka määrittää kuinka monta naapuria jokaisella pisteellä oletetaan olevan. Käytännössä naapureiden määrällä säädellään sitä, halutaanko visualisaatiossa havainnollistaa enemmän datan globaalia rakennetta vai lokaalia rakennetta. Pienemmillä arvoilla saavutetaan tarkempi kuva lokaalista rakenteista, ja suurilla arvoilla globaalista rakenteesta. Muita esimerkkejä hyödyllisistä parametreista ovat ulottuvuuksien määrä ja metriikka. Ulottuvuuksien määrä tarkoittaa sitä, kuinka moneen ulottuvuuteen data projisoidaan (1-3 ulottuvuutta hyödyllisimmät visualisointiin), ja metriikka taas tarkoittaa sitä mitä tapaa käytetään etäisyyksien mittaamiseen datapisteiden välillä lähtöavaruudessa. Eri tavat mitata etäisyyksiä lähtöavaruudessa vaikuttavat merkittävästi siihen, mihin pisteet projisoituvat kohdeavaruudessa. UMAP hyödyntää myös satunnaisuutta jonkin verran, koska on osittain myös optimointialgoritmi. Tämä satunnaisuus kuitenkin aiheuttaa sen, että algoritmin tarkkoja tuloksia ei voi toisintaa täsmällisesti. Kuitenkin satunnaissiemenen voi asettaa vakioksi, jolloin lopputulos tulee olemaan täysin sama (McInnes 2018).

UMAP-algoritmia voidaan hyödyntää myös klusterointiin. Vaikka algoritmi itsessään ei tee klusterointia, sen tuottamaan visualisaatioon voidaan soveltaa sopivaa klusterointialgoritmia. Klusteroinnin kannalta tärkein parametri on naapurien määrä. Väärillä naapureiden määrillä algoritmi tuottaa visualisaatioita, jotka voivat olla aliklusteroituja tai yliklusteroituja. Yleensä on parempi asettaa naapurien määrä hieman suuremmaksi kuin pienemmäksi, koska sil-

loin visualisaatio tuo paremmin esille mahdolliset klusterit helpottaen siten klusterointialgoritmin toimintaa. Samalla periaatteella voidaan etsiä poikkeamia datasta käyttämällä LOF (Local Outlier Factor) algoritmia tuotettuun visualisaatioon. Sillä voidaan havaita pisteet, jotka eivät kuulu mihinkään klustereihin. Edellisiin liittyen, visualisaatio saattaa olla harhaanjohtava johtuen datasta. Jos datassa on hyvin tiheitä klustereita, UMAP tapaa puskea kyseisiä pisteitä hieman erilleen visualisaatiossa antaen vaikutelman suunnilleen yhtä tiheistä klustereista vaikka näin ei olisi. Tähän ongelmaan ratkaisuna toimii DensMAP, joka on UMAP algoritmin muokattu versio. DensMAP säilyttää paremmin paikalliset tiheydet lähtöavaruudessa johtaen todenmukaisempaan visualisaatioon. Tuotettu visualisaatio ei kuitenkaan ole välttämättä parempi esimerkiksi klusterointitarkoituksiin, koska DensMAP myös lisää hajontaa pisteisiin, mikäli tarkasteltava klusteri on harva lähtöavaruudessa. (McInnes 2018)

3 Tapaustutkimus

Tutkimus päädyttiin suorittamaan tapaustutkimuksena, jossa tarkastelun kohteena on kehitetty artefakti. Artefakti itsessään kehitettiin noudattaen DSR-menetelmää, joka puolestaan kuuluu suunnittelutieteen menetelmiin. Kehitetty artefakti tuli kiinteäksi osaksi UTV-alustaa, ja sen kehitys jatkuu alustan kehityksen mukana.

3.1 Urheilun tietovaranto

Urheilun tietovarannnon kehitysprojekti on hanke, jossa tavoitteena on rakentaa urheiludatan keräämisen mahdollistava tekninen ratkaisu. (University of Jyväskylä 2024b). Kerätty urheiludata tallennetaan tietoaltaaseen pääosin siinä muodossa kuin se tulee laitevalmistajilta. Motivaationa tähän hankkeeseen on ollut urheiludatan hajanaisuus digitaalisessa maailmassa. Erilaiset datalähteet, kuten eri laitevalmistajat, urheilutulokset ja urheilijoiden omat muistiinpanot muodostavat arvokkaan datakokonaisuuden, jonka hyödyntäminen edellyttää datan keräämistä yhteen paikkaan. Tämä datan hajanaisuus korostuu erityisesti huippu-urheilun parissa, jossa urheilijoilla on monesti useiden eri laitevalmistajien mittauslaitteita, ja valmentajat käyttävät monia eri tapoja tallentaa havaintojaan. Datan hyödyntämisen kannalta on merkittävästi tehokkaampaa, jos kaikki relevantti data on tallennettuna yhteen keskitettyyn datavarastoon. On siis selkeä tarve mahdollistaa hajanaisen urheiludatan yhteenkerääminen. Tätä urheiludatan keräämiseen tähtäävää teknistä ratkaisua kutsutaan nimellä Urheilun Tietovaranto, eli UTV.

Dataa kerätään alustalle useista eri lähteistä. Näihin hajanaisiin lähteisiin kuuluvat erilaiset laitevalmistajat, kuten esimerkiksi Polar, Oura ja Garmin. Useimmat urheilulaitevalmistajat tarjoavat ohjelmointirajapinnan, jonka kautta käyttäjien autorisoimat tahot voivat hakea heidän dataansa. UTV-alusta hyödyntää näitä rajapintoja tapauksissa joissa niitä on tarjolla. Laitevalmistajien tarjoamien rajapintojen lisäksi alustalle tullaan kehittämään geneerinen käyttöliittymä minkä vain urheiludatan tallentamiseen. Tämä rajapinta olisi juuri henkilökohtaisia muistiinpanoja varten. Näitä ovat esimerkiksi Excel-taulukot ja csv-tiedostot.

Datan hallinnointi on merkittävä osa UTV-hanketta. Kun data ladataan laitevalmistajalta, tai

vastaanotetaan käyttäjältä, se pyritään säilyttämään mahdollisimman lähellä alkuperäistä tiedostomuotoaan. Valtaosa laitevalmistajilta ladattavasta datasta saadaan json-tiedostomuodossa, mikä helpottaa sen käsittelyä. Data pääasiallisesti lajitellaan tietokantaan sen omistajan ja päivämäärän mukaan helpottamaan myöhempiä tietokantahakuja. Olennainen osa datan hallinnointia on se, että jokainen käyttäjä voi kontrolloida omaa dataansa. Ilman toimivaa mekanismia datan pääsyoikeuksien hallintaan, ei olisi mahdollista toteuttaa alustaa ollenkaan. Alusta täyttää GDPR:n vaatimat asetukset datan hallinnan suhteen (European Commission 2016).

Datan hallinnointi on toteutettu järjestelmässä mahdollistamalla niinkutsuttujen dataresurssien luominen. Dataresurssi ei itsessään sisällä dataa, vaan se on vain osoite, joka määrittelee jonkin tietyn datakokonaisuuden. Tähän osoitteeseen sitten voidaan sallia pääsy käyttäjän itsensä valitsemille käyttäjille ja kohderyhmille. Kyseinen osoite sisältää tiedon siitä, mistä datasta on kyse ja miltä aikaväliltä. Lisäksi UTV-alusta tarjoaa mahdollisuuden datan hienojakoisempaankin hallinnointiin mahdollistamalla myös datan jakamisen rajoittamisen datatyyppien, kuten sykedatan tasolla. Datatyypit riippuvat laitevalmistajan rajapinnan tarjoamista datatyypeistä. Esimerkiksi Oura tarjoaa pääsyn käyttäjän jatkuviin sykemittauksiin ja unidataan. Konkreettisena esimerkkinä, dataresurssi voi siis olla esimerkiksi käyttäjän X Ouran sykedataa aikaväliltä 10.10.2022 - 10.11.2022. Tähän resurssiin käyttäjä X voi sitten sallia pääsyn käyttäjälle Y, ja yritykselle Z. Pääsyn dataresurssiin voi jakaa käyttäjä itse, tai joku joka haluaa pääsyn käyttäjän dataan voi lähettää käyttäjälle pyynnön.

Kohderyhmät UTV-hankkeelle on jaettu kolmeen osaan perustuen siihen milloin järjestelmä tulee käytettäväksi kyseiselle kohderyhmälle. Ensimmäinen kohderyhmä on kilpa- ja huippu-urheilu, sekä siihen liittyvä tutkimustyö. UTV-hanke on suunnattu tukemaan nimenomaan korkean tason urheilua ja sen valmennustoimintaa, tavoitteena parantaa urheilutuloksia. UTV-järjestelmän päälle rakennettavat sovellukset tarjoavat huippu-urheilun parissa työskenteleville laadukkaita työkaluja valmentamisen ja harjoittelun tehokkuuden lisäämiseen. Toinen kohderyhmä on aktiiviliikkujat. Tällä tarkoitetaan ihmisiä jotka eivät kuulu huippu-urheilun pariin, mutta liikkuvat aktiivisesti ja pitävät kirjaa suorituksistaan. Tälle ryhmälle UTV-järjestelmä suunnataan vasta siinä vaiheessa kun järjestelmä on havaittu toimivaksi ja hyödylliseksi ensimmäisen ryhmän (eli kilpa- ja huippu-urheilun) parissa. Kol-

mas ja viimeinen ryhmä kattaa kaikki muut, jotka eivät kuuluneet ensimmäisiin kahteen ryhmään. Tälle ryhmälle UTV-järjestelmän päälle rakennetut sovellukset voivat tarjota esimerkiksi työkaluja oman kehittymisen seurantaan ja ennustamiseen aktiivisemman elämäntavan aloitusvaiheessa, sekä yleisiä ohjeita kannattavimpiin harjoittelutapoihin henkilön omasta tilanteesta riippuen. Huolimatta siitä missä vaiheessa alustan kehitys etenee kohderyhmien näkökulmasta, alustalle pääsee rekisteröitymään avoimesta verkosta käsin kuka vain. Kuitenkin niin kauan kuin alusta on kehitysvaiheessa, datan säilytyksen suhteen ei anneta mitään takeita vaan tietokannat saatetaan nollata kehitysprosessin niin vaatiessa.

Alusta on rakennettu noudattamaan mikropalveluarkkitehtuuria. Datan tallennukseen ja hakemiseen liittyvät komponentit on eriytetty omiksi mikropalveluikseen, ja samalla tavalla muutkin toiminnallisuudet kuten käyttäjänhallinta, työjonot ja tietokanta ovat omia palveluitaan alustalla. Jokainen palvelu on sijoitettu omaan konttiinsa, ja ne kommunikoivat keskenään HTTP kutsuilla REST-rajapintojen kautta. Joissakin tapauksissa, kuten työjonot, tarkka kommunikaatioprotokolla on abstrahoitu valmiiden kirjastojen taakse, jotka pitävät itse huolen konttien välisen kommunikaation yksityiskohdista.

Koneoppimista ei ole alustan perustoiminnallisuuteen toteutettu. Kuitenkin alustan rakenne tukee hyvin erilaisten mahdollisten koneoppimisputkien rakentamista sen päälle. UTV-alustalle rakennettu artefakti on kirjoitushetkellä ainoa palvelu alustalla, joka sisältää koneoppimistoiminnallisuutta. Myöhemmin alustan päälle kehitettävät muut artefaktin kaltaiset sovellukset saattavat sisältää lisää koneoppimista.

3.2 Tutkimusongelma ja tutkimuskysymys

Tutkimusongelma nousee esiin UTV-alustan tavoitteesta kyetä tuottamaan hyötyä loppukäyttäjille heidän datansa pohjalta, vaikka alustan tarkoitus on olla vain tietovaranto, jossa käyttäjät voivat jakaa toisilleen omaa dataansa. Pelkän raakadatan jakaminen ei kuitenkaan tarjoa uutta arvoa loppukäyttäjille. Todellinen arvo syntyy UTV-alustan päälle rakennettavista sovelluksista, jotka hyödyntävät alustalle tallennettua dataa. Johtuen datan suuresta määrästä, hyödyntäminen tapahtuisi pääasiassa koneoppimisen keinoin. Siten tutkimusongelma on kuinka UTV-alustan kaltaiselle alustalle voidaan toteuttaa koneoppimissovelluksia parhai-

ten.

Edellisen kohdan pohjalta tutkimuskysymys voidaan jakaa kahteen osaan:

1. Mitä hyötyjä ja haittoja mikropalveluarkkitehtuurista on koneoppimissovelluksille?
2. Mitä hyötyjä ja haittoja konttitekniikasta on koneoppimissovelluksille?

3.3 Tutkimusmenetelmä

Tutkimuksen menetelmäksi valittiin DSR menetelmä (DSRM - Design Science Research Methodology) johtuen sen erityisestä sopivuudesta artefaktien kehitykseen (Peffers, Tuunanen ja Niehaves 2018). DSR menetelmä kuuluu suunnittelutieteen menetelmien piiriin, joka on yksi informaatioteknologian erilaisista tutkimusparadigmoista (Patas ja Goeken 2011). Muihin menetelmiin suunnittelutieteiden parissa kuuluvat esimerkiksi IT suunnitteluteoria (IS design theory - ISDT), ja suunnitteluorientoitunut IT tutkimus (Design-oriented IS research - DOIS).

Suunnittelutieteen prosessi on menetelmästä riippumatta jaettavissa kahteen pääaktiviteettiin, jotka ovat rakentaminen ja arviointi. Muitakin aktiviteetteja on olemassa, mutta nämä kaksi ovat merkittävimmät. Rakennusvaiheessa artefaktia kehitetään kohti jotain ennaltamäärättyä tavoitetta, ja arviointivaiheessa pyritään selvittämään onko kyseiseen tavoitteeseen päästy. Artefaktia kehitettäessä rakentaminen ja arviointi tapaavat vuorotella ja vaikuttaa toinen toisiinsa, kun arviointivaihe antaa paremman ymmärryksen rakennusvaiheeseen asetetuista tavoitteista. Tarkat arviointimenetelmät ja -kriteerit vaihtelevat artefaktin luonteen ja menetelmän mukaan, ja riippuvat monista muista tekijöistä kuten ympäristöstä, johon artefakti on kehitetty(March ja Smith 1995).

Artefaktit voidaan jakaa neljään erilaiseen ryhmään käyttötapojen perusteella, jotka ovat rakenteet, mallit, metodit ja implementaatiot(Hevner ym. 2004). **Rakenteet** kattavat erilaiset symbolit ja ammattisanastot. Nämä ovat kohdealueesta riippuvia kokonaisuuksia, joita kyseisen kohdealueen ammattilaiset hyödyntävät esimerkiksi kuvaillessaan kohdealueeseen kuuluvaa ongelmaa. Rakenteiden ja konseptien avulla voidaan siis taata tehokas ja täsmällinen kommunikointi jollakin tietyllä kohdealueella. **Mallit** ovat abstrahointeja tai kuvauksia

rakenteiden välisistä suhteista. Malleihin kuuluvat esimerkiksi matemaattiset mallit simulaatioissa ja kuvakset kohdealueen asioiden välisistä suhteista. Mallit ovat siis yleisiä kuvauksia jostakin kohteesta siten, että kyseistä mallia voidaan käyttää kuvaamaan muitakin samankaltaisia kohteita. **Metodit** kattavat erilaiset algoritmit ja käytänteet ja nämä voivat myös olla suunnittelutieteen prosessin lopputuloksena. Metodit ovat vaiheittaisia ohjeita joiden avulla voidaan toteuttaa jokin ennaltamäärätty tehtävä. **Implementaatiot** ovat käytännön toteutuksia kohdeympäristössä. Näihin kuuluvat kokonaiset järjestelmät ja erilaiset prototyypit. Käytännön implementaatioiden tarkoituksena on demonstroida implementaatioon liittyvien metodien ja mallien toimivuus. Voi siis olla, että prosessin tuloksena syntyvä artefakti on jokin metodi ja käytännön implementaatio tehdään osoittamaan kyseisen metodin toimivuus. Huomiotavaa on, että artefakti voi kuulua useampaan ryhmään kuin vain yhteen(March ja Smith 1995). Tällöin tosin saattaa olla kyseessä tapaus, jossa on luotu kaksi artefaktia yhden sijaan. UTV-alustalle kehitetty artefakti kuuluu edellä luetelluista kategorioista implementaatioiden kategoriaan.

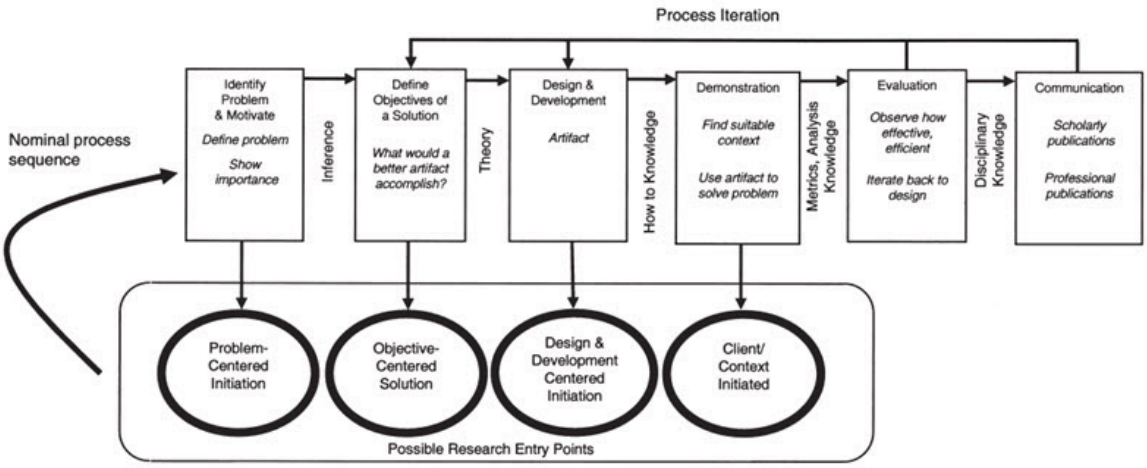
Artefaktien kehitykseen keskittyvä DSRM kehitettiin metodologiaksi suunnittelutieteen saralla, koska suunnittelutieteen parissa on ollut puutetta yleisesti hyväksytyistä toimintamalleista(Peffers ym. 2007). DSRM kehityksen tavoitteena oli tuottaa malli joka ohjaa tutkimusprosessia ja johtaa laadukkaisiin tuloksiin sekä tarjoaa mielikuvan siitä, miltä laadukkaan suunnittelutieteen tutkimuksen tulosten kuuluisi näyttää. Lisäksi tavoitteena oli että DSRM olisi yhteensopiva aiemman suunnittelutieteeseen itseensä liittyvän laadukkaan tutkimuksen kanssa ja hyödyntäisi kyseisiä tutkimustuloksia mallin rakenteessa.

DSRM malli koostuu useista eri aktiviteeteista sekä erilaisista mahdollisista tutkimustyön aloituspisteistä. Koska artefaktien kehittäminen on iteratiivista, mallissa voidaan palata myöhemmistä aktiviteeteista aikaisempiin aktiviteetteihin takaisin tarpeen niin vaatiessa. Mahdollisia aktiviteetteja on yhteensä kuusi:

1. **Ongelman määrittely ja rajaaminen** sekä siihen liittyvän tutkimustyön arvon perustelu.
2. **Vaatimusmäärittely** artefaktille ongelman kuvauksen pohjalta. Vaatimukset voivat olla kvantitatiivisia tai kvalitatiivisia, esimerkiksi järjestelmän suorituskyvyn kasvaminen (kvantitatiivinen).

3. **Artefaktin suunnittelu ja rakentaminen**.
4. **Artefaktin toiminnallisuuden demonstrointi**. Tässä voidaan hyödyntää esimerkiksi simulaatioita, tapaustutkimusta, tai formaalimpia todistusmenetelmiä.
5. **Artefaktin arviointi** sen perusteella kuinka hyvin se ratkaisee aiemmin määritellyn ongelman ja kuinka hyvin se täyttää aiemmin asetetut vaatimukset. Arvioinnissa voidaan hyödyntää mitä vain tapaukseen sopivia metriikoita, kuten suorituskykyä tai käytetyn muistin määrää.
6. **Tulosten kommunikointi** laajemmalle yleisölle. Tällä tarkoitetaan tulosten esittämistä kyseessä olevan tieteenalan käytänteiden mukaisesti.

Tutkimustyön mahdolliset aloituspisteet ovat kukin liitoksissa tiettyihin aktiviteetteihin. Tämä havainnollistaa sitä että eri tutkimuksilla voi olla eri lähtökohta tutkimustyössä. Esimerkiksi osa tutkimuksista alkaa ongelman havaitsemisesta, joka johtaa tutkimustyöhön (ongelmakeskeinen aloitus) ja osa alkaa jostain tarpeesta, joka nousee esille toimialan parista (tavoitelähtöinen aloitus). Seuraava kuva havainnollistaa edellämainittuja aktiviteetteja, aloituskohtia ja niiden välisiä polkuja:



Kuvio 2. Kuvassa näkyy DSRM-prosessimallin eri vaiheet, mukaanlukien prosessin eri aloituspisteet(Peffers ym. 2007).

Kuvan perusteella voidaan todeta tässä tutkimuksessa kehitettävän artefaktin lähtökohdan olevan sovelluskehityskeskeinen (Design and Development Centered Initiation). Siten ensimmäinen askel artefaktin kehityksessä tulisi olla sen toteutus. DSRM prosessimalli ei kui-

tenkaan kiellä tekemästä ensin vaatimusmäärittelyä artefaktille, mikä onkin tarpeen tehdä, jotta myöhemmin voidaan arvioida artefaktin kehityksen onnistuneisuutta.

4 Toteutus

Tässä luvussa kuvataan artefaktin kehitysprosessi DSRM-prosessimallin mukaan. Artefaktille suoritetaan ensin vaatimusmäärittely, jossa pyritään vastaamaan kysymykseen siitä miltä valmiin artefaktin tulisi näyttää ja mihin sen tulisi kyetä. Sen jälkeen suoritetaan artefaktin varsinainen toteutus, ja kuvataan lyhyesti toteutuksen eri vaiheet. Kolmantena on kuvaus toteutetun artefaktin rakenteesta ja sen tarjoamasta toiminnallisuudesta, ja lopuksi arvioidaan täyttääkö tuo toiminnallisuus artefaktille asetetut vaatimukset.

4.1 Vaatimusmäärittely

Vaatimussuunnittelu on ohjelmistosuunnittelun osa-alue, jossa tavoitteena on selvittää toteutettavaan kokonaisuuteen kohdistuvat vaatimukset. Nämä vaatimukset vaihtelevat hyvin korkean ja abstraktin tason vaatimuksista matalan tason tarkkoihin teknisiin vaatimuksiin. Sovelluskehitykseen liittyvät eri sidosryhmät omaavat erilaisia vaatimuksia ja kommunikoivat niitä eri tavoin. Vaatimussuunnittelun tavoitteena on siis kerätä kaikki eri sidosryhmien vaatimukset yhdeksi koherentiksi kokonaisuudeksi, joka määrittelee kuinka tavoite saavutetaan. Tavoitteet ja vaatimukset eivät ole samoja. Tavoitteet kuvaavat päämäärän, johon luotavalla kokonaisuudella pyritään, kun taas vaatimukset kertovat miten tähän tavoitteeseen päästään. (Laplante 2018)

UTV-alustan tapauksessa artefaktin kehityksessä ei ole erillisiä sidosryhmiä kehitysprojektin henkilöstön ulkopuolella, joten tavoitteiden ja vaatimusten selvittäminen on helpohko prosessi. Tavoitteena on yksinkertaisesti tuottaa artefakti, jonka kehityksen pohjalta pystytään vastaamaan aiemmin esitettyihin tutkimuskysymyksiin mikropalveluarkkitehtuurin ja konttiteknologian hyödyistä ja haitoista AI/ML sovelluksille. Artefaktin vaatimukset puolestaan syntyvät alustan käyttötarkoituksen pohjalta, ja siitä minkälainen artefakti voisi yhtäaikaa saavuttaa edellisen tavoitteen tutkimuskysymykseen vastaamisesta, ja samalla tuomaan jotain hyödyllistä ja uutta UTV-alustalle urheiludatan visualisaation muodossa.

Alla oleva vaatimusdokumentti on muokattu versio IEEE 29148 mukaisesta vaatimusdokumentista. Dokumentin päätarkoitus on huolellinen vaatimusten tarkastelu ennen artefak-

tin toteuttamista siten vähentäen tarvetta muuttaa artefaktin suunnitelmaa myöhemmin (ks. Laplante 2018, luku 4). Dokumentin rakennetta on muokattu, koska jokaista aspektia, mitä IEEE 29148 mallidokumentti käsittelee, ei ole tarpeen ottaa huomioon tässä kontekstissa (esimerkiksi kustannukset).

1. **Johdanto**

   Tässä dokumentissa kuvatut vaatimukset kohdistuvat UTV-alustalle kehitettävään artefaktiin, ja siten UTV-alustan asettamat vaatimukset yhdessä tutkimuskysymyksen kanssa määrittävät artefaktiin kohdistuvat vaatimukset. Toteutettava artefakti tulee olemaan yhdestä tai useammasta mikropalvelusta koostuva kokonaisuus joka täyttää tässä dokumentissa listatus vaatimukset.

   1.1. Tarkoitus

   Artefaktin tarkoituksena on vastata tutkimuksessa esitettyihin tutkimuskysymyksiin mikropalveluarkkitehtuurin ja konttiteknologian hyödyllisyydestä koneoppimissovelluksille.

   1.2. Laajuus

   Kehitettävä artefakti tulee olemaan yhden tai useamman mikropalvelun muodostama kokonaisuus UTV-alustalla, joka sisältää tarvittavan toiminnallisuuden urheiludatan visualisaation toteuttamiseen. Tähän toiminnallisuuteen kuuluu sekä verkkosivu, joka toimii artefaktin käyttöliittymänä, että palvelu joka suorittaa visualisaation vaatiman laskennan. Tämä kokonaisuus tarjoaa rekisteröityneelle käyttäjälle mahdollisuuden visualisoida omaa urheiludataa selaimessa.

   1.3. Käytetyt lyhenteet

   UTV – Urheilun Tietovaranto.

   UMAP – Uniform Manifold Approximate Projection

1.4. Yleiskatsaus

1.4.1. Artefaktin perspektiivi

Artefakti tulee olemaan kiinteä osa UTV-alustaa, mutta se ei kuitenkaan ole osa ydintoiminnallisuutta. Se myös demonstroi sitä, kuinka alustan toiminnallisuutta voi hyödyntää rakennettaessa sovelluksia sen päälle. Esimerkiksi käyttäjänhallinta on UTV-alustan tarjoama palvelu, jota artefakti tulee hyödyntämään.

1.4.2. Artefaktin toiminnallisuudet

Artefakti kykenee visualisoimaan Ouran pilvipalvelusta UTV-alustalle ladattavaa urheiludataa hyödyntäen visualisaation tuottamisessa UMAP-algoritmia. Laskenta tapahtuu palvelimella, josta laskennan tulos siirretään käyttäjälle selaimeen. Artefakti piirtää selaimeen visualisaation palvelimelta haetun datan pohjalta.

1.4.3. Artefaktin käyttäjäryhmät

Artefaktin käyttö on rajoitettu ainoastaan UTV-alustalle rekisteröityneille käyttäjille.

1.4.4. Rajoitteet

Artefaktiin kohdistuu rajoitteita UTV-alustan teknisistä vaatimuksista, käyttöehdoista sekä GDPR:n asettamista vaatimuksista.

1.4.5. Oletukset ja riippuvuudet

Artefakti ei kykene toimimaan itsenäisesti ilman UTV-alustaa. Se on riippuvainen sen tarjoamista rajapinnoista ja käyttäjänhallinnasta. Lisäksi urheiludatan saatavuus riippuu kolmesta tekijästä. Ne ovat Ouran APIn toimivuus ja jatkuva olemassaolo, sekä se että käyttäjällä on aktiivisessa käytössä Ouran mittauslaite. Oletuksiin sisältyy myös UTV-alustan käyttämän palvelimen keskeytyksetön ja häiriötön toiminta.

2. Tarkat vaatimukset

2.1. Ulkopuoliset rajapinnat

Kontin näkökulmasta ulkopuolisiin rajapintoihin kuuluvat sen käyttämät rajapinnat UTV-alustan sisällä. UTV-alustan ulkopuolisia rajapintoja artefaktin ei tule käyttää. UTV-alustalla on oma rajapinta käyttäjän datan hallintaan, hakemiseen ja lataamiseen. Artefaktin tulee hyödyntää UTV-alustan rajapintaa sen sijaan, että toteuttaisi kyseiset toiminnallisuudet itse.

2.2. Toiminnallisuuden vaatimukset

Artefakti kykenee tuottamaan visualisaation Ouran pilvipalvelusta UTV-alustalle ladatun yksittäisen käyttäjän urheiludatan pohjalta. Visualisaation tulee olla käyttäjän ymmärrettävissä ilman erillistä koulutusta.

2.3. Käytettävyyden vaatimukset

Artefaktin verkkokäyttöliittymän tulee olla käyttökelpoinen eri näyttökokojen kanssa, myös mobiiililaitteiden. Käyttökelpoisuudella tarkoitetaan tässä vain sitä, että sivun komponentit skaalautuvat ja asettuvat eri näyttökoilla tarpeen mukaan. Käyttökelpoisuudella ei viitata mihinkään kiinteään metriikkaan, jolla sivun laatua mitattaisiin.

2.4. Suorituskyvyn vaatimukset

Artefaktina kehitetään mikropalvelukokonaisuus konttiympäristössä, mutta sitä ei tulla toteuttamaan tavalla, jossa se skaalautuisi suuren käytön alla. Tämä on suoraa seurausta UTV-alustan teknisestä tilasta, jossa skaalautuvuus ei kuulu tämänhetkisen version ominaisuuksiin.

Koneoppimisalgoritmin käyttöä rajataan yhteen laskentatehtävään per mikropalvelu kerrallaan suoritustehon säästämiseksi. Tätä varten hyödynnetään alustan ominaisuuksia siten, että peräkkäiset visualisointipyynnöt tallennetaan työjonoon, josta koneoppimisalgoritmista vastaava prosessi ottaa aina seuraavaan laskentatehtävän suoritettavaksi, kun edellinen on valmis. Kuitenkin laskentaa tekeviä identtisiä mikropalveluja tulee olla enemmän kuin yksi siten että ne toimivat rin-

nakkain tarpeen niin vaatiessa.

2.5. Tietokantaan ja tallennustilaan kohdistuvat vaatimukset

Visualisaatiossa käytettävää raakadataa ei tallenneta levylle sellaisenaan, vaan se säilyy ajonaikaisessa muistissa siihen asti, että visualisaatio on valmis. Valmiin visualisaation data säilyy palvelimella, kunnes käyttäjä itse poistaa sen.

Artefaktista itsestään ei kohdistu alustan tietokantaan uusia vaatimuksia. Tietokantaan tallennettu Ouran pilvipalvelusta saatu urheiludata esikäsitellään artefaktin toimesta muotoon, jossa se on UMAP-algoritmin hyödynnettävissä.

2.6. Rajoitteet

Artefaktin tulee noudattaa samoja periaatteita kuin muukin UTV-alusta. Sillä ei siis saa esimerkiksi pystyä tarkastelemaan muiden käyttäjien dataa ilman lupaa, eikä se saa välittää dataa kolmansille osapuolille mitään tarkoitusta varten. Artefakti ei myöskään saa tehdä mitään tietokantaoperaatioita alustan sisällä suoraan. Käytännössä tämä tarkoittaa, että artefaktiin kuuluvat mikropalvelut saavat kommunikoida artefaktin ulkopuolisten mikropalvelujen kanssa vain ja ainoastaan UTV-alustan tarjoamien rajapintojen kautta.

Toinen rajoite on se, että kontissa käytettävän koneoppimisalgoritmin suorituksen tulee tapahtua palvelimen puolella kontin sisällä eikä esimerkiksi visualisointia käyttävän käyttäjän selaimessa.

3. **Verifikaatio**

Artefaktin toimivuutta tarkastellaan vertaamalla sen ominaisuuksia ja toimintoja vaatimuksiin, joita on esitetty tässä vaatimusdokumentissa. Mikäli artefakti täyttää tässä listatut vaatimukset ja on toimiva kokonaisuus, se tulkitaan onnistuneeksi.

4.2 Suunnittelu ja toteutus

Artefaktin toteuttaminen alkoi hieman vaatimussuunnittelun valmistumisen jälkeen, kun UTV-alustan kehityksestä vapautui riittävästi aikaa. Artefaktin kehityksen ohella myös muita alustan toiminnallisuuksia kehitettiin, joten aika jakautui artefaktin ja alustakehityksen välillä. Artefaktille ei sen pienen kokoluokan johdosta tehty erillistä suunnitelmaa. Lisäksi artefaktin käyttämät teknologiat määräytyvät vaatimuksista suoraan, joten tarkalle suunnitelmalle ei myöskään ollut tarvetta. Kokonaisuudessaan artefaktin toteutukseen kului aikaa hieman reilu kuukausi.

Toteutus eteni siten, että ensin UMAP-algoritmin käyttöä testattiin lokaalisti kiinteällä datasetillä. Data oli Ouran pilvipalvelusta UTV-alustalle ladattua urheiludataa. Syynä datasetin rajaamiseen oli se, että oli helpompaa laatia Ouran dataan sopiva esikäsittely kiinteällä datasetillä ilman UTV-alustan asettamia rajoituksia. Testauksen seurauksena saatiin riittävä ymmärrys siitä mitä vaaditaan algoritmin toteutukseen UTV-alustalla. Sitten toteutettiin passiivinen käyttöliittymä artefaktia varten ja tämän aikana päädyttiin lopulta käyttämään Pythonin Dash-kirjastoa käyttöliittymän toteuttamiseen. Syynä tähän valintaan oli Dash-kirjaston kyvykkyys luoda automaattisesti laadukkaita käyttöliittymiä visualisaatioita varten. Lisäksi kyseiset käyttöliittymät skaalautuvat automaattisesti, jolloin kyseisiä ominaisuuksia ei tarvinnut alkaa käsin toteuttamaan käytettävyyden takaamiseksi.

Tässä vaiheessa artefakti koostui yhdestä mikropalvelusta, joka oli käyttöliittymä artefaktille. Sen jälkeen lisättiin uusi mikropalvelu, jonka roolina oli suorittaa varsinainen laskenta visualisaatiota varten. Aluksi kyseinen palvelu ohitti UTV-alustan rajapinnat niiden puutteiden vuoksi dataa haettaessa. Puutteiden korjauksen jälkeen palvelu päivitettiin käyttämään datahauissa vain valmista rajapintaa alustalla. Lisäksi havaittiin, että käyttäjän kannalta on tarpeettoman monimutkaista sallia visualisoitavien datasettien määrittely. Tämän takia käyttöliittymä päivitettiin sellaiseksi, että voidaan visualisoida vain ennaltamäärättyjä datatyyppejä Ouran datasta.

Artefakti päivitettiin käyttämään alustan tarjoamia työjonoja siten, että jokainen visualisointipyyntö menee ensin työjonoon, josta laskentapalvelu poimii sen ja palauttaa sen käyttöliittymäpalvelulle toisen jonon kautta. Tämän jälkeen laskentakonttien määrä nostettiin kol-

meen. Lisäksi mallien ja visualisaatioiden laskeminen eriytettiin eri tehtäviksi laskentapalveluissa ja mallien laskenta ajoitettiin tapahtumaan uusien käyttäjien saapuessa käyttämään artefaktia sekä tietyin ennaltamäärätyin väliajoin. Vaatimusmäärittelystä poiketen jonoihin lisättiin myös prioriteettiominaisuus. Jonojen priorisointi mahdollistaa yhden laskentatehtävän priorisoimisen toisen edelle.

Artefaktin kehitys jätettiin pisteeseen, jossa se on täysin integroituna alustaan ja kykenee visualisoimaan käyttäjän oura-dataa käyttäjälle itselleen. Kehitys saattaa jatkua tulevaisuudessa, mikäli artefakti halutaan siirtää osaksi jotain suurempaa koneoppimisputkea tai se halutaan päivittää demonstraatiotarkoituksia varten.

4.3 Artefaktin kuvaus

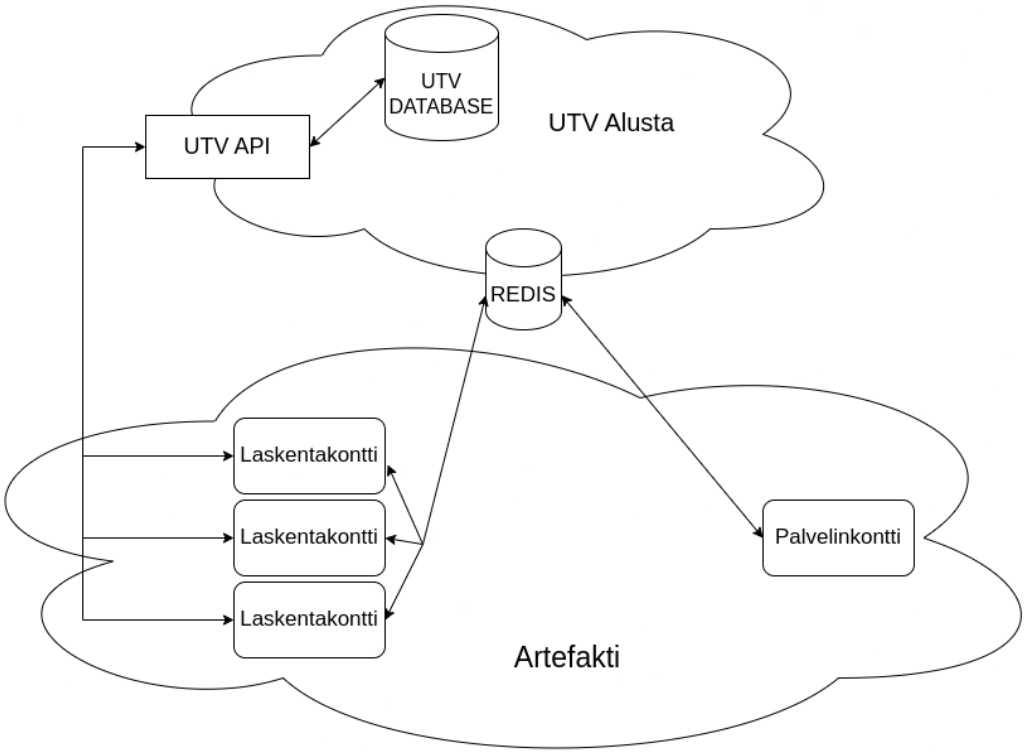
Kehitetty artefakti koostuu kahdesta mikropalvelusta, jotka hyödyntävät UTV-alustan tarjoamia eri palveluja kuten työjonoja ja käyttäjänhallintaa. Ensimmäinen mikropalvelu on palvelin, jonka vastuulla on tarjoilla sovelluksen verkkosivu sovelluksen käyttäjille ja asettaa visualisointitehtäviä työjonoon laskentaa varten. Toinen on laskentapalvelu, jonka tehtävänä on suorittaa UMAP-algoritmi annetuilla parametreilla.

Artefaktin käyttöliittymästä vastaava palvelu käyttää pythonilla toimivaa Dash kirjastoa verkkosivujen tarjoamiseen (“Dash docs” 2023). Kun käyttäjä verkkosivuilla valitsee, mitä haluaa visualisoitavan, käyttöliittymäpalvelu laittaa kyseisen visualisointitehtävän työjonoon. Työjonot on toteutettu UTV-alustalla Keydb:n avulla (“Keydb docs” 2023). Kun laskentatehtävä on laitettu työjonoon, käyttöliittymäpalvelu jää kuuntelemaan vastausta toisesta työjonosta. Tällä tavoin laskentatehtävät saadaan toimitettua laskentapalveluille, ja laskentapalvelut osaavat palauttaa tehtävän vastauksen oikealle käyttäjälle.

Laskentapalvelun sisältäviä kontteja on artefaktissa 3 kappaletta (Ks kuvio 3 sivulla 29), ja ne ovat keskenään identtisiä. Palvelu on monistettu Docker Composen tarjoaman replicas-ominaisuuden avulla. Niiden rakenne sallii määrän muuttamisen dynaamisesti tarpeen mukaan, mutta UTV-alustan rajoituksista johtuen tätä ominaisuutta ei ole toteutettu artefaktiin. Jokainen laskentakontti on identtinen muiden kanssa. Laskentakontit kuuntelevat artefaktin työjonoa (REDIS-palvelu), ja poimivat tehtäviä työjonosta heti, kun sellaisia sinne il-

mestyy. Suoritettuaan laskentatehtävän ja palautettuaan vastauksen tehtävässä määriteltyyn vastausjonoon, laskentakontit jatkavat työjonon kuuntelua. Laskentakontit suorittavat kahdenlaisia tehtäviä. Ne laskevat malleja eri visualisointityypeille käyttäen hyödyksi kaikkien luvan antaneiden käyttäjien dataa, ja ne laskevat visualisaatiot olemassaolevien mallien avulla hyödyntäen visualisaatiota pyytäneen käyttäjän dataa. Luvan antamiseksi lasketaan tässä vaiheessa sovelluksen käyttö ja myöhemmin erillinen luvan antaminen (kuten OAuth2 prosessissa), kun sovellusten hallinta on toteutettu UTV-alustalle. Kun sovellus on saanut käyttäjältä luvan, se hakee UTV-alustalta käyttäjän kaiken Oura-datan, ja laskee sen pohjalta pyydetyn visualisaation. Kuvaus siitä miten palvelut kommunikoivat keskenään tuottaessaan käyttäjälle visualisaation on kuviossa 4 sivulla 34.

Alla on kuvaus siitä, mitä reittejä pitkin artefaktin ja UTV-alustan palvelut kommunikoivat keskenään:



Kuvio 3. Konttien ja alustan relaatiot.

Mallien laskeminen joka kerralla erikseen, erityisesti useamman käyttäjän datalla on varsin hidasta. Siksi jokaiselle visualisointityypille lasketaan tietyin väliajoin malli valmiiksi kaik-

kien sovelluksen käytettävissä olevien käyttäjien datojen pohjalta. Nämä valmiiksi lasketut mallit tallennetaan sitten UTV-alustan tietokantaan, josta ne voidaan lukea silloin, kun kyseisiä malleja tarvitaan. Tämä on nopeampaa kuin koko mallin laskeminen uudestaan joka kerta, kun käyttäjä pyytää uutta visualisaatiota.

Laskentapalvelu sisältää myös esikäsittelykomponentin Ouran datan esikäsittelyä varten, koska UMAP tarvitsee datan sellaisessa matriisimuodossa, joka sisältää vain reaalilukuja. Esikäsittelyä ei ole eriytetty omaan mikropalveluunsa. Ouran pilvipalvelusta voi tällä hetkellä ladata useita erilaisia datatyyppejä kuten sykedataa ja unidataa, ja kaikki eri datatyyppien sisältämä data ei ole numeerista. Tarkempi dokumentaatio mukaanlukien jokaisen datatyypin tarkka skeema löytyy Ouran verkkosivuilta (“Ouran V2 apikuvaus” 2023). Jokaisesta datatyypistä UMAP-algoritmiin syötetään vain sellaiset arvot, jotka ovat reaalilukuja tai muunnettavissa sellaisiksi järkevällä tavalla. Muuntamisella tarkoitetaan tässä sitä että jos ei-numeerisella datatyypillä on rajallinen ja ennaltamäärätty joukko arvoja, joita se voi saada, niin näille ennaltamäärätyille tyypeille asetetaan arvoksi kokonaisluku nollasta alkaen. Jotta UMAP algoritmin tulos on ymmärrettävissä, sille syötettävä matriisi esikäsitellään sellaiseksi, että jokainen rivi vastaa yhtä datapistettä, ja jokainen sarake jotain tiettyä ominaisuutta, kuten esimerkiksi päivittäistä kalorien kulutusta. Kun raakadatasta on muodostettu sopiva matriisi, jokainen matriisiin sijoitettu pystyrivi eli ominaisuus standardisoidaan. Tämä tarkoittaa sitä, että jokaiselle matriisin pystyriville suoritetaan operaatiot, joissa niiden keskiarvot ja varianssit asetetaan nollaan. Näiden toimenpiteiden lisäksi ei suoriteta muuta esikäsittelyä.

Artefaktissa on lisäksi laskentapalveluja varten toteutettu prioriteettijono, joka sallii joidenkin laskentatehtävien priorisoinnin toisten edelle. Tämä simuloi sitä mahdollisuutta, että järjestelmän itsensä aloittamat tehtävät, kuten mallien ajoittaiset päivittämiset, menevät käyttäjän aloittamien tehtävien edelle ja sitä, että sovelluksille on mahdollista toteuttaa hierarkia, jossa esimerkiksi maksavien käyttäjien pyynnöt priorisoidaan sovellusta ilmaiseksi käyttävien käyttäjien edelle. Prioriteettijonot toteuttavat tällä hetkellä vain mallien uudelleenlaskennan priorisoinnin käyttäjien edelle, eikä käyttäjien välille ole toteutettu minkäänlaista hierarkiaa.

Käytännössä siis artefakti toimii niin, että käyttäjän moniulotteinen Oura-data pudotetaan

kahteen tai kolmeen ulottuvuuteen UMAP-algoritmin avulla, ja kyseinen pudotettu data visualisoidaan selaimessa. Artefaktin toiminnallisuutta voidaan kuvata myös 4+1 kuvausmallilla, jossa artefaktin rakennetta tarkastellaan viiden eri näkymän kautta:

1. **Looginen näkymä**. Artefakti tarjoaa käyttäjille käyttöliittymän Pythonin Dash kirjaston avulla. Käyttöliittymästä he voivat valita, mitä he haluavat visualisoitavan (Ks kuvio 5 sivulla 35). Visualisointipyyntö välittyy selaimesta palvelimelle, ja visualisaatio piirtyy heti, kun palvelin on suorittanut sen vaatiman laskentatehtävän.

2. **Prosessinäkymä**. Artefaktin käyttöliittymän toiminnallisuus sijaitsee eri kontissa kuin visualisaation vaatima laskentatoiminnallisuus. Käyttöliittymäkontti laittaa työjonoon visualisointitehtävän käyttäjän pyynnöstä, jonka jokin kolmesta identtisestä laskentakontista suorittaa. Sen jälkeen vastaus palautetaan käyttöliittymäkontille, joka puolestaan päivittää visualisaation käyttäjän selaimeen.
   Lisäksi järjestelmä ajaa automaattisia päivityksiä visualisaation vaatimiin malleihin tietyin väliajoin. Tämä ajastettu laskenta tapahtuu aina käyttöliittymäkontin aloitteesta, ja kuten aiemmassa skenaariossa laskentakontit suorittavat mallin päivityksen heti, kun ovat vapaana.
   Työjonot sisältävät priorisointiominaisuuden, josta seuraa, että järjestelmän aloittamat mallin päivitykset suoritetaan ennen käyttäjien antamia laskentatehtäviä.

3. **Kehitysnäkymä**. Artefaktissa on eriytetty osat siten, että se mahdollistaa Ouran tarjoamien datatyyppien päivittämisen artefaktiin helposti silloin, kun uusia datatyyppejä tulee Ouralle. Esityskerroksen hoitaa käyttöliittymäkontti, sovelluskerroksen ja datakerroksen hoitaa laskentakontit. Muiden datalähteiden kuin Ouran lisääminen artefaktiin onnistuu myös lisäämällä ne osaksi esikäsittelykomponenttia ja määrittelemällä sen, kuinka uudesta datasta voidaan muodostaa UMAP-algoritmille kelpaava matriisi.

4. **Fyysinen näkymä**. Koko artefakti sijaitsee samalla yksittäisellä palvelimella virtuaalikoneen sisällä, missä muu UTV-alusta myös sijaitsee. Artefakti koostuu useasta kontista, jotka eivät sijaintinsa puolesta eroa muista UTV-alustan konteista.

5. **Skenaariot**. Normaali käyttötilanne artefaktissa on tilanne, jossa käyttäjä UTV-alustan kautta kirjautuu sisään, pyytää visualisaation ja artefakti kommunuikoi UTV-alustan kanssa saadakseen tarpeellisen datan visualisaation tuottamiseksi (Ks kuvio 4 sivulla 34).

4.4 Artefaktin arviointi

Artefaktin arviointi suoritetaan tarkastamalla täyttyvätkö vaatimusdokumentin eri osioissa esitetyt vaatimukset sekä testaamalla artefaktin toimivuus käytännössä.

Vaatimusten täyttyminen, kategoriat ovat suoraan vaatimusdokumentin luvusta 2:

- **Ulkopuoliset rajapinnat**. Artefakti ei käytä UTV-alustan ulkopuolisia rajapintoja, vaan hyödyntää ainoastaan alustan tarjoamia sisäisiä rajapintoja. Tältä osaa vaatimukset siis täyttyvät.

- **Toiminnallisuus**. Artefakti tuottaa visualisaation Ouran datan pohjalta sille käyttäjälle, joka sovellusta sillä hetkellä käyttää (Ks kuvio 5 sivulla 35). Tuotettu visualisaatio ei kuitenkaan ole selkeästi ymmärrettävissä, joten toiminnalisuuden osalta kaikki vaatimukset eivät täyty.

- **Käytettävyys**. Dash takaa riittävän käyttökelpoisuuden eri kokoisilla näytöillä ja eri laitteilla automaattisesti, joten tämä vaatimus täyttyy.

- **Suorituskyky**. Artefakti toimii mikropalveluympäristössä ilman skaalautuvuusominaisuuksia ja jokainen laskentakontti suorittaa korkeintaan yhtä laskentatehtävää kerrallaan. Kaikki suorituskykyyn liittyvät vaatimukset siis täyttyvät.

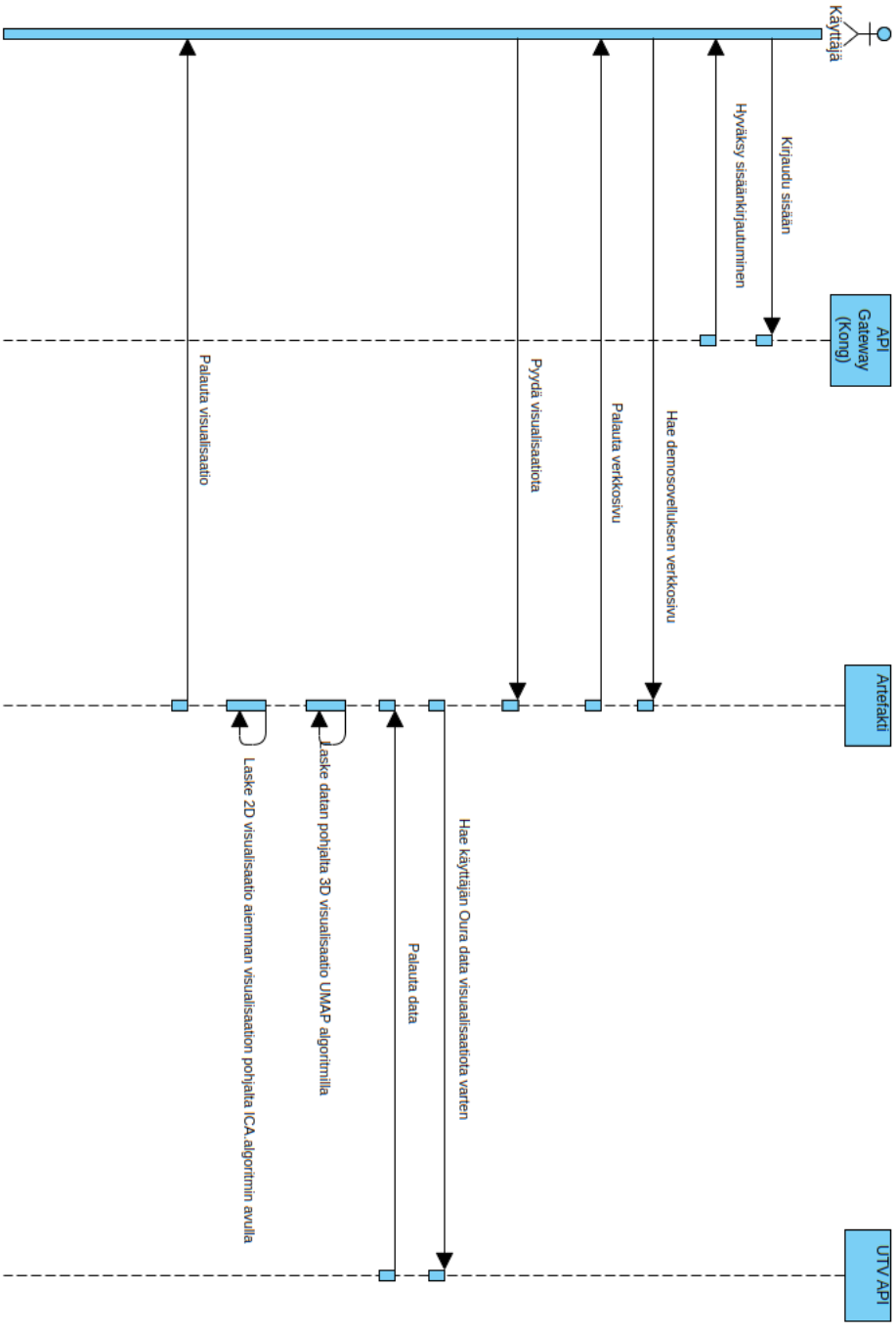
- **Tietokanta ja tallennustila**. Vaatimukset täyttyvät lukuunottamatta visualisaatioiden tallentumista palvelimelle. Tätä ominaisuutta ei missään vaihessa artefaktiin toteutettu koska se olisi tuonut tarpeetonta monimutkaisuutta käyttöliittymään.

- **Rajoitteet**. Artefakti ei noudata kaikkia rajoitteita UTV-alustan sisäisten rajapintojen osalta. Tämä johtuu siitä, ettei artefaktin kehityksen aikana UTV-alustalla ollut vielä toimivaa sovellushallintaa. Tästä seurasi, että joidenkin sovelluksen toimivuuden kannalta elintärkeiden toimintojen toteuttaminen vaati sellaisten alustan sisäisten apukirjastojen käyttöä, jotka eivät lopullisessa alustan versiossa tule olemaan missään nimessä sovelluskehittäjien suorassa käytössä. Tätä UTV-alustasta itsestään johtuvaa puutetta lukuunottamatta rajoitteisiin liittyvät vaatimukset täyttyvät.

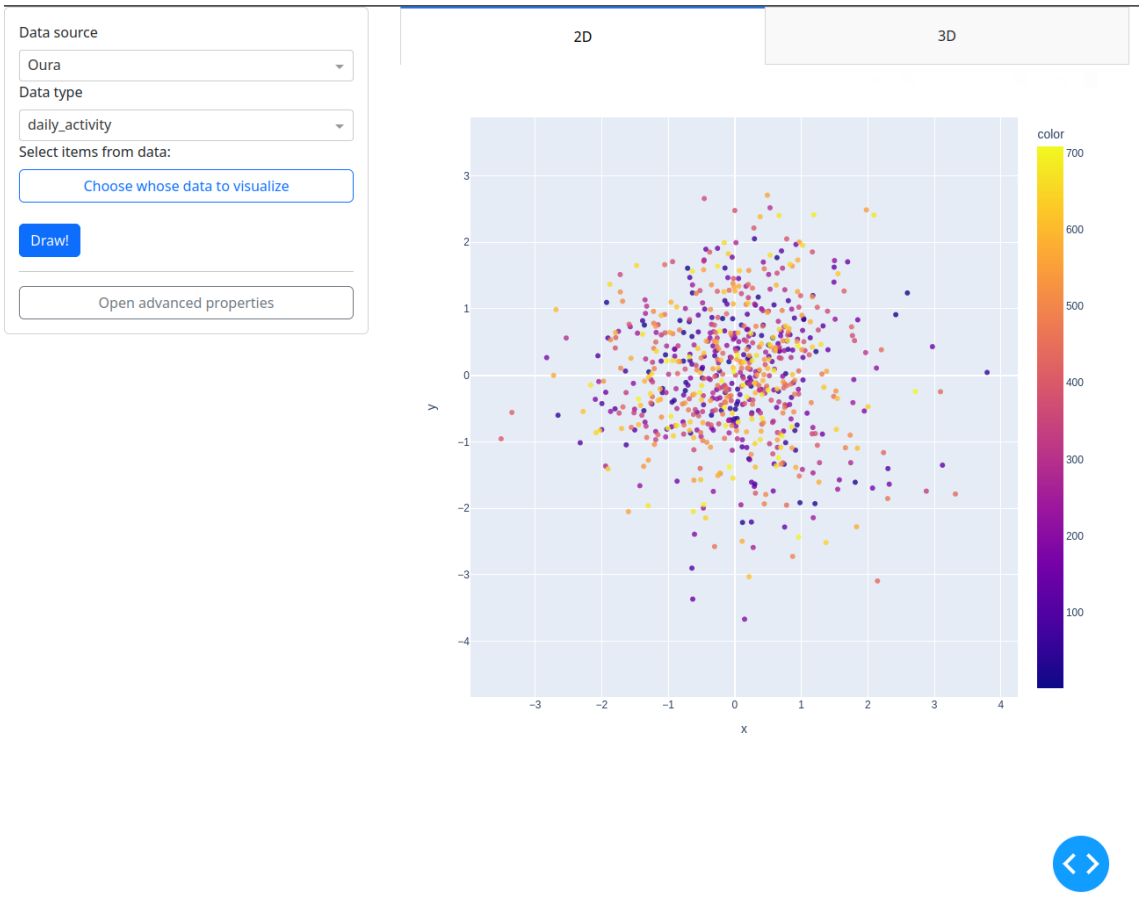
Vaatimusten osalta voidaan siis havaita että artefaktissa on pieniä puutteita, ja lisäksi käytännön syistä on jouduttu toimimaan toisin kuin vaatimusdokumentissa on esitetty.

Artefaktille tehtiin lyhyt testaus. Sen käyttöliittymän normaali käyttäminen ei aiheuta ajonaikaisia virheitä, joten artefakti on normaaliolosuhteissa toimiva. Mikäli käyttäjä toimii tarkoituksella väärin, eli yrittää käyttää artefaktia ilman että on rekisteröitynyt alustalle, se aiheuttaa ajonaikaisen virheen ilman kunnollista virheenkäsittelyä. Se ei kuitenkaan kaada artefaktia. Käytettävyyden testauksen jälkeen mikropalvelujen toimivuutta testattiin asettamalla työjonoon useita laskentatehtäviä yhtäaikaa simuloimaan sekä suuria käyttäjämääriä että eri tasoisten käyttäjien tehtävien priorisointia. Tältä osaa artefaktin toiminnassa ei havaittu mitään puutteita.

Yhteenvetona voidaan siis todeta, että artefaktissa on pieniä puutteita sekä vaatimusten että toimivuuden kanssa. Puutteet ovat kuitenkin pieniä eivätkä estä artefaktia toimimasta tarkoitetulla tavalla. Artefakti voidaan todeta tutkimuksen kannalta riittävän onnistuneeksi.



Kuvio 4. Normaalia käyttöä havainnollistava kuva.



Kuvio 5. Artefaktin verkkokäyttöliittymä.

5 Pohdinta

Artefaktin kehityksessä voitiin havaita, että taustakirjallisuuden väitteet tutkimuskysymyksiin liittyen pitävät paikkansa. Jokaiseen löydetyistä väitteistä ei kuitenkaan voida ottaa kantaa, koska kaikkia esitettyjen hyötyjen ja haittojen osa-alueita ei työssä tullut vastaan.

Väitteisiin, joihin ei artefaktin kehityksen nojalla voida ottaa kantaa, lukeutuvat tietoturva, tehokkuus ja tehottomuus. Jokainen näistä vaatisi erityistä vertailua tai jonkin erityisen tilanteen, jossa kyseinen aspekti esiintyisi. Esimerkiksi, artefaktin kehityksen aikana ei esiintynyt tilannetta, joka olisi antanut näkökulmaa tietoturvaan. Mitä taas tulee konttien tehokkuuteen virtuaalikoneisiin verrattuna, tähän ei voida ottaa kantaa ilman erillistä vertailua. Sama tilanne on tehottomuuden kanssa; ilman erillistä monoliittista versiota sekä artefaktista että UTV-alustasta ei voida sanoa kumpi arkkitehtuuri olisi tehokkaampi.

Muihin väitteisiin voidaan ottaa kantaa edellä olevaa perustellummin:

1. **Hyödyt**

    - **Soveltuvuus**. Ottaen huomioon eri kirjastojen suuren määrän, joita sekä artefakti että UTV-alusta vaativat toimiakseen, on ehdottomasti helpoin vaihtoehto asentaa kohdelaitteelle Docker Engine, joka hoitaa konttien itsensä asennusprosessin itsenäisesti. Lisäksi konttiympäristön etuna on, ettei artefaktin kehityksessä tarvinnut ottaa ollenkaan huomioon eri suoritusympäristöjä.

    - **Skaalautuvuus**. Vaikka dynaamista skaalausta ei alustalla ole käytössä, lisättiin artefaktissa olevien laskentakonttien määrää Docker Composen "replicas-komennon avulla helposti. On selvää että palvelujen dynaaminen skaalaus on merkittävästi helpompaa kontteihin sijoitettujen mikropalvelujen avulla kuin ilman.

    - **Vaatimusten hallinta ja ylläpito**. Tämä pitää paikkansa erityisen hyvin. Artefaktia pystyttiin kehittämään osana UTV-alustaa ilman, että alustan muita osia tarvitsi käynnistää uudelleen artefaktia päivitettäessä. Koska artefaktin vaatimat kirjastot ovat eristyksissä alustan muista osista, vältyttiin erilaisilta konflikteilta myös.

- **Viansieto**. Artefaktin kehityksen aikana laskentakontit kaatuivat useamman kerran virheitä etsittäessä, mutta tämä ei kuitenkaan vaarantanut alustan toimintaa millään tavalla.

2. **Haitat**

   - **Kompleksisuus**. Mikropalvelujen ja konttien käyttö tuo ehdottomasti oman lisänsä kehitettävän artefaktin kompleksisuuteen. Esimerkiksi datan siirto järkevällä tavalla kontista toiseen vaatii aina enemmän harkintaa ja tekemistä kuin vastaava tilanne monoliittisessa arkkitehtuurissa. Onnistunut sovelluskehitys MSAn ja konttien parissa vaatii aiempaa kokemusta molemmista.

   - **Tarpeettomuus**. Artefakti itsessään olisi voitu kehittää alustasta erillisenä, pienenä monoliittisena sovelluksena. Tällöin kehitys olisi ollut hieman helpompaa ja nopeampaa. Jos taas ajatellaan suurempaa esimerkkiä, kuten itse UTV-alustaa, on selvää, että mikropalveluarkkitehtuuri ja konttiympäristö ovat tarpeen. Toisin sanoen alkuperäinen väite siitä, että pienissä sovelluksissa monoliittinen arkkitehtuuri on parempi, pitää todennäköisesti paikkansa.

Kirjallisuudessa esitetyt väitteet eivät ole ristiriidassa artefaktin kehityksessä havaittujen tilanteiden kanssa. Tutkimuskysymyksiin saatiin vastaus suoraan kirjallisuuden pohjalta, ja vastaus vahvistettiin toteuttamalla koneoppimista hyödyntävä artefakti mikropalveluarkkitehtuuriin. Lisäksi voidaan väitteiden ja artefaktin kehityskokemuksen pohjalta sanoa, että mikropalveluarkkitehtuurin ja konttiteknologian hyödyt ylittävät haitat pienemmissä koneoppimistehtävissä. Erittäin suurissa laskentatehoa vaativissa tehtävissä voi kuitenkin olla tehokkaampaa käyttää monoliittista arkkitehtuuria. Esimerkkinä juuri tästä on reunalaskennan sovellutukset, jossa pienet laskentatehtävät voidaan suorittaa reunalla, kun taas suuremmat tehtävät suoritetaan pilvessä keskitetysti.

Myös artefaktin toteutuksessa olisi voinut hyödyntää reunalaskentaa. Käyttäjän pyytämä visualisaatio voitaisiin laskea suoraan selaimessa vähentäen siten palvelimen työtaakkaa. Tämä vaatisi kuitenkin visualisaation tarvitseman mallin kopioinnin palvelimen tietokannasta

käyttäjän laitteelle, ja sen päivittämisen tarpeen mukaan, mikä ei enää olisi käytännöllistä. Kuitenkin, jos UTV-alustaa tullaan joskus kehittämään hajautetunpaan suuntaan, tällainen versio kehitetystä artefaktista voi tulla ajankohtaiseksi.

Jos tämän tutkimuksen tulosten luotettavuutta haluttaisiin parantaa, tulisi valmistaa kaksi artefaktia, jotka molemmat toteuttavat identtisen koneoppimistehtävän. Toinen pohjautuisi mikropalveluarkkitehtuuriin ja olisi toteutettu konttiteknologian avulla, kun taas toinen olisi monoliittinen sovellus. Näiden artefaktien erot sitten mitattaisiin tarkasti, jolloin voitaisiin varmuudella sanoa, mitä eroa ja kuinka paljon näillä eri arkkitehtuureilla on koneoppimissovellusten suhteen.

6 Yhteenveto

Tutkimuksessa tarkasteltiin mikropalveluarkkitehtuurin ja konttiteknologian hyötyjä ja haittoja koneoppimissovellusten kehitykselle verrattuna monoliittiseen arkkitehtuuriin. Tutkimus suoritettiin kolmessa osassa.

Ensin etsittiin taustakirjallisuudesta tietoa MSAn ja konttiteknologian hyödyistä ja haitoista. Toiseksi, kehitettiin artefakti noudattaen DSRM prosessimallia. Artefakti kehitettiin UTV-alustalle, joka noudatta mikropalveluarkkitehtuuria ja hyödyntää konttiteknologiaa. Artefaktin kehitysprosessissa havaitut hyödyt ja haitat ovat siten vertailukelpoisia taustakirjallisuuden väitteiden kanssa. Kolmanneksi, artefaktin kehityksessä havaittuja hyötyjä ja haittoja verrattiin taustakirjallisuudessa esitettyihin.

Artefaktissa hyödyiksi havaittiin kehitettävien sovellusten agnostisuus alustan suhteen, palvelujen skaalautuvuus, helpompi ylläpito sekä parempi vikasieto. Haitoiksi havaittiin konttiteknologian tuoma lisääntynyt kompleksisuus, sekä sovellustilanteesta riippuva mahdollinen tarpeettomuus. Havaitut hyödyt ja haitat ovat siis linjassa kirjallisuudesta löydetyn kanssa. Artefaktin kehityksessä ei havaittu sellaisia hyötyjä tai haittoja joita ei olisi löytynyt taustakirjallisuudesta.

Lähteet

Amazon. 2024. “What is SOA” [kielellä en-US]. https://aws.amazon.com/what-is/service-oriented-architecture/.

Amazon Web Services. 2021. “What is a data lake”. Viitattu 8. elokuuta 2022. https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/.

B.J. Copeland. 2022. “Artificial intelligence”. Viitattu 10. lokakuuta 2022. https://www.britannica.com/technology/artificial-intelligence.

Blinowski, Grzegorz, Anna Ojdowska ja Adam Przybyłek. 2022. “Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation”. *IEEE Access* 10. ISSN: 2169-3536. https://doi.org/10.1109/ACCESS.2022.3152803.

Cao, Keyan, Yefan Liu, Gongjie Meng ja Qimeng Sun. 2020. “An Overview on Edge Computing Research” [kielellä en]. *IEEE Access* 8:85714–85728. ISSN: 2169-3536. https://doi.org/10.1109/ACCESS.2020.2991734.

Chae, MinSu, HwaMin Lee ja Kiyeol Lee. 2019. “A performance comparison of linux containers and virtual machines using Docker and KVM”. *Cluster Computing* 22 (1): 1765–1775.

CNCF. 2022. “Kubernetes”. Viitattu 30. elokuuta 2022. https://kubernetes.io/.

“Dash docs”. 2023. Viitattu 29. marraskuuta 2023. https://dash.plotly.com/.

European Commission. 2016. “General data protection regulation”. European Commission. Viitattu 8. elokuuta 2022. https://eur-lex.europa.eu/legal-content/FI/TXT/?uri=uriserv:OJ.L_.2016.119.01.0001.01.ENG&toc=OJ:L:2016:119:TOC.

Google. 2022. “What are containers”. Viitattu 29. elokuuta 2022. https://cloud.google.com/learn/what-are-containers.

Hevner, Alan, Alan R, Salvatore March, Salvatore T, Park, Jinsoo Park, Ram ja Sudha. 2004. “Design Science in Information Systems Research”. *Management Information Systems Quarterly* 28 (maaliskuu): 75–.

IBM. 2020. “What is machine learning”. Viitattu 30. elokuuta 2022. https://www.ibm.com/cloud/learn/machine-learning.

Joy, Ann Mary. 2015. “Performance comparison between linux containers and virtual machines”. Teoksessa *2015 international conference on advances in computer engineering and applications,* 342–346. IEEE.

Kaplan, Andreas, ja Michael Haenlein. 2019. “Siri, Siri, in my hand: Who’s the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence”. *Business Horizons* 62 (1): 15–25. ISSN: 0007-6813. https://doi.org/https://doi.org/10.1016/j.bushor.2018.08.004. https://www.sciencedirect.com/science/article/pii/S0007681318301393.

Kate Brush, Brian Kirsch. 2021. “Virtualization”. Viitattu 29. elokuuta 2022. https://www.techtarget.com/searchitoperations/definition/virtualization.

“Keydb docs”. 2023. Viitattu 29. marraskuuta 2023. https://docs.keydb.dev/docs/about.

Laplante, kirjoittaja, Phillip A. 2018. *Requirements engineering for software and systems.* Third edition. Boca Raton, FL: Taylor & Francis, CRC Press.

Li, Shanshan, He Zhang, Zijia Jia, Chenxing Zhong, Cheng Zhang, Zhihao Shan, Jinfeng Shen ja Muhammad Ali Babar. 2021. “Understanding and addressing quality attributes of microservices architecture: A Systematic literature review”. *Information and Software Technology* 131 (maaliskuu): 106449. ISSN: 0950-5849. https://doi.org/10.1016/j.infsof.2020.106449.

Lingeswaran R. 2017. “Para virtualization vs Full virtualization vs Hardware assisted Virtualization”. Viitattu 23. elokuuta 2022. https://www.unixarena.com/2017/12/para-virtualization-full-virtualization-hardware-assisted-virtualization.html/.

March, Salvatore T., ja Gerald F. Smith. 1995. “Design and natural science research on information technology”. *Decision Support Systems* 15 (4): 251–266. ISSN: 0167-9236. https://doi.org/https://doi.org/10.1016/0167-9236(94)00041-2. https://www.sciencedirect.com/science/article/pii/0167923694000412.

Marinescu, Dan C. 2018. “Chapter 10 - Cloud Resource Virtualization”. Teoksessa *Cloud Computing (Second Edition),* Second Edition, toimittanut Dan C. Marinescu, 365–402. Morgan Kaufmann. ISBN: 978-0-12-812810-7. https://doi.org/https://doi.org/10.1016/B978-0-12-812810-7.00013-3. https://www.sciencedirect.com/science/article/pii/B9780128128107000133.

McInnes, Leland. 2018. “Umap documentation”. Viitattu 10. heinäkuuta 2023. https://umap-learn.readthedocs.io/en/latest/.

McInnes, Leland, John Healy ja James Melville. 2020. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.* arXiv: 1802.03426 [stat.ML].

Oracle. 2011. “Brief history of virtualization”. Viitattu 23. elokuuta 2022. https://docs.oracle.com/cd/E26996_01/E18549/html/VMUSG1010.html.

“Ouran V2 apikuvaus”. 2023. Oura. Viitattu 12. syyskuuta 2023. https://cloud.ouraring.com/v2/docs.

Pahl, Marc-Oliver, ja Markus Loipfinger. 2018. “Machine learning as a reusable microservice”. Teoksessa *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium,* 1–7. https://doi.org/10.1109/NOMS.2018.8406165. https://ieeexplore.ieee.org/abstract/document/8406165.

Patas, Janusch, ja Matthias Goeken. 2011. “Paradigms and Information Systems as an applied discipline - A model-based representation, problems, and suggested solutions”. Tammikuu.

Peffers, Ken, Tuure Tuunanen ja Björn Niehaves. 2018. “Design science research genres: introduction to the special issue on exemplars and criteria for applicable design science research”. *European Journal of Information Systems* 27 (2): 129–139. https://doi.org/10.1080/0960085X.2018.1458066. eprint: https://doi.org/10.1080/0960085X.2018.1458066. https://doi.org/10.1080/0960085X.2018.1458066.

Peffers, Ken, Tuure Tuunanen, Marcus A. Rothenberger ja Samir Chatterjee. 2007. “A Design Science Research Methodology for Information Systems Research”. *Journal of Management Information Systems* 24 (3): 45–77. https://doi.org/10.2753/MIS0742-1222240302. eprint: https://doi.org/10.2753/MIS0742-1222240302. https://doi.org/10.2753/MIS0742-1222240302.

Pietikäinen, Matti, ja Olli Silven. 2019. *Tekoälyn haasteet - koneoppimisesta ja konenäöstä tunnetekoälyyn.* University of Oulu, Center for Machine Vision / Signal Analysis. ISBN: 978-952-62-2482-4. http://jultika.oulu.fi/files/isbn9789526224824.pdf.

Rani Osnat. 2020. “A Brief History of Containers”. Viitattu 30. elokuuta 2022. https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016.

Torvekar, Nupura, ja Pravin Game. 2019. “Microservices and Its Applications An Overview”. *International Journal of Computer Sciences and Engineering* 7:803–809. https://doi.org/10.26438/ijcse/v7i4.803809.

University of Jyväskylä. 2024a. [Kielellä fi]. University of Jyväskylä. Viitattu 24. tammikuuta 2024. https://www.jyu.fi/fi/hankkeet/lth-utv-alustan-hyodyntaminen-liikunnan-terveyden-edistamisen-ja-hyvinvoinnin-alalla.

———. 2024b. “Urheilun tietovaranto ja ekosysteemi (UTV)”. University of Jyväskylä. Viitattu 24. tammikuuta 2024. https://www.jyu.fi/fi/hankkeet/urheilun-tietovaranto-ja-ekosysteemi-utv.

Urias, Igor, ja Rogério Rossi. 2023. “Evaluation of Frameworks for MLOps and Microservices” [kielellä en]. *EAI Endorsed Transactions on Smart Cities* 7 (3). ISSN: 2518-3893. https://doi.org/10.4108/eetsc.3661. https://publications.eai.eu/index.php/sc/article/view/3661.

Wang, Yingying, Harshavardhan Kadiyala ja Julia Rubin. 2021. “Promises and challenges of microservices: an exploratory study” [kielellä en]. *Empirical Software Engineering* 26 (4): 63. ISSN: 1573-7616. https://doi.org/10.1007/s10664-020-09910-y.

Weisstein, Eric W. 2023. “Manifold”. Viitattu 21. heinäkuuta 2023. https://mathworld.wolfram.com/Manifold.html.

VMware. 2017. “What is a virtual machine”. Viitattu 24. elokuuta 2022. https://www.vmware.com/topics/glossary/content/virtual-machine.html.