Mikropalvelut ja monoliittinen arkkitehtuuri web-kehityksessä

TURUN YLIOPISTO
Tietotekniikan laitos
TkK-tutkielma
Tietotekniikka
Marraskuu 2025
Anton Fredriksson

TURUN YLIOPISTO
Tietotekniikan laitos

ANTON FREDRIKSSON: Mikropalvelut ja monoliittinen arkkitehtuuri web-kehityksessä

TkK-tutkielma, 21 s.
Tietotekniikka
Marraskuu 2025

Mikropalveluarkkitehtuuri on vakiinnuttanut asemansa yhtenä keskeisenä lähestymistapana web-pohjaisten sovellusten kehityksessä. Perinteiseen monoliittiseen arkkitehtuuriin perustuvan sovelluksen kehitys ja käyttöönotto on yksinkertaista, mutta sovelluksen kasvaessa saatetaan kohdata esimerkiksi skaalautuvuus- ja ylläpito-ongelmia. Mikropalveluilla pyritään ratkaisemaan monoliittisen arkkitehtuurin haasteita jakamalla sovellus useisiin itsenäisiin palveluihin. Toisaalta se monimutkaistaa järjestelmää ja tuo mukanaan omat haasteensa.

Tässä tutkielmassa pyritään kirjallisuuskatsauksen keinoin esittelemään kummankin arkkitehtuurimallin ominaispiirteitä ja selvittämään näiden kahden lähestymistavan hyötyjä ja haasteita. Tavoitteena on muodostaa kokonaiskuva siitä, miten ne eroavat toisistaan muun muassa kehityksen, ylläpidon ja suorituskyvyn näkökulmasta. Lisäksi pyritään tunnistamaan tilanteita, joissa tietyn arkkitehtuurimallin valinta voisi olla perusteltua.

Kirjallisuudessa esitettyjen tulosten ja näkökulmien perusteella päätellään, että mikropalvelut voivat helpottaa erityisesti suurten järjestelmien ja niitä kehittävien organisaatioiden hallintaa. Toisaalta monoliittisen arkkitehtuurin yksinkertaisuus on eduksi tilanteissa, joissa sovelluksen jako mikropalveluiksi ei ole välttämätöntä. Arkkitehtuurimallin valinta tulisi näin ollen tehdä aina tapauskohtaisesti.

Asiasanat: ohjelmistoarkkitehtuurit, monoliittinen arkkitehtuuri, mikropalveluarkkitehtuuri

Sisällys

1 Johdanto

Mikropalvelut ovat joukko pieniä, erillisiä palveluita, jotka muodostavat vaihtoehtoisen arkkitehtuurin monoliitille; yhdelle yhtenäiselle sovellukselle, jossa toiminnallisuuksia on vaikea irrottaa toisistaan. Mikropalveluarkkitehtuuri on herättänyt paljon huomiota jo noin kymmenen vuoden ajan. Se on kasvattanut suosiotaan erityisesti suurten järjestelmien kehityksessä, jossa itsenäiset palvelut voivat joustavoittaa ohjelmiston kehitystä ja skaalausta. [1] Mikropalvelujen hyödyistä ja haasteista, sekä niiden kehitystä tukevista käytännöistä ja teknologioista on käyty paljon keskustelua viime vuosina, mutta sopivan arkkitehtuurimallin valinta saattaa silti olla joissain tilanteissa haastavaa [2]. Kasvaneen suosion myötä mikropalveluja on käytetty myös projekteissa, joihin ne eivät sovellu, tai joissa niiden tuomat haasteet ovat osoittautuneet hyötyjä merkittävämmäksi [3]. Monoliittinen arkkitehtuuri on yksinkertaisempi lähestymistapa, joka saattaa osoittautua edullisemmaksi ratkaisuksi tilanteissa, joissa mikropalveluille ei ole selkeää tarvetta. Siirtyminen arkkitehtuurimallista toiseen voi olla työläs prosessi, joten ohjelmiston suunnitteluvaiheessa tehtäviä valintoja tulisi harkita tarkkaan. [4]

Tutkielmassa pyritään vastaamaan seuraaviin tutkimuskysymyksiin:

- **Tutkimuskysymys 1**: Mitkä ovat mikropalveluarkkitehtuurin ja monoliittisen arkkitehtuurin hyödyt ja haitat?
- **Tutkimuskysymys 2**: Milloin mikropalvelut ovat järkevä vaihtoehto?

Kysymyksiin pyritään vastaamaan selvittämällä, millaisia mikropalveluarkkitehtuuri ja monoliittinen arkkitehtuuri ovat, miten ne eroavat toisistaan ja millä tavalla ne vaikuttavat sovelluksen toimintaan, sen kehittämiseen ja ylläpitoon.

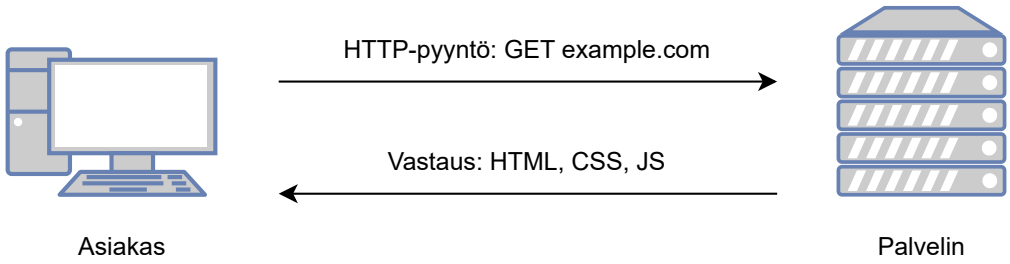
Tämä tutkielma on kirjallisuuskatsaus, jonka lähdeaineisto koostuu tieteellisistä julkaisuista, kirjallisuudesta ja verkkoartikkeleista. Tiedonhakuun käytettiin IEEE Xplore- ja Web of Science -tietokantoja, sekä Turun Yliopiston Volter-palvelua. Tietoa haettiin avainsanoilla: *”microservice* AND monolith*”* ja esimerkiksi vertailevia julkaisuja hakulauseella *”monolith* AND microservice* AND (compar* OR vs.)”*. Löydettyä aineistoa karsittiin ensin otsikon ja tiivistelmän perusteella, jonka jälkeen valittiin aiheen kannalta relevantit julkaisut niitä lukemalla. Näiden julkaisujen lähteitä tarkastelemalla löytyi vielä työhön soveltuvaa aineistoa.

Luvussa 2 annetaan taustatietoja web-sovelluksista, sekä ohjelmistoarkkitehtuuriin ja ohjelmistojen laatuun liittyvistä seikoista. Luvussa 3 käydään läpi monoliittisen arkkitehtuurin määritelmä, hyödyt ja haasteet. Luvussa 4 määritellään mikropalveluarkkitehtuuri ja tarkastellaan sen tuomia hyötyjä ja haasteita. Luvun 5 pohdinnassa vertaillaan näitä kahta arkkitehtuurimallia pääosin suorituskyvyn näkökulmasta. Luvussa 6 tehdään johtopäätökset ja vastataan tutkimuskysymyksiin.

2 Web-sovellukset

2.1 Web-sovellukset

Web-sovellukset noudattavat tyypillisesti kuvassa 2.1 näkyvää asiakas-palvelin-arkkitehtuuria, jossa asiakasohjelma, yleensä verkkoselain, on yhteydessä fyysisesti eri paikassa sijaitsevaan palvelimeen. Asiakkaan ja palvelimen välinen kommunikointi tapahtuu usein verkon yli HTTP-protokollaa käyttäen. Esimerkiksi verkkosivuille mentäessä selain lähettää HTTP-pyynnön palvelimelle, pyyntö kulkee verkon välityksellä web-palvelimelle, joka käsittelee pyynnön ja lähettää vastauksena pyydetyt resurssit. Vastaus voi olla esimerkiksi HTML-tiedosto, jonka avulla selain esittää verkkosivun sisällön käyttäjälle. Sisällön esittämiseen saatetaan tarvita myös muita resursseja, kuten HTML-tiedostoon linkitetyt kuvat ja tyylitiedostot, joiden hakemiseksi selain tekee uudet pyynnöt palvelimelle. Perinteiset web-palvelimet tarjoavat staattisia sivuja ja selaimen tehtävä on yksinkertaisesti esittää palvelimelta haettu sisältö. Nykyään asiakasohjelmassa saatetaan suorittaa logiikkaa sisällön dynaamiseen muokkaamiseen, ja palvelimella sijaitseva sovellus voi olla useaan loogiseen kerrokseen jaettu järjestelmä, joka sisältää monimutkaista tiedon käsittelyyn ja tietokannan käyttöön liittyvää toimintalogiikkaa. [5]



Kuva 2.1: Asiakas-palvelin-arkkitehtuuri

2.2 Ohjelmistoarkkitehtuuri ja laatu

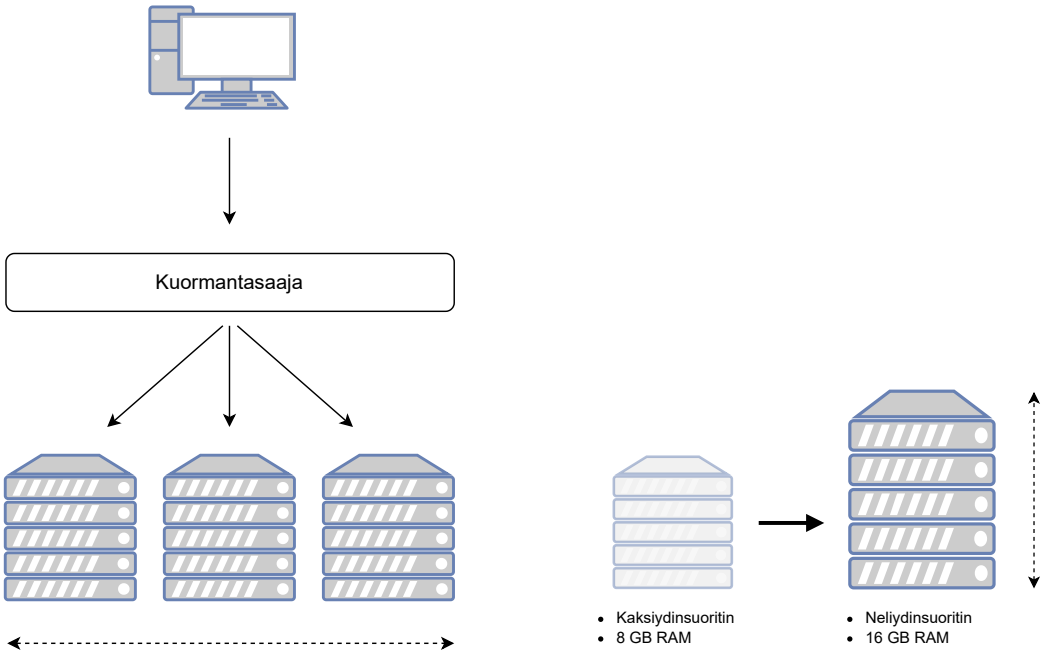
Ohjelmistojen toteutustapoja ja suunnittelussa huomioon otettavia seikkoja on lukuisia. Ohjelmistokehityksessä käytettävät menetelmät ja työkalut muuttuvat nopeasti. Vakiintuneita tekniikoita parannellaan ja ajoittain suosioon nousee täysin uusia ideoita. Tietyt hyväksi todetut periaatteet, kuten myös jotkin vakiintuneet käytännön tekniikat, ovat kuitenkin säilyttäneet paikkansa siitä huolimatta.

Ohjelmistoarkkitehtuuri kuvaa ohjelmiston rakennetta ja ominaispiirteitä. Arkkitehtuurityylien (engl. *architectural styles*) ja suunnittelumallien (engl. *design patterns*) avulla ohjelmistoa voidaan jäsennellä siten, että kokonaisrakennetta ja sen sisältämiä elementtejä, sekä niiden välisiä suhteita on helpompi ymmärtää. Asioiden eriyttäminen (engl. *separation of concerns*) eli ongelmien jakaminen pienempiin osiin on yleinen suunnitteluperiaate (engl. *design principle*), joka ilmentyy monessa muodossa osana muita suunnittelumalleja ja -periaatteita. Näistä yleisimpänä voidaan pitää modulaarisuutta. Modulaarisuudella tarkoitetaan ohjelmiston toiminnallisuuden jakamista pienempiin osiin, joista kokonaisuus koostetaan. Yhden osan (moduuli, komponentti) on tarkoitus toteuttaa vain jokin tietty toiminnallisuus. Rakenteen selkeyttämisen lisäksi modulaarisuudella voidaan tietyissä tapauksissa mahdollistaa osien parempi uudelleenkäytettävyys. Modulaarisuutta voidaan pitää ohjelmiston ylläpidettävyyden kannalta tärkeänä ominaisuutena. [6]

Ohjelmiston laatua voidaan arvioida monesta eri näkökulmasta. Ohjelmistotuotteen kan-

nalta laadulla viitataan usein ominaispiirteisiin, joita on määritelty esimerkiksi ISO/IEC 25010 -standardin laatumallissa. Useimmille web-sovelluksille tärkeitä laatuattribuutteja ovat esimerkiksi luotettavuus, tehokkuus ja ylläpidettävyys. Luotettava ohjelmisto toimii vaatimusten ja käyttäjän odotusten mukaisella tavalla. Sen tulisi sietää käyttäjän ja laitteiston aiheuttamia virheitä, sekä estää väärinkäyttöä. Luotettavan toiminnan ylläpidon ja ohjelmiston jatkokehityksen kannalta on tärkeää, että ohjelmistoon on helppo tehdä muutoksia. Ylläpidettävyys on laatuattribuutti, jolla viitataan etenkin edellä mainittuun muutettavuuteen. Tehokkuudella puolestaan tarkoitetaan ohjelmiston suorituskykyä tai resurssien käyttöä. Web-pohjaisissa sovelluksissa suorituskykyä voidaan arvioida esimerkiksi mittaamalla palvelimen vasteaikaa, eli asiakkaan tekemän pyynnön ja vastauksen saamisen välistä aikaa. Suorituskyvyn tulisi pysyä kohtuullisena myös olosuhteiden muuttuesssa. Kun ohjelmistoon kohdistuvan verkkoliikenteen tai käsiteltävän datan määrä kasvaa, sen toiminta saattaa hidastua huomattavasti. Skaalautuvuudella kuvaillaan järjestelmän kykyä kestää näitä muutoksia siihen kohdistuvan kuorman määrässä. [7]

Järjestelmän skaalaus, eli käytettävissä olevien resurssien lisääminen tai muuttaminen voidaan jakaa kahteen lähetymistapaan, jotka näkyvät kuvassa 2.2. Vertikaalinen skaalaus tarkoittaa palvelinkoneen laskentaresurssien lisäämistä. Se on yleensä yksinkertaisin tapa parantaa suorituskykyä, mutta yhden palvelinkoneen päivittäminen tehokkaampaan on kannattavaa tai ylipäätään mahdollista vain tiettyyn pisteeseen asti. Horisontaalisessa skaalauksessa ei rajoituta yhden palvelinkoneen tai ohjelmainstanssin käyttöön. Tätä lähestymistapaa voidaan hyödyntää esimerkiksi monistamalla ohjelmisto usealle palvelinkoneelle ja jakamalla työkuorma tasaisesti kuormantasaajan (engl. *load balancer*) avulla. [1]

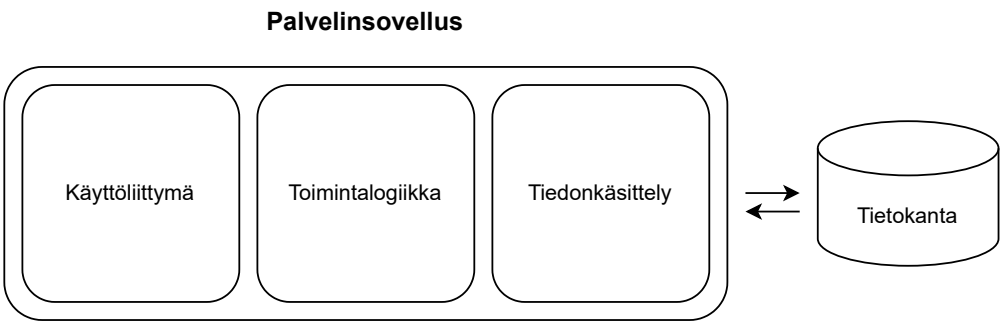


Kuva 2.2: Horisontaalinen ja vertikaalinen skaalaus

3 Monoliittinen arkkitehtuuri

3.1 Määritelmä

Monoliitti on ohjelmistosovellus, jonka moduuleja ei voi suorittaa itsenäisesti [8]. Tyypillinen web-pohjainen sovellus koostuu käyttöliittymästä, palvelinsovelluksesta ja tietokannasta. Palvelinsovellus sisältää ohjelmalogiikan ja toiminnallisuuden, joiden avulla käsitellään dataa käyttöliittymän ja tietokannan välillä. Monoliittisella sovelluksella tarkoitetaan erityisesti sitä, että on vain yksi palvelinsovellus, joka sisältää kaiken ohjelman kannalta olennaisen toiminnallisuuden. Käyttöliittymän mielletään usein kuuluvan samaan kokonaisuuteen, mutta tietokanta on yleensä oma erillinen prosessinsa [4], [9]. Tämä riippuu kuitenkin toteutustavasta ja määritelmästä. Sovelluksen osat ovat vahvasti sidoksissa toisiinsa ja ne suoritetaan yhtenä prosessina. [10] Monoliittinen sovellus voi olla rakenteeltaan kuvan 3.1 kaltainen.



Kuva 3.1: Monoliittinen arkkitehtuuri

3.2 Hyödyt

Monoliittisen sovelluksen merkittävimpänä vahvuutena voidaan pitää sen yksinkertaisuutta. Kehittäminen, testaaminen, ja käyttöönotto on yksinkertaista, kunhan sovelluksen koko ja kompleksisuus pysyy maltillisena. Tämän vuoksi monoliittinen arkkitehtuuri on säilyttänyt asemansa hyvin yleisenä tapana toteuttaa web-pohjaisia sovelluksia. Se voidaan mieltää ikään kuin vakiintuneena perusvaihtoehtona, jota voidaan onnistuneesti käyttää myös joissain suuremmissa ja monimutkaisemmissa ohjelmistoprojekteissa.

Monoliittinen sovellus on parhaimmillaan vakaa, luotettava ja suorituskykyinen. Sovelluksen moduulit tai komponentit julkaistaan yhtenä kokonaisuutena, jonka vuoksi osien välinen kommunikaatio tapahtuu saman prosessin sisällä, esimerkiksi funktiokutsuilla. Kaikki sovelluksen käsittelemä data voidaan myös säilöä yksittäiseen tietokantaan. Välttämällä erillisten osien välistä verkon yli tapahtuvaa kommunikaatiota voidaan nopeuttaa tiedonkulkua ja säästytään ongelmilta, joita voi esiintyä hajautettujen järjestelmien välisessä kommunikoinnissa. Monoliittista sovellusta voidaan skaalata melko vaivattomasti lisäämällä palvelinresursseja vastaamaan kasvaneeseen työkuormaan. [1]

3.3 Haasteet

Monoliittisen arkkitehtuurin haasteet ilmenevät erityisesti silloin, kun sovellus kasvaa kooltaan ja kompleksisuudeltaan. Vahvan sidosteisuuden takia muutokset yhdessä osassa sovellusta saattavat aiheuttaa ongelmia muualla. Yhteen osaan kohdistuvaan lisääntyneeseen kuormaan ei voida vastata lisäämällä resursseja kohdistetusti, sillä monoliittista sovellusta voi skaalata vain koko sovelluksen tasolla. Kaikki palvelinsovelluksen osat tulee myös toteuttaa samalla ohjelmointikielellä ja tekniikoilla, joten kehitystyössä hyödynnettävien työkalujen ja erikoisosaamisen määrä on rajattu. Ison monoliittisen sovelluksen

kehittäminen vaatii usein myös suurta määrää kehittäjiä. Sovelluksen osien väliset riippuvuudet ja pienienkin muuotosten vaatima uuden kokonaisuuden hidas käyttönotto vaikeuttavat jatkuvaa kehittämistä. Tällöin itse kehitystyölle jää vähemmän aikaa ja tuottavuus laskee.

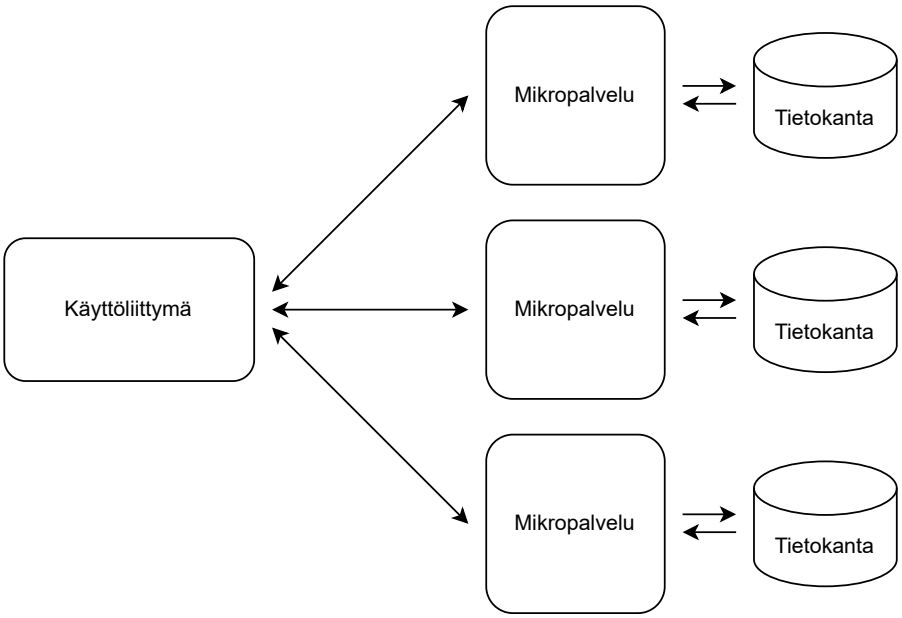
Näiden seikkojen vuoksi sovellusta kehittävää henkilöstöä on vaikea jakaa itsenäisiin, tiettyä toiminnallisuutta kehittäviin tiimeihin. Suuren monoliitin keskitetty, vahvasti sidosteinen rakenne rajoittaa myös sitä kehittävän organisaation tehokkuutta ja rakennetta. Monoliittisen arkkitehtuurin yksinkertaisuuteen pohjautuvat edut muuttuvatkin ongelmiksi, kun kehitettävä sovellus kasvaa suuremmaksi ja monimutkaisemmaksi. [1], [4]

4 Mikropalveluarkkitehtuuri

4.1 Määritelmä

Mikropalveluarkkitehtuuri voidaan määritellä arkkitehtuurimallina, jossa sovelluksen toiminnallisuus hajautetaan erillisiin palveluihin, joita kutsutaan mikropalveluiksi. Sovellus koostuu itsenäisistä, toiminnallisuuden mukaan rajatuista osista. Mikropalvelut suoritetaan omina prosesseinaan ja niiden välinen kommunikaatio tapahtuu verkkoprotokollien avulla. [8], [11]

Mikropalveluarkkitehtuurin avulla pyritään tekemään kehitystyöstä joustavampaa. Jokaista sovelluksen osaa voidaan kehittää ikään kuin omana sovelluksenaan, sillä niiden toiminta on pääsääntöisesti muista osista riippumatonta. Yhden mikropalvelun on tarkoitus toteuttaa jokin tietty, tarkkaan rajattu tehtävä. Palvelu sisältää kaiken siihen tarvittavan toiminnallisuuden ja sillä on yleensä oma tietokanta. Nämä itsenäisesti julkaistut palvelut tarjoavat tyypillisesti rajapinnan, jonka avulla esimerkiksi muut palvelut voivat käyttää sitä [1]. Mikropalveluista koostuvaa sovellusta on havainnollistettu kuvassa 4.1.



Kuva 4.1: Mikropalveluarkkitehtuuri

4.2 Hyödyt

Mikropalveluarkkitehtuurin hajautettu rakenne tekee siitä houkuttelevan vaihtoehdon monimutkaisten sovellusten kehittämiseen. Yksi keskeisimmistä eduista on se, että varsinkin suurempien sovellusten ylläpito, uusien ominaisuuksien kehittäminen ja muutosten julkaisu on nopeampaa. Yhden mikropalvelun sisäiseen rakenteeseen voidaan usein tehdä muutoksia muista palveluista välittämättä, kunhan sen tarjoama rajapinta ei muutu. Rakenne soveltuu hyvin nykyaikaisiin ketteriin kehitysmenetelmiin, joissa painotetaan nopeaa muutosten julkaisua ja tehokasta, usein myös automatisoitua julkaisuprosessia. [10], [12]

Mikropalveluille ominainen osien välinen löyhä sidosteisuus tarjoaa joustavuutta myös toteutusmenetelmiin ja työkaluihin. Palvelua kehittävällä tiimillä on vapaammat kädet tehdä päätöksiä käytettävän ohjelmointikielen, kehitysmenetelmien ja työkalujen suhteen. Itsenäistä mikropalvelua kehittäessä voidaan valita parhaat toteutustavat juuri siihen käyttö-

tarkoitukseen, muista palveluista riippumatta. Mikropalvelujen välinen kommunikaatio tapahtuu niiden tarjoamien rajapintojen välillä yleisiä protokollia käyttäen, joten eri palvelujen kehityksessä ei olla lukittuna saman teknologian käyttöön, kuten monoliittisessa arkkitehtuurissa. [1], [10]

Eristetyt, itsenäiset sovelluksen osat helpottavat myös vikatilanteissa ja sovelluksen skaalauksessa. Jos yksi mikropalvelu lopettaa toimintansa, se ei välttämättä vaikuta muiden palvelujen toimintaan. Vikatilanteiden tarkkailu ja virheiden korjaus on myös luonnollisesti nopeampaa ja helpompaa, kun käsiteltävä kokonaisuus on pienempi. Palvelimeen kohdistuvan kuorman kasvaessa mikropalveluarkkitehtuuri mahdollistaa joustavamman skaalauksen. Mikropalveluja voidaan skaalata itsenäisesti, eli jokaisen palvelun ylläpitoon käytettäviä resursseja voi hallita kohdistetusti. Jos jokin tietty sovelluksen osa vaatii enemmän laskentatehoa kuin muut, koko sovelluksen sijaan resursseja voi kohdistaa tämän tietyn osan, eli mikropalvelun käyttöön. [1], [10]

Nämä teknologiset vahvuudet heijastuvat myös sovellusta kehittävän organisaation toimintaan. Sovelluskokonaisuuden jakaminen mikropalveluiksi edesauttaa henkilöstön jakoa itsenäisempiin tiimeihin, joilla on pienempi vastuualue. Kehittäjien erikoisosaamista voidaan hyödyntää tehokkaammin jos he työskentelevät sellaisen mikropalvelun parissa, jonka toimintaympäristö ja käytettävät työkalut vastaavat paremmin omaan osaamiseen. Mikropalvelujen välisen riippumattomuuden ansiosta kommunikoinnissa ja erinäisissä kehitysvaiheissa saatetaan säästää aikaa. [1], [10], [12]

4.3 Haasteet

Mikropalveluarkkitehtuuri ei ole kaikkiin tilanteisiin sopiva ratkaisu. Tarpeettomasti käytettynä tai huonosti toteutettuna mikropalvelut voivat aiheuttaa enemmän haasteita kuin hyötyä.

Sovelluksen komponenttien rajaaminen ja jako sopivan kokoisiin mikropalveluihin on haastavaa. Mikropalveluarkkitehtuurin periaatteen mukaan sovelluksen tulisi koostua pienistä palveluista, mutta toisaalta liian pieniin palveluihin jako voi vaikeuttaa kehitystä ja aiheuttaa hajautetuille järjestelmille ominaisia ongelmia. Suurin osa mikropalveluarkkitehtuurin teknisistä haasteista liittyy näihin ongelmiin, joista merkittävimpänä voidaan pitää riippuvuutta ajoittain epäluotettavasta verkkoyhteydestä. Jonkin operaation suorittaminen saattaa esimerkiksi edellyttää verkkopyyntöjä useaan mikropalveluun, jonka vuoksi virhealttius kasvaa ja muun muassa pyyntöjen vasteajat heikentävät suorituskykyä. Tämän lisäksi järjestelmän osien välisen synkronisaation ja datan eheyden kanssa voi ilmetä ongelmia, kun sovellus on hajautettu monelle palvelimelle ja data eri tietokantoihin. [4], [7], [12]

Mikropalveluja kehittävän organisaation rakenteen tulisi muistuttaa mikropalveluarkkitehtuurin rakennetta. Organisaation henkilöstö voidaan jakaa itsenäisiin, yhdestä mikropalvelusta vastuussa oleviin tiimeihin sen sijaan, että jako tehtäisiin perinteisesti roolin tai erikoisosaamisen perusteella. Tiimien välinen kommunikaatio on välttämätöntä rakenteesta riippumatta, mutta kehitysprosessia hidastavia konflikteja ja ylimääräistä kommunikaatiota voidaan vähentää järjestämällä organisaation rakenne kehitettävän järjestelmän mukaan. Conwayn lain mukaan organisaatio, joka suunnittelee järjestelmän tuottaa rakenteen, joka vastaa organisaation rakennetta. Rakenteiden erot voivat vaikeuttaa kehitystyötä. [4], [12]

5 Pohdinta

Ohjelmiston suorituskykyä voidaan tarkastella esimerkiksi seuraamalla resurssien käyttöä, virheiden määrää tai käytettävyysaikaa (engl. *uptime*). Varsinkin web-pohjaisten sovellusten kohdalla tarkastellaan näiden lisäksi yleensä verkkoliikenteeseen liittyviä mittareita, kuten suoritustehoa (engl. *throughput*) ja vasteaikoja [1], [13]. Suoritustehoa mitataan usein seuraamalla järjestelmän käsittelemien verkkopyyntöjen määrää tietyn ajanjakson aikana.

Monoliittisen arkkitehtuurin ja mikropalveluarkkitehtuurin suorituskykyvertailuja tarkastellessa tulee pitää mielessä se, että tulokset riippuvat monesta asiasta. Tuloksiin saattaa vaikuttaa esimerkiksi testiympäristö, kuten testattavaan sovellukseen ja työkuormiin liittyvät yksityiskohdat. Tästä syystä näiden arkkitehtuurien suorituskykyä vertailevassa kirjallisuudessa saattaa esiintyä vaihtelevia tuloksia [2] ja yleispätevien johtopäätösten tekeminen on haastavaa.

Blinowski ym. [1] vertasivat näiden kahden arkkitehtuurin suorituskykyä ja kustannuksia eri ympäristöissä. Testeissä käytettävästä sovelluksesta toteutettiin kahdella eri teknologialla molempiin arkkitehtuurimalleihin perustuvat versiot, eli yhteensä neljä toiminnallisuudeltaan identtistä sovellusta. Jokainen sovellus tarjoaa päätepisteet kahden erityyppisen palvelun tai toiminnon käyttöön: yksinkertaisen kyselyn suorittava City-palvelu ja laskennallisesti intensiivinen Route-palvelu. Testit suoritettiin lokaalisti ja

Azure-pilvipalveluissa. Lokaalissa ympäristössä (Windows PC, yksi instanssi per palvelu) monoliittinen versio oli suoritusteholtaan (pyyntöjä per sekunti) molemmissa palveluissa, molemmilla teknologioilla parempi. Vastaavasti Azure Spring Cloud ja Azure App Service -pilvipalveluissa monoliittinen sovellus oli yhden palvelinkoneen konfiguraatioissa keskimäärin hieman mikropalveluja tehokkaampi. Skaalaamalla mikropalvelusovellusta horisontaalisesti ja vertikaalisesti saavutettiin paras mahdollinen suoritusteho, mutta monoliittinen versio osoittautui useimmissa tapauksissa kustannustehokkaammaksi ratkaisuksi.

Villamizar ym. [13] selvittivät web-sovellusten taloudellisia kustannuksia ja suorituskykyä vertailemalla AWS EC2 -pilvipalvelussa suoritettavia mikropalveluja ja monoliittista sovellusta, sekä palvelimettomaan tietojenkäsittelyyn (engl. *serverless computing*) ja ”Function as a Service” -malliin perustuvan AWS Lambdan operoimia mikropalveluja. Tulosten mukaan AWS EC2 -palvelussa mikropalveluilla voitiin saavuttaa parempi suoritusteho (tuettu pyyntöjen määrä per minuutti) hieman pienemmillä kustannuksilla kuin monoliittisella sovelluksella. Toisaalta keskimääräiset vasteajat osoittautuivat korkeammiksi mikropalveluita käytettäessä. Merkittävin ero ilmeni kuitenkin AWS Lambdaa hyödyntävien mikropalvelujen tapauksessa, jossa vastaava suorituskyky voitiin saavuttaa jopa 77 % pienemmillä kustannuksilla.

Berry ym. [14] päättelivät testitulostensa perusteella, että mikropalveluarkkitehtuuri parantaa sovelluksen suorituskykyä ja saatavuutta (engl. *availability*). Mikropalvelusovelluksen saatavuusasteessa näkyvät parannukset todettiin selittyvän osittain API-yhdyskäytävän (engl. *API gateway*) käytöllä pyyntöjen reitityksessä. Vastaavasti monoliittisen sovelluksen saatavuutta ja suoritustehoa saatiin parannettua hieman sijoittamalla se kuormantasaajan taakse. Horisontaalisen skaalauksen todettiin olevan suorituskyvyn ja saatavuuden kannalta hyödyllistä, arkkitehtuurimallista riippumatta. Optimaalisiin suorituskyky- ja saatavuusarvoihin vaadittiin mikropalveluversion tapauksessa kuitenkin

vähemmän replikaatteja monoliittiseen versioon verrattuna. Laskentaresurssien (suorittimien ja muistin) käyttöasteissa ja energiankulutuksessa monoliitti oli yksinkertaisissa konfiguraatioissa mikropalveluja edullisempi vaihtoehto, mutta optimaalisen suorituskyvyn ja saatavuuden mahdollistavissa konfiguraatioissa tilanne oli päinvastainen.

Taulukkoon 5.1 on tiivistetty mikropalveluarkkitehtuurin ja monoliittisen arkkitehtuurin hyötyjä ja haasteita.

Arkkitehtuuri	**Hyödyt**	**Haasteet**
Monoliittinen arkkitehtuuri	Kehittämisen ja muutosten julkaisun yksinkertaisuus Testaus ja virheenjäljitys suoraviivaista Hyvä suorituskyky, etenkin yksinkertaisissa palvelinkonfiguraatioissa Vakiintunut, yksinkertainen ja luotettava rakenne	Suuren monoliittisen sovelluksen kehitys on hidasta Keskitetty rakenne ja vahva sidosteisuus Rajoitettu skaalautuvuus Heikko virheiden eristys: virheet voivat kaataa koko järjestelmän Sitoutuminen samaan teknologiapinoon kaikissa osissa
Mikropalvelu-arkkitehtuuri	Palvelujen itsenäinen kehitys ja julkaisu Skaalautuvuus Teknologinen joustavuus Soveltuu hyvin nykyaikaisiin, nopeita muutoksia ja iteratiivisuutta painottaviin kehitysmenetelmiin	Kompleksisuus Työläämpi ja vaikeampi toteuttaa Mahdollisesti heikompi suorituskyky ja korkeammat ylläpitokustannukset Hajautettu rakenne: ylimääräinen kommunikaatio osien välillä, verkon luotettavuus, hajautettu data

Taulukko 5.1: Monoliittisen ja mikropalveluarkkitehtuurin hyödyt ja haasteet

Vertailun pohjalta voidaan todeta, että mikropalveluarkkitehtuurin joustavuudella ja skaalautuvuudella voidaan hallita ohjelmiston kasvavia vaatimuksia, monimutkaisuutta ja laajuutta. Mikropalveluille ominaiset haasteet voivat osoittautua helpommin ratkaistaviksi kuin ne, joita monoliittinen arkkitehtuuri aiheuttaa kasvavassa ohjelmistoprojektissa ja organisaatiossa. Toisaalta monoliittinen arkkitehtuuri on yksinkertaisempi lähestymistapa, joka toimii monessa tilanteessa hyvin. Yhtenäisen monoliitin kehitys voi parhaimmillaan olla hyvin suoraviivaista ja tehokasta.

6 Johtopäätökset

TK1: Mitkä ovat mikropalveluarkkitehtuurin ja monoliittisen arkkitehtuurin hyödyt ja haitat?

Monoliittisen arkkitehtuurin yksinkertainen rakenne helpottaa ohjelmiston kehitystyötä ja -operaatioita. Prosessin sisäinen kommunikaatio ja pienempi verkkoliikenteen määrä saattaa parantaa suorituskykyä, luotettavuutta ja tietoturvaa hajautettuun järjestelmään verrattuna. Monoliittista sovellusta voidaan myös skaalata kokonaisuutena helposti lisäämällä palvelinresursseja ja se voidaan tarpeen vaatiessa monistaa useaksi instanssiksi.

Monoliittisen rakenteen yksinkertaisuus voi kuitenkin osoittautua haasteelliseksi, kun sovelluksen koko, monimutkaisuus tai sitä kehittävien henkilöiden määrä kasvaa. Suureen, vahvasti sidosteiseen kokonaisuuteen on vaikeampaa ja hitaampaa tehdä muutoksia, sillä siihen kuuluvat komponentit ovat toisistaan riippuvaisia, ja muutosten käyttöönotto vaatii koko sovelluksen uudelleenjulkaisua. Sovelluksen osia ei voi skaalata kohdistetusti, eikä niiden kehityksessä voida hyödyntää muista osista poikkeavaa teknologiaa, esimerkiksi eri ohjelmointikieltä.

Mikropalveluarkkitehtuuria hyödyntämällä voidaan ratkaista yleisimpiä monoliittisen sovelluksen kehitykseen liittyviä haasteita. Hajauttamalla sovelluksen osia itsenäisiksi palveluiksi voidaan mahdollistaa niiden kehitys, muutosten julkaisu ja palvelun skaalaus muista järjestelmän osista riippumatta. Kehitystyön tehokkuutta voidaan parantaa, kun

erilliset tiimit voivat keskittyä työskentelemään oman vastuualueensa parissa. Tämä mahdollistaa myös eri teknologioiden käytön eri palveluissa, jonka vuoksi tiimit voivat valita itselleen ja palvelulle parhaiten soveltuvat työkalut. Lisäksi vikatilanteissa yhden mikropalvelun kaatuminen ei välttämättä aiheuta koko järjestelmän kaatumista, vaan siitä riippumattomat palvelut voivat toimia normaalisti.

Toisaalta mikropalveluihin perustuva kokonaisuus on monimutkaisempi ja haastavampi toteuttaa. Palveluiden rajat ja vastuut tulee määritellä tarkkaan, jotta järjestelmän rakenne pysyy hallinnassa. Mikropalveluarkkitehtuuri tukee modulaarisuutta ja vahvaa rajausta, mutta se saattaa aiheuttaa hyötyjen sijaan lähinnä lisähaasteita, jos rajojen ja vastuualueiden määrittely epäonnistuu. Palveluiden välinen viestintä voi edellyttää myös tarkempaa suunnittelua tietoturvan varmistamiseksi, sekä aiheuttaa lisäkuormaa, joka saattaa heikentää suorituskykyä. Palvelukohtainen kehitys vaikeuttaa kokonaisuuden hallintaa ja kasvattaa operatiivista työtä. DevOps-käytäntöjen merkitys korostuu, ja automatisointi, kuten CI/CD-putket ovat lähes välttämättömiä. Kehitys- ja ylläpitoprosessit, sekä esimerkiksi monitorointi tulee määritellä palvelukohtaisesti. Näistä syistä projektin käynnistäminen ja sen ylläpito edellyttää tarkempaa suunnittelua ja lisäresursseja.

TK2: Milloin mikropalvelut ovat järkevä vaihtoehto?

Mikropalveluarkkitehtuuri ei ole joka tilanteeseen sopiva ratkaisu. Monoliittinen arkkitehtuuri vaikuttaa paremmalta valinnalta, kun mikropalveluiden käytölle ei ole selkeää tarvetta. Sen yksinkertaisuudesta johtuen projektin käynnistäminen, sekä sen kehitys ja ylläpito on melko nopeaa ja vaivatonta. Yhdessä prosessissa suoritettava sovellus on joissain tapauksissa suorituskyvyltään parempi ja kustannuksiltaan edullisempi. Yksinkertaisuudella voidaan monesti saavuttaa huomattavia etuja, jos monimutkaisemmalle ratkaisulle ei ole tarvetta. Tällöin hyvin suunniteltu modulaarinen monoliitti saattaa olla parempi ratkaisu. Modulaarisella rakenteella ja tarkalla järjestelmän osakomponenttien vastuualuei-

den rajauksella voidaan helpottaa monoliittisen sovelluksen ylläpidon lisäksi myös tarpeen vaatiessa osien siirtoa mikropalveluiksi.

Mikropalveluarkkitehtuuri on järkevä vaihtoehto silloin, kun järjestelmän jako pieniin, itsenäisiin palveluihin on mikropalveluiden aiheuttamasta lisätyöstä ja haasteista huolimatta kannattavaa. Järjestelmän suuri koko ja kompleksisuus, sekä paremman skaalautuvuuden ja kehitystyön joustavuuden tarve ovat yleisiä syitä, joiden vuoksi itsenäisiin palveluihin perustuva lähestymistapa voi olla suotuisa ratkaisu. Mikropalveluarkkitehtuuri tukee myös nykyaikaista, nopeita muutoksia ja jatkuvaa käyttöönottoa suosivaa ohjelmistokehitystä, koska yhtä rajattua toiminnallisuutta vastaavaa sovelluksen osaa pystytään muuttamaan ja näitä muutoksia julkaisemaan vaikuttamatta muihin osiin. Lisäksi mikropalveluihin perustuvaa toteutusta voi tukea tarve joustavuudelle teknologiavalintojen suhteen, sekä mahdollisuus eristää muita sovelluksen osia esimerkiksi virhetilanteiden aiheuttamilta katkoksilta. Tämä arkkitehtuurivalinta liittyy vahvasti myös järjestelmää kehittävän organisaation rakenteeseen. Itsenäiset palvelut mahdollistavat, sekä usein edellyttävät kehittäjien jakoa itsenäisiin tiimeihin, jotka ovat vähemmän riippuvaisia toisistaan.

Lopuksi

Arkkitehtuurivalinnoilla voi olla suuri merkitys ohjelmiston kehitystyön, kustannusten ja suorituskyvyn kannalta. Täten on erityisen tärkeää tuntea eri vaihtoehdot, sekä se, että miksi ja miten niitä käytetään. Arkkitehtuurimallista riippumatta on noudatettava hyväksi todettuja käytäntöjä, sillä huono toteutus voi esimerkiksi tehdä mikropalveluista huonomman vaihtoehdon tilanteessa, jossa niiden käyttö on perusteltua. Eri toteutustavoista opitaan myös jatkuvasti lisää ja niitä tukevat teknologiat kehittyvät. Mikropalveluiden suosion myötä niiden kehitystä ja ylläpitämistä helpottavat työkalut, ohjelmakirjastot ja pilvipalvelut ovat parantuneet huomattavasti. Toisaalta tietoisuus mikropalveluiden haasteista ja epäonnistuneista toteutuksista on lisääntynyt, jonka vuoksi monoliittinen arkkitehtuuri ja eräänlaiset näiden kahden arkkitehtuurimallin välimuodot ovat nostaneet päätään. Yksi

tulevaisuuden kehityssuunta saattaakin olla hybridimallit, joissa yhdistetään monoliittisia ja mikropalvelun periaatteita – esimerkiksi ydinlogiikka voi olla monoliittisessa muodossa, kun taas nopeasti muuttuvat tai erilliset toiminnallisuudet on toteutettu mikropalveluina.

Lähdeluettelo

[1] G. Blinowski, A. Ojdowska ja A. Przybyłek, ”Monolithic vs. microservice architecture: A performance and scalability evaluation”, *IEEE Access*, vol. 10, s. 20 357–20 374, 2022.

[2] H. Rodrigues, A. Rito Silva ja A. Avritzer, ”Performance Comparison of Monolith and Microservice Architectures: An Analysis of the State of the Art”, teoksessa *European Conference on Software Architecture*, Springer, 2023, s. 185–199.

[3] N. C. Mendonça, C. Box, C. Manolache ja L. Ryan, ”The monolith strikes back: Why istio migrated from microservices to a monolithic architecture”, *IEEE software*, vol. 38, nro 5, s. 17–22, 2021.

[4] M. Kalske, N. Mäkitalo ja T. Mikkonen, ”Challenges when moving from monolith to microservice architecture”, teoksessa *Current Trends in Web Engineering: ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine*, Springer, 2018, s. 32–47.

[5] MDN contributors, *An overview of HTTP*, 2023, Viitattu 11.8.2025, https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

[6] R. S. Pressman, *Software Engineering: A Practitioner's Approach, 7th Edition*. 2009.

[7] M. Kleppmann, *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. O'Reilly Media, 2017.

[8] N. Dragoni et al., "Microservices: yesterday, today, and tomorrow", *Present and ulterior software engineering*, s. 195–216, 2017.

[9] M. Milić ja D. Makajić-Nikolić, "Development of a Quality-Based Model for Software Architecture Optimization: A Case Study of Monolith and Microservice Architectures", *Symmetry*, vol. 14, nro 9, s. 1824, 2022.

[10] AWS, *Microservices*, 2023, https://aws.amazon.com/microservices/, Viitattu 11.8.2025.

[11] M. Fowler ja J. Lewis, *Microservices: a definition of this new architectural term*, https://martinfowler.com/articles/microservices.html, 2014.

[12] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis ja S. Tilkov, "Microservices: The journey so far and challenges ahead", *IEEE Software*, vol. 35, nro 3, s. 24–35, 2018.

[13] M. Villamizar et al., "Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures", *Service Oriented Computing and Applications*, vol. 11, s. 233–247, 2017.

[14] V. Berry, A. Castelltort, B. Lange, J. Teriihoania, C. Tibermacine ja C. Trubiani, "Is it Worth Migrating a Monolith to Microservices? An Experience Report on Performance, Availability and Energy Usage", teoksessa *2024 IEEE International Conference on Web Services (ICWS)*, IEEE, 2024, s. 944–954.