



DevOps ja tuotannon monitorointiratkaisu

Jaakko Halttunen

Opinnäytetyö, AMK

Toukokuu 2022

Insinööri (AMK), Tieto- ja viestintätekniikan tutkinto-ohjelma

Halttunen, Jaakko

DevOps ja tuotannon monitorointiratkaisu

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2022, 51 sivua.

Tekniikan ala. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

Ohjelmistotuotannon juuret ovat 1940-luvulla, kun ensimmäinen ohjelma kirjoitettiin ja suoritettiin onnistuneesti. Ohjelmistotuotannon menetelmät ovat kehittyneet noilta ajoilta vesiputousmallista, prototyyppi-kehitykseen ja siitä aina ketteriin kehitysmenetelmiin asti. Nykyään DevOps-toimintamalli on hyvin suosittu sen käytännöllisyyden takia.

Opinnäytetyön tavoitteena oli selvittää mitä sovelluksen monitorointiin käytettäviä monitorointiratkaisuja oli saatavilla. Lisäksi tarkoitus oli selvittää mikä monitorointiratkaisu soveltuu parhaiten toimeksiantajan käyttöön, miten sen käyttöönotto tapahtuu, millä tavoin monitorointiratkaisu tuottaa haluttuja tietoja ja miten valittu monitorointiratkaisu hyödyttää toimeksiantajaa.

Työ oli kehittämistutkimus, jossa tavoiteltiin prosessin kehitystä ja pyrittiin löytämään sitä tukevia uusia työkaluja. Työ toteutettiin suorittamalla häiriötilanteiden prosessin ja monitoroinnin nykytilat, joiden perusteella työlle saatiin vaatimukset. Tämän jälkeen suoritettiin monitorointiratkaisujen vertailu pohjautuen aiemmin saatuihin vaatimuksiin. Vertailun tuloksena valittiin yksi monitorointiratkaisu testikäyttöön. Testikäytön aluksi tehtiin monitorointiratkaisun käyttöönotto ja sen jälkeen suoritettiin testikäyttö.

Tutkimuksen tuloksena saatiin valittua toimeksiantajalle parhaiten sopiva monitorointiratkaisu, joka oli Azure Monitor. Tämän lisäksi tutkimuksen tuloksena saatiin dokumentoitua Azure Monitorin käyttöönotto sekä selvitettyä monitorointiratkaisun hyödyt toimeksiantajalle. Lisäksi selvitettiin miten Azure Monitor täyttää siihen kohdistuvat vaatimukset.

Tutkimuksen toteutus onnistui hyvin. Parhaiten toimeksiantajan käyttöön soveltuva monitorointiratkaisu saatiin valittua. Monitorointiratkaisun käyttöönotto saatiin dokumentoitua niin, että sen perusteella voidaan suorittaa käyttöönotto. Näiden lisäksi saatiin selvitettyä, miten toimeksiantaja hyötyy monitorointiratkaisun käytöstä sekä löydettiin tapoja käyttää sitä tehokkaasti häiriötilanteiden selvityksessä.

Avainsanat (asiasanat)

DevOps, Ohjelmistotuotanto, Monitorointi

Muut tiedot (salassa pidettävät liitteet)

Ei salassa pidettäviä tietoja

Halttunen, Jaakko

DevOps and solution for service monitoring

Jyväskylä: JAMK University of Applied Sciences, May 2022, 51 pages.

Engineering and technology. Degree Programme in Information and Communications Technology. Bachelor's thesis.

Permission for web publication: Yes

Language of publication: Finnish

Abstract

The history of software engineering starts in 1940 century. In that time the first computer program was written and executed. Since then, methodologies of software engineering have changed. Waterfall model was first introduced methodology on 1970 century. After that there was prototyping methodology which leads into the Agile development and publishing Agile Manifesto in 2001. In these days DevOps is one of the most popular methodologies in the software engineering.

The goal of the thesis was to find out what monitoring applications are available. One goal was to find out which monitoring application be most suitable for the client's use. One goal was to find out how to bring monitoring application into service. One goal was to find out what are the ways that monitoring application full fills its requirements. The last goal was to find out how monitoring application benefits the client.

Thesis was implemented as a research-based development assignment method which targets to improve process of the client and offers new tools for the process. The first task of the thesis was to find out status for the failure process and for the monitoring at the current point. Based on the outcome of the first task requirements were defined. The third task was to compare different monitoring applications based on requirements. As a result of that comparison one monitoring application was selected for test use. Test use was divided into two phases that were bringing it into service and actual test use.

As a result of thesis Azure Monitor was selected for the client. Another result was documentation for bringing Azure Monitor into service in the client's environment. One most important goal was achieved by finding out the benefits of the Azure Monitor that brings some value to the client. Also, goal about full filling requirements was achieved.

The execution of the thesis was successful. Monitoring application which is most suitable for the client's use was selected successfully. Documentation for bringing monitor application into service was also achieved. Additionally benefits for the client about monitoring application were recognized and the ways to use it for the failure process was also recognized.

Keywords/tags (subjects)

DevOps, Software Developing, Monitoring

Miscellaneous (Confidential information)

No confidential information

Sisältö

1	Johdanto	7
1.1	Tausta	7
1.2	Opinnäytetyö.....	7
1.3	Jydacom Oy	8
2	Tutkimusasetelma	9
2.1	Tutkimusongelma.....	9
2.2	Tutkimusmenetelmä	10
2.3	Rajaukset	10
2.4	Tutkimuksen toteutus	10
3	Ohjelmistotuotanto	11
3.1	Määritelmä	11
3.2	Historia	11
3.2.1	Vesiputousmalli	12
3.2.2	Prototyyppekehitys.....	14
3.2.3	Ketterä kehitys.....	15
4	DevOps.....	16
4.1	Miksi DevOps on olemassa?.....	16
4.2	DevOps ja sen suosio.....	17
4.3	DevOpsin määritelmä.....	18
4.4	DevOpsin vaiheet	18
4.4.1	Suunnittelu (plan)	19
4.4.2	Koodaus (code)	19
4.4.3	Kääntäminen (build)	19
4.4.4	Testaaminen (test).....	19
4.4.5	Julkaiseminen (Release).....	20
4.4.6	Käyttöönotto (deploy)	20
4.4.7	Käyttö (operate).....	20
4.4.8	Monitorointi (monitor)	20
5	Toteutus	23
5.1	Nykytilan selvitys.....	23
5.2	Monitoroinnille asetetut vaatimukset	24
5.3	Monitorointiratkaisujen vertailu	28
5.4	Monitorointiratkaisun testikäyttö.....	30

5.4.1	Azure Monitor	30
5.4.2	Application Insights-palvelun lisääminen koodiin	32
5.4.3	Vaatimusten toteutuminen Application Insights:ssa	37
6	Johtopäätökset.....	45
6.1	Monitorointiratkaisun soveltuvuus toimeksiantajalle	45
6.2	Monitorointiratkaisun käyttöönotto.....	46
7	Pohdinta.....	46
7.1	Jatkokehityskohteet	47
	Lähteet	48
	Liitteet	50
	Liite 1. Application Insights Javascript API wrapper-luokka.....	50
	Liite 2. ErrorHandlerService luokka keskitettyyn virheiden kirjaamiseen	51

Kuviot

Kuvio 1. Vesiputousmalli (Royce 1970).....	13
Kuvio 2. Iteratiivisempi versio vesiputousmallista (Royce 1970)	14
Kuvio 3. DevOpsin suosio (A Maturing DevSecOps landscape – 2021 Global Survey results N.d., 5)	17
Kuvio 4. DevOps-kaavio (What is DevOps? 2020)	19
Kuvio 5. Monitoroinnin yleiskuva web-ympäristössä	25
Kuvio 6. Vaatimusten täyttyminen monitorointiratkaisuissa	29
Kuvio 7. Azure Monitor (Azure Monitor overview 2021)	31
Kuvio 8. Resurssien luonti Azure portaalissa	33
Kuvio 9. Application Insights nuget paketti	33
Kuvio 10. Application Insights:n lisäys koodiin	34
Kuvio 11. Application Insights -palvelun connectionstring AppSettings.json-tiedostossa.....	34
Kuvio 12. ASP.NET Core -sovelluksen transaktioita testauksen aikana Azuressa.....	35
Kuvio 13. Sovelluspakettien lisäys Angularia varten.....	35
Kuvio 14. Application Insights:n connectionstring environment.json-tiedostossa	36
Kuvio 15. Monitorointiluokan ilmentymän luonti ohjelman käynnistyessä.....	36
Kuvio 16. Mukautetun virheen käsittelyn käyttöönotto	37
Kuvio 17. Serilogin nuget-paketti Applicaiton Insights:a varten.....	38
Kuvio 18. LogSettings.json tiedosto Serilogin konfigurointiin	39
Kuvio 19. Lisätty IIS web-palvelimen suorituskyvyn monitorointi.....	39

Kuvio 20. Azuren Live Metrics näkymä	40
Kuvio 21. Azure Metrics suorituskkyvalintoja	40
Kuvio 22. Application Insights -resurssin näkymä taustasovelluksen päätepisteiden suoritusajoista	41
Kuvio 23. Taustasovelluksen virheet näkymä Application Insights -resurssissa.....	42
Kuvio 24. Taustasovelluksen päätepisteiden suoritusajojen muutos.....	43
Kuvio 25. Näkymän vaihdon seuranta	44

Taulukot

Taulukko 1. Saavutettavuuden vaatimukset.....	26
Taulukko 2. Sovellusympäristöön liittyvät vaatimukset	27
Taulukko 3. Monitorointiratkaisuun liittyvät toiminnalliset vaatimukset	28

1 Johdanto

1.1 Tausta

Ohjelmistokehitys sai alkunsa 1940-luvulla, kun ensimmäinen ohjelma kirjoitettiin ja suoritettiin onnistuneesti. 1970-luvulla tietokoneet alkoivat yleistyä teollisuudessa kovaa vauhtia, jolloin myös ohjelmistojen tarve ja suosio kasvoivat. Ohjelmistokehityksen työkaluksi kehitettiin vesiputousmalli, joka jäsenteli ohjelmistokehityksen vaiheet perättäisiksi vaiheiksi ja antoi näin ollen selkeät raamit kehitykselle. 1980-luvulla prototyyppikehitys kasvatti nopeasti suosiotaan johtuen muun muassa sen tavasta tuottaa aluksi nopeasti ja edullisesti ohjelmiston prototyyppi, joka simuloi kuitenkin tarpeeksi tarkasti lopputuotetta. Siitä saadun palautteen perusteella voitiin kehittää lopputuote, joka vastasi paremmin asiakkaiden toiveita. 1990-luvun loppupuolella kehittäjät ja yritykset alkoivat kokeilemaan erilaisia tapoja tehdä ohjelmistoja nopeammin ja joustavammin. Tämän tuloksena vuonna 2001 julkaistiin ketterän kehityksen julistus, joka toimii perustana ketterille kehitysmenetelmille kuten Scrum ja Lean. (A Brief History of Software Development Methodologies 2021).

Nykyään ketterä kehitys ja DevOps ovat saavuttaneet vahvan suosion ohjelmistokehityksessä ja DevOps onkin suosituin käytäntö ohjelmistokehityksessä (A Maturing DevSecOps landscape – 2021 Global Survey results N.d., 5). DevOps-toimintamalli yhdistää tehokkaalla tavalla kehityksen ja ylläpidon niin, että ohjelmistoa voidaan kehittää jatkuvasti perustuen muun muassa tuotannossa havaittuihin asioihin. Yritykset ovat implementoineet DevOps-toimintamallia käyttöönsä monesti sen mukaan mikä heidän tarpeisiinsa on sopivaa. Jotta DevOps-toimintamallista saadaan ohjelmistokehityksessä irti kaikki se potentiaali mikä siinä piilee, niin sen käyttöön on sitouduttava ja se on pyrittävä implementoimaan yrityksen toimintaan mahdollisimman laajasti sekä kokonaisvaltaisesti.

1.2 Opinnäytetyö

Tämän opinnäytetyön toimeksiantajalla Jydacom Oy:llä DevOps-toimintamalli on ollut implementoituna ohjelmistokehitykseen soveltuvien osien. Siinä DevOps-mallia on hyödynnetty muilta osin paitsi monitoroinnin osalta, joka on oikeastaan se osa DevOps-toimintamallissa, joka kytkee ylläpidon takaisin kehitykseen antaen lähtötiedot kehitystarpeille. Tämä on aiheuttanut sen, että ohjelmiston suorituksen seuraaminen tuotannossa on jouduttu toteuttamaan pitkälti manuaalisesti.

Toimeksiantaja haluaa täydentää DevOps-toimintamalliaan lisäämällä siihen tuotantoympäristön monitoroinnin, joka täydentää toimintamallia lähemmäksi "täydellistä" DevOps-ajatusmaailmaa. Tutkimuksen tavoite on kartoittaa tarjolla olevia monitorointiratkaisuja ja niiden tarjoamia ominaisuuksia. Tutkimuksen aikana monitorointiratkaisuja verrataan keskenään ja suoritetaan testikäyttöä valituille ratkaisuille. Tutkimuksen lopputuloksena selvitetään yhden tai useamman monitorointiratkaisun soveltuvuus vertailun ja testikäytön perusteella toimeksiantajan tarpeisiin ja vaatimukseen nähden sekä lisäksi selvitetään, miten monitorointiratkaisu otetaan käyttöön.

1.3 Jydacom Oy

Yritys perustettiin vuonna 1981 Jyväskylässä, jolloin sen nimi oli Jyväsdata Oy. Alusta lähtien yrityksen tavoite oli palvella rakennusalan yrityksiä ja tuottaa alalle räätälöityjä ohjelmistoja, jotka tukivat rakennusliikkeiden liiketoimintaa (Yritys 2022). Yrityksen tarjonta on laajentunut vuosien saatossa ja nykyään sillä on useita ratkaisuja rakennusalan:

- Tarjouslaskenta
- Tuotannonhallinta
- Työmaatuotteet
- Taloushallinto
- Palkat ja HR
- Raportit
- Hankinta ja ostosopimukset

Vuonna 2011 pohjoismaalainen ohjelmistoyritys EVRY osti yrityksen liiketoiminnan, jonka omistuksessa yritys toimi aina vuoteen 2019. Vuonna 2019 suomalainen Tieto Oyj ja norjalainen EVRY yhdistyivät suureksi konserniksi. Yhdistymisen tuloksena perustettiin uusi yritysnimeltään TietoEVRY

Oyj, jonka omistukseen Jydacom siirtyi. Vuonna 2021 tanskalainen EG osti Jydacomin liiketoiminnan ja se siirtyi osaksi pohjoismaiden johtavaa rakennusalan ohjelmistotaloa EG Constructionia. (Yritys 2022).

2 Tutkimusasetelma

2.1 Tutkimusongelma

Toimeksiantajalla on kiinnostusta aloittaa ohjelmiston monitorointi tuotantoympäristössä, jotta ohjelmiston suorituskyvystä ja tapahtuvista virheistä saadaan täsmällisempää tietoa. Tarkoituksena on myös tuottaa tietoa tuotekehitykselle siitä missä kohdin ohjelmaa esiintyy mahdollisia hitautta aiheuttavia ongelmakohtia. Toimeksiantaja haluaa ensin kuitenkin selvittää mitä mahdollisia ratkaisuja ohjelmiston monitorointiin on saatavilla ja miten hyvin ne tuottavat sitä tietoa mitä kävataan eli miten hyvin ne vastaavat toimeksiantajan vaatimuksiin. Lisäksi toimeksiantaja haluaa tietää mitä monitorointiratkaisun käyttöönotto vaatii.

Tutkimusongelmaa tarkasteltiin ja se purettiin neljäksi tutkimuskysymykseksi, joihin tässä tutkimuksessa pyritään vastaamaan:

- **Q1.** Mikä monitorointiratkaisu soveltuu parhaiten toimeksiantajan käyttöön?
- **Q2.** Miten valitun monitorointiratkaisun käyttöönotto tapahtuu?
- **Q3.** Miten valittu monitorointiratkaisu täyttää toimeksiantajan vaatimukset?
- **Q4.** Miten valittu monitorointiratkaisu hyödyttää toimeksiantajaa?

Monitorointiratkaisun soveltuvuudella (Q1) tarkoitetaan sitä, että halutaan selvittää mikä monitorointiratkaisu sopii toimeksiantajan tarpeisiin parhaiten, kun otetaan huomioon niihin kohdistuvat vaatimukset. Monitorointiratkaisun käyttöönoton (Q2) selvityksen tarkoituksena on saada selvitettyä, miten monitorointiratkaisu otetaan käyttöön ja samalla dokumentoida sen käyttöönotto. Tutkimuskysymyksen Q3 tarkoittaa sitä, että tarkastellaan millä tavoin monitorointiratkaisu täyttää siihen kohdistuvat vaatimukset. Monitorointiratkaisun hyödyt (Q4) toimeksiantajalle ovat tärkeä

osa koko työtä. Sen tarkoituksena on saada selville, miten ja millä keinoin toimeksiantaja voi parhaansa mukaan hyötyä ja jopa tehostaa toimintaansa käyttämällä monitorointiratkaisua.

2.2 Tutkimusmenetelmä

Tutkimuksessa on kyseessä kehittämistutkimuksesta, koska sen tavoitteena on parantaa toimeksiantajan prosessia ja antaa uusia työkaluja sen kehittämiseen (Kananen 2012, 19). Tutkimuksen tavoitteena on myös selvittää monitorointiratkaisujen sopivuus toimeksiantajan käyttöön, sekä kuvata sellaisen käyttöönotto toimeksiantajan ympäristössä. Lisäksi tarkoituksena on tarkastella mitä hyötyjä toimeksiantaja monitorointiratkaisusta saa.

Tutkimuksessa kerätään aiheeseen liittyvää tietoa sekä kirjallisista että sähköisistä lähteistä ja syvennyttään aiheeseen sillä tavalla. Lähteinä ovat esimerkiksi monitorointiratkaisujen dokumentaatiot sekä niihin liittyvät julkaisut ja artikkelit. Opinnäytetyö itsessään on kaksivaiheinen tarkoittaen sitä, että ensimmäisessä vaiheessa toteutetaan itse työ ja toisessa vaiheessa kirjoitetaan opinnäytetyöraportti. Tiedonhankinta kuuluu työvaiheeseen, koska sillä luodaan koko työn perusta ja tarpeeksi syvä ymmärrys aihealueesta.

2.3 Rajaukset

Tutkimuksessa keskitytään tuotannossa suoritettavan ohjelmiston monitorointiratkaisuihin eli siihen vaiheeseen DevOps-toimintamallia, jossa ohjelmisto on jo siirretty tuotantoon ja se pyörii siellä. Myöskään tarkoituksena ei ole etsiä monitorointiratkaisua, jolla voidaan monitoroida tuotantoympäristöä vaan keskittyä ratkaisuihin, joilla monitoroidaan itse ohjelmistoa ja sen koodin suorittamista. Tämän rajauksen tarkoitus on keskittää tutkimus siihen, että miten ohjelmiston koodin suorituksesta ja mahdollisista virhetilanteista saadaan kerättyä tietoa tuotekehitykseen ohjelmiston jatkokehitystä varten.

2.4 Tutkimuksen toteutus

Tutkimuksen toteutus alkaa nykytilan kartoittamisella, joka toteutetaan pitämällä palaveri, jossa käydään läpi sitä, miten järjestelmää monitoroidaan tällä hetkellä tuotantoympäristössä. Tämän

avulla on tarkoitus saada ymmärrys siitä, että mitä tietoja monitoroinnilla halutaan kerätä ja samalla saadaan lähtötilanne, johon voidaan verrata monitorointiratkaisun tuomaa hyötyä. Tutkimuksen alussa on myös tavoite kerätä ylös ne kriteerit ja tarpeet, joita monitorointiratkaisuun kohdistuu.

Alkukartoituksen ja nykytilan selvityksen jälkeen tehdään selvitystyötä eri monitorointiratkaisuista, joista sitten tämän selvitystyön perusteella valitaan muutama tarkempaan vertailuun. Kriteerit, joilla tämä ensimmäinen valinta suoritetaan, tarkentuvat esiselvitystyön aikana. Monitorointiratkaisujen vertailussa valittuja ratkaisuja verrataan suhteessa niihin tarpeisiin ja kriteereihin mitä alkukartoituksessa nousee esiin. Vertailun tuloksena saadaan suositus siitä, että mikä / mitkä monitorointiratkaisut olisi hyvä ottaa koekäyttöön.

Vertailun perusteella koekäyttöön valitut monitorointiratkaisut koekäytetään testiympäristössä ja samalla selvitetään, miten niiden käyttöönotto tapahtuu. Tutkimuksen lopputuloksena saadaan selvitettyä monitorointiratkaisujen soveltuvuus toimeksiantajan tarpeisiin.

3 Ohjelmistotuotanto

3.1 Määritelmä

Ensimmäinen määritelmä ohjelmistotuotannosta esiteltiin NATOn konferenssissa vuonna 1968. Sen määriteltiin olevan kurinalaista insinööriyttä, joka pyrkii tarjoamaan keinoja valmistaa laadukas ja tehokas ohjelmisto taloudellisesti. Määritelmä oli tuolloin aikaansa edellä ja pätee hyvin edelleen. (Taina 2006). Ohjelmistotuotanto haluttiin nähdä samantapaisena prosessina kuin sillan tai talojen rakentaminen, jossa lähtökohtana on teoreettinen perusta ja sen lisäksi hyödynnetään vakiintuneita käytäntöjä sekä hyväksi todettuja toimintatapoja (van Vliet 2007, 3).

3.2 Historia

Ohjelmistokehityksen juuret ovat 1940-luvun lopulla, jolloin Tom Killburn kirjoitti maailman ensimmäisen ohjelman, joka suoritettiin 21. kesäkuuta 1948. Ohjelma suoritettiin yhdessä varhaisimmista tietokoneista, jonka olivat rakentaneet itse Tom Killburn ja hänen kollegansa Freddie Williams. Ohjelma oli tehty suorittamaan matemaattisia laskelmia käyttäen konekoodiohjeita.

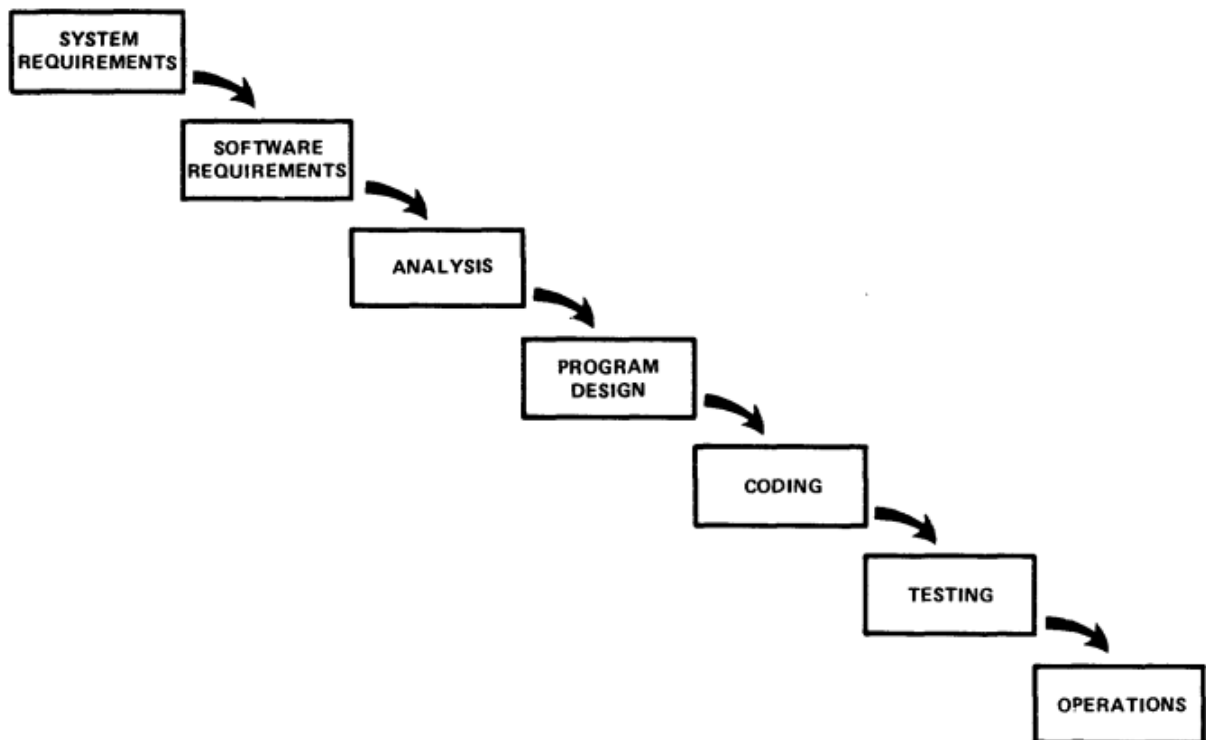
Ensimmäisen ohjelman suoritukseen kului aikaa ”vain” 52 minuuttia, jonka aikana se laski 2:n suurimman jakajan luvun 18 potenssiin. (Yost 2018).

Pitkän aikaa tämän merkittävän tapahtuman jälkeen tietokoneisiin ohjelmoitiin reikäkortteja, joissa reiät tarkoittivat tiettyjä konekoodikäskyjä. Yksi ensimmäisistä julkaistuista korkeamman tason ohjelmointikielistä oli Fortran, joka julkaistiin vuonna 1957. Muut merkittävät ohjelmointikieliset kuten Cobol, BASIC, Pascal ja C saapuivat seuraavan kahden vuosikymmenen aikana. (Yost 2018).

1970-luvulla tietokoneet alkoivat lisääntyä ja lähes kaikilla teollisuudenaloilla otettiin käyttöön tietotekniikkaa, jotta asiakkaille pystyttiin tarjoamaan tehokkaampaa palvelua. Siitä lähtien tietotekniikan teknologinen kehittyminen on ollut todella nopeaa. Tämä tarkoitti myös ohjelmistokehityksen kasvua, jolloin ohjelmistoyrityksiä syntyi nopeasti ja ne alkoivat tarjoamaan asiakkailleen ohjelmia, jotka automatisoivat ja paransivat prosesseja. Ohjelmistojen kasvavan suosion myötä niiden kehitykseen tarvittiin menetelmiä. (A Brief History of Software Development Methodologies 2021).

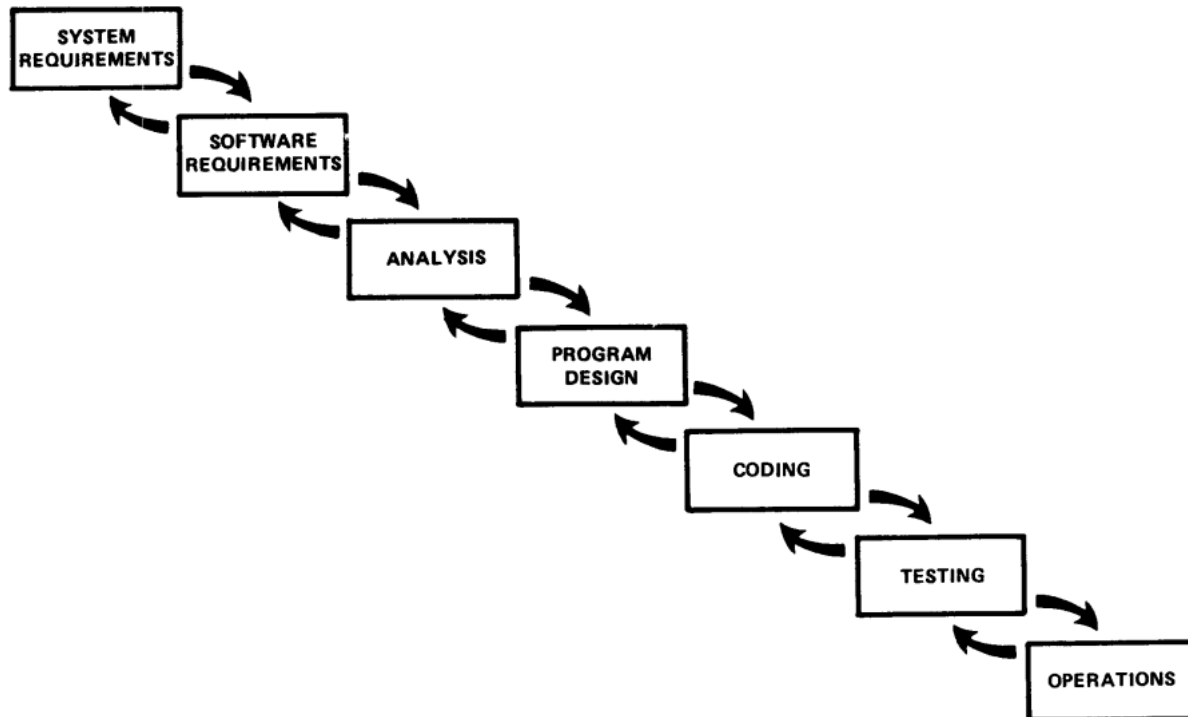
3.2.1 Vesiputousmalli

Winston Royce esitteli vesiputousmallin (kuvio 1) vuonna 1970 ja häntä pidetään sen kehittäjänä, vaikka siihen liittyviä julkaisuja oli julkaistu jo 1960-luvun alkupuolella. (van Vliet 2007, 48). Royce ei itse edes käyttänyt termiä vesiputousmalli vaan hän esitteli prosessin, jossa edetään vaiheesta toiseen, kun edellinen vaihe on kokonaan valmis (Fowler 2019).



Kuvio 1. Vesiputousmalli (Royce 1970)

Royce uskoi mallin ideaan, mutta hän ei kehunut mallia vaan piti sitä riskialttiina ja herkkänä virheille. Tämä perustui siihen, että testaus oli hyvin myöhäisessä prosessin vaiheessa ja mahdolliset virheet havaitaan prosessin lopussa, jolloin niiden korjaaminen on työlästä ja kallista. Pienentääkseen riskiä siitä, että kehityksen kulkua ei saada muutettua tarpeeksi ajoissa Royce kehitti mallista iteratiivisemmän mallin, joka on nähtävillä kuviossa 2. Siinä ideana on, että tieto kulkee myös "vasta virtaan", jolloin edellisessä vaiheessa tehtyt virheet voidaan havaita mahdollisimman aikaisessa vaiheessa ja ne voidaan palauttaa korjattavaksi. (Royce 1970).



Kuvio 2. Iteratiivisempi versio vesiputousmallista (Royce 1970)

Vesiputousmalli sopii käytettäväksi pienissä ohjelmistoprojekteissa, joissa vaatimusmäärittely on tehty huolella ja vaatimukset ovat selkeitä. Jokaisen vaiheen jälkeen tulee tarkastella, eteneekö projekti suunnitellusti vai tarvitseeko se lisää aikaa tai pitääkö se jopa lopettaa. Vesiputousmallin heikkouksina pidetään yleisesti sen joustamattomuutta, vaiheiden riippuvuussuhteita, lopputuloksen selkeyden puuttumista ja hankalaa sekä kallista virheiden korjausta. Sen etuina taas ovat helppo ymmärrettävyys ja selkeä rakenne. (Projektien vesiputousmalli ja sen viisi heikkoutta 2016).

3.2.2 Prototyypikehitys

Ohjelmistokehittäjät alkoivat ymmärtämään, että idean testaaminen käyttäjillä ennen kuin lopullinen ohjelmisto on valmis auttaa kehittämään ohjelmistoja paremmin markkinoille sopiviksi. Tämä oivallus johti uuteen lähestymistapaan ohjelmistotuotannossa, jota kutsutaan prototyypikehitykseksi. (A Brief History of Software Development Methodologies 2021). Siinä tarkoituksena on tuottaa ohjelmasta tai jostakin sen toiminnosta alustava versio, joka esittelee ohjelman toiminnal-

lisuutta, mutta johon ei ole toteutettu ohjelman logiikkaa. Tämän avulla asiakkaille pystytään näyttämään ohjelman tulevaa toiminnallisuutta, josta asiakkaat voivat antaa palautetta. Tämä auttaa kehittäjiä ymmärtämään mitä ohjelmistolta halutaan. (SDLC - Software Prototype Model n.d.).

Prototyypikehitys alkoi kasvattamaan suosiotaan 1980-luvulla ja ajoi nopeasti vesiputousmallin ohi sen suosiossa. Sen myötä ohjelmistoja alettiin kehittämään paljon iteratiivisemmin, kun prototyyppejä testautettiin asiakkailta ja niistä saatujen palautteiden mukaan tehtiin muutoksia. Prototyypikehitys ei kuitenkaan ollut lopullinen ratkaisu ohjelmistokehitykseen vaan 1990-luvulla ohjelmistojen pitkä kehitysaika kehittyi kriisiksi, kun yritykset alkoivat laajentumaan ja liikkumaan nopeammin. Monet projektit jouduttiin keskeyttämään kesken kehitysprosessin, koska ne olivat vanhentuneet. Samaan aikaan teollisuudessa tapahtui teknistä kehitystä, jonka ansiosta yritykset pystyivät tuottamaan parempia tuotteita ja toimittamaan ne markkinoilla entistä nopeammin. Tämä ajanjakso tuli tunnetuksi sovelluskehityksen kriisinä. 1990-luvun lopulla monet ohjelmistokehittäjät ja yritykset alkoivat kokeilla uusia prosesseja, joilla saavutetaan lisää joustavuutta ohjelmistokehitykseen. Tämän seurauksena vuonna 2001 julkaistiin ketterän ohjelmistokehityksen julistus. (A Brief History of Software Development Methodologies 2021).

3.2.3 Ketterä kehitys

Ketterä kehitys tunnetaan paremmin terminä Agile development. Se sai alkunsa vuonna 2001 kun ketterän kehityksen julistus julkaistiin. Julistuksen taustalla on 12 periaatetta, jotka ohjeistavat miten ketterän kehityksen pitäisi toimia. Ketterä kehitys on iteratiivinen lähestymistapa ohjelmistokehitykseen, joka auttaa tiimejä tuottamaan valmiita tuotteita asiakkaille nopeammin. Ketterän kehityksen ideana on tuottaa pieniä osia ohjelmaa kerrallaan, joka sitten kasvaa isoksi kokonaisuudeksi. Näitä lyhyitä jaksoja, jolloin kehitystyötä tehdään, kutsutaan sprinteiksi. Tämä on lähes täysi vastakohta vesiputousmallille, jossa koko ohjelmisto tehdään valmiiksi ennen sen julkaisua. Ketterässä kehityksessä vaatimuksia, suunnitelmia ja tuloksia arvioidaan jatkuvasti, jolloin myös niiden suuntaa voidaan korjata, mikäli muun muassa asiakkaan vaatimukset muuttuvat. (What is Agile? n.d.).

Perinteisessä vesiputousmallissa yksi tiimi osallistuu projektiin ja kun sen osuus on valmis niin projekti "heitetään seinän yli" seuraavalle tiimille. Pahimmassa tapauksessa nämä samaan projektiin

osallistuvat tiimit eivät kommunikoi keskenään millään tavalla ja tiimillä, joka vastaanottaa projektin seuraavaan vaiheeseen ei ole tietoa siitä, miten siihen tilanteeseen on päädytty. Ketterässä kehityksessä tiimien välinen yhteistyö on tärkeässä roolissa. Ketterä kehitys keskittyy erityisesti asiakkaan ja kehityksen väliseen kommunikointiin, jolloin ohjelmistojen tekijöillä säilyy kuva siitä mitä asiakas haluaa. (Hamilton 2022a).

Ketterää kehitystä sovelletaan monissa nykyäänkin käytössä olevissa menetelmissä. Kaikkien näiden menetelmien taustalla on edelleen ne 12 periaatetta, jotka ovat myös ketterän kehityksen juuristuksen takana. Ketterän kehityksen tunnettuja menetelmiä ovat muun muassa Scrum, SAFe ja Kanban. (Hamilton 2022b).

4 DevOps

4.1 Miksi DevOps on olemassa?

Ennen eikä niin kovin kauan aikaa sitten monilla yrityksillä oli tiimeilleen selkeät vastualueet, joissa toinen tiimi hoiti ohjelmiston kehittämisen ja toinen tiimi varmisti ohjelmiston toimimisen tuotannossa. Näiden tiimien voitiin ajatella toimivan omissa silloissaan, ottamatta huomioon sitä mitä toiset tiimit tekevät ja keskittyvän oman vastualueensa hoitamiseen. Tämä johti herkästi tiimien välisen kommunikoinnin puutteeseen, joka aiheutti ongelmia niin kehityksessä kuin tuotannossa. (7 Reasons Why DevOps Matter 2019).

Samalla kun markkinat kasvoivat ja odottivat yhä nopeampaa toimitusta ohjelmistoille, nousi esiin idea siitä, että perinteistä ohjelmistokehityksen mallia pitää uudistaa niin, että ohjelmistokehitys ja tuotannon ylläpito eivät ole toisistaan niin kaukana ja toimisivat yhdessä. Tällä tavoin saavutettaisiin tehokkaampi sekä nopeampi tapa kehittää ohjelmistoja, jolloin markkinoille päästään nopeammin, mutta pienemmällä työmäärällä. (7 Reasons Why DevOps Matter 2019).

DevOps on siis kehitetty ohjelmistokehityksen tehostamiseksi. Sen avulla saavutetaan nopeampi kehityksen läpimenoaika, jolloin ohjelmisto saadaan julkaistua nopeammin. DevOps on käytännöllinen, joten sen periaatteisiin on helppo päästä kiinni. Sen avulla tiimit voivat ratkaista ongelmia nopeammin, koska kehitysiteraatiot pidetään lyhyinä. Se myös tehostaa tuottavuutta, koska useat automatisoidut tehtävät säästävät työaikaa. Edellä mainitut asiat ovat pääsääntöisesti mitattavia,

joten DevOpsin tuottama hyöty on helppo osoittaa todeksi, kun asiat ovat mittavissa. Näiden lisäksi DevOpsin yksi isoimmista asioista on tiimien välinen yhteistyön lisääminen. DevOps rikkoo siilomaisen työskentelyn ajatusmaailman täysin ja lähtee liikkeelle siitä, että kehitys- ja ylläpito -tiimit tekevät tiivistä yhteistyötä ja kommunikoivat keskenään. Joskus tiimit on voitu sulauttaa yhdeksi tiimiksi, jolloin yksi tiimi hoitaa ohjelmiston kehittämisen ja sen ylläpidon.

4.2 DevOps ja sen suosio

DevOps on tällä hetkellä yksi käytetyimmistä ja tunnetuimmista ohjelmakehityksen toimintatavoista (kuvio 3). Sen suosio perustuu sen helppoon omaksumiseen ja muovattavuuteen. Kukin yritys voi ottaa DevOpsin käyttöönsä parhaalla näkemällään tavalla. Osalle se tarkoittaa automatisoituja prosesseja, joilla ohjelmisto siirretään kehityksestä tuotantoon automaattisesti. Osalle se tarkoittaa ajatusmaailman muuttamista niin, että tiimit tekevät tiivistä yhteistyötä. Joillekin se tarkoittaa näitä molempia yhdessä.

MOST PRACTICED DEVELOPMENT METHODOLOGIES:

35.9% DevOps/DevSecOps

31.78% Agile/Scrum

13.02% Kanban

10.02% Waterfall

5.01% Water/Scrum/Fall

4.20% Lean

Kuvio 3. DevOpsin suosio (A Maturing DevSecOps landscape – 2021 Global Survey results N.d., 5)

Lisäksi DevOpsin suosion taustalla on lisääntynyt laatu ohjelmiston koodissa, nopeampi markkinoille pääsy, tietoturva, parantunut yhteistyö ja tyytyväisemmät kehittäjät. (A Maturing DevSecOps landscape – 2021 Global Survey results N.d., 5)

4.3 DevOpsin määritelmä

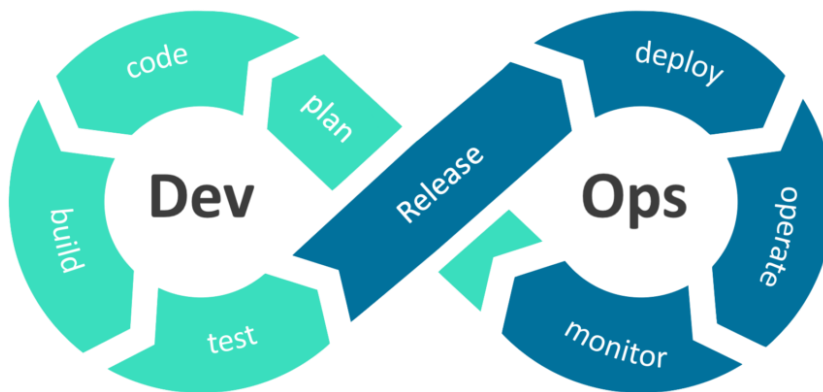
DevOpsin määrittely on monesti hiukan erilainen riippuen lähteestä. Toiset ajattelevat DevOpsin olevan ketju automatisoituja toimintoja, joiden avulla ohjelmisto viedään kehityksestä testauksen kautta tuotantoon mahdollisimman automaattisesti. Toiset taas ajattelevat DevOpsin olevan filosofinen ajatusmalli, jossa seurataan jotakin prosessia orjallisesti ja näin ollen saadaan ohjelmistotuotannosta tehokasta ja ketterää.

Kummatkin näkökulmat ovat tavallaan oikeassa, sillä DevOps on yhdistelmä kulttuurifilosofioita, työkaluja ja käytäntöjä, jotka kasvattavat yrityksen kykyä toimittaa ohjelmistoja ja palveluja nopeammin ja tehokkaammin kuin perinteisiä ohjelmistokehitysprosesseja käyttävät organisaatiot. Saavutetut nopeus ja tehokkuus ohjelmistokehityksessä mahdollistavat yritykselle paremmat lähtökohdat palvella asiakkaita ja kilpailla markkinoilla. (What is DevOps? 2022).

DevOps-termi tulee sanoista development (kehitys) ja operations (tuotanto). Tuohon termiin tiivistyy myös DevOpsin ydin. DevOps-toimintamallissa kehitys- ja tuotantotiimi eivät ole enää toisistaan irrallisia, omissa siiloissaan olevia tiimejä, vaan ne ovat joissakin tapauksissa jopa yhdistetty yhdeksi tiimiksi, jossa saman tiimin työntekijät työskentelevät yli ohjelmiston koko elinkaaren. (7 Reasons Why DevOps Matter 2019).

4.4 DevOpsin vaiheet

DevOps esitetään yleensä perättäisinä vaiheina, jotka muodostavat sulkeutuvan kehän. Tämä tarkoittaa sitä, että viimeisestä vaiheesta saadaan tietoja, jotka ovat lähtökohtana taas ensimmäiselle vaiheelle uudessa syklistä. Kuviossa 4 on esitetty perinteinen tapa, miten DevOpsin vaiheet esitetään.



Kuvio 4. DevOps-kaavio (What is DevOps? 2020)

4.4.1 Suunnittelu (plan)

Suunnitteluvaiheessa tehdään ohjelma tai ominaisuuden vaatimusmäärittely, jossa huomioidaan sen vaatimukset ja tarpeet. Lisäksi tässä vaiheessa ohjelma tai ominaisuus määritellään tarkasti, jotta kehittäjä tietää miten toteuttaa se. (What is DevOps? N.d.)

4.4.2 Koodaus (code)

Koodausvaiheessa kehittäjä toteuttaa ohjelman tai sen ominaisuuden edellisessä vaiheessa luodun määrittelyn mukaisesti ja siirtää koodit lopuksi versionhallintaan. (What is DevOps? N.d.)

4.4.3 Kääntäminen (build)

Tässä vaiheessa ohjelman lähdekoodi käännetään tietokoneen suoritettavaksi koodiksi käyttäen automaattisia kääntöohjelmia. Käännetty koodi paketoidaan ja se on valmiina siirrettäväksi ympäristöön, jossa sitä suoritetaan. (What is DevOps? N.d.)

4.4.4 Testaaminen (test)

Tässä vaiheessa suoritetaan ohjelmiston testaus, joka voidaan suorittaa joko automaatiotestein tai manuaalitestein. Hyvin usein testaus toteutetaan näiden molempien kombinaationa. Testauksella

varmistetaan ohjelmiston toiminta ja saadaan selville virheet ennen kuin se menee tuotantoon.
(What is DevOps? N.d.)

4.4.5 Julkaiseminen (Release)

Tässä vaiheessa hyväksytyn testauksen jälkeen ohjelma julkaistaan saataville tuotantoympäristöön, joka tarkoittaa sitä, että se voidaan ottaa käyttöön.

4.4.6 Käyttöönotto (deploy)

Tässä vaiheessa ohjelma otetaan käyttöön eli asennetaan tuotantoympäristöön ja tehdään tarvittavat konfiguroinnit (What is DevOps? N.d.).

4.4.7 Käyttö (operate)

Tässä vaiheessa ohjelma on käytössä ja käytön tukemiseksi tehdään tarvittavia toimenpiteitä ympäristössä, jotta ohjelma pysyy käytettävissä (What is DevOps? N.d.).

4.4.8 Monitorointi (monitor)

Tässä vaiheessa ohjelmistosta kerätään tietoja, että miten se toimii tuotantoympäristössä. Samalla myös tunnistetaan, mikäli ohjelmassa esiintyy virheitä käytönaikana tai ohjelma on muuten hidas. (What is DevOps? N.d.) Ohjelman lisäksi voidaan tarkkailla ympäristöä ja palvelinta, jolla ohjelmaa suoritetaan. Koska ohjelma ja varsinkin web-ohjelma on monesti kaksi jakoinen eli siinä on käyttöliittymä, jonka kanssa käyttäjä vuoro vaikuttaa. Sen lisäksi siinä on taustasovellus, jonka kanssa käyttöliittymä vuoro vaikuttaa käyttäjän toimiin perustuen. Näin ollen web-ohjelman monitorointi voidaan jakaa kahteen osaan, jotka ovat APM (Application Performance Monitoring) ja RUM (Real User Monitoring).

APM (Application Performance Monitoring)

APM eli Application Performance Monitoring tai Application Performance Management on suunniteltu ohjelman suorituskyvyn monitorointia tai hallinnointia. APM:n tavoite on ylläpitää ohjelmiston tavoiteltua palvelun tasoa. APM keskittyy tunnistamaan ja diagnosoimaan ohjelman

suorituskykyongelmat mahdollisimman aikaisessa vaiheessa, jotta ne ehditään korjata ennen kuin ne aiheuttavat ongelmia käyttäjille. (Watts 2018)

Altvaterin (2015) mukaan APM:n voidaan ajatella olevan toimialan luoma termi kaikelle, joka liittyy koodin, sovellusriippuvuuksien, tapahtuma-aikojen ja yleisen käyttäjäkokemuksen seurantaan. APM keskittyy sovellusten suorituskyvyn heikkenemisen havainnointiin ja diagnosointiin, ennen kuin ne kasvavat suuremmiksi ja alkavat vaikuttamaan ohjelman käytettävyyteen. Tällaisia ongelmia voivat olla esimerkiksi koodin suorituskyvyn ongelmat, sovellusten väliset riippuvuudet, hitaastuneet tapahtuma-ajat ja yleinen käyttökokemuksen lasku. Hyvän APM-ratkaisun tulisi löytää syy ongelmaan, ei vain ratkaisua siihen. (Watts 2018)

IT-alan tutkimusyritys Gartner määrittelee usein teknologiakäsitteiden standardimääritelmiä. Se on määritellyt APM:n olevan mikä tahansa ohjelmisto- tai laitteistokomponentti, joka valvoo suorituskykyä viidellä avainalueella:

1. **Loppukäyttäjäkokemuksen monitorointi ja todellisen käyttäjän monitorointi.** Nämä työkalut tarjoavat tietoa vasteajoista ja virheistä, jotka liittyvät työpöytä-, web- tai mobiilisovellukseen, jonka kanssa käyttäjä on vuorovaikutuksessa. Näiden tietojen avulla voidaan esimerkiksi tunnistaa liittyvätkö ohjelmassa havaitut virheet ja hitaudet ohjelmistokomponenttien yhteensopivuuteen tai tiettyihin sivuihin ja niiden toimintoihin. Ne antavat myös arvokasta tietoa, miten käyttäjät käyttävät ohjelmaa.
2. **Ajonaikainen ohjelmistoriippuvuuksien tunnistaminen.** Tämän avulla voidaan tunnistaa sekä havainnollistaa kuinka paljon ja kuinka usein ohjelmistossa olevat riippuvuudet kommunikoivat keskenään ohjelman suorituksen aikana. Tämä auttaa tunnistamaan mahdollisia ongelmakohtia.
3. **Käyttäjämääritteinen tapahtumaprofilointi.** Tämä seuraa jokaisen käyttäjän toimia ohjelmassa. Käyttäjän alkaessa käyttämään ohjelmaa luodaan uusi tapahtuma, johon liittyen kerätään kaikki tieto siitä mitä käyttäjä ohjelmassa tekee sekä mitä taustatoimintoja käyttäjän toimiin liittyen suoritetaan.
4. **Ohjelmistokomponenttien perusteellinen monitorointi.** Tämä auttaa selvittämään hankalampien ongelmien selvitystä, kuten koodin suoritukseen liittyvät ongelmat. Tässä pureudutaan ohjelmiston koodin tasolle, jossa mitataan sen suorittamisnopeutta sekä kerätään talteen koodin suoritusjärjestys. Tämän lisäksi muun muassa SQL-kysely otetaan talteen.

5. **Analysointi.** Edellä mainittujen tietojen kerääminen ei yksin riitä vaan niitä on myös analysoitava, jotta tiedetään mitä ongelmia ohjelmistossa esiintyy ja miten niitä voidaan korjata. (Watts 2018)

APM:llä on määriteltynäkin varsin lakea merkitys, joten toimijoista riippuen sillä voidaan tarkoittaa eri asioita ja se voidaan toteuttaa eri tavalla (Altvater 2015). Riippumatta APM:n toteutuksesta ja siitä mitä sillä tarkoitetaan, sen avulla pyritään tunnistamaan monia ohjelmistojen yleisiä ongelmia, kuten

- löytää hitaat SQL-kyselyt
- löytää hitaat metodit, joissa ohjelman suoritus kestää
- löytää paikat koodista, joissa virheet tapahtuvat
- tunnistaa ohjelman yleinen suorituskyky ja sen muutokset sekä
- selvittää käyttäjän tapa käyttää ohjelmaa.

RUM (Real User Monitoring)

RUM eli Real User Monitoring on suorituskyvyn seurannan tyyppi, joka tallentaa ja analysoi kaikki käyttäjän suorittamat toiminnot ohjelman käyttöliittymässä. Sitä käytetään mittaamaan käyttäjäkokemusta, keskeisten mittareiden kuten latausaikojen ja tapahtumapolkujen avulla. Se on tärkeä osa ohjelmiston suorituskyvyn monitorointia (APM) kun halutaan monitoroida suorituskykyä käyttäjän näkökulmasta. (Altvater 2020)

RUM on passiivinen web-monitoroinnin muoto, koska sen toiminta perustuu palveluihin, jotka tarkkailevat ohjelmaa taustalla ja seuraavat ohjelman saavutettavuutta, toimivuutta ja reagoitua. RUM kerää kaikki tarvittavat tiedot suoraan käyttäjän selaimesta tai muusta käyttöliittymästä. Selaimissa suoritettavissa ohjelmissa pieni määrä Javascript-koodia on upotettuna jokaiseen näytettävään sivuun. Tämä skripti sitten kerää dataa ja tallentaa käyttäjän toimet lähetettäväksi ne sitten analysoidavaksi monitorointipalveluun. RUM:n avulla voidaan jälkikäteen nähdä mitä käyttäjä on tehnyt ohjelmassa ja millainen käyttäjän kokemus ohjelman suorituskyvystä on ollut. Tarkemmin eriteltynä sen avulla voidaan muun muassa

- kerätä talteen web-ohjelmasta taustasovellukselle lähetetyt kyselyt ja niiden data
- nähdä käyttäjän toimet koko istunnon aikana
- havaita ja tallentaa epänormaalit tapahtumat ohjelmassa, kuten hitaat vasteajat sekä selaimessa tapahtuvat virheet ja
- raportoida ohjelman suorituskyvystä ja saavutettavuudesta. (Altvater 2020)

5 Toteutus

5.1 Nykytilan selvitys

Työ aloitettiin suorittamalla monitorointiin liittyvän nykytilan selvitys. Selvityksessä tutustuttiin prosessiin, jolla häiriötilanteet selvitetään. Tällä pyrittiin siihen, että saadaan ymmärrys siitä, miten monitorointia ja sen tuottamaa tietoa voitaisiin hyödyntää. Lisäksi selvitettiin, miten ohjelmaa monitoroidaan toimeksiantajan toimesta eli miten tietoa kerätään ja miten sitä hyödynnetään, kun ohjelma pyörii tuotannossa. Selvitys toteutettiin palavereiden yhteydessä, joissa käytiin läpi häiriötilanteiden prosessi ja itse monitorointi.

Häiriötilanteiden prosessi

Tämä häiriötilanteiden prosessi kuvaa sitä tilannetta, kun asiakas havaitsee ohjelmassa häiriön ja ilmoittaa siitä eteenpäin eikä tässä oteta huomioon sitä, miten prosessi etenee, jos häiriö havaitaan toimeksiantajan toimesta.

Kun asiakas havaitsee ohjelmassa häiriön, hän on yhteydessä asiakaspalveluun, joka luo häiriöstä tiketin Zendesk-palveluun. Asiakaspalvelu aloittaa häiriön selvittämisen ja mikäli kyse on ohjelman virheellisestä käytöstä he opastavat asiakasta ohjelman käytössä. Mikäli kyse on ohjelman laajemmasta ongelmasta, joka aiheuttaa merkittävää haittaa ohjelman käyttämiselle, asiakaspalvelu laittaa viestiä Häiriötilanteet Teams-kanavalle, joka toimii sisäisenä ilmoituskanavana häiriöille. Riippuen ohjelman tyypistä sitä aletaan selvittämään joko käyttöympäristöt ja/tai tuotekehityksen

henkilöiden toimesta. Riippuen häiriöstä ja sen laadusta häiriötilanteen selvityksessä voidaan käyttää avuksi tietokantaan ja tekstitiedostoihin tallennettuja lokitietoja. Näiden tietojen läpi käynti on hyvin työlästä sillä, se on tehtävä manuaalisesti, ja monesti tietoja joutuu yhdistelemään.

Monitorointi

Heti alussa selvisi, että ohjelmistoa ei monitoroida erillisiä ohjelmia hyödyntäen. Ympäristön monitoroinnin toteutti palveluntarjoaja omilla monitoreillaan, mutta itse ohjelman monitorointi oli puutteellinen. Ohjelmasta kyllä kerättiin lokeja virheistä tai hitaista kutsuista, mutta niitä ei talletettu keskitetysti. Mikäli taustasovellukselle suoritettu kutsu päättyi virheeseen tai se kesti yli 3 sekuntia se lokitettiin tietokantaan. Tietokantaan lokitettiin myös kaikki ulkoisiin palveluihin tehtävät kutsut. Taustasovelluksessa tapahtuvat virheet lokitettiin myös tekstitiedostoihin palvelimelle, joihin talletettiin tietoa muun muassa siitä missä kohti koodia virhe tapahtui. Selainpään ohjelmassa tapahtuvat virheet näkyvät ainoastaan käyttäjälle ja ne päättyvät selaimen konsoliin, joten niitä ei saatu kerättyä talteen.

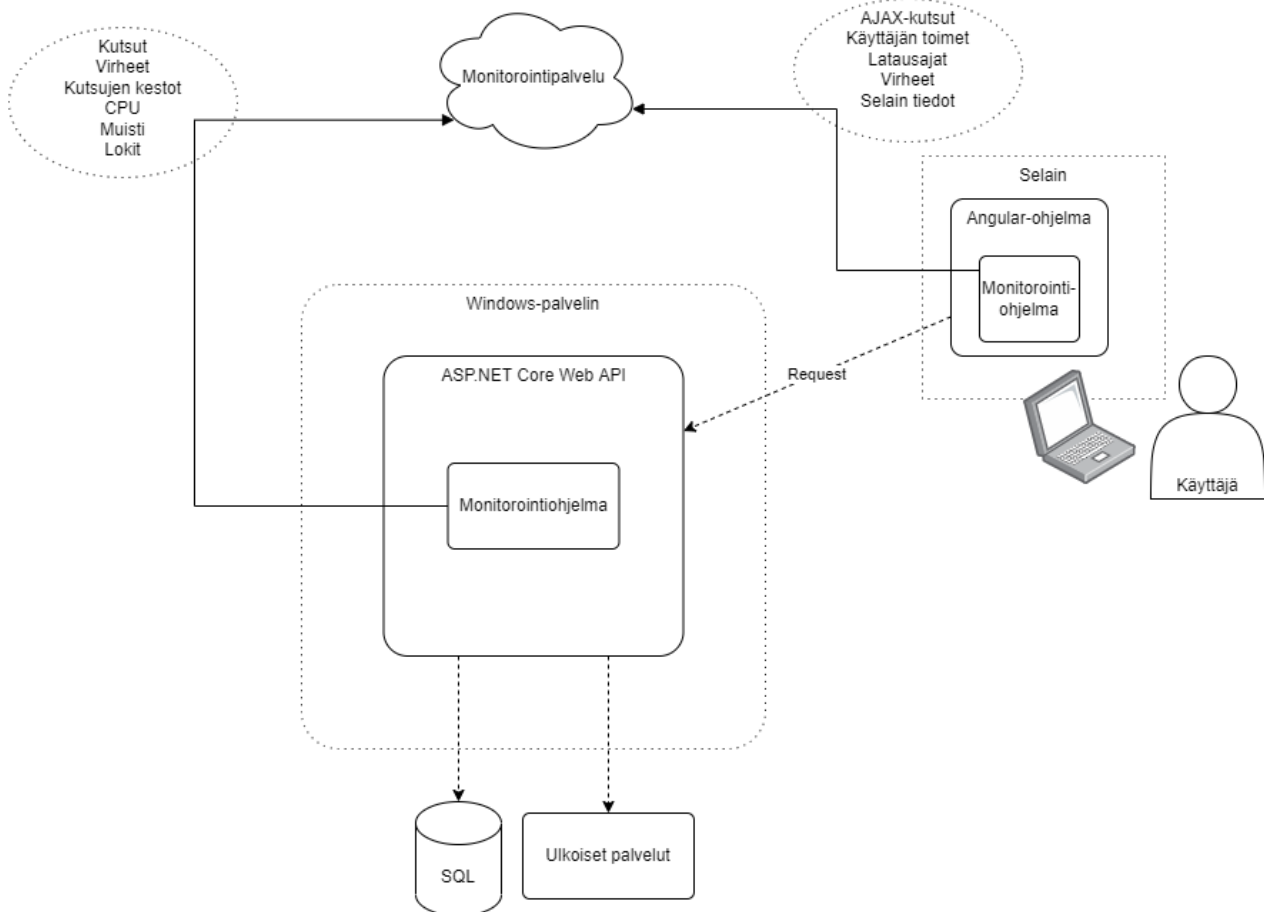
Nykytila ohjelmiston monitoroinnin osalta on siis hyvin manuaalinen eli varsinaista suorituksen aikaista monitorointia ei ole. Tämän takia tietoa ohjelman suorituskyvyn heikkenemisestä tai tapahtuvista virheistä on hyvin hankala saada ennen kuin asiakas niistä ilmoittaa. Tämä aiheuttaa sen, että muun muassa virhetilanteiden selvittäminen on hidasta ja vaikeaa sillä välttämättä kaikki tarvittava tieto ei ole saatavilla, kuten selainpään virhetiedot. Muutenkin tiedon yhdistäminen niin tekstitiedostoihin kerätyistä lokeista kuin tietokantaan tallennettavista lokeista on työlästä. Lisäksi tietokanta kyselyiden tekeminen vaatii SQL-osaamista, joten kaikki eivät sinne tallennettuja tietoja pysty katselemaan.

5.2 Monitoroinnille asetetut vaatimukset

Nykytilan selvityksen perusteella oli selkeää, että monitorointiratkaisulle on selvästi tarvetta. Virheet ja ohjelman yleinen suorituskyvyn heikkeneminen olisi hyvä saada selville ennen kuin asiakas sen kokee. Tähänhän myös DevOps tähtää. Toimeksiantajalle on muodostunut varsin hyvä kuva siitä, mitä asioita ohjelmassa halutaan monitoroida. Tämä helpotti huomattavasti vaatimusten

määrittelyä. Vaatimusten määrittelyn ideana oli kerätä kasaan ne kriteerit, joiden pohjalta monitorointiratkaisuja arvioidaan ja saadaan muodostettua kuva siitä mitä ja miten niiden avulla monitorointia voitaisiin suorittaa.

Toimeksiantajalla on eri teknologioilla toteutettuja ohjelmistokokonaisuuksia. Tässä työssä keskityttiin niistä uusimpaan, web-pohjaiseen ratkaisuun. Siinä taustalla toimii ASP.NET Corella tehty REST API ja käyttöliittymänä Angularilla tehty ohjelma. Sovelluspalvelimen käyttöjärjestelmänä toimii Windows. Monitorointiratkaisu on siis tarkoitus lisätä sekä ASP.NET Core -sovelluksen että Angular-ohjelman koodiin, joista se lähettää tietoja eteenpäin. Kuviossa 5 on esitetty monitoroinnin periaate web ympäristössä.



Kuvio 5. Monitoroinnin yleiskuva web-ympäristössä

Näiden tietojen pohjalta lähdettiin keräämään vaatimuksia, joita monitorointiratkaisuihin liittyi. Vaatimuksia määriteltäessä otettiin huomioon toimeksiantajan tarpeet sekä myös käytettävän ympäristön ja käytettävien teknologioiden luomat vaatimukset. Vaatimukset määriteltiin lopputuloskeskeisesti. Tämä tarkoittaa sitä, että listattiin asiat mitä ohjelmasta halutaan saada selville monitorointia hyödyntämällä. Vaatimusmäärittelyn pohjana käytettiin dokumenttia, joka oli luotu erillisen projektiryhmän palaverin aikana. Siihen oli kirjattuna niitä ominaisuuksia ja tarpeita mitä monitoroinnilla halutaan saavuttaa. Näitä asioita sitten tarkennettiin tai muunnettiin paremmin vaatimuksiksi sopiviksi ja niistä muodostui monitorointiratkaisuihin kohdistuvat vaatimukset. Tällaisia kirjattuja asioita olivat esimerkiksi ohjelmassa tapahtuvat virheet ja kutsujen vasteajat. Vaatimuksia oli myös erityyppisiä, sen mukaan mihin osa-alueeseen ne liittyvät. Taulukossa 1 on nähtävillä monitorointipalvelun saavutettavuuteen liittyvät vaatimukset.

Taulukko 1. Saavutettavuuden vaatimukset

Tunniste	Tyyppi	Vaatus	Selite	Kohde
AvailReq001	Saavutettavuus	Monitorointipalveluun pitää kirjautua		Monitorointipalvelu (käyttöliittymä)
AvailReq002	Saavutettavuus	Monitorointipalveluun pitää olla henkilökohtaiset tunnukset		Monitorointipalvelu (käyttöliittymä)
AvailReq003	Saavutettavuus	Monitorointipalveluun on päästävä selaimen kautta		Monitorointipalvelu (käyttöliittymä)

Taulukossa 2 on esitetty sovellusympäristöön ja käytettyihin teknologioihin liittyvät vaatimukset. Nämä vaatimukset ovat pakollisia monitorointiratkaisulle, mikäli se halutaan ottaa käyttöön, sillä ilman näiden täyttymistä monitorointiratkaisua ei pysty ottamaan käyttöön toimeksiantajan sovellusympäristössä.

Taulukko 2. Sovellusympäristöön liittyvät vaatimukset

Tunniste	Tyyppi	Vaatus	Selite	Kohde
EnvReq001	Ympäristö	Pystyttävä monitoroimaan ASP.NET Core ohjelmaa		Monitorointiratkaisu
EnvReq002	Ympäristö	Pystyttävä monitoroimaan Angular ohjelmaa		Monitorointiratkaisu
EnvReq003	Ympäristö	Pystyttävä monitoroimaan IIS web-palvelinta		Monitorointiratkaisu
EnvReq004	Ympäristö	Toimittava Windows-palvelimella		Monitorointiratkaisu
EnvReq005	Ympäristö	Toimittava itse hostatulle palvelimella (On-Premise)		Monitorointiratkaisu

Taulukossa 3 on näkyvillä monitorointiratkaisuihin liittyvät toiminnalliset vaatimukset. Näiden vaatimuksien tarkoituksena on määrittää mitä ominaisuuksia monitorointiratkaisuissa halutaan olevan eli mitä tietoja monitoroinnilla halutaan saada ohjelmistosta. Vaatimukset on kohdennettu taulukossa tarkemmin käyttäen Kohde-saraketta kuvaamaan sitä, että mistä ohjelmiston osasta tietoja halutaan saada.

Taulukko 3. Monitorointiratkaisuun liittyvät toiminnalliset vaatimukset

Tunniste	Tyyppi	Vaatus	Selite	Kohde
FuncReq001	Toiminnallisuus	Monitoroitava CPU-kuormaa	Palvelimen CPU	Palvelin
FuncReq002	Toiminnallisuus	Monitoroitava muistin käyttöä	Palvelimen muisti	Palvelin
FuncReq003	Toiminnallisuus	Lokitiedot pitää saada keskitettyä	Kaikki lokitiedot (tietokanta ja tekstitiedosto) saatava koottua yhteen paikkaan	Taustasovellus (API)
FuncReq004	Toiminnallisuus	Monitoroitava API-kutsujen kesto		Taustasovellus (API)
FuncReq005	Toiminnallisuus	Kerättävä talteen API-kutsujen virheet		Taustasovellus (API)
FuncReq006	Toiminnallisuus	Mitattava API:n endpointtien suorituskykyä (trendaavuus)	Miten nopeus muuttuu yli versioiden	Taustasovellus (API)
FuncReq007	Toiminnallisuus	Mitattava sivujen latausajat		Käyttöliittymä (Frontend)
FuncReq008	Toiminnallisuus	Mitattava toimintojen suoritusajat		Käyttöliittymä (Frontend)
FuncReq009	Toiminnallisuus	Kerättävä talteen selainpään virheet		Käyttöliittymä (Frontend)
FuncReq010	Toiminnallisuus	Monitoroitava käyttäjän toimet		Käyttöliittymä (Frontend)

5.3 Monitorointiratkaisujen vertailu

Kun vaatimukset olivat listattuina, lähdettiin kartoittamaan mahdollisia monitorointiratkaisuja.

Monitorointiratkaisuja on tarjolla paljon ja hyvin nopeasti oli selvää, että liian montaa ei kannata

valita vertailuun mukaan vaan jonkinasteinen esikarsinta täytyy suorittaa ennen varsinaista vertailua. Monitorointiratkaisuissa on paljon samoja ominaisuuksia, mutta ne on saatettu toteuttaa eri tavoilla. Vertailuun haluttiin valita ratkaisut, jotka eroavat toisistaan kuitenkin jonkin verran. Vertailuun valittiin Azure Monitor, Dynatrace, New Relic ja Elastic. Näistä Azure Monitor edustaa puhtaasti pilviympäristössä toimivaa monitorointiratkaisua, joka toimii Azuressa. New Relic on erillinen myöskin pilvessä toimivia monitorointiratkaisu. Elastic ja Dynatrace taas ovat monitorointiratkaisuja, jotka voidaan asentaa tarvittaessa omille palvelimille, jolloin data ei siirry organisaation ulkopuolelle.

Vertailu suoritettiin pohjautuen siihen tietoon mitä vertailuun valittujen ratkaisujen dokumentaatiosta ja nettisivustoilta oli saatavissa. Ratkaisuja ei suoranaisesti verrattu toisiinsa vaan tehtiin lähinnä tutkimuksena, että täyttävätkö ne niille asetetut vaatimukset (taulukko 1) niiden dokumentaation perusteella. Kuviossa 6 on nähtävillä, miten monitorointiratkaisut täyttävät vaatimukset. Kuten kuviosta 6 nähdään kaikista monitorointiratkaisuista, löytyi lähes kaikki vaatimuksia vastaavat ominaisuudet. Vaatimus AvailReq004 eli monitorointipalvelun asennus omille palvelimille oli ainut vaatimus, jota kaikki ratkaisut eivät täyttäneet. Vertailun tuloksena voidaan todeta, että vaatimukset olivat hyvin tavanomaisia monitorointiratkaisuilla, koska ne kaikki täyttivät vaatimukset.

	AvailReq				EnvReq					FuncReq									
	001	002	003	004	001	002	003	004	005	001	002	003	004	005	006	007	008	009	010
Azure Monitor	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Dynatrace	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
New Relic	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
Elastic	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Kuvio 6. Vaatimusten täyttyminen monitorointiratkaisuissa

Testikäyttöön valittiin Azure Monitor. Valintaan vaikuttivat vertailun tuloksien lisäksi muun muassa se, että toimeksiantajalla on jo olemassa olevia toimintoja Azuressa, joten resurssien perustaminen sinne käy helposti. Lisäksi siihen kuuluu 5 GB dataa kuukautta ja tilausta kohden, joten monitoroinnin testaus on varsin edullista eikä se sido mihinkään jatkoa ajatellen.

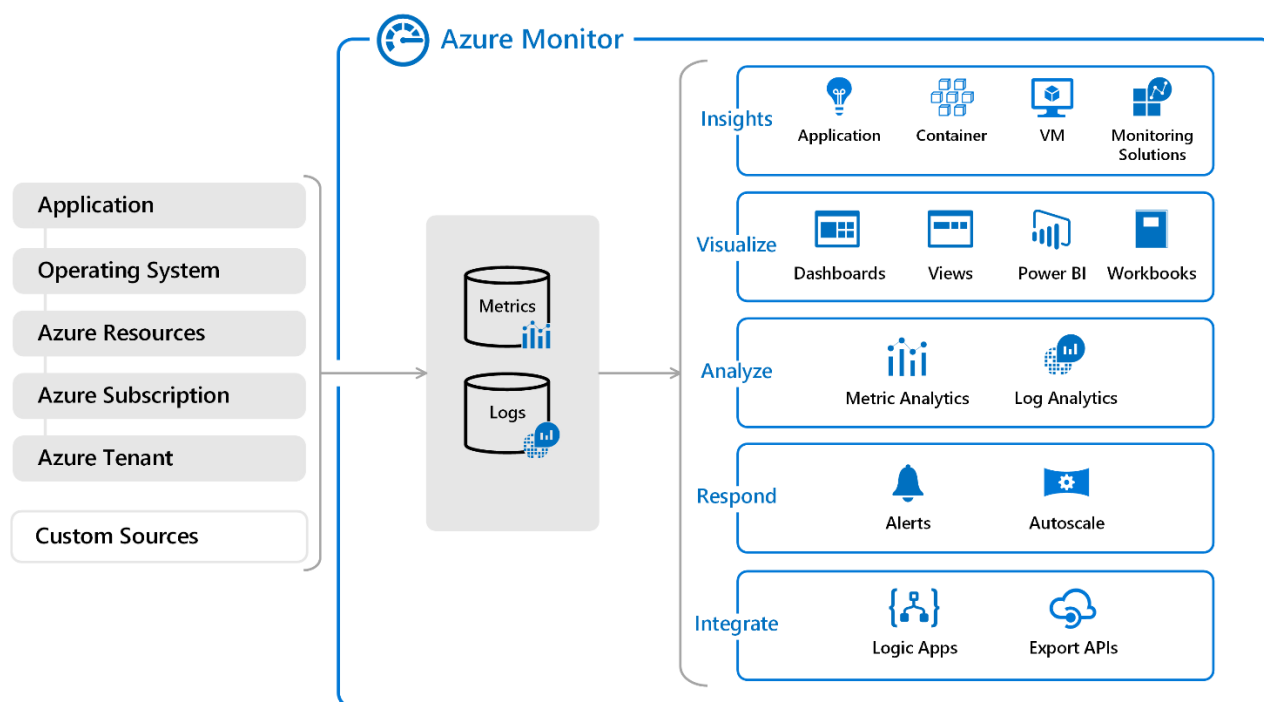
5.4 Monitorointiratkaisun testikäyttö

Testikäytön tarkoituksena oli selvittää, miten monitorointiratkaisu otetaan käyttöön ja mitä sen käyttöönotto edellyttää. Lisäksi halutaan selvittää käytännössä mitä asioita pystytään monitorimaan ja miten monitorointi hyödyttää toimeksiantajaa. Näiden avulla saadaan kuva siitä, minkälainen hyöty monitoroinnista olisi toimeksiantajalle. Testikäyttö tapahtuu ottamalla Azure Monitor käyttöön testiympäristössä ja suorittamalla testiympäristön monitorointia sillä.

5.4.1 Azure Monitor

Azure Monitor on Microsoftin pilvipohjainen monitorointiratkaisukokonaisuus, jossa kerätty data tallennetaan Azuren pilvipalveluun ja käsitellään siellä. Se auttaa maksimoimaan sovellusten ja palveluiden saatavuuden sekä suorituskyvyn. Sen avulla sovelluksesta saadaan kattavasti kerättyä ja analysoitua tietoa, toimii sovellus sitten pilvessä tai paikallisessa ympäristössä. Tiedot auttavat ymmärtämään kuinka sovellus toimii ja tunnistamaan ennakoivasti siihen kohdistuvat ongelmat. (Azure Monitor overview 2021).

Azure Monitor on laajakokonaisuus, jonka avulla voidaan monitoroida niin sovellusta kuin myös sitä ympäristöä, jossa sovellusta suoritetaan. Kuviossa 7 on esitetty Azure Monitorin sisältämät toiminnot. Vasemmalla ovat kohteet, joista dataa voidaan monitoroinnin avulla kerätä. Kuten kuvioista nähdään ne eivät ole osa Azure Monitoria, mutta kuvaavat hyvin sitä, mistä kaikkialta dataa voidaan kerätä.



Kuvio 7. Azure Monitor (Azure Monitor overview 2021)

Keskimmäisenä kuviossa ovat datavarastot, jotka edustavat kahdentyyppistä dataa, jota Azure Monitoriin tallennetaan: metriikka- ja lokitieto. Metriikkatieto on numeerista dataa, joka kuvaa sovelluksen tilaa numeerisesti tietyssä ajanhetkenä, kuten esimerkiksi käytettyä muistia. Ne kerätään aina tietyin ajanvälein ja ne ovat yksilöity aikaleimalla. Metriikat tallennetaan Azuressa aikasarjatietokantaan, joka on optimoitu analysoimaan aikaleimattua dataa. Metriikkatietoja voidaan käsitellä algoritmeilla, jotta niistä saadaan hyödynnettävää tietoa. Niitä voidaan muun muassa verrata toisiinsa tai niiden perusteella saadaan selville suorituskyvyn trendi, eli se onko suorituskyky huonontunut vai parantunut tietyssä ajanjaksossa. Metriikkatieto on kevyttä ja se mahdollistaa myös lähes reaaliaikaisen seurannan muun muassa palvelimen suorituskyvystä tai sovelluksen vastaajista (Azure Monitor Metrics overview 2021).

Lokitieto taas pitää sisällään kaiken muun datan kuten lokit, virheet, kutsut ja kaiken muun mitä halutaan kerätä. Sitä voidaan rikastaa helposti lisäämällä omia tietoja lähetettävän datan mukaan. Azure Monitorin toiminnot käyttävät lokitietoa, kun ne lähettävät dataa palveluun. Lokitietoja voidaan helposti suodattaa, järjestää, vertailla tai visualisoida Azuren Kusto Query Language avulla, joka on kehitetty lokitietojen kyselyjä varten. (Azure Monitor Logs overview 2022).

Oikealla kuviossa ovat Azure Monitorin palvelut, joiden avulla dataa voidaan muun muassa kerätä, analysoida sekä visualisoida. Application Insights on Azuren palvelu, jolla sovelluksia monitoroidaan niiden suorituksen aikana. Se on APM-ratkaisu, joka muun muassa,

- tunnistaa poikkeavuudet sovelluksen toiminnassa automaattisesti
- auttaa tunnistamaan ongelmat analyysityökalujen avulla ja
- auttaa näkemään mitä käyttäjä oikeasti tekee (RUM).

Application Insights -palvelua voidaan käyttää kahdella tavalla. Ensimmäinen tapa on asentaa instrumentointipaketti sovellukseen eli lisätä monitorointi koodin sekaan. Toinen tapa on asentaa Application Insights agentti palvelimelle, jossa sovellusta ajetaan. Agentti mahdollistaa myös niin kutsutun koodittoman monitoroinnin eli itse sovelluksen lähdekoodiin ei tarvitse tehdä muutoksia. Kooditon monitorointi ei tue niin montaa ympäristöä kuin perinteinen tapa. Ja lisäksi se luo rajoitteita kustomoitujen tietojen keräämiseen. Application Insights tukee monia ohjelmointikieliä, kuten Javascriptia jolloin myös käyttäjän käyttämää web-ohjelmaa voidaan monitoroida. (Application Insights overview 2022).

Tässä työssä keskitytään itse sovelluksen monitorointiin, joten tietojen keräämiseen sovelluksesta käytetään Application Insights -palvelua. Tämän vuoksi Azure Monitorin muut Insights-palvelut jätettiin huomioimatta.

5.4.2 Application Insights-palvelun lisääminen koodiin

Azure Monitorin testaamista varten Azureen luotiin tarvittavat resurssit, jotta monitorointia voidaan käyttää (kuvio 8). Azureen luotiin Application Insights -resurssi, joka on Azuren APM-ratkaisu, jolla suoritetaan sovelluksen monitorointia.

Application Insights

Monitor web app performance and usage

Basics Tags Review + create

Create an Application Insights resource to monitor your live web application. With Application Insights, you have full observability into your application across all components and dependencies of your complex distributed architecture. It includes powerful analytics tools to help you diagnose issues and to understand what users actually do with your app. It's designed to help you continuously improve performance and usability. It works for apps on a wide variety of platforms including .NET, Node.js and Java EE, hosted on-premises, hybrid, or any public cloud. [Learn More](#)

PROJECT DETAILS

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ ↓
 Resource Group * ⓘ ↓
 (New) AppInsights_Test
[Create new](#)

INSTANCE DETAILS

Name * ⓘ ✓
 AppInsights_Test
 Region * ⓘ ↓
 (Europe) North Europe
 Resource Mode * ⓘ Classic Workspace-based

WORKSPACE DETAILS

Subscription * ⓘ ↓
 *Log Analytics Workspace ⓘ ↓
 DefaultWorkspace-~~XXXXXXXXXX~~-NEU [north...]

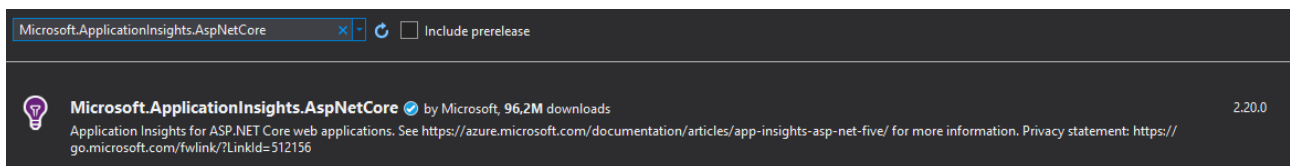
Review + create

< Previous

Next: Tags >

Kuvio 8. Resurssien luonti Azure portaalissa

Kuten kuviosta 8 nähdään, luotiin samalla myös uusi tarvittava workspace, jota käytetään tietojen tallentamiseen. Kun tarvittavat resurssit oli luotu Azureen alettiin Application Insights -palvelua lisäämään ASP.NET Core -sovelluksen koodiin, jotta monitorointi onnistuu. Tarvittavat ohjelmistopakettit lisättiin Nuget-paketteina (kuvio 9).



Kuvio 9. Application Insights nuget paketti

Kun tarvittavat ohjelmistopakettit oli lisätty, voitiin aloittaa itse koodin muokkaaminen. Ensimmäisenä ohjelman startup-luokkaan ja siellä ConfigureServices-metodiin lisättiin kuvan 10 mukaisesti kaksi metodikutsua, joilla saadaan lisättyä Application Insights mukaan ohjelmaan. Ensimmäisellä perustetaan Application Insights -palvelu, joka hoitaa tiedon keräämisen ja lähettämisen Azureen.

Toisella konfiguroidaan palvelua niin, että se kerää SQL-kyselyn talteen ja lähettää sen Azureen, silloin kun SQL-kysely tehdään.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddApplicationInsightsTelemetry(
        new ApplicationInsightsServiceOptions {
            ConnectionString = Configuration["ApplicationInsights:ConnectionString"]
        });

    services.ConfigureTelemetryModule<DependencyTrackingTelemetryModule>(
        (module, o) => {
            module.EnableSqlCommandTextInstrumentation = true;
        });

    services.AddControllers();
}
```

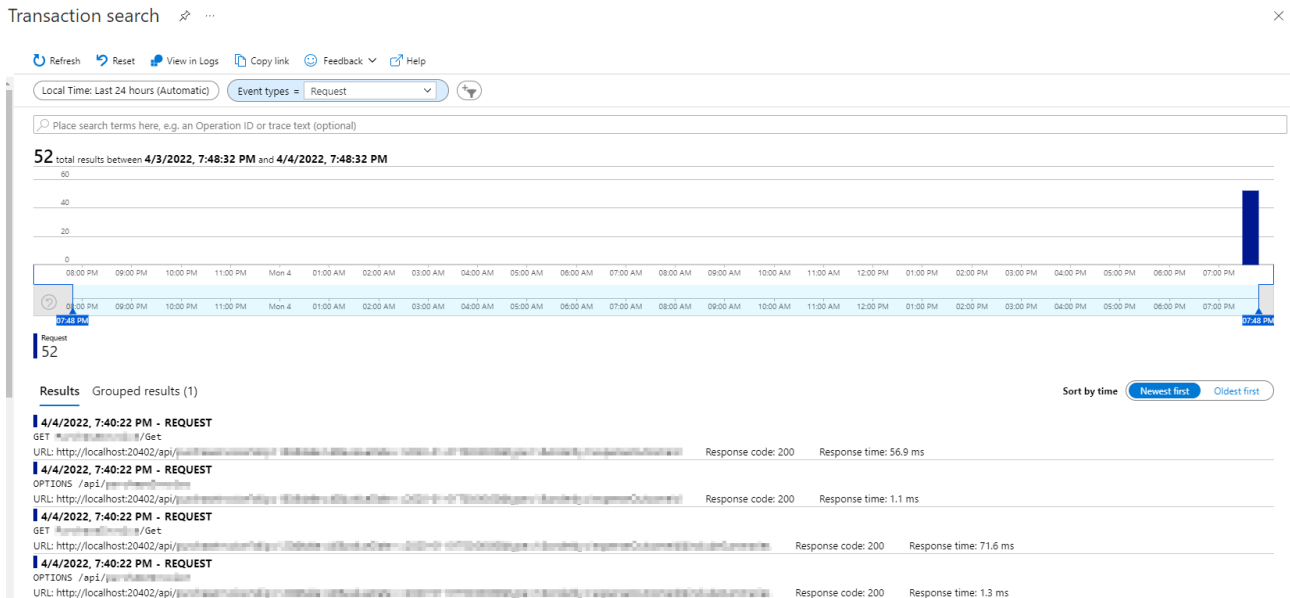
Kuvio 10. Application Insights:n lisäys koodiin

Jotta Application Insights -palvelun rekisteröinti onnistuu, tarvitsee se Connectionstring:n, jonka avulla se ottaa yhteyden Azureen. Connectionstring lisättiin ohjelman AppSettings.json-tiedostoon kuvion 11 mukaisesti.

```
"ApplicationInsights": {
  "ConnectionString": "<Put connectionstring from Azure here>"
}
```

Kuvio 11. Application Insights -palvelun connectionstring AppSettings.json-tiedostossa

Tämän jälkeen Application Insights on valmiina käyttöön ja se alkaa lähettää ASP.NET Core -sovelluksesta kerättyjä tietoja Azureen kun ohjelma käynnistetään ja siihen lähetetään kutsuja käyttöliittymästä. Tässä vaiheessa ASP.NET Core -sovelluksen monitoroinnin toimintaa testattiin käynnistämällä ohjelma ja käyttämällä sitä normaalisti käyttöliittymän kautta. Testissä huomattiin, että Application Insights lähetti onnistuneesti tietoja Azureen (kuvio 12) ASP.NET Core -sovelluksesta, joten taustasovelluksen monitoroinnin voitiin todeta onnistuvan.



Kuvio 12. ASP.NET Core -sovelluksen transaktioita testauksen aikana Azuressa

Seuraavaksi Application Insights lisättiin käyttöliittymän Angular-sovellukseen käyttäen Application Insights Javascript SDK:ta. Azuren tarvittavat resurssit on jo luotu, joten voitiin aloittaa suoraan Application Insights:n lisääminen koodiin. Ensin tarvitsi lisätä tarvittavat sovelluspaketit, jotka Angularin puolella hoidetaan npm:llä (Nuget Package Manager). Application Insights -palvelun sovelluspaketit lisättiin kuvion 13 mukaisella komennolla, joka suoritettiin terminaalista.

```
C:\Users\haltt\Documents\Omat>npm i --save @microsoft/applicationinsights-web
```

Kuvio 13. Sovelluspakettien lisäys Angularia varten

Sovelluspakettien lisäyksen jälkeen päästiin lisäämään monitorointia itse ohjelmaan. Angularissa kuten ASP.NET Core -sovelluksessaakin yhteyden muodostaminen Azureen tapahtuu ConnectionString:n avulla. Angularissa ConnectionString lisättiin environment.json-tiedostoon kuvion 14 mukaisesti.

```
"appInsights" : {
  "connectionString": "<Put connectionString from Azure here>"
}
```

Kuvio 14. Application Insights:n connectionString environment.json-tiedostossa

Application Insights -palvelun käyttöä varten luotiin Angular-projektiin wrapper-luokka MonitoringService (Liite 1), joka pitää sisällään Application Insights Javascript API:n sekä toteuttaa halutut metodit monitorointia varten. Ideana wrapper-luokassa on se, että Application Insights -palveluun liittyvät asiat saadaan keskitettyä yhteen luokkaan, joka on vastuussa monitoroinnista. Mikäli monitorointia, kuten toteutettuja metodeja halutaan muokata, voidaan se tehdä tässä luokassa. Jotta ohjelman monitorointi alkaa heti kun ohjelma on käynnistetty, pitää wrapper-luokasta luoda ilmentymä heti kun ohjelma käynnistetään. Se onnistuu ohjelman AppComponent-luokassa kuvion 15 mukaisesti lisäämällä monitorointi luokka AppComponent-luokan konstruktoriin, jolloin Angular luo siitä automaattisesti ilmentymän, kun ohjelma käynnistyy eli kutsuu kyseistä konstruktoria.

```
export class AppComponent implements OnInit{
  constructor(private monitoringService: MonitoringService){}

  ngOnInit() {
  }
}
```

Kuvio 15. Monitorointiluokan ilmentymän luonti ohjelman käynnistyessä

Ohjelmaan lisättiin vielä yleinen virheiden käsittely, jotta käsittelemättömät virheet saadaan talteen ja ne lähetetään Azureen monitoroinnin toimesta. Tätä varten ohjelmaan luotiin liitteen 2 mukainen ErrorHandlerService-luokka, joka hoitaa virheen lokituksen käyttämällä Application Insights API:n logException-metodia. Ohjelma asetetaan käyttämään haluttua virheen käsittelyä AppModule-tiedostossa kuvion 16 mukaisesti.

```
import { ErrorHandlerService } from './services/errorHandler.service';
import { ErrorHandler, NgModule } from '@angular/core';

@NgModule({
  providers: [
    {provide: ErrorHandler, useClass: ErrorHandlerService}
  ],
})
```

Kuvio 16. Mukautetun virheen käsittelyn käyttöönotto

Tämän jälkeen Application Insights on lisätty sekä ASP.NET Core -sovellukseen, että käyttöliittymään eli Angular-ohjelmaan. Monitorointia testattiin tässä kohdin, jotta varmistuttiin että Application Insights lähettää tietoja molemmista sovelluksista onnistuneesti Azureen. Testeissä kävi ilmi, että tiedot lähetettiin Azureen ilman ongelmia, mutta Application Map, joka näyttää kutsujen ja monitoroinnissa mukana olevien sovelluksien riippuvuudet toisiinsa ei osannut näyttää niitä oikein. Selvittelyn jälkeen selvisi, että Application Map näkymässä on oletusarvoisesti uuden version esikatselutila päällä ja uudessa versiossa on bugi, joka aiheuttaa sen, että kaikkia riippuvuuksia ei tunnisteta oikein.

5.4.3 Vaatimusten toteutuminen Application Insights:ssa

Kun Application Insights oli saatu toimimaan ja lähettämään dataa Azureen oli aika alkaa tarkastella miten Azure Monitor kokonaisuutena vastaa monitoroinnin vaatimuksiin.

Käyttöympäristö

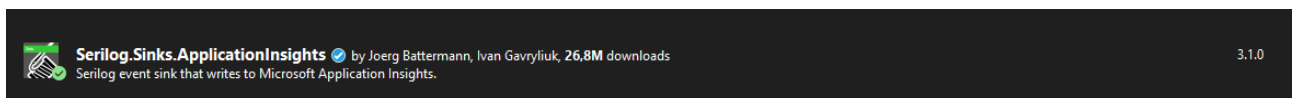
Käyttöympäristöön liittyvät vaatimukset (EnvReq001, EnvReq002, EnvReq003 ja EnvReq004) olivat lähinnä sellaisia, että valitun monitorointiratkaisun täytyy tukea Windows-ympäristöä ja IIS web-palvelinta sekä olla otettavissa käyttöön taustasovelluksen ASP.NET Core ja käyttöliittymän Angular -ohjelmistokehyksissä. Application Insights täytti kaikki käyttöympäristöön liittyvät vaatimukset, kun se on onnistuneesti otettu käyttöön.

Saavutettavuus

Saavutettavuuden vaatimuksina (AvailReq001, AvailReq002 ja AvailReq003) oli, että palveluun pitää kirjautua henkilökohtaisilla tunnuksilla ja sinne on päästävä selaimen kautta. Azure toimii puhtaasti selaimesta ja sinne vaaditaan kirjautuminen omilla tunnuksilla. Käyttäjille voidaan antaa Azureen tarvittavat oikeudet, joten käyttäjät pääsevät näkemään ja tekemään vain omiin rooleihinsa sopivia asioita. Näihin ominaisuuksiin verrattuna Azure Monitor täyttää saavutettavuuden vaatimukset. AvailReq004 vaatimuksena oli, että palvelu on asennettavissa omille palvelimille, jolloin taataan, että kerätty data ei siirry organisaation ulkopuolelle. Azure Monitor on täysin pilviympäristössä toimiva monitorointiratkaisu, joten sitä ei ole mahdollista asentaa omille palvelimille. Tästä johtuen Azure Monitor ei täytä vaatimusta AvailReq004.

Lokitus

Yhtenä tärkeimmistä vaatimuksista oli, että lokitiedot pitää pystyä keskittämään (FuncReq003) niin, että ne kaikki löytyvät Azuresta, kun ne nyt ovat hajautettuina joko tietokantaan tai tekstitiedostoihin. ASP.NET Core -sovelluksen lokitus virheiden osalta oli toteutettu käyttämällä Serilog-ohjelmistokehystä, joka kirjoittaa lokitiedot haluttuun paikkaan ja sitä pystytään konfiguroimaan logSettings.json-tiedostoa käyttäen. Serilogiin on saatavilla suoraan laajennusosa nuget-pakettina (kuvio 17), joka mahdollistaa lokitietojen lähettämisen Serilogista suoraan Azureen käyttämällä Application Insights -palvelua.



Kuvio 17. Serilogin nuget-paketti Application Insights:a varten

Serilogin konfigurointi lokitietojen lähettämiseksi hyödyntämällä Application Insights -palvelua tapahtuu suoraan logSettings.json-tiedostoon kuvion 18 mukaisesti.

```

"Serilog": {
  "Using": [
    "Serilog.Sinks.ApplicationInsights"
  ],
  "MinimumLevel": {
    "Default": "Error",
    "Override": {
      "Microsoft": "Error",
      "System": "Error"
    }
  },
  "WriteTo": [
    {
      "Name": "ApplicationInsights",
      "Args": {
        "restrictedToMinimumLevel": "Error",
        "instrumentationKey": "<Put Instrumentationkey from Azure here>",
        "telemetryConverter": "Serilog.Sinks.ApplicationInsights.Sinks.ApplicationInsights.TelemetryConverters.TraceTelemetryConverter, Serilog.Sinks.ApplicationInsights"
      }
    }
  ]
}

```

Kuvio 18. LogSettings.json tiedosto Serilogin konfigurointiin

Kun Serilogin luomat lokitiedot saatiin siirrettyä lähetettäväksi suoraan Azureen täyttää monitorointiratkaisu lokitukseen liittyvät vaatimukset sillä lokitiedot, jotka tallennettiin tietokantaan menevät jo Application Insights -palvelun telemetriatietojen mukana Azureen.

Palvelin

Yhtenä vaatimuksena oli, että palvelinta, jolla ohjelmisto pyörii pitää pystyä monitoroimaan, sillä tasolla, että tiedetään miten ohjelma käyttää muun muassa muistia (FuncReq002) tai miten ohjelma kuormittaa palvelimen CPU:ta (FuncReq001). Azure Monitorissa on olemassa oma toiminto palvelimen monitorointia varten, mutta Application Insights kykenee myös monitoroimaan palvelinta tietyllä tasolla. Application Insights saatiin monitoroimaan palvelinta lisäämällä kuvion 19 mukainen koodin pätkä ohjelman startup-luokkaan.

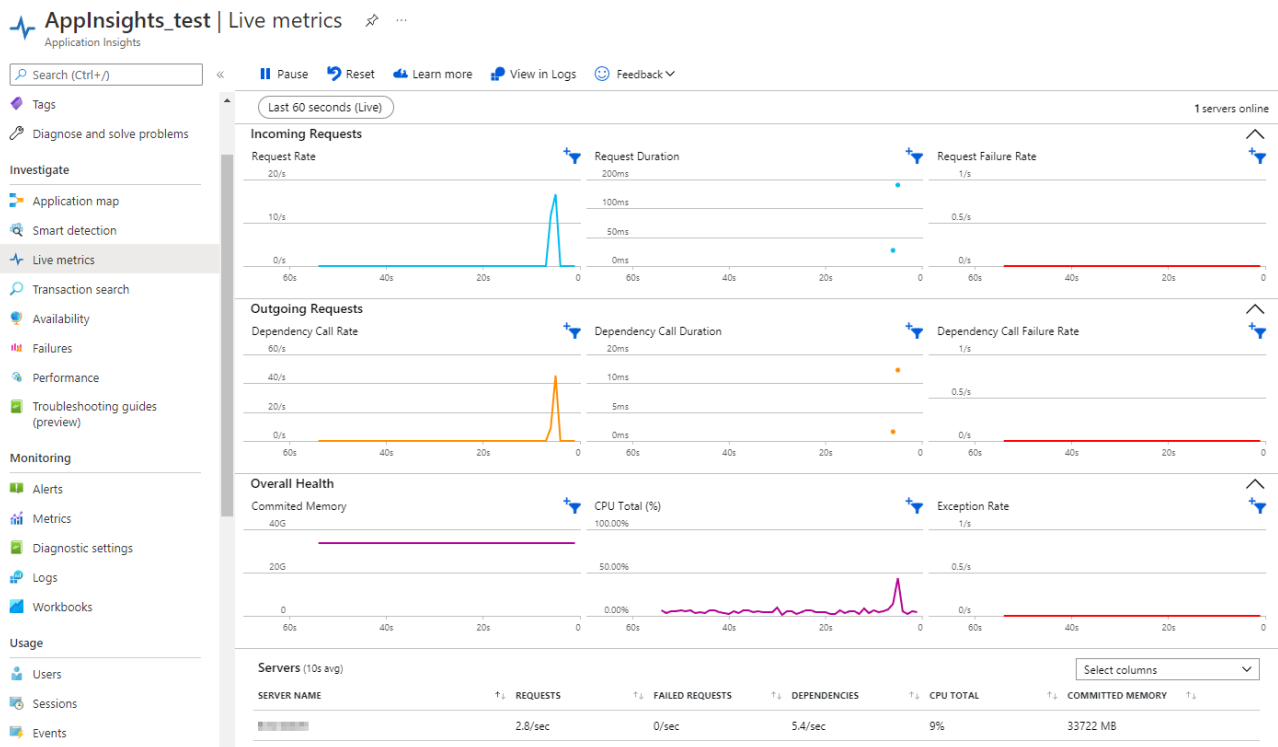
```

services.AddApplicationInsightsTelemetry(new ApplicationInsightsServiceOptions { ConnectionString = Configuration["ApplicationInsights:ConnectionString"] });
services.ConfigureTelemetryModule<DependencyTrackingTelemetryModule>((module, o) => { module.EnableSqlCommandTextInstrumentation = true; });
services.ConfigureTelemetryModule<PerformanceCollectorModule>((module, o) => { module.EnableIISExpressPerformanceCounters = true; });

```

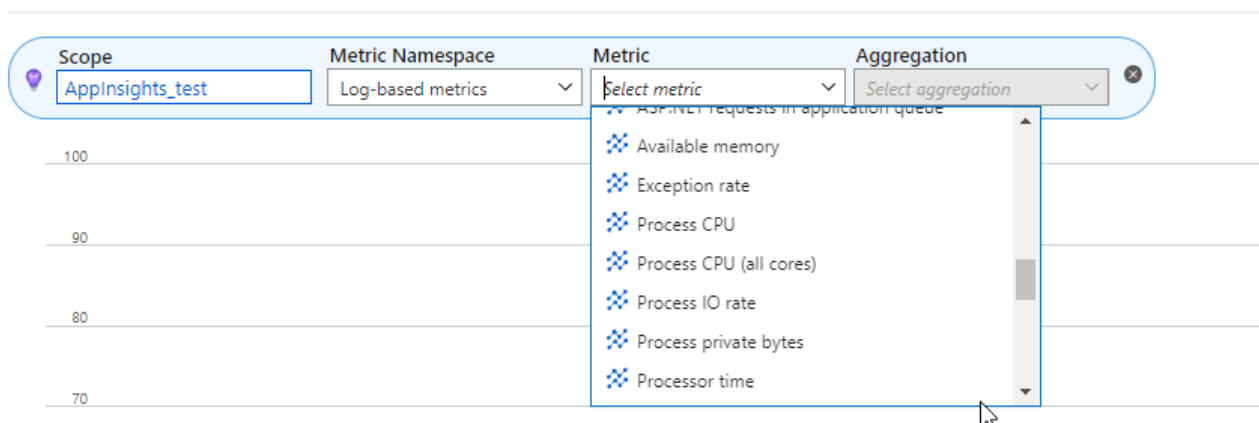
Kuvio 19. Lisätty IIS web-palvelimen suorituskyvyn monitorointi

Tämän jälkeen Application Insights alkoi lähettämään suorituskykytietoja Azureen. Azuressa helppoin tapa nähdä palvelimen tämänhetkinen tilanne on katsoa Application Insights -resurssin Live Metrics näkymää (kuvio 20). Näkymässä on nähtävillä graafisesti muun muassa pyyntöjen tiheys, pyyntöjen kesto, virhetiheys sekä käytetty muisti.



Kuvio 20. Azuren Live Metrics näkymä

Yksityiskohtaisempaa suorituskyvyn tutkimista, kuten jälkikäteen selvitettävää tietyn ajanhetken suorituskkyä voi tutkia Application Insights -resurssin Metrics näkymässä (kuvio 21), jossa voidaan valita tarkempia suorituskymittareita.

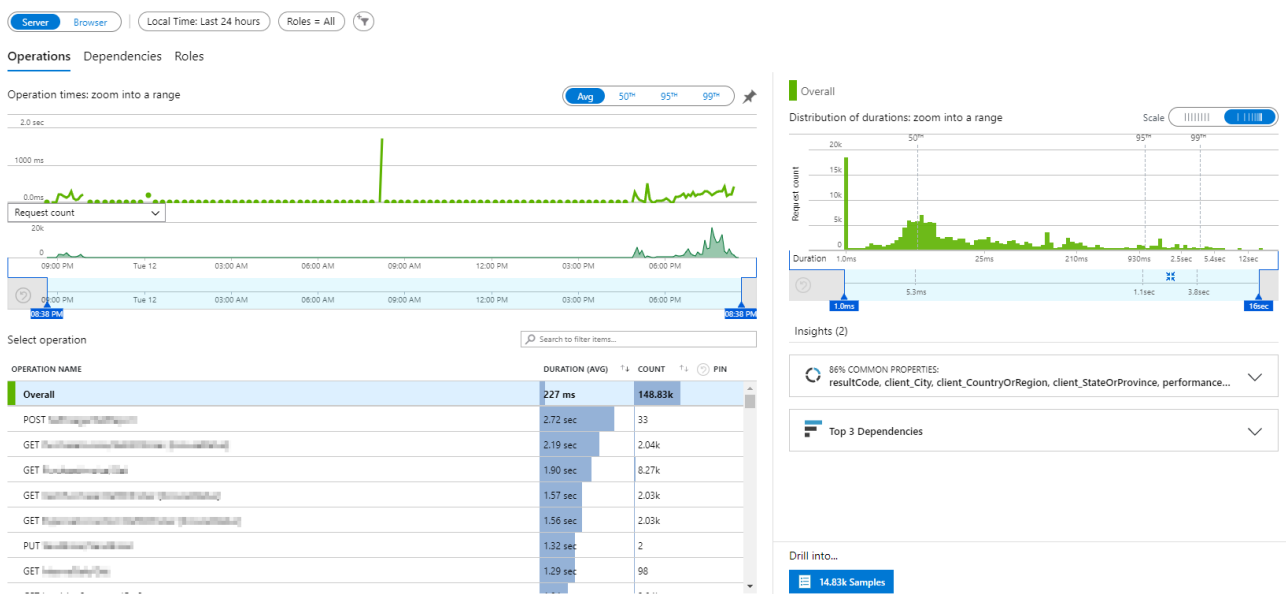


Kuvio 21. Azure Metrics suorituskymittarit

Application Insights täyttää näin ollen myös vaatimuksien FuncReq001 ja FuncReq002 kriteerit.

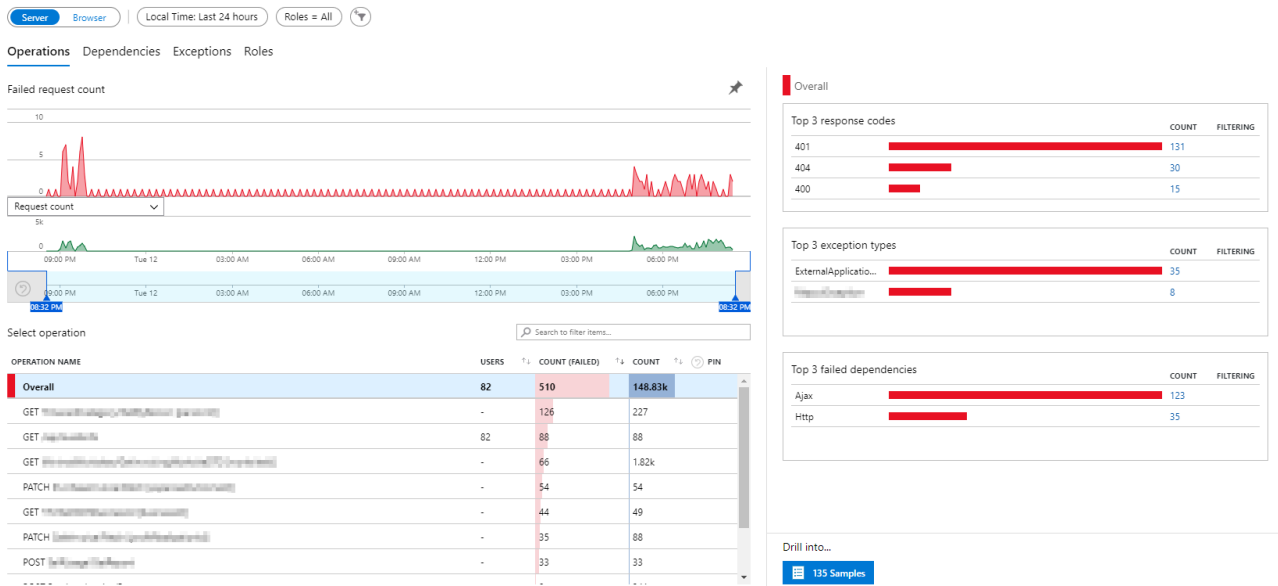
ASP.NET Core -sovelluksen monitorointi

ASP.NET Core -sovellukseen liittyviä vaatimuksia olivat FuncReq004, FuncReq005 ja FuncReq006. Ne keskittyivät siihen mitä tietoja taustasovelluksen toiminnoista halutaan saada kuten päätepis- teiden kesto, virheet ja suorituskykytietoa. Application Insights ei vaadi mitään erikoista konfigu- rointia, jotta nämä tiedot saadaan käyttöön vaan ne näkyvät Azuressa automaattisesti. Kuviossa 22 on nähtävillä, kuinka pääte pisteiden kestoja pystytään seuraamaan yhdestä näkymästä.



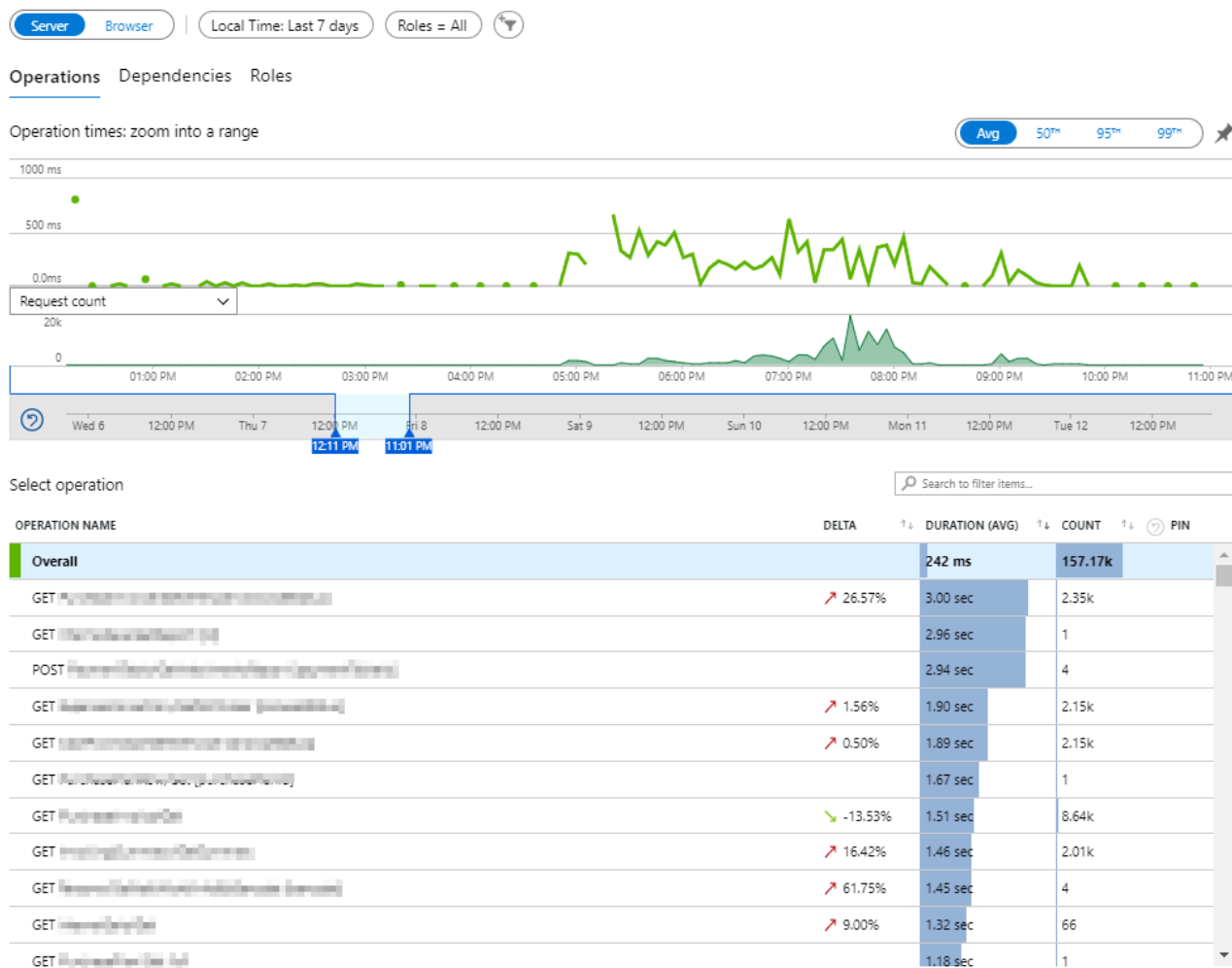
Kuvio 22. Application Insights -resurssin näkymä taustasovelluksen pääte pisteiden suoritusajoista

Taustasovelluksessa tapahtuvat virheet ovat samoin nähtävillä yhdestä näkymästä (kuvio 23) Ap- plication Insights -resurssissa. Näkymä näyttää muun muassa yleisimmät taustasovelluksen vas- tauskoodit ja virhetyypit. Tämän lisäksi ylimmästä graafista voidaan nähdä, milloin virheitä on ta- pahtunut eniten. Lisäksi graafien alla olevasta kutsuttujen pääte pisteiden listasta voidaan vaihtaa tarkasteluun vain haluttu pääte piste.



Kuvio 23. Taustasovelluksen virheet näkymä Application Insights -resurssissa

Taustasovelluksen päätepisteiden suorituskyvyn seuranta eli se miten niiden vasteajat ovat kehittyneet saadaan näkyviin samaan näkymään kuin itse päätepisteiden suoritusajat rajaamalla tarkasteltavaa aikajaksoa (kuvio 24). Päätepisteiden listaukseen ilmestyy tällöin sarake (Delta), joka kuvaa prosentuaalisesti sitä onko päätepisteiden suoritusajat hidastuneet vai nopeutuneet valittuna ajanjaksona verrattuna kokonaisuuteen. Tämän avulla voidaan tarkastella esimerkiksi versiomuutoksen vaikutusta suoritusajoihin.



Kuvio 24. Taustasovelluksen päätepisteiden suoritusaikojen muutos

Application Insights täyttää kaikki ASP.NET Core -sovellukseen liittyvät vaatimukset. Näiden lisäksi on mahdollista kerätä lisää tietoja taustasovelluksen toiminnasta muun muassa lisäämällä kusto-
moituja tietoja telemetritietoihin.

Käyttöliittymän monitorointi

Käyttöliittymään eli Angular-ohjelmaan liittyviä vaatimuksia olivat FuncReq007, FuncReq008, FuncReq009 ja FuncReq010. Ne olivat suunnattu siihen, että käyttöliittymästä saadaan tietoa, miten käyttäjä kokee ohjelman käytön sekä mitä virheitä hän kohtaa ohjelmaa käyttäessään.

Johtuen siitä, että Angular-ohjelma on SPA-ohjelma (Single Page Application) sivujen latausta ei saada automaattisesti seurattua. Tätä varten koodiin jouduttiin tekemään muutoksia (kuvio 25)

niin, että kun sivua vaihdetaan eli selaimen osoiterivillä polku muuttuu, niin käynnistetään näkymän seuranta koodissa, jolloin Application Insights kerää sivun latautumiseen liittyvää dataa kuten latausajan. Ohjelman auetessa ja pääsivulle tultaessa tämä jo toimii koska kun MonitoringService-luokka (Liite 1) perustetaan, heti ohjelmaa avattaessa, niin kutsutaan samalla trackPageView-metodia, joka käynnistää sivun seurannan ohjelman ensimmäiselle sivulle. Tämän muutoksen avulla pystyttiin seuraamaan mitä käyttäjä on tehnyt käyttöliittymässä esimerkiksi juuri ennen virheen tapahtumista.

```
this.router.events.subscribe(event =>
{
  if (event instanceof NavigationStart)
  {
    this.startNavigationEvent(event.url);
  }
  else if (event instanceof NavigationEnd)
  {
    this.endNavigationEvent(event.url);
    this.appInsights.trackPageView();
  }
});
```

Kuvio 25. Näkymän vaihdon seuranta

Toimintojen suoritusajat ja selainpään virheiden keräys ovat Application Insights:lle perusominaisuuksia. Toimintojen suoritusajat ovat nähtävillä samasta näkymästä kuin taustasovelluksen, vaihtamalla vain tarkastelu kohteeksi selain. Sama pätee myös selainpään virheisiin. Testauksen yhteydessä huomattiin, että Application Insights tuottaa paljon muutakin hyödyllistä tietoa käyttöliittymän toiminnasta kuin mitä vaatimuksissa oli määritelty. Tällaisia tietoja ovat muun muassa käyttäjien selain versiot, käyttöjärjestelmät ja laitetyyppiin.

6 Johtopäätökset

Työn tuloksena saatiin selvitettyä mitä monitorointiratkaisuja on tarjolla ja mitä niillä voi tehdä. Lisäksi saatiin selvitettyä, miten sellainen otetaan käyttöön ja miten toimeksiantaja hyötyy siitä.

6.1 Monitorointiratkaisun soveltuvuus toimeksiantajalle

Valituille monitorointiratkaisuille suoritettiin vertailu, jossa selvitettiin miten niiden ominaisuudet vastaavat toimeksiantajan vaatimuksia. Vertailu oli varsin tasaväkinen ja kaikki monitorointiratkaisut toteuttivat lähes kaikki niihin kohdistuneet vaatimukset. Vertailun lisäksi valittaessa monitorointiratkaisua testikäyttöön otettiin huomioon toimeksiantajan olemassa olevat toiminnot Azuressa sekä kokemukset sen käytöstä. Tutkimuskysymykseen mikä monitorointiratkaisu soveltuu parhaiten toimeksiantajan käyttöön (Q1), vastattiin onnistuneesti, kun vertailun lopputuloksena testikäyttöön valittiin Azure Monitor.

Azure Monitor vastaa lähes kaikkiin vaatimuksiin, joita sitä kohtaan oli asetettu. Monitorointipalvelun asentaminen omille palvelimille ei ole Azuren tapauksessa mahdollista, mutta sitä ei koettu ratkaisevaksi tekijäksi. Azure Monitor täyttää muut vaatimukset ja se tarjoaa lisäksi paljon jatkokehitys mahdollisuuksia muun muassa kerätyn datan visualisoinnissa ja tietokannan monitoroinnissa. Vaatimuksien täyttymisen lisäksi työssä tarkasteltiin sitä, millä tavoin Azure Monitor täyttää nämä vaatimukset, kuten esimerkiksi miten vasteajat näkyvät Azuressa ja millä tavoin lokitiedot käyttäen Serilog:a saadaan vietyä Azureen. Näin ollen myös tutkimuskysymykseen miten valittu monitorointiratkaisu vastaa toimeksiantajan tarpeisiin saatiin vastattua.

Monitorointiratkaisu hyödyttää tilaajaa muun muassa sillä, että häiriötilanneprosessiin on nyt tarjolla työkalu, jonka avulla päästään käsiksi sen ajanhetken tilanteeseen, kun häiriö on sovelluksessa esiintynyt tai alkanut. Tämän on havaittu jo testikäytön aikana nopeuttavan häiriötilanteiden selvitystä. Se myös nopeuttaa tarvittavien korjausten saamista tuotantoon, kun häiriöisen juurisyyt pystytään paikantamaan entistä nopeammin. Tavoiteltava hyöty on myös se, että kun monitorointiratkaisua opitaan hyödyntämään kokonaisvaltaisesti niin häiriöihin ja niiden syntyyn päästään puuttumaan jo ennen kuin käyttäjät kokevat käyttäjäkokemuksen heikkenemistä ohjelmaa käytettäessä. Tutkimuskysymykseen miten valittu monitorointiratkaisu hyödyttää tilaajaa (Q4) vastattiin myös onnistuneesti.

6.2 Monitorointiratkaisun käyttöönotto

Application Insights -palvelun käyttöönotto testikäyttöä varten onnistui hyvin niin ASP.NET Core -sovellukseen kuin käyttöliittymän Angular-ohjelmaan. Monitorointiratkaisun käyttöönotto on suoraan rutiinista eikä se vaadi suuria muutoksia ohjelmakoodiin. Tässä työssä esitetty käyttöönotto kuvaa miten monitorointiratkaisun saa otettua käyttöön toimeksiantajan käyttämään ympäristöön. Tutkimuskysymykseen Q2 vastattiin onnistuneesti, kun käyttöönotto dokumentoitiin tämän työn osatuotoksena.

7 Pohdinta

Tutkimuksen tavoitteena oli selvittää mitä monitorointiratkaisuja on tarjolla, ja mikä tai mitkä niistä olisivat sopivia toimeksiantajan käyttöön. Lisäksi tavoitteena oli selvittää mitä monitorointiratkaisun käyttöönotto vaatii, miten se tapahtuu sekä mitä hyötyä se tuottaa toimeksiantajalle.

Tutkimuksen tavoitteet saavutettiin hyvin ja kaikkiin tutkimuskysymyksiin saatiin vastattua tutkimuksen aikana. Tutkimuksessa suoritettiin monitorointiratkaisujen vertailu, joka antoi tietoa siitä mitä ratkaisuja markkinoilla on. Lisäksi se antoi tietoa, että monitorointiratkaisut ovat perusominaisuuksiltaan melko samanlaisia, mutta siinä miten ne toteuttavat ohjelman monitoroinnin on jonkin verran eroja. Vertailun tuloksien jälkeen valittiin Azure Monitor ja sen Application Insights -palvelu tarkempaan testikäyttöön. Testikäytön aikana suoritettiin monitorointiratkaisun käyttöönotto sekä testattiin miten se toteuttaa siihen kohdistetut vaatimukset kuten vasteaikojen, virheiden ja käyttäjän kokeman suorituskyvyn monitoroinnin. Testikäytön perusteella voitiin todeta, että Application Insights toteuttaa kaikki ne vaatimukset, jotka sille oli asetettu. Tämän lisäksi Application Insights tarjoaa paljon sellaista tietoa ohjelmasta mitä ei osattu ennakkoon ajatella ja josta tulee olemaan suurta hyötyä, kun halutaan seurata ohjelman suorituskykyä.

Sovellusten monitorointi on vielä varsin uusi asia ja siitä ei löydy hirveästi tietoa kirjallisuudesta. Tämänkin työn teoriatausta monitoroinnin osalta koostuu siksi hyvin pitkälti verkkojulkaisuista ja monitorointiratkaisujen toimittajien materiaaleista. Niitä tarkasteltaessa on pidettävä mielessä se, että ne on saatettu kirjoittaa aina tekijän suosimasta näkökulmasta. Se ei kuitenkaan tarkoita sitä, että ne eivät olisi relevantteja.

7.1 Jatkokehityskohteet

Tutkimuksessa keskityttiin loppujen lopuksi melko kapeaan alueeseen monitoroinnista, kun keskiössä pidettiin lähes kokonaan ohjelmistoon keskittyvä monitorointi (APM ja RUM). Jatkokehityksenä monitoroinnin mahdollisuuksia voitaisiin tutkia muillakin osa-alueilla kuten palvelimen tarkemmassa monitoroinnissa. Tämän lisäksi syvällisempi SQL-palvelimen monitorointi voisi olla hyvä kohde tarkempaan monitorointiin, sillä siihen liittyy paljon suorituskykyyn liittyviä kysymyksiä, joiden tueksi olisi hyvä saada mitattua dataa, jota voitaisiin tuottaa monitoroimalla. Azuressa on tällä hetkellä esikatseluvaiheessa SQL Insights -palvelu, joka mahdollistaa SQL:n tarkemman monitoroinnin, joten siihen perehtyminen olisi yksi käytännön esimerkki tutkimuksen jatkokehittämisestä.

Lähteet

A Brief History of Software Development methodologies. 2021. Julkaisu Growin:n verkkosivuilla 07.12.2021. Viitattu 30.04.2022. <https://www.growin.com/blog/history-of-software-development-methodologies/>

Hamilton, T. 2022a. Agile Methodology: What is Agile Model in Software Testing? Julkaisu Guru99:n verkkosivuilla 23.04.2022. Viitattu 04.05.2022. <https://www.guru99.com/agile-scrum-extreme-testing.html>

Hamilton, T. 2022b. Agile Vs. DevOps: What's the difference? Julkaisu Guru99:n verkkosivuilla 30.04.2022. Viitattu 04.05.2022. <https://www.guru99.com/agile-vs-devops.html>

Altwater, A. 2015. What is APM? Overview, Common Terms, and 10 Critical APM Features. Kolumni Stackify:n verkkosivulla, 14.09.2015. Viitattu 23.02.2022. <https://stackify.com/what-is-apm/>

Altwater, A. 2020. What Is Real User Monitoring? How It Works, Examples, Best Practices, and More. Kolumni Stackify:n verkkosivuilla, 29.01.2020. Viitattu 10.04.2022. <https://stackify.com/what-is-real-user-monitoring/>

A Maturing DevSecOps landscape – 2021 Global Survey results. N.d. GitLabin julkaisu. Viitattu 01.02.2022. <https://learn.gitlab.com/c/2021-devsecops-report?x=u5RjB>

Application Insights overview. 2022. Azuren dokumentaatio Microsoftin sivuilla, 23.02.2022. Viitattu 07.03.2022. <https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview>

Azure Monitor Logs overview. 2022. Azuren dokumentaatio Microsoftin verkkosivuilla, 23.02.2022. Viitattu 07.03.2022. <https://docs.microsoft.com/en-us/azure/azure-monitor/logs/data-platform-logs>

Azure Monitor Metrics overview. 2021. Azuren dokumentaatio Microsoftin verkkosivuilla, 12.07.2021. Viitattu 07.03.2022. <https://docs.microsoft.com/en-us/azure/azure-monitor/essentials/data-platform-metrics>

Azure Monitor overview. 2021. Azure monitorin dokumentaatio Microsoftin verkkosivuilla, 11.09.2021. Viitattu 06.03.2022. <https://docs.microsoft.com/en-us/azure/azure-monitor/overview>

Fowler, M. 2019. WaterfallProcess. Martin Fowlerin artikkel verkkosivuilla, 13.11.2019. Viitattu 03.05.2022. <https://martinfowler.com/bliki/WaterfallProcess.html>

Kananen, J. 2012. Kehittämistutkimus opinnäytetyönä. Jyväskylän Ammattikorkeakoulu.

Projektien vesiputousmalli ja sen viisi heikkoutta, 2016. Artikkel Thinking Portfolion verkkosivuilla, 19.07.2016. Viitattu 03.05.2022. <https://thinkingportfolio.com/projektien-vesiputousmalli-ja-sen-viisi-heikkoutta/>

Royce, W. 1970. Managing The Development Of Large Software Systems. Proceedings IEEE WESCON, Los Angeles

SDLC - Software Prototype Model. N.d. Verkkajulkaisu Tutorialspoint:n verkkosivuilla. Viitattu 03.05.2022. https://www.tutorialspoint.com/sdlc/sdlc_software_prototyping.htm

Taina, J. 2006. Ohjelmistotuotannon luentomateriaali. <https://www.cs.helsinki.fi/u/taina/ohtu/k-2006/pdf/Ohjelmistotuotanto-2006-2.pdf>

van Vliet, H. 2007. Software Engineering: Principles and Practice. <https://citeserx.ist.psu.edu/viewdoc/download?doi=10.1.1.128.2614&rep=rep1&type=pdf>

Yritys. N.d. Jydacomin verkkosivut. Viitattu 30.01.2022. <https://www.jydacom.fi/yritys/>

Watts, S. 2018. What is Application Performance Management? APM Explained. Artikkelit ITSM:n verkkosivuilla, 15.03.2018. Viitattu 10.02.2022. <https://itsm.tools/what-is-application-performance-management-apm-explained/>

What is Agile? N.d. Julkaisu Atlassian:n verkkosivuilla. Viitattu 04.05.2022. <https://www.atlassian.com/agile>

What is a monitoring environment? 2022. Dynatracen dokumentaation Dynatracen tukisivuilla, N.d. Viitattu 19.04.2022. <https://www.dynatrace.com/support/help/get-started/monitoring-environment>

What is DevOps? 2022. Artikkelit AWS:n verkkosivuilla. Viitattu 24.01.2022. <https://aws.amazon.com/devops/what-is-devops>

What is DevOps? 2020. Artikkelit KAASM:n verkkosivuilla, 29.01.2020. Viitattu 24.01.2022. <https://kaasm.com/devops/what-is-devops>

What is DevOps? N.d. Artikkelit NetApp:n verkkosivuilla. Viitattu 09.02.2022. <https://www.netapp.com/devops-solutions/what-is-devops/>

Yost, M. 2018. A Brief History of Software Development. Artikkelit medium.com:n verkkosivuilla, 25.01.2018. Viitattu 29.04.2022. <https://medium.com/@micahyost/a-brief-history-of-software-development-f67a6e6ddae0>

7 Reasons Why DevOps Matter. 2019. Artikkelit Agileit:n verkkosivuilla 04.12.2019. Viitattu 26.01.2022. <https://www.agileit.com/news/7-reasons-devops-matter>

Liitteet

Liite 1. Application Insights Javascript API wrapper-luokka

```

1  import { Injectable } from '@angular/core';
2  import { ApplicationInsights, DistributedTracingModes } from '@microsoft/applicationinsights-web';
3  import { environment } from '../../environments/environment';
4
5  @Injectable({
6    providedIn: 'root'
7  })
8
9  export class MonitoringService
10 {
11   private appInsights: ApplicationInsights;
12
13   constructor()
14   {
15
16     this.appInsights = new ApplicationInsights({
17       config: {
18         connectionString: environment.appInsights.connectionString,
19         enableAutoRouteTracking: true,
20         autoTrackPageVisitTime: true,
21         enableRequestHeaderTracking: true,
22         enableResponseHeaderTracking: true,
23         enableCorsCorrelation: true,
24         distributedTracingMode: DistributedTracingModes.AI_AND_W3C
25       }
26     });
27
28     this.appInsights.loadAppInsights();
29     this.appInsights.trackPageView();
30
31   }
32
33   public logPageView(name?: string, url?: string)
34   {
35     this.appInsights.trackPageView({
36       name: name,
37       uri: url
38     });
39   }
40
41   public logEvent(name: string, properties?: { [key: string]: any })
42   {
43     this.appInsights.trackEvent({ name: name }, properties);
44   }
45
46   public logMetric(name: string, average: number, properties?: { [key: string]: any })
47   {
48     this.appInsights.trackMetric({ name: name, average: average }, properties);
49   }
50
51   public logException(exception: Error, severityLevel?: number)
52   {
53     this.appInsights.trackException({ exception: exception, severityLevel: severityLevel });
54   }
55
56   public logTrace(message: string, properties?: { [key: string]: any })
57   {
58     this.appInsights.trackTrace({ message: message }, properties);
59   }
60 }

```

Liite 2. ErrorHandlerService luokka keskitettyyn virheiden kirjaamiseen

```
1  import { ErrorHandler, Injectable } from '@angular/core';
2  import { MonitoringService } from './monitoring.service';
3
4  @Injectable({
5    providedIn: 'root'
6  })
7
8  export class ErrorHandlerService extends ErrorHandler
9  {
10
11    constructor(private monitoringService: MonitoringService)
12    {
13      super();
14    }
15
16    public handleError(error: Error)
17    {
18      this.monitoringService.logException(error);
19    }
20  }
```