Tampereen yliopisto

Ilpo Viertola

SUORITUSKYVYN SEURANNAN INTEGROINTI SOVELIA® PLM -JÄRJESTELMÄÄN

Tekniikan ja luonnontieteiden tiedekunta
Kandidaatintyö
Huhtikuu 2021

TIIVISTELMÄ

Ilpo Viertola: Suorituskyvyn seurannan integrointi Sovelia® PLM -järjestelmään
Integration of application performance management into Sovelia® PLM -system
Kandidaatintyö
Tampereen yliopisto
Automaatiotekniikan tutkinto-ohjelma
Huhtikuu 2021

Huono suorituskyky saattaa tehdä muuten toimivasta sovelluksesta käyttökelvottoman. Erityisesti kaupallisten järjestelmien suorituskyvyn monitorointi on tärkeää, jotta mahdollisimman korkea asiakastyytyväisyys voidaan taata. Suorituskyvyn aktiivisella ja reaaliaikaisella monitoroinnilla voidaan havaita indikaatioita poikkeavasta toiminnasta ennen vakavampia seurauksia, kuten koko järjestelmän kaatumista. Tällöin syntyviä ongelmia saadaan ehkäistyä proaktiivisesti, ennen kuin käyttäjä huomaa poikkeavaa toimintaa järjestelmässä ja ohjelmisto saadaan pidettyä saatavilla käyttäjille.

Tämän työn tarkoituksena on toteuttaa järjestelmä, jolla voidaan monitoroida Sovelian suorituskykyä reaaliaikaisesti. Sovelia on tuotteen elinkaaren hallintaan kehitetty kaupallinen ohjelmisto. Suorituskykyyn liittyvää informaatiota kerätään Sovelian palvelimelta ja lähetetään eteenpäin monitoroitavaksi ja analysoitavaksi Microsoftin Azure-pilvipalveluihin kuuluvaan Application Insights -palveluun. Application Insights on sovelluksen tehokkuuden hallinnointiin tarkoitettu palvelu, joka sisältää työkaluja suorituskykyinformaation esittämiseen ja siinä esiintyvien poikkeavuuksien havaitsemiseen. Tämän työn tärkeimpänä tuloksena oli valmis järjestelmä, joka onnistuneesti kerää suorituskykyyn liittyvää informaatiota Soveliasta ja lähettää tämän eteenpäin Application Insights -palveluun.

Avainsanat: sovelluksen suorituskyvyn hallinnointi, suorituskyky, ohjelmisto

ALKUSANAT

Tämän kandidaatintyön mahdollistamisesta haluan kiittää Symetri Oy:tä.

Tampereella, 23.4.2021

Ilpo Viertola

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

API	engl. Application Programming Interface, ohjelmointirajapinta
APM	engl. Application Performance Management, sovelluksen tehokkuuden hallinta
CPU	engl. Central Processing Unit, keskusyksikkö
HTML	engl. Hypertext Markup Language, hypertekstin merkintäkieli
HTTP	engl. Hypertext Transfer Protocol, hypertekstin yhteyskäytäntö
KPI	engl. Key Performance Indicator, keskeinen suorituskykyindikaattori
PLM	engl. Production Lifecycle Management, tuotteen elinkaaren hallinta
SDK	engl. Software Development Kit, ohjelmistokehityspaketti
SQL	engl. Structured Query Language, relaatiotietokannan kysely- ja merkintäkieli
WWW	engl. World Wide Web, internetin hypertekstijärjestelmä

1. JOHDANTO

Hyvän käyttökokemuksen takaamiseksi ohjelmiston täytyy olla suorituskykyinen. Vaikka tarjottu järjestelmä olisi kaikin puolin toimiva, saattavat hitaat vasteajat käyttäjän pyyntöihin pilata käyttökokemuksen. Huonot käyttäjäkokemukset maksavat yrityksille myös asiakassuhteissa, jos luvattuun suorituskykyyn ei päästä. Toisaalta merkittävä suorituskyvyn lasku saattaa indikoida myös ohjelmiston puutteellisesta toiminnasta ja tarjota arvokasta tietoa sen kehityksessä. Tämän työn tavoitteena on tuottaa järjestelmä, jonka avulla monitoroitavan järjestelmän suorituskyvystä saadaan reaaliaikaista tietoa. Toteutettu järjestelmä mahdollistaa suorituskyvyn laskuun puuttumisen ennen kuin suurempaa vahinkoa, kuten järjestelmän kaatumista, pääsee tapahtumaan. Suorituskykydata tarjoaa myös arvokasta tietoa järjestelmän tapahtumista ennen mahdollista vikatilaa, joka on johtanut merkittävään suorituskyvyn laskuun.

Tässä työssä integroidaan sovelluksen tehokkuuden hallinta -työkalut osaksi tuotteen elinkaaren hallintajärjestelmää, joka tunnetaan nimellä Sovelia. Tarkemmin Soveliaa työn toimintaympäristönä käsitellään alaluvussa 2.3. Sovelluksen tehokkuuden hallinta, (APM, engl. application performance management), pitää sisällään metodeita, tekniikoita ja työkaluja, joiden tarkoituksena on jatkuvasti monitoroida ohjelmiston tilaa ja käyttöastetta sekä tunnistaa, diagnosoida ja ratkaista suorituskykyyn liittyviä ongelmia kerätyn monitorointidatan perusteella [8]. Laajempi katsaus tähän aiheeseen on alaluvussa 2.2.

Tässä työssä käytetty APM-järjestelmä on Microsoftin Azuren tarjoama Application Insights. Tarkoituksena on lähettää suorituskykyyn liittyvää informaatiota Sovelia-palvelimelta Azuren Application Insightsiin, jossa dataa varastoidaan, monitoroidaan sekä analysoidaan. Tarkempi kuvaus Azuren Application Insightsin tuomista mahdollisuuksista sekä toiminnasta on alaluvussa 3.1.

Ennen kuin voidaan tutkia ohjelmiston suorituskykyä, on määritettävä ohjelmistolle suorituskykyindikaattorit. Suorituskykyindikaattori tarkoittaa ohjelmistosta kerättävää informaatiota, joka antaa käsityksen ohjelmiston sen hetkisestä suorituskyvyn tilasta [4, s. 1]. Alaluvussa 3.2 perehdytään Sovelian suorituskykyindikaattoreihin sekä niiden valintaperusteisiin.

Tässä työssä seurataan Sovelian APM-järjestelmän kehitystä, joka tunnetaan nimellä Sovelia Monitor. Sovelia Monitorin päätarkoitus on kerätä ja lähettää dataa yllä esiteltyyn Azuren Application Insightsiin. Tässä työssä ei pureuduta kehitettyyn järjestelmään lähdekooditasolla, vaan tutkitaan sen yleistä rakennetta alaluvussa 3.3 ja kehitysprosessia alaluvussa 4.1. Alaluku 4.2 pitää sisällään tarkemman kuvauksen Sovelia Monitorista sekä sen erilaisista konfigurointimahdollisuuksista, joiden avulla käyttäjä voi räätälöidä järjestelmän erilaisiin Sovelia-ympäristöihin.

Työn tulokset esitellään alaluvussa 4.3. Sovelia Monitor on yhä testausvaiheessa, joten asiakasympäristöistä saatuja konkreettisia hyötyjä ei ole vielä syntynyt. Yksi tärkeimpiä työssä saavutettuja tuloksia on syventynyt ymmärrys tiettyjen Sovelian toimintojen huonosta suorituskyvystä. Tämä tarkoittaa yksinkertaisten operaatioiden pitkää kestoa, joita voidaan optimoida tehokkaammiksi. Toinen tärkeä tulos on myös parantunut käsitys Sovelian yleisrasitteesta, joka tarkoittaa järjestelmän rasitetta sen ollessa käyttämättömänä.

2. TYÖN MOTIIVI JA LÄHTÖKOHDAT

Motiivi projektin aloittamiselle oli havaittu tarve Sovelia-palvelimen tilan monitoroinnille. Virhetilan sattuessa palvelimella yleinen ongelma on myös vian selvittäminen jälkeenpäin runsaista sekä sekavista lokitiedostoista. Palvelimen seuraaminen ja tilan analysoiminen auttaa tunnistamaan sekä ehkäisemään mahdollisia ongelmia proaktiivisesti, ennen kuin vikatila aiheuttaa suuria vaikutuksia ohjelmistossa. Eräs esimerkki suuresta vaikutuksesta on palvelimen kaatuminen.

2.1 Ongelman asettelu

Ohjelmiston suorituskyvyllä ja toimivuudella on suora yhteys asiakastyytyväisyyteen sekä ohjelmiston käytettävyyteen. Tuotantokäytössä olevan ohjelmiston suorituskykyongelmat saattavat aiheuttaa asiakkaiden sekä tuottojen menetystä [7][8]. Ohjelmiston suorituskyvyllä on suuri rooli asiakastyytyväisyyden takaamisessa. Tästä on esimerkkinä Googlen vuonna 2018 tekemä tutkimus, jonka mukaan 53 % mobiilikäyttäjistä poistui sivuilta, mikäli sivun latautuminen kesti yli 3 sekuntia [7, katso 2]. Voidaan siis todeta, että yleisesti käyttäjät arvostavat nopeutta. Liian hidas ohjelmisto on lähes käyttökelvoton, vaikka se olisi muuten toimiva ja luotettava.

Eräänä monitorointiin velvoittavana tekijänä voidaan pitää yrityksen ja asiakkaan välille sovittua palvelutasosopimusta. Palvelutasosopimuksessa määritellään palvelu ja sen laatu, jota asiakkaalle tarjotaan [18, s. 246]. Tässä sopimuksessa määritellään muun muassa, kuinka saatavilla ohjelmiston on oltava asiakkaalle. Jos kyseiseen palvelutasoon ei päästä, seuraa tästä palvelun tarjoajalle sopimuksessa määritetty sanktio. Sovelian kohdalla tämä palvelutaso vaihtelee asiakaskohtaisesti, mutta pääsääntöisesti Sovelia on määritelty korkean saatavuuden ohjelmistoksi. Korkea saatavuus tarkoittaa, että ohjelmistossa ei tulisi olla suunnittelemattomia katkoja ja ohjelmisto tulisi olla saatavilla aina sovittuina ajanjaksoina [15]. Esimerkiksi ohjelmisto tulisi olla aina käyttäjien saatavilla aikavälillä 06.00–18.00.

Kun järjestelmässä tapahtuu odottamaton virhe, alkaa vikatilan juurisyyn selvittäminen ja ratkominen. Kun vikatila huomataan, ohjelmiston käyttäjä on jo saattanut uudelleen käynnistää palvelimen ja tyhjentää kaatumista ennen kirjoitetut paikalliset lokitiedostot. Tämä luo haasteita virheenkorjaukseen, sillä ongelman uudistaminen tai juurisyyn selvittäminen saattavat olla lähes mahdottomia tehtäviä. Ongelman juurisyyn selvittämistä helpottaa tieto, mitä ohjelmistossa on tapahtunut tunti tai kymmenen

minuuttia ennen palvelimen kaatumista. Tämä historiatieto on arvokasta, koska sen avulla saadaan selville mitä käyttäjä on tehnyt, mitä palvelimella on tapahtunut ja mitkä näistä tapahtumista ovat johtaneet kaatumiseen. Kun tiedetään, mitkä tapahtumat saattavat johtaa vikatilaan, voidaan tapahtumiin reagoida ja vaikutukset estää, ennen kuin ohjelmiston käyttäjä huomaa ohjelmistossa poikkeavaa toimintaa. [15][17]

Eräs toinen saavutettu hyöty on asiakkaan palvelimen tilaan sekä tapahtuneisiin virheisiin käsiksi pääsy, ilman suoraa yhteyttä palomuurien takana olevaan palvelimeen tai ilman fyysistä pääsyä palvelimelle. Tämä nopeuttaa virheen selvitystä, kun analysointi voidaan aloittaa ilman etäyhteyksien muodostamisia. Myös suorituskyvyn historiadata saattaa olla arvokasta ohjelmistokehityksessä, koska sen avulla voidaan tunnistaa pullonkauloja ohjelmiston toiminnassa.

Nykyaikaiset APM-työkalut tarjoavat ratkaisuja edellä esitettyihin ongelmiin. APM tarjoaa työkaluja ohjelmiston suorituskyvyn hallinnointiin keräämällä dataa ohjelmistosta, analysoimalla sitä ja tuottamalla datasta käyttäjäystävällisen näkymän. Tämä data on reaaliaikaista, mutta siihen tulisi olla sulautettuna myös vanhempaa dataa palvelimen tilasta. APM siis sisältää erilaisia metodeita, tekniikoita ja työkaluja ohjelmiston tilan ja käytön jatkuvaan monitorointiin sekä suorituskykyyn liittyvien ongelmien havainnointiin, analysointiin sekä selvittämiseen. [7][8][17] Näitä asioita käsitellään tarkemmin alaluvussa 2.2.

2.2 Application performance management

Kuten alaluvussa 2.1 todettiin, APM sisältää metodeita, tekniikoita ja työkaluja, joiden tarkoituksena on jatkuvasti monitoroida ohjelmiston tilaa ja käyttöastetta sekä tunnistaa, diagnosoida ja ratkaista suorituskykyyn liittyviä ongelmia kerätyn monitorointidatan perusteella. Ohjelmiston suorituskyvyn monitoroinnin työnkulku voidaan jakaa neljään suurempaan kokonaisuuteen:

1. datan keruu
2. datan varastointi sekä prosessointi
3. datan esittäminen
4. datan tulkinta sekä käyttö.

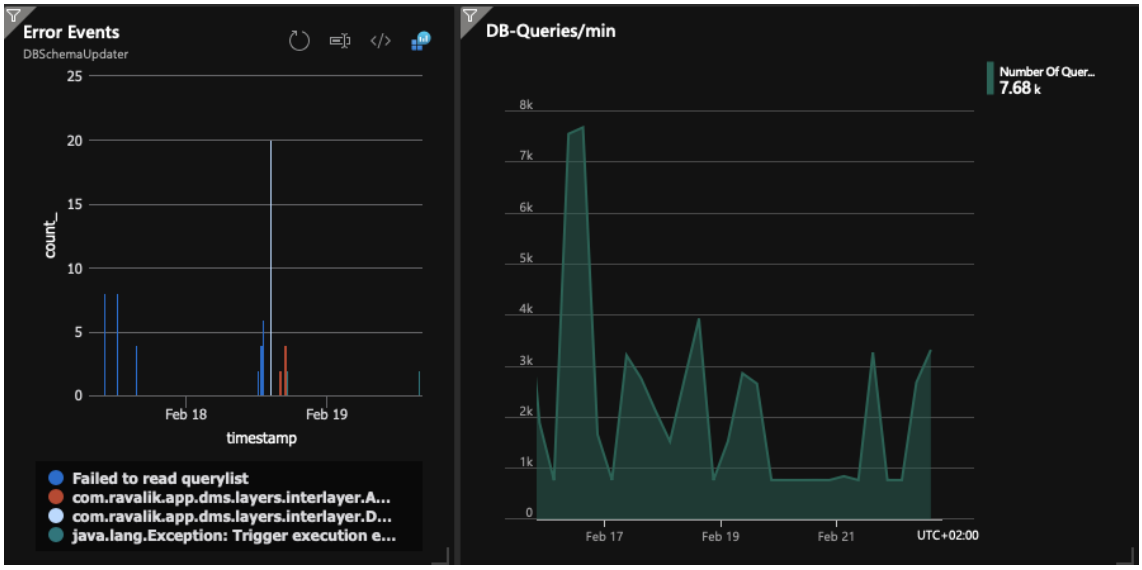
Nämä neljä vaihetta toistuvat jatkuvasti ja samanaikaisesti ohjelmaa monitoroidessa. [7][8]

Ennen datan keräämistä on kuitenkin määritettävä, mitä informaatiota ohjelmistosta kerätään. Tämä informaatio tunnetaan myös nimellä keskeinen suorituskykyindikaattori

(KPI, engl. key performance indicator). KPI tarkoittaa säännöllisin ajanjaksoin kerättävää informaatiota, joka kuvaa sen hetkistä ohjelmiston suorituskykyä [4, s. 1]. KPI:t saattavat vaihdella laajasti ohjelmiston toteutuksen ja käytettyjen tekniikoiden perusteella. Yleistä ohjelmiston suorituskyvystä kertovaa informaatiota ovat esimerkiksi palvelimen keskusyksikön käyttöprosentti, ohjelmiston käyttämä muisti, tietokantakyselyiden määrä sekä näiden kyselyiden kesto. Tätä informaatiota kutsutaan myös suorituskykymetriikaksi tai metriikaksi [17]. Dataa voidaan myös kerätä aktiivisesti tai passiivisesti. Aktiivisella keräyksellä viitataan tietyin ajanjaksoin tapahtuvaan näytteenottoon ohjelmiston käyttämistä resursseista, joita edellä lueteltiin. Passiivinen datan keräys tarkoittaa informaation taltioimista, kun tietty tapahtuma ohjelmistossa tapahtuu. Esimerkiksi, kun tietty virhe tapahtuu tai käyttäjä painaa tiettyä nappia ohjelmiston käyttöliittymästä. [7][8]

Datan keräyksen jälkeen se varastoidaan sekä prosessoidaan paremmin analysoitavaan muotoon. Data varastoidaan tietokantaan, josta APM-työkalun avulla sitä voidaan yhä jatkojalostaa entistä käytettävämmäksi. Esimerkiksi tietokantayhteyksien määrää minuutissa voi olla mielenkiintoisempaa seurata kuin tietokantayhteyksien kokonaismäärää tietyn ajanjakson aikana.

Tämän jälkeen data esitetään monitoroijalle erilaisissa, helposti tulkittavissa, muodoissa. Kuva 1 esittää kaksi erilaista diagrammia APM-työkalun käyttöliittymässä. Vasemmalla on passiivisesti kerättyjä virheilmoituksia ajan funktioina ja oikealla aktiivisesti kerättyjä tietokantakyselyiden määriä minuutissa, myös ajan funktiona. Jotta datasta olisi konkreettista hyötyä, on sitä myös tulkittava ja analysoitava. Datan tulkintaa voidaan tehdä automaattisesti tai manuaalisesti.



Kuva 1. *Suorituskykyindikaattorien kuvaajia ajan funktiona.*

Datan automaattisella tulkinnalla tai analysoinnilla tarkoitetaan esimerkiksi automaattisten hälytysten määrittämistä tietyille raja-arvoille. Hälytys voidaan sitoa yhteen tai useampaan suorituskykyindikaattoriin. Raja-arvojen määrittämiseen on yleensä kaksi erilaista lähestymistapaa: vertailukohtapohjainen tai kynnysarvopohjainen lähestymistapa. Vertailukohtapohjaisessa lähestymistavassa raja-arvo määritetään historiatietojen perusteella. Kun on saatu yleiskäsitys siitä, missä arvoissa tietty KPI liikkuu esimerkiksi työaikoina, voidaan tämä asettaa hälytyksen raja-arvoksi. Jos ohjelman havaittu suorituskyky laskee, korreloi tämä KPI:n arvon kanssa. Käytännössä tämä tarkoittaa KPI:n arvon nousua tai laskua. Raja-arvo säännön rikkoutuessa APM-työkalu lähettää automaattisen hälytysviestin esimerkiksi sähköpostitse vastuussa olevalle taholle. Kynnysarvopohjainen lähestymistapa tarkoittaa sellaisen arvon määrittämistä, jonka ylittämisen jälkeen ohjelmistoa pidetään käyttökelvottomana [1]. Esimerkkinä tällaisesta KPI:sta on tietokantakyselyiden kesto. Jos yksi tietokantakysely kestää enemmän kuin kymmenen sekuntia, voidaan olettaa, että ohjelmiston suorituskyky on merkittävästi laskenut, jolloin tästä voidaan lähettää hälytys.

Kehittynyt ennakoiva analytiikka on eräs tapa lähestyä monitorointiin kuuluvaa datan analysointiongelmaa. Siinä ideana on kehitellä tekoälymalleja tukemaan suurien datamäärien analysointia, joka manuaalisesti olisi varsin uuvuttavaa. Oikein konfiguroituna ja käytettynä, ennakoivaa analytiikkaa tukevat mallit voivat havaita ohjelmiston, verkkoyhteyden ja tietokantaan liittyviä ongelmia, mutta myös käyttäjän kokemia suorituskyvyn puutteita. [18, s. 239] Tässä työssä ei kuitenkaan perehdytä näihin ratkaisuihin syvällisemmin.

Ennen APM:n implementointia ohjelmistoon, on hyvä tuntea ohjelmiston arkkitehtuuri ja käyttötarkoitukset, jotta implementointiin osataan valita oikeat ohjelmistokehitykselliset tekniikat. Toinen tärkeä asia on KPI:den valinta. Ilman tietoa ohjelmistosta tai sen toiminnasta, on keskeisien suorituskykyindikaattorien määrittäminen liki mahdotonta. Tämän vuoksi alaluvussa 2.3 perehdytään pintapuolisesti Soveliaan. Tämä katsaus on hyvin kevyt ja sen ainoana tarkoituksena on tuottaa käsitys projektin toimintaympäristöstä, jotta seuraavissa luvuissa esitetyt teknologiset sekä monitorointiratkaisut ovat ymmärrettävämpiä.

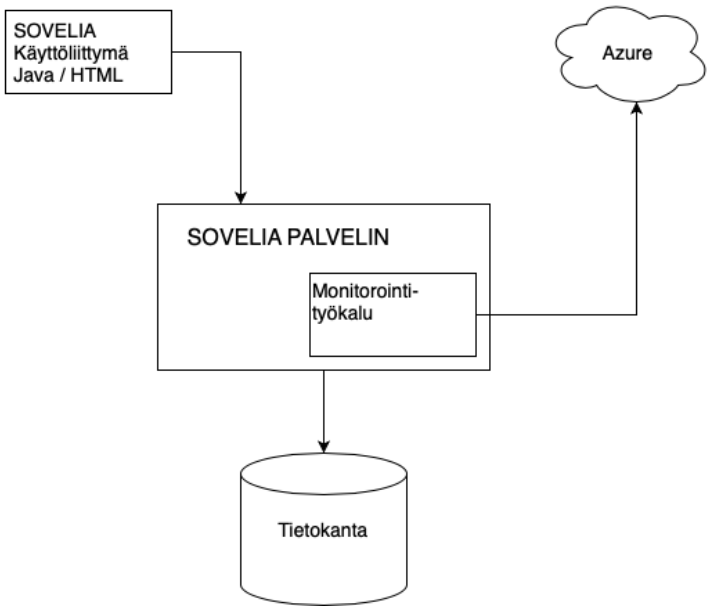
2.3 Sovelia PLM monitoroinnin toimintaympäristönä

Sovelia on tuotteen elinkaaren hallintaan (PLM, engl. product lifecycle management) tarkoitettu järjestelmä, lyhyemmin PLM-järjestelmä. Sovelia mahdollistaa tehokkaan suunnitteluprosessien ja tuotetiedon hallinnan sekä ajantasaisen tiedon jakamisen hajautetusti koko organisaatiossa. Sovelian käytön avulla yritys vähentää turhia kuluja ja

viiveitä suunnittelu- ja tuotantoprosesseissaan, kun tuotetieto on ajantasaista. Tämä lisää saavutettua liikevoittoa, koska tuotteet saadaan mahdollisimman ajoissa markkinoille. Tämän projektin kannalta ei kuitenkaan ole keskeistä ymmärtää PLM-järjestelmiä tai niiden toimintaa, vaan ymmärtää Sovelia isompana kokonaisuutena.

Kuvassa 2 on esitetty yksinkertaistettu kuvaus Sovelian arkkitehtuurista. Keskeisenä kuvassa on Java-pohjainen Sovelia-palvelin, joka kommunikoi järjestelmän eri osien välillä, tarjoten loppupään (engl. back end) ratkaisun ohjelmistolle. Loppupäällä tarkoitetaan ohjelmiston osaa, joka ei näy käyttäjälle ja jonka kanssa käyttäjä ei ole suoraan yhteydessä. Loppupää pitää huolen muun muassa datan käsittelystä ja vastaa käyttäjän käyttöliittymässä esittämiin pyyntöihin. [14] Sovelia-palvelimen sisässä on monitorointityökalu, joka puolestaan huolehtii suorituskykyyn liittyvän datan keruusta, esikäsittelystä ja lähetyksestä. Tämän monitorointityökalun kehitystä seurataan tässä työssä.

Oikeassa yläkulmassa oleva pilvi edustaa Microsoftin Azure-pilvipalveluita, joita tässä työssä käytetään. Pilvipalvelulla (engl. cloud computing) tarkoitetaan verkkoyhteyden välityksellä käytettävää ja tarpeen mukaan saatavilla olevaa tietoteknistä palvelua [12]. Tämä tarkoittaa, että palvelun käyttäjän ei tarvitse itse omistaa tai hallinnoida esimerkiksi palvelimia, jossa käytetty palvelu sijaitsee vaan käyttäjä maksaa palvelusta käytön mukaan. Lisäksi pilvipalveluille tyypillisiä ominaisuuksia ovat muun muassa niiden saatavuus ja skaalautuvuus. Pilvipalvelu on saatavilla mistä vain verkkoyhteyden ylitse ja se on käyttäjän käytettävissä lähtökohtaisesti milloin tahansa. Skaalautuvuudella tarkoitetaan palvelun koon kasvattamista. Esimerkiksi APM-palvelun kokoa tai palveluiden määrää voidaan kasvattaa, kun käyttö kasvaa. [16, s. 2–18] Azuren pilvipalvelut sisältävät datan käsittelyyn, esittämiseen ja analysointiin käytetyn APM-työkalun nimeltään Application Insights [6]. Monitorointityökalun kehityksessä käytettyjä tekniikoita esitellään luvuissa 3 ja 4.



Kuva 2. *Yksinkertaistettu kuva Sovelian arkkitehtuurista.*

Kuvan vasemmassa yläkulmassa on esitetty Sovelian käyttöliittymä sekä alimpana tietokanta. Myös palvelimen ja tietokannan välistä liikennettä monitoroidaan. Käyttöliittymän kautta käyttäjä operoi järjestelmää. Sovelia tarjoaa kaksi erilaista käyttöliittymäratkaisua: Java-ohjelmointikielellä toteutetun työpöytäkäyttöliittymän sekä HTML-kielellä (engl. hypertext markup language) toteutetun WWW-käyttöliittymän (engl. world wide web). Käyttöliittymässä tapahtuvaa toimintaa, kuten tietyn napin klikkauksia, ei monitoroida. Kohteena on ainoastaan palvelimen suorituskyvyn monitorointi, joka korreloi käyttäjän havaitseman ohjelmiston suorituskykyyn kanssa.

Sovelian taustalla toimii SQL (engl. structured query language) kysely- ja määrityskieltä käyttävä relaatiotietokanta. SQL-kieli on relaatiotietokantojen luontiin ja niiden ylläpitoon kehitetty ohjelmointikieli. Käskyä, jolla tietokannasta haetaan tietoa, tai tietokantaa muokataan, kutsutaan SQL-lauseeksi tai kyselyksi. [13]

Kuten edellä todettiin, Sovelia on Java-pohjainen ohjelmisto. Tämän vuoksi myös monitorointi päädyttiin toteuttamaan Javalla, sillä se mahdollistaa monitoroinnin sulauttamisen olemassa olevan lähdekoodin sekaan. Tämä kuitenkin rajoittaa myös toteutusta. Suorituskykymetriikkaa lähetetään APM-palvelulle Sovelia-palvelimelta hyödyntäen Azuren tarjoamaa ohjelmistokehityspakettia (SDK, engl. software development kit). SDK:n avulla yhteyden luominen Sovelian ja Azuren välille on suoraviivaista. Azuren tarjoamassa Java SDK:ssa on kuitenkin rajoitteita muiden ohjelmointikielien SDK:ihin verrattaessa.

Tässä luvussa tarkasteltiin projektin taustalla olevia tekijöitä ja toimintaympäristöä. Tämä tarjoaa pohjan ongelman ymmärtämiselle ja ratkaisujen omaksumiselle, joka on tulevien osioiden kannalta tärkeää.

3. SOVELIA MONITOR KOKONAISUUDEN YLEISKUVAUS

Tässä luvussa otetaan ensi katsaus kehitettyyn monitorointijärjestelmään. Ensin keskitytään käytetyn APM-palvelun, Azure Application Insightsin, ominaisuuksiin sekä siihen, miten niitä on tämän projektin tapauksessa hyödynnetty.

3.1 Azure Application Insights

Application Insights on Microsoftin Azure Monitor -tuoteperheeseen kuuluva kattava APM-palvelu. Sen avulla voidaan automaattisesti havaita ohjelmiston suorituskyvyn poikkeamia erilaisten analysointi- ja monitorointityökalujen avulla. Application Insightsin on tarkoitus tarjota ohjelmiston kehittäjille reaaliaikaista tietoa ohjelmiston suorituskyvyn muutoksista ja apua suorituskyvyn sekä käytettävyyden parantamiseen. [6]

APM-palveluksi valittiin Application Insights muun muassa sen helpon käyttöönoton ja laajan analysointityökalutarjonnan vuoksi. Lisäksi Application Insightsin toimii osana Azuren pilvipalveluita. Palvelulla voidaan tutkia asiakkaan Sovelia-ympäristön suorituskykymetriikkaa ilman fyysistä- tai etäyhteyttä asiakkaan palvelimelle. Koska Application Insights toimii osana Azuren pilvipalveluita, on sen kokoa helppo kasvattaa monitoroitavien Sovelia-palvelimien lisääntyessä. Toinen vaikuttava tekijä oli Application Insightsin tuoma mahdollisuus implementoida APM palvelut Java-pohjaiseen ohjelmistoon, koska se toimii myös Sovelian loppupään ohjelmistokielenä. Tämä helpottaa suorituskykymetriikan keräystä ja lähetystä suoraan Sovelia-palvelimelta APM-työkaluun.

Application Insights tukee sekä agentillista että agentitonta monitorointia [5][6]. Agentilla tarkoitetaan monitoroitavalle palvelimelle liitettävää järjestelmää, joka tallentaa sekä lähettää automaattisesti dataa ohjelmiston suorituskyvystä. Automaattiseen suorituskyvyn mittaukseen käytetään yleisesti tunnettuja suorituskykyindikaattoreita [17]. Näihin kuuluvat muun muassa palvelimelle lähetettyjen hypertekstin yhteyskäytäntöpyyntöjen (HTTP, engl. hypertext transfer protocol) määrä, palvelimen vasteajat HTTP-pyyntöihin ja epäonnistuneet HTTP-pyynnöt [6]. Agentittomassa monitoroinnissa ohjelmoija on vastuussa oikean datan keräämisestä. Tällöin voidaan kerätä ohjelmistokohtaisesti tärkeää suorituskykymetriikkaa [17]. Datan lähetys tapahtuu APM-palvelutarjoajan kehittämän ohjelmointirajapinnan (API, engl. application

programming interface) avulla. Tässä tapauksessa käytetty avoimen lähdekoodin API on Microsoft Application Insights Java SDK Core [10].

Mitä Application Insights oikeastaan monitoroi? Seuraava listaus sisältää tämän projektin kannalta tärkeimmät KPI:t, joita on mahdollista Application Insightsilla monitoroida:

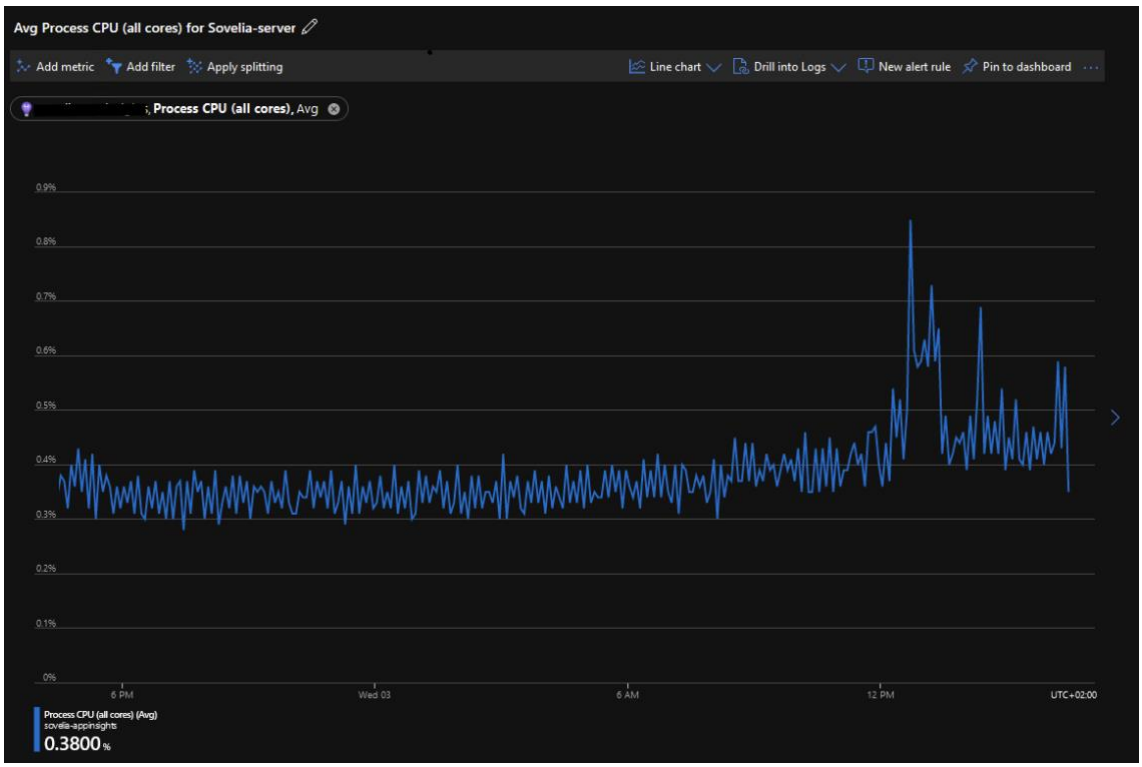
1. palvelimelle lähetettyjen HTTP-pyyntöjen määriä, palvelimen vasteaikoja pyyntöihin sekä epäonnistuneita pyyntöjä ja näiden määriä
2. ulkoisille palveluille (muun muassa tietokantaan) lähetettyjen pyyntöjen määriä, palvelun vasteaikoja pyyntöihin sekä epäonnistuneita pyyntöjä ja näiden määriä
3. ohjelmiston käytössä syntyneitä virheitä ja näiden esiintymismääriä
4. ohjelmiston käyttämä palvelimen muistin määrä, palvelimen keskusyksikön (CPU, engl. central processing unit) rasitus tai verkkoliikenteen määrä
5. kustomoituja suorituskykymetriikoita.

Yllä olevan listan kohtien 1 ja 2 vasteajoilla tarkoitetaan palvelimelta tai tietokannalta kestävää aikaa vastata sille esitettyyn pyyntöön. Esimerkki tunnetusta vasteajasta on verkkosivun lataamiseen kuluva aika. Käyttäjä esittää pyynnön siirtyä verkkosivulle kirjoittamalla verkkosivun osoitteen selaimen URL-kenttään. Palautteena pyyntöön verkkosivu latautuu käyttäjälle. Vasteaika on näiden kahden toiminnon välissä kulunut aika.

Ulkoisten palveluiden monitorointi on suorituskyvyn näkökulmasta kiinnostavaa, koska sillä saadaan selvitettyä ohjelmiston toimintaa hidastavat tietokantahaut. Myös kustomoitujen KPI:den monitorointi on tärkeää. Ne ovat ohjelmistokohtaisia mittareita, jotka korreloivat suorituskyvyn heikkenemisen kanssa. [6] Ohjelmistokohtaisen suorituskykymetriikan implementointi on tärkeää, sillä vakiomittareilla ei ole aina mahdollista saavuttaa kattavaa käsitystä palvelimen suorituskyvyn tilasta.

Seuraavaksi esitetään, miten suorituskykymetriikka näkyy Azuren Application Insights -palvelussa ja tarkastellaan muutamaa yleisempää tapaa monitoroida keskeisiä suorituskykyindikaattoreita. Alla oleva kuva 3 esittää perinteisen viivadiagrammin Sovelia-palvelimen keskusyksikön käyttöasteesta vuorokaudessa. Application Insights generoi viivadiagrammin automaattisesti datan perustella. Tämä data sisältää numeerisen arvon lisäksi myös aikaleiman, milloin data on palvelimelta kerätty. Aikaleiman avulla Application Insights piirtää automaattisesti diagrammin halutulta aikaväliltä. Tällöin kuvaajasta voidaan helposti tulkita keskusyksikön käyttöastetta

tiettynä ajanhetkenä. Tämä on yleinen datan esitystyyli erityisesti aktiivisesti monitoroitavalle metriikalle.

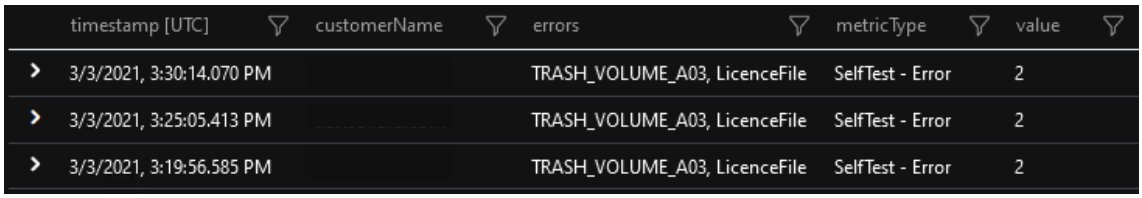


Kuva 3. *Application Insightin generoima viivadiagrammi.*

Passiivisesti kerättävän metriikan esittämiseen käytetään usein pylväsdiagrammeja. Kuva 4 esittää Application Insightsin generoimaa pylväsdiagrammia palvelimella syntyneille *java.lang.Exception*-virheille vuorokauden aikana.

Kuva 4. *Application Insightsin generoima pylväsdiagrammi.*

APM-työkaluun lähetetty data voi myös sisältää lisäinformaatiota, kuten virheviestejä, palvelimen nimen tai SQL-lauseen. Näiden virheviestien lukemiseksi eräs käytetty datan esityskeino on taulukko. Kuva 5 esittää taulukkomuotoista informaatiota eräästä KPI:sta. Jokainen datapiste on esitetty omalla rivillään ja jokaiseen datapisteeseen on liitettynä virheviesti, joka löytyy taulukon kolmannesta sarakkeesta.



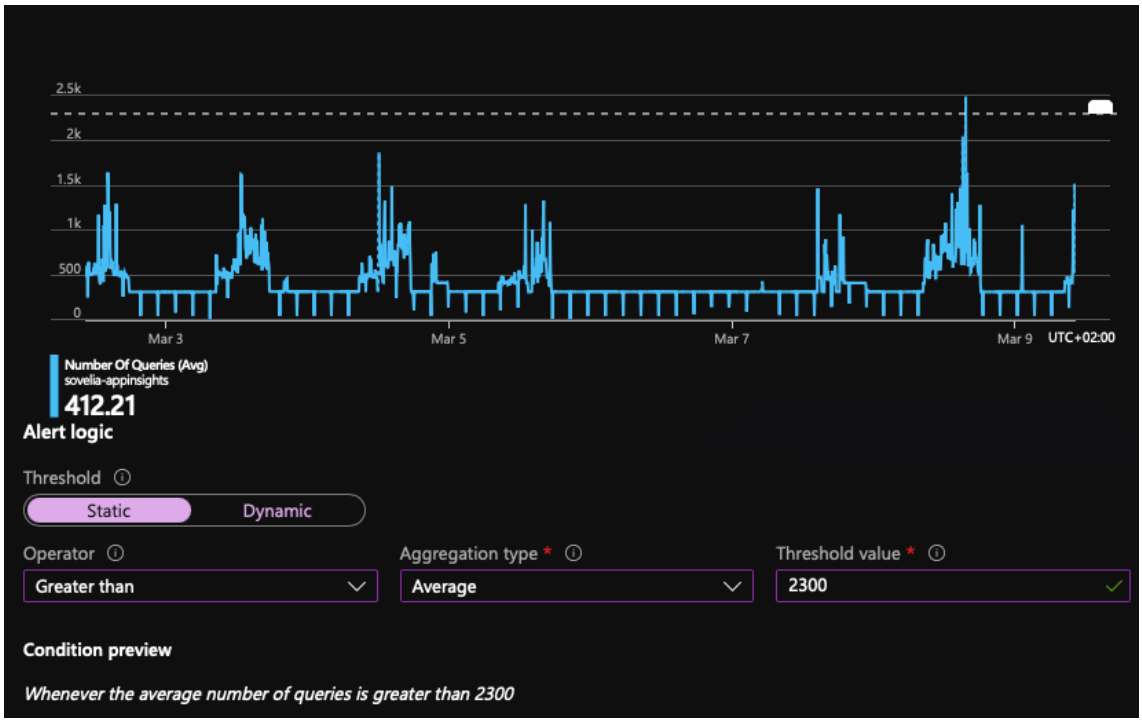
timestamp [UTC]	customerName	errors	metricType	value
3/3/2021, 3:30:14.070 PM		TRASH_VOLUME_A03, LicenceFile	SelfTest - Error	2
3/3/2021, 3:25:05.413 PM		TRASH_VOLUME_A03, LicenceFile	SelfTest - Error	2
3/3/2021, 3:19:56.585 PM		TRASH_VOLUME_A03, LicenceFile	SelfTest - Error	2

Kuva 5. *Application Insightsin taulukko.*

Näiden erilaisten esitysmuotojen avulla voidaan generoida erilaisia näkymiä. Näitä näkymiä kutsutaan kojelaudoiksi (engl. dashboard) [4][5]. Kojelauta on täysin kustomoitava näkymä, jonka voi täyttää erilaisilla diagrammeilla, taulukoilla ja tietokentillä. Tämä näkymä voidaan jakaa organisaation sisällä, joka mahdollistaa muille käyttäjille helpon pääsyn monitoroitavan palvelimen diagnostiikkaan. Kuvassa 6 on esimerkki erään Sovelia-palvelimen kojelautanäkymästä. Kojelaudat helpottavat datan tulkintaa ja monitorointia, sekä tarjoavat paljon informaatiota yhdellä vilkauksella. Kaavioita klikkaamalla käyttäjän on mahdollista pureutua monitorointidataan tarkemmin. Hälyttävä piikki kaaviossa saattaa antaa aihetta tarkemmalle analysoinnille.

Kuva 6. *Esimerkki kojelautanäkymästä.*

Dataa voidaan siis analysoida manuaalisesti kaavioita katsomalla, mutta tämä voi osoittautua työlääksi, kun monitoroitavia palvelimia on useampia ja datan määrä on suuri. Application Insights tarjoaa tähän ongelmaan ratkaisun: automaattiset hälytykset. Kuten alaluvussa 2.2 kerrottiin, hälytys voidaan sitoa yhteen tai useampaan KPI:n arvoon. Hälytys laukeaa tietyn raja-arvon ylittyessä, joka voidaan pääsääntöisesti määrittää kahdella eri tavalla: vertailukohtapohjaisesti tai kynnysarvopohjaisesti. Kynnysarvopohjainen hälytys tunnetaan Application Insightsissa nimellä static. Kuvassa 7 on esitettynä eräälle suorituskykyindikaattorille asetettu kynnysarvopohjainen hälytys. Jos KPI:n arvo ylittää tämän staattisen raja-arvon, lähettää APM-työkalu hälytyksen esimerkiksi sähköpostilla. [3]



Kuva 7. *Kynnysarvopohjaisen hälytyksen määrittäminen Application Insightsissa.*

Kuvassa 8 on esitetty vertailukohtapohjaista raja-arvoa hyödyntävän hälytyksen määrittely samalle suorituskykyindikaattorille. Application Insightsissa tämän raja-arvon nimi on dynamic. Työkalu oppii raja-arvon analysoimalla KPI:n historiadataa käyttäjän määrittelemältä aikaväliltä ja mallintamalla sitä käyttämällä joukkoa erilaisia algoritmeja sekä metodeita. Se oppii muun muassa datan kausittaista vaihtelua esimerkiksi tunti-, päivä- ja viikkotasolla ja pystyy oppimaan mallin jopa hyvin kohinaisesta sekä hajaantuneesta datasta. Kynnysarvot ovat siis valittu siten, että poikkeamat niistä osoittavat poikkeavuuksia myös KPI:n käyttäytymisessä. [9]

Kuva 8. *Vertailukohtapohjaisen hälytyksen määrittely Application Insightsissa.*

Tässä luvussa keskityimme tarkemmin Application Insightsiin ja sen ominaisuuksiin. Alaluvussa 3.2 tarkastellaan tarkemmin Sovelialle ominaisia suorituskykyindikaattoreita ja niiden valintaperusteita.

3.2 Sovelia-palvelimen suorituskykyindikaattorit

Alaluvussa 3.1 tarkasteltiin Azuren APM-työkalua ja sen tarjoamia metriikan keräys-, esitys- ja analysointimahdollisuuksia. Tässä alaluvussa keskitytään Sovelia-palvelimelle ominaisiin suorituskykyindikaattoreihin, niiden valintaperusteisiin, monitorointiin ja analysointiin.

Projektin alkaessa organisaation sisällä keskusteltiin, mitä palvelimen tilasta halutaan tietää ja miten palvelimen elinvoimaisuutta voidaan määrittää mahdollisimman tarkasti. Näihin mittareihin päädyttiin asiakasrajapinnassa työskentelevien työntekijöiden ja ohjelmistokehitystiimin konsultaation jälkeen. Alla on listattuna tärkeimmät monitoroitavat KPI:t, joita Sovelia-palvelimelta lähetetään eteenpäin Azuren Application Insights -työkaluun analysoitavaksi:

1. lisenssien kokonaismäärä
2. käytössä olevien lisenssien määrä
3. epäonnistuneet self-test-testit

4. tietokantakyselyiden määrä minuutissa
5. tietokantatransaktioiden kesto ja määrä minuutissa
6. epäonnistuneiden transaktioiden määrä minuutissa
7. pitkien tietokantatransaktioiden määrä
8. palvelimelle lähetettyjen HTTP-pyyntöjen määrä, palvelimen vasteajat pyyntöihin sekä epäonnistuneet pyynnöt ja näiden määrä
9. ohjelmiston käytössä syntyneet virheet ja näiden esiintymismäärä
10. palvelimen syke
11. palvelimen käyttämä muistinmäärä ja palvelimen CPU:n rasitus.

Lisenssien kulutus kertoo, kuinka monta käyttäjää Soveliaan on kirjautuneena. Tämä korreloi palvelimen kuormituksen kanssa, sillä mitä enemmän käyttäjiä, sitä enemmän palvelimella on ruuhkaa. Lisenssien kokonaismäärän ja käytön seuraaminen tarjoavat myös asiakkaalle arvokasta tietoa ostettujen lisenssien käyttöasteesta.

Self-test-testeillä testataan palvelimen suorituksen aikaista ympäristöä ja Sovelian konfiguraatiota, minkä tarkoituksena on löytää kyseisen Sovelia-konfiguraation tai -asennuksen aiheuttamat ongelmat. Esimerkkejä Self-test-testeistä ovat: onko tietokantayhteydet määritetty oikein ja onnistuuko palvelin luomaan yhteyden tietokantaan. Testeillä pyritään selvittämään, että lähtökohdat ohjelmiston toiminnalle ovat kunnossa. Self-test-testin epäonnistuminen tarkoittaa ongelmaa, joka voi ilmentyä ohjelmiston suorituskyvyn laskuna tulevaisuudessa.

Yllä olevassa luettelossa kohdat 4–7 liittyvät ulkoisen palvelun, tietokannan, monitorointiin. Tietokantakyselyllä tarkoitetaan yhtä SQL-lausetta, joka ajetaan tietokannassa. Transaktiolla taas viitataan joukkoon peräkkäisiä SQL-lauseita, jotka suoritetaan joko kaikki, tai virheen sattuessa koko transaktio kumotaan aiheuttamatta muutoksia tietokantaan. [13] Hidas tietokantahaku saattaa laskea käyttäjän havaitsemaa ohjelmiston suorituskykyä. Hidas tietokantayhteys saattaa ilmentyä esimerkiksi hakutulosten pitkittyneenä valmistumisena. Erityisen kiinnostava mittari on pitkien transaktioiden määrä. Pitkä transaktio saattaa kestää esimerkiksi muutamia minuutteja. Jos ohjelmistossa havaitaan pitkä transaktio, lähetetään siitä välittömästi tieto APM-työkaluun. Pitkä transaktio varaa yhden tai useamman tietokantayhteyden, jolloin näitä yhteyksiä ei voida käyttää muuhun tietoliikenteeseen ohjelmiston ja tietokannan välillä. Tämä laskee lähes poikkeuksetta havaittua suorituskykyä ja saattaa aiheutua esimerkiksi ohjelmiston jumiutumisesta.

Palvelimelle lähetetyt pyynnöt antavat tietoa siitä, mitä käyttäjä on ohjelmistolla tehnyt. Kun esimerkiksi pitkittynyt tietokantatransaktio havaitaan, voidaan katsoa, mitä käyttäjä on tehnyt viiden minuutin aikavälillä ennen havaintoa. Palvelimen vasteajan mittaaminen pyyntöön voi paljastaa myös optimointimahdollisuuksia ohjelmistokooditasolla. Epäonnistuneet pyynnöt voivat johtua esimerkiksi käyttäjän toiminnasta, muun muassa väärän salasanan syöttämisestä tai järjestelmässä syntyneestä virheestä.

Ohjelmiston virhetilojen monitorointi helpottaa muun muassa vianselvitystä. Jos palvelin esimerkiksi kaatuu, voidaan tarkastaa sitä ennen tapahtuneet virheet.

Palvelimen sykkeellä tarkoitetaan tasaisin väliajoin lähetettävää metriikkaa APM-palveluun. Tämän metriikan tarkoituksena ei ole mitata mitään tiettyä arvoa vaan toimia indikaattorina siitä, että palvelin on pystyssä. Sykkeen lähettäminen ei siis riipu mistään ulkoisesta tekijästä eikä sitä voi kytkeä pois päältä, jos monitorointipalvelut ovat käytössä.

Äkillinen piikki palvelimen keskusyksikön käyttöasteessa tai varatussa muistissa on selvä merkki lisääntyneestä rasituksesta. Rasituksen monitorointi voi antaa perusteita esimerkiksi palvelimen komponenttien päivittämiseen tai tarkempaan analysointiin, mitä palvelimella on tapahtunut. Jokaisen KPI:n analysointi sekä monitorointi tehdään Application Insightsissa. Monitorointiin käytetään kuvassa 6 esitettyä kojelautanäkymää. Analysointi tehdään Application Insightsin tarjoamien työkalujen avulla, joista kerrottiin alaluvussa 3.1.

3.3 Sovelia Monitor

Sovelia Monitor viittaa Sovelian moduuliin, joka on vastuussa datan keräämisestä ja lähetyksestä Sovelia-palvelimelta. Koska APM-palveluksi valittiin valmis Microsoftin Azuren tarjoama järjestelmä, ei suorituskykymetriikan analysointiin ja monitorointiin ollut tarvetta itse kehittää minkäänlaista sovellusta.

Kuten alaluvussa 3.1 mainittiin, Azuren Application Insights tukee agentillista sekä agentitonta monitorointia. Koska Azure tarjoaa avoimen lähdekoodin järjestelmän (agentin) tiettyjen KPI:den keräämiseen ja lähetykseen, ei ole järkevää tehdä kaikkea itse [10][11]. Taulukossa 1 on esitetty agentin ja toteutettavan järjestelmän vastuujako KPI:den keräyksessä ja lähetyksessä. Sovelia Monitor on vastuussa kustomoidun suorituskykymetriikan keräämisestä ja lähetyksestä käyttäen alaluvussa 3.1 mainittua Microsoft Application Insights Java SDK Corea.

Taulukko 1. *Java-agentin ja Sovelia Monitorin vastuujako.*

Agentin keräämät ja lähettämät KPI:t	**Sovelia Monitorin keräämät ja lähettämät KPI:t**
HTTP-pyynnöt	Lisenssien kokonaismäärä
Palvelimen vasteajat HTTP-pyyntöihin	Käytössä olevien lisenssien määrä, sekä käyttäjät
Epäonnistuneet HTTP-pyynnöt	Self-test tulokset
Palvelimen käyttämä muistin määrä	Tietokantakyselyt minuutissa
Palvelimen CPU:n käyttöaste	Tietokantatransaktiot minuutissa
Tietokantakyselyt	Epäonnistuneet transaktiot minuutissa
	Ohjelmiston käytössä syntyneet virheet
	Palvelimen syke

Sovelia Monitor ei ole riippuvainen Azuren tarjoamasta Java-agentista. Se on itsenäinen järjestelmänsä, jota on mahdollista käyttää myös ilman agenttia. Tällöin lähetetyn datan määrä pienenee huomattavasti. Jos Java-agentti kytketään pois päältä, Sovelia Monitor lähettää yhä oikeassa sarakkeessa luetellut KPI:t Application Insightsiin. Jos taas Sovelia Monitor kytketään pois päältä, lähettää Java-agentti yhä vasemmassa sarakkeessa määritetyt KPI:t. Seuraavassa luvussa tarkastellaan Sovelia Monitorin kehitysprosessia, rakennetta ja sen erilaisia konfiguraatiomahdollisuuksia.

4. TEKNIIKAT JA TULOKSET

Edellisissä luvuissa käsiteltiin mikä on APM ja miten sitä voidaan hyödyntää ohjelmiston monitoroinnissa. Sitten esiteltiin Soveliaa, sekä Sovelia-palvelimen keskeisiä suorituskykyindikaattoreita. Lisäksi tarkasteltiin Azuren Application Insightsia ja sen mahdollisuuksia palvelimen monitoroinnissa sekä tutustuttiin, mikä Sovelia Monitor oikeastaan on. Tässä luvussa käsitellään Sovelia Monitorin kehitysprosessia, rakennetta, toimintaa ja konfigurointia tarkemmin, jotta saadaan parempi käsitys sen tarjoamista mahdollisuuksista. Alaluvussa 4.3 esitellään työn tulokset.

4.1 Sovelia Monitorin kehitysprosessi

Kuten luvussa 2 kerrottiin, työn lähtökohtana oli tuottaa Java-pohjaisen ohjelmiston suorituskyvyn monitorointiin soveltuva järjestelmä. Tämän vuoksi myös monitorointityökalun toteutuskieleksi valittiin alusta asti Java, jotta sen sulauttaminen Sovelia-palvelimen loppupään lähdekoodiin olisi helppoa.

Alustavasti suorituskykymetriikan lähettäminen sulautettiin olemassa oleviin moduuleihin, joista sitä kerätään. Tämä aiheutti kuitenkin vaikeuksia, koska monitorointityökalu ei ollut selvästi oma kokonaisuutensa. Siksi päädyttiin toteuttamaan monitorointi kokonaan omana moduulinaan, Sovelia Monitorina. Pääsääntöisesti Sovelian eri moduuleja pystytään konfiguroimaan kulloisenkin ympäristön vaatimuksien mukaan moduulien omien INI-tiedostojen avulla. Sovelia Monitorin toteuttaminen omana moduulinaan mahdollisti siis myös työkalulle laajemmat konfigurointimahdollisuudet. Alaluvussa 4.2 tutustutaan tarkemmin näihin.

Sovelia Monitor kerää dataa, joka voi olla asiakkaille sellaisenaan ilman analysointia hyödyllistä, esimerkiksi lisenssien käyttöön liittyvä metriikka. Lisenssien käyttöä seuraamalla asiakas pystyy itse arvioimaan tarvettaan ostaa uusia lisenssejä. Tämän vuoksi on tärkeää, että he pääsevät siihen suoraan käsiksi. Sovelia Monitorin toteuttaminen omana moduulinaan antoi mahdollisuuden luoda useampi Application Insights -yhteys samalta palvelimelta. Suorituskykymetriikkaa voidaan lähettää myös asiakkaan omaan Application Insightsiin, jolloin heille ei tarvitse myöntää käyttöoikeuksia organisaation omaan Azure-pilvipalveluun. Pääsääntöisesti myös kaikkein Sovelia-palvelimien monitorointidata lähetetään samaan Application Insightsiin analysoitavaksi, joten asiakkaille ei haluta antaa pääsyä muiden Sovelia-ympäristöjen suorituskykymetriikkaan. Datan lähettäminen asiakkaan omaan Azure-pilvipalveluun

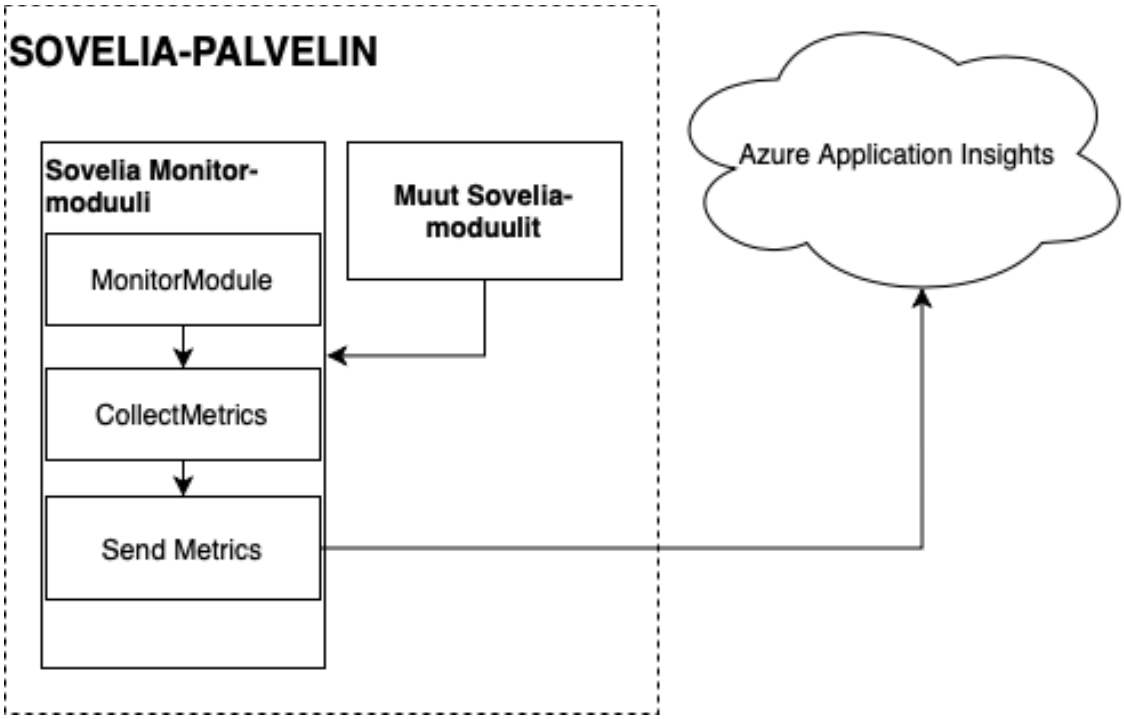
helpottaa käyttöoikeuksien hallinnointia. Alaluvussa 4.2 esitellään, kuinka useampia Application Insights yhteyksiä voidaan konfiguroida ja miten määritetään, mitä suorituskykydataa näihin lähetetään.

Seuraavassa alaluvussa 4.2 esitellään Sovelia Monitorin rakennetta yksityiskohtaisemmin. Käydään läpi muun muassa moduulien eri luokkien vastuualueita metriikan keräyksessä ja lähetyksessä. Asian selkeyttämiseksi listataan myös kaikki konfiguraatioparametrit selityksineen ja vakioarvoineen.

4.2 Sovelia Monitor työkaluna järjestelmän suorituskyvyn monitoroinnissa

Tässä tutkielmassa ei esitellä Sovelia Monitorin toteutusta lähdekooditasolla. Sen sijaan keskitytään sen yleiseen rakenteeseen, toimintaan ja erilaisiin konfiguraatiomahdollisuuksiin.

Kuvassa 9 on yksinkertainen kuvaus Sovelia Monitorin arkkitehtuurista. Moduulikokonaisuus koostuu kolmesta Java-luokasta: *MonitorModule*, *CollectMetrics* ja *SendMetrics*. MonitorModulen päävastuut ovat käyttäjän konfiguraation päivittäminen ja aktiivisesti, eli tasaisin ajanjaksoin, kerättävän datan hakeminen järjestelmän muista moduuleista ja sen välittäminen sellaisenaan CollectMetrics-luokalle. MonitorModule pitää siis huolen Sovelia Monitorin kokonaisuudesta.



Kuva 9. *Sovelia Monitor -moduulin yksinkertaistettu arkkitehtuuri.*

CollectMetricsin pääasialliset tehtävät ovat varastoida kerätty data ja mahdollisesti muokata sitä ennen sen lähetystä. Se siivoaa dataa kevyesti, ennen kuin sitä lähetetään eteenpäin. CollectMetrics voi liittää metriikkaan lisätietoja, joita alaluvussa 3.1 esiteltiin. Tämä luokka taltioi passiivisesti kerättävän metriikan, jota syntyy tietyn tapahtuman sattuessa muissa Sovelian moduuleissa. Kun kaikki tarvittava metriikkadata on lopulta kerätty ja modifioitu, CollectMetrics luokka lähettää datapaketit eteenpäin SendMetricsille. Käyttäjä voi myös suodattaa tietyt datatyypit pois lähetettävästä metriikasta, kuten self-test-metriikan. Tällöin kyseisen datatyypin paketit jätetään välittämättä SendMetrics-luokalle.

SendMetrics lähettää metriikan APM-palveluun. Se myös tarkastaa, että yhteys on konfiguroitu käyttäjän asettamien parametrien mukaisesti ja että monitorointipalvelut ovat käytössä. Jos käyttäjä on asettanut monitoroinnin pois päältä tai yhteyden konfiguraatio on viallinen, datapaketit jätetään lähettämättä. Yksi SendMetrics-luokan instanssi huolehtii yhdestä yhteydestä Azureen, sillä Sovelia Monitor mahdollistaa myös useamman yhteyden käytön samanaikaisesti. SendMetrics luo CollectMetricsin välittämästä datapaketista metriikkaobjektin, joka sisältää KPI:n nimen, arvon, keräysajankohdan sekä mahdolliset lisätiedot, ja lähettää tämän Application Insightsille monitoroitavaksi sekä analysoitavaksi. SendMetrics luokka ylläpitää myös lokia lähetetystä datasta.

Yllä on viitattu moneen kertaan erilaisiin käyttäjän konfiguraatiomahdollisuuksiin. Taulukossa 2 on esitelty Sovelia Monitorin konfiguraatioparametrit, niiden merkitykset sekä vakioarvot. Parametreja säädellään tekstipohjaisessa INI-muotoisessa konfiguraatiotiedostossa. Mikäli jokin avain–arvo pari puuttuu tai arvo on virheellinen, käytetään avaimen vakioarvoa.

Taulukko 2. *Sovelia Monitorin konfiguraatioparametrit.*

Avain	Selitys	Vakioarvo
DEBUG	Lokitus päällä/pois	FALSE
LOG	Lokitiedoston polku	$INST_DIR$/logs/ monitoring.log
RESET_LOG	Lokin resetointi palvelimen uudelleenkäynnistyksessä	TRUE
USER_INITIALS_TO_DEBUG	Sisällytetäänkö Sovelia-käyttäjien käyttäjänimet lokiin	FALSE
SEND_TO_AZURE	Monitorointi päällä/pois	FALSE
INSTRUMENTATION_KEY_XXX	XXX-konfiguraation Application Insights yhteysavain	Undefined
METRICS_FILTER_XXX	XXX-konfiguraation metriikkasuodatin	None
CUSTOMER_NAME	Palvelimen tunniste	Undefined
CONNECTION_SEND_INTERVAL	Tietokantayhteys-metriikan keräysintervalli minuuteissa	5
SELFTEST_SEND_INTERVAL	Self-test-metriikan keräysintervalli minuuteissa	5
SESSIONS_SEND_INTERVAL	Käyttäjä-metriikan keräysintervalli minuuteissa	5
TRANSACTION_THRESHOLD	Pitkän tietokantatransaktion raja-arvo minuuteissa	5

Sovelia Monitor voi lähettää suorituskykymetriikan useaan Azuren Application Insights palveluun, kun konfiguraatiotiedostoon lisätään uudet *INSTRUMENTATION_KEY_XXX* ja *METRICS_FILTER_XXX* parametrit ja korvataan *XXX* halutulla tunnisteella. Keräysintervallit määritetään ainoastaan aktiivisesti kerättävälle metriikalle, koska passiivisesti kerättävä metriikka – kuten ohjelmistossa suorituksen aika tapahtuneet virheet – taltioidaan vasta tietyn tapahtuman tapahtuessa. *CUSTOMER_NAME*-parametri identifioi palvelimelta lähetetyn metriikan. Tämän avulla Application Insightsissa voidaan suodattaa kaikesta suorituskykymetriikasta vain tiettyä palvelinta koskevat arvot, jos samaan Application Insights -palveluun lähetetään dataa useammalta palvelimelta.

Kuten alaluvussa 3.3 todettiin, suorituskykymetriikan keräämiseen ja lähettämiseen on myös mahdollista käyttää Azuren tarjoamaa Java-agenttia joko yhdessä tai ilman Sovelia Monitoria. Java-agentin konfigurointi tapahtuu omalla konfigurointitiedostolla. Taulukossa 2 esitetyillä parametreilla ei ole vaikutusta agentin toimintaan, koska tämä on kokonaan oma järjestelmänsä. Tässä työssä ei käsitellä Java-agenttiin liittyviä konfigurointimahdollisuuksia. Seuraavaksi tarkastellaan tämän työn tuloksia.

4.3 Tulokset

Sovelia Monitor on uusi lisäys Sovelia-tuotteeseen, joten se ei ole vielä käytössä asiakasympäristöissä. Tällä hetkellä Sovelia Monitor on testausvaiheessa ja aktiivisena muutamilla testipalvelimilla. Testipalvelimien ongelmana on, että palvelimien kuorma on vähäisempää kuin todellisissa asiakasympäristöissä. Tämä johtuu siitä, että testiympäristöissä Sovelian käyttö on vähäisempää kuin oikeassa asiakasympäristössä. Vähäisen kuormituksen vuoksi selkeä suorituskyvyn lasku testiympäristöissä on harvinaista. Tämä vaikeuttaa esimerkiksi datan analysointia, koska hälytysten raja-arvojen määrittäminen on hankalaa, eikä kipurajaa tiedetä. Merkittäviä tuloksia on kuitenkin saatu.

Tärkeimpänä työn tuloksena on moduuli, joka onnistuneesti kerää ja lähettää suorituskykymetriikkaa palvelimelta Azuren Application Insightsiin. Tästä datasta on rakennettu erilaisia diagrammeja ja taulukoita, joista on luotu erilaisia kojelautanäkymiä palvelimen elinvoimaisuuden kuvaamiseksi. Ne tarjoavat reaaliaikaisen näkymän palvelimen tapahtumiin ja sen liikenteeseen. Kojelautanäkymää tarkkailemalla on helppo havaita esimerkiksi piikkejä ohjelmiston suorituksen aikana syntyneiden virheiden määrissä. Selkeästi erottuvaa piikkiä kannatta analysoida tarkemmin, jotta saadaan selville, mitä virhetilanteita ohjelmistossa on tapahtunut.

Sovelia-palvelimen vasteaikaa, eli aikaa, joka palvelimella kestää vastata sille esitettyyn pyyntöön, monitoroimalla on saatu selville pullonkauloja ohjelmiston toiminnassa. Pullonkaulalla tarkoitetaan ohjelmiston osaa, joka estää ohjelmiston toiminnan täydellä kapasiteetilla. Jos yhteen tiettyyn pyyntöön vastaamiseen kuluu palvelimelta paljon aikaa, aiheuttaa se suorituskyvyn laskua. Jos tämä pyyntö lähetetään usein, saattaa se kerryttää palvelimen kuormitusta. Pullonkauloja voidaan saada poistettua tai niiden vaikutuksia pienennettyä optimoimalla ohjelmistokoodia.

Monitorointityökalulla on myös saatu tarkempi käsitys Sovelian ja tietokannan välisestä yleisrasitteesta. Yleisrasitteella tarkoitetaan tietokantakyselyiden määrää, joita suoritetaan, vaikka Sovelia ei olisi aktiivisessa käytössä. Yleisrasite muodostuu siis ajanjaksolta, jolla ohjelmistoa ei käytetä.

5. YHTEENVETO

Työn tuloksena saatiin Sovelian Java-pohjaisen palvelimen suorituskykyä monitoroiva moduuli, Sovelia Monitor. Moduuli on upotettu osaksi palvelimen lähdekoodia ja tämän vuoksi reaaliaikaisen suorituskykymetriikan kerääminen, muokkaaminen ja lähettäminen APM-palveluun suoraan palvelimelta on suoraviivaista. Datan analysointiin käytettiin Microsoftin Azure-pilvipalveluihin kuuluvaa APM-palvelua nimeltään Application Insights, joka mahdollistaa suorituskyvyn poikkeamien automaattisen havaitsemisen erilaisilla analysointi- ja monitorointityökaluilla. Se tarjoaa reaaliaikaista tietoa ohjelmiston suorituskyvyn muutoksista sekä apua suorituskyvyn että käytettävyyden parantamiseen. [6]

Tällä hetkellä testausvaiheessa oleva monitorointityökalu on jo tuottanut arvokasta tietoa Sovelian käyttäytymisestä. Sen avulla on havaittu pullonkauloja ohjelmiston toiminnassa, minkä lisäksi käsitys Sovelian ja tietokannan välisestä yleisrasitteesta on parantunut. Sovelia Monitorin päätarkoituksena on havaita palvelimen suorituskyvyn lasku, jotta siihen voidaan puuttua ennen vakavampia seurauksia. Se tarjoaa myös arvokasta historiatietoa palvelimen käyttäytymisestä, mitä voidaan hyödyntää myöhemmin Sovelian kehityksessä, palvelimella sattuneiden virhetilojen selvittämisessä sekä ongelmien proaktiivisessa ehkäisemisessä.

Kehitystyön seuraavat vaiheet ovat:

1. Kerätä lisää laadukasta suorituskykydataa.
2. Syventää ymmärrystä, kuinka analysoida kerättyä dataa tehokkaammin.
3. Määrittää, kuinka automaattisesti reagoida dataan.

Laadukkaan suorituskykydatan kerääminen on tärkeää, jotta suorituskykyindikaattoreiden raja-arvot voidaan määrittää. Raja-arvolla tarkoitetaan arvoa, jonka ylittyessä tai alittuessa ohjelmiston suorituskyky tulkitaan heikentyneen. Tässä kehityksen vaiheessa kaikille suorituskykyindikaattoreille ei ole täysin selvää, minkä pisteen jälkeen ohjelmiston suorituskyky on selkeästi heikentynyt.

Suorituskykydatan analysoinnin kehittäminen on tärkeää, jotta poikkeamat suorituskyvyssä saadaan havaittua mahdollisimman aikaisin. Tämä tarkoittaa erilaisten hälytyksien ja tekoälymallien kehittämistä datan manuaalisen analysoinnin tueksi. Manuaalisesti suurien tietomäärien analysoiminen on raskasta ja vaikeaa. Application Insightsissa on mahdollista sitoa erilaisia automaattisia hälytyksiä tiettyyn

suorituskykymetriikkaan, sekä opettaa eri malleja tuntemaan tietyn metriikan arvon vaihteluita näin havaiten poikkeamia suorituskyvyssä.

Automaattinen reagointi vaatii myös tutkimustyötä. On tärkeää määrittää toimintamalli reagoinnille, kun asiakkaan Sovelia-ympäristöstä saadaan indikaatio suorituskyvyn laskusta. Suorituskyvyn laskuun reagointi vaatii suoria toimia asiakkaan omalla palvelimella tai palvelimen tilan tarkempaa tarkkailua sekä tutkimista, ettei vakavia virhetiloja pääse syntymään. Näin pystytään takaamaan, että Sovelia pysyy saatavilla asiakkaalle.

LÄHTEET

[1] T.M. Ahmed, C. Bezemer, T. Chen, A. E. Hassan, W. Shang, Studying the Effectiveness of Application Performance Management (APM) Tools for Detecting Performance Regressions for Web Applications: An Experience Report, International Conference on Mining Software Repositories, IEEE/ACM, Austin, Texas, USA, 2016, pp. 1–12.

[2] D. An, Find out how you stack up to new industry benchmarks for Mobile Page Speed, Google, 2018. Saatavissa (viitattu 22.2.2021): https://www.thinkwithgoogle.com/intl/en-154/marketing-strategies/app-and-mobile/how-stack-up-new-industry-benchmarks-for-mobile-page-speed.

[3] R.J. Boucher, B. Wren, D. Coulter, Understand how metric alerts work in Azure Monitor, Microsoft, 2021. Saatavissa (viitattu 9.3.2021): https://docs.microsoft.com/en-us/azure/azure-monitor/alerts/alerts-metric-overview.

[4] C. Fitz-Gibbon, Performance indicators, Multilingual Matters, Clevedon England, 1990, 109 p.

[5] L. Gayhardt, B. Wren, J. Martinez, D. Coulter, R.J. Boucher, M. Goedtel, et al., Application Insights Overview dashboard, Microsoft, 2019. Saatavissa (viitattu 9.3.2021): https://docs.microsoft.com/en-us/azure/azure-monitor/app/overview-dashboard.

[6] L. Gayhardt, B. Wren, M. McCleary, J. Martinez, D. Coulter, K. Sharkey, et al., What is Azure Application Insights?, Microsoft, 2019. Saatavissa (viitattu: 28.2.2021): https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview.

[7] Google Data, n=3,700 aggregated, anonymised Google Analytics data from a sample of mWeb sites opted into sharing benchmark data, Google, 2016.

[8] C. Heger, A. van Hoorn, M. Mann, D. Okanović, Application Performance Management: State of the Art and Challenges for the Future, International Conference on Performance Engineering, ACM, 2017, pp. 429–432.

[9] Y. Lavi, D. Coulter, B. Wren, Metric Alerts with Dynamic Thresholds in Azure Monitor, Microsoft, 2021. Saatavissa (viitattu: 9.3.2021): https://docs.microsoft.com/en-us/azure/azure-monitor/alerts/alerts-metric.

[10] Maven Repository: com.microsoft.azure >> applicationinsights-core, Maven, 2020. Saatavissa (viitattu 28.2.2021): https://mvnrepository.com/artifact/com.microsoft.azure/applicationinsights-core.

[11] MS-jgol, T. Stalnaker, H. Y, M. Perfetti, PRMerger19, D. Coulter, et al., Java codeless application monitoring Azure Monitor Application Insights, Microsoft, 2020. Saatavissa (viitattu: 8.3.2021): https://docs.microsoft.com/en-us/azure/azure-monitor/app/java-in-process-agent.

[12] The NIST Definition of Cloud Computing, SP 800-145, National Institute of Standards and Technology, 2011. Saatavissa (viitattu: 26.3.2021): https://csrc.nist.gov/publications/detail/sp/800-145/final.

[13] A. Oppel, SQL: A Beginner's Guide, 4th Edition, McGraw-Hill, 2015.

[14] Back end, Oxford Advanced Learner’s Dictionary, Oxford University Press. Saatavissa (viitattu 26.3.2021): https://www.oxfordlearnersdictionaries.com/definition/english/back-end_1.

[15] F. Piedad, M. Hawkins, High availability: design, techniques, and processes, 1st Edition, Prentice Hall PTR, 2001, 266 p.

[16] N. Ruparelia, Cloud computing, The MIT Press, Cambridge Massachusetts, USA, 2016, 278 p.

[17] M. Sahasrabudhe, M. Panwar, S. Chaudhari, Application performance monitoring and prediction, International Conference on Signal Processing, Computing and Control, IEEE, 2013, pp. 1–6.

[18] R. Sturm, C. Pollard, J. Craig, Application Performance Management (APM) in the Digital Enterprise: Managing Applications for Cloud, Mobile, IoT and EBusiness, Elsevier Science & Technology, San Francisco, 2017, 304 p.