jamk.fi

Elastic APM-palvelu

Sovellushallinnan hyödyntäminen kehitystyössä

Juuso Oksanen

Opinnäytetyö
Kesäkuu 2020
Liiketalouden ala
Tradenomi (AMK), tietojenkäsittelyn tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Oksanen, Juuso	Opinnäytetyö, AMK	Kesäkuu 2020
	Sivumäärä 58	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Elastic APM-palvelu
Sovellushallinnan hyödyntäminen kehitystyössä

Tutkinto-ohjelma
Tietojenkäsittelyn tutkinto-ohjelma

Työn ohjaaja(t)
Kiviaho, Niko

Toimeksiantaja(t)
Kansaneläkelaitos, IT-palvelujen tulosyksikkö

Tiivistelmä

Sovellushallintaan tarkoitetut automatisoidut sovellukset on kehitetty monimutkaisten sovellusarkkitehtuurien hahmottamisen ja seurannan avuksi. Automaattiset työkalut parantavat pilvi- ja verkkopalveluiden hallintaa. Sovellushallinta on tarpeellista tuotantoympäristöjen hallintaan ja monitorointiin. Hyödyntämällä sovellushallintaa kehityksen aikana varmistutaan ennalta sen toimivuudesta tuotannossa ja saadaan samalla kehitystiimille sen tuomat monitorointihyödyt. Yksi näistä ratkaisuista on Application Performance Management eli APM.

Tutkimuksen tavoitteena oli perehtyä Elastic APM-palveluun ja sen soveltuvuuteen kehitystiimin työvälineeksi. APM-palvelun käyttöönoton jälkeen selvitettiin kehitystiimin saamia hyötyjä uudesta työvälineestä. Lisäksi selvitettiin Elastic APM:n toimintojen kattavuus suhteessa tyypillisimpiin APM-ratkaisujen ominaisuuksiin.

Tutkimuksen aikana asennettiin Elastic APM-palvelu kehitystiimin käyttöön. Tulokset saatiin antamalla kehitystiimille oikeus käyttää APM-palvelua palvelukokonaisuuteensa monitorointiin sekä haastattelemalla kehitystiimiä sen tuomista hyödyistä. Elastic APM-palvelun todettiin täyttävän APM:lle asetettujen avaintoimintojen kriteerit.

Tutkimuksesta voitiin vetää johtopäätös, että APM-palvelua käyttämällä kehitystiimi pystyi kehittämään sovellushallintaansa, sekä kehittämiensä sovelluksien suorituskyvyn laatua. Muutkin kehitystiimit voisivat tehostaa kehitystyötänsä instrumentoimalla sovelluksensa APM-palveluun.

Avainsanat (asiasanat)
Elastic, Java, sovellushallinta, APM, instrumentointi, suorituskyvyn monitorointi

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s)	Type of publication	Date
Oksanen, Juuso	Bachelor's thesis	June 2020
		Language of publication: Finnish
	Number of pages 58	Permission for web publication: x

Title of publication

Elastic APM Service

Utilizing application management in software development

Degree programme

Business Information Technology

Supervisor(s)

Kiviaho, Niko

Assigned by

The Social Insurance Institution of Finland

Abstract

Automated solutions for application management have been developed to aid understanding and monitoring of complex application architectures. Automated tools improve the management of cloud and web services. Application management is necessary for the management and monitoring of production environments, and by utilizing it during development, its functionality in production is ensured in advance, and the development team gets the benefits of monitoring. One of these automated solutions is Application Performance Management, or APM.

The aim of the study was to get acquainted with the Elastic APM and evaluate its suitability as a tool for the development team. Following the introduction of the APM service, the benefits of the new tool for the development were investigated. In addition, the coverage of Elastic APM's functions of the most typical features of APM solutions was investigated.

During the study, the Elastic APM service was installed for the development team's use. The results were obtained by giving the development the right to use the APM service to monitor their service under development and by interviewing the development team about the perceived benefits. The Elastic APM service was found to meet the criteria for key functions defined for APM.

The development team was able to enhance their application management, as well as the performance of their applications. Other development teams could as well improve their development work by instrumenting their services to the APM service.

Keywords/tags (subjects)

Elastic, Java, application management, APM, instrumentation, performance monitoring

Miscellaneous (Confidential information)

Sisältö

Taulukot

Kuviot

Käsitteet

JVM eli Java Virtual Machine on sovellus, joka toteuttaa Javan ajoympäristön ja suorittaa Java-sovellukset. JVM on suunniteltu virtualisoimaan Java-sovellukset niin että sovellukset toimivat käyttöjärjestelmästä riippumatta. (Niemeyer 2013 s. 68)

JAR eli Java Archive on ZIP-tiedostoformaattiin perustuva tiedosto, joka sisältää mahdollisesti META-INF-kansion. Ajettavan Java-sovelluksen tulee sisältää manifestitiedosto, joka määrittelee sovelluksen Main-luokan. (JAR File Specification N.d)

JPA eli Java Persistence API on ohjelmointirajapinta, joka tarjoaa relaatiotietokantojen taulujen sisältämän datan muodostamisen olioiksi ja takaisin. (Introduction to the Java Persistence API N.d)

HTTP (Hyper Text Transfer Protocol) on asiakaskoneen ja palvelinkoneen välinen protokolla. Koneiden välinen kommunikaatio tapahtuu lähettämällä pyyntö (HTTP Request) ja vastaanottamalla vastaus (HTTP Request). (What is HTTP? N.d)

API eli Application Programming Interface on ohjelmointirajapinta. API on joko sovelluksen sisäisen käytön rajapinta tai ulkoinen rajapinta, jolla sovelluksen keskustelevat keskenään. (Korhonen 2018)

REST eli Representational State Transfer on arkkitehtuurityyli, joka perustuu HTTP-protokollaan ja sen standardimetodeihin. API:t julkaistaan yleensä hyödyntäen REST-arkkitehtuurityyliä. (Korhonen 2018)

JSON eli Javascript Object Notation on tiedon näyttämiseen käytetty tietomalli. JSON hyödyntää nimi/arvo (key/value) pareja ja taulukoita tiedon näyttämiseen. (Korhonen 2018)

Inline on funktio, jolla kääntäjä (compiler) vaihtaa funktion kutsun itse funktion koodiin. (The __inline Keyword for Inline Functions N.d)

Mikropalvelut tai mikropalveluarkkitehtuuri on arkkitehtuurillinen tyyli, joka jäsentää sovelluksen palveluiden kokoelmaksi. (Richardson N.d)

Sovelluspalvelin on Java EE-spesifikaation mukainen Java-ajoympäristö web-sovelluksille. (Java EE Servers N.d)

Aggregointi on tiedon yhdistelyä ja summaamista yleiskuvan esittämiseksi. (Keskeiset käsitteet N.d)

1 Johdanto

Web-sovelluskehitys on saavuttanut pisteen, jossa sovellukset jakautuvat pienempiin osiin, mutta samalla niiden väliset riippuvuudet monimutkaistuvat. Yksittäinen sovellus on yksinkertaisempi ja helpompi kehittää, mutta kokonaisuuden hallinta on haastavampaa. Järjestelmien kehittyessä monimutkaisemmiksi kokonaisuuksiksi, valvonta ja suorituskykyongelmien ratkominen vaikeutuu jatkuvasti, jolloin manuaalisesti toteutettu lokitus ei ole enää tarpeeksi tehokas tai joustava tapa hallita ongelmatilanteita. Itse toteutettu monitorointi vaatii sovelluskehittäjiltä valtavaa huolellisuutta ja tietoa siitä, mitä tietoja lokitetaan. Ennakoimattomien ongelmien ilmetessä sovellukset vaatisivat lisää käsin tehtävää sovelluslokitusta, jos ongelma ei satu osumaan jo olemassa oleviin käyttölokituksiin.

Sovellushallintaan tarkoitetut automatisoidut sovellukset on kehitetty monimutkaisten sovellusarkkitehtuurien hahmottamisen ja seurannan avuksi. Automaattiset työkalut parantavat pilvi- ja verkkopalveluiden hallintaa. Sovellushallinta on tarpeellista tuotantoympäristöjen hallintaan ja monitorointiin. Hyödyntämällä sovellushallintaa kehityksen aikana varmistutaan ennalta sen toimivuudesta tuotannossa ja samalla saadaan kehitystiimille sen tuomat monitorointihyödyt. Yksi näistä ratkaisuista on Application Performance Management eli APM.

2 Tutkimusasetelma

2.1 Toimeksiantaja

Tutkimuksen toimeksiantajana toimii Kelan IT-palvelujen tulosyksikkö. Kela ylläpitää ja kehittää Suomen sosiaaliturvan järjestelmiä. Kelan IT-palvelujen tulosyksissä työskentelee yli 700 henkilöä. Työntekijöiden toimipaikka sijaitsee pääosin Jyväskylässä ja Helsingissä. (IT-palvelut 2020)

2.2 Tutkimusongelma

Toimeksiantaja on kiinnostunut APM-ratkaisun hankinnasta ja toimeksiantajalla on kehitystiimi, jolla on tarve tehokkaampaan sovellushallintaan. Kehitystiimillä on ollut käytössään sovelluslokien vasteaikojen monitorointi, mutta kyseinen palvelu nojaa sovelluskehittäjien omatekoisen lokituksen toteuttamiseen. Lisäksi monitorointipalvelua ei enää jatkokehitetä, joten uusien raportointinäkymien tekeminen tai tietueiden visualisoiminen ei ole enää mahdollista.

Tutkimuksen tarkoituksena on tutkia Elastic NV:n tarjoamaa Elastic APM-palvelua. Tutkimuksessa selviää, mitä hyötyjä toimeksiantaja saisi verkkopalveluidensa monitoroinnista, ja kuinka kerättyä dataa voitaisiin hyödyntää muun muassa suorituskykyongelmien selvittämiseen ja verkkoliikenteen profilointiin.

Tutkimuksessa haetaan vastauksia seuraaviin kysymyksiin:

- Mitä on APM?
- Mikä on Elastic APM-palvelu?
- Miten kehitystiimi voi hyödyntää APM-palvelua kehitystyössään?
- Miten Elastic APM-palvelu toteuttaa APM:n avaintoiminnot?

2.3 Rajaukset

APM-ratkaisun tutkiminen on rajattu yhteen APM-ratkaisuun. Tällöin ei saada vertailtua eri APM-ratkaisujen keskinäistä paremmuutta, mutta pystytään syventymään yhteen APM-ratkaisuun ja vastaamaan toimeksiantajan antamiin kysymyksiin Elastic APM-palvelusta tarkemmin.

Elastic APM-palvelu toteutetaan Elasticin ilmaislisenssillä. Tämä johtaa siihen, etteivät kaikki tuotteen maksullisen lisenssin ominaisuudet ole käytettävissä. Tutkimuksessa selviää täyttääkö Elasticin ilmaislisenssi tarvittavat kriteerit tuotantokäyttöön, vai tarvitseeko toimeksiantaja jatkokehitystä varten maksullisen lisenssin.

Tutkimus on rajattu yhden nimetyn Java-ohjelmistokehitystiimin näkökulmaan, jolloin keskitytään kehitystiimin saamiin hyötyihin tutkimuksen tuloksissa.

2.4 Tutkimusote

Tutkimuksessa on kyseessä kehittämistutkimus, koska tutkimuksen tuotteena kehittyy palvelu, jota kehitystiimi, sekä toimeksiantaja pystyvät hyödyntämään. (Kananen 2012, 19) Tutkimuksen tavoitteena on myös arvioida Elasticin tarjoamien suorituskyvynmonitorointityökalujen soveltuvuutta tutkimuksen kohdeympäristöjen valvontaan ja ylläpitoon, ja kuvailla monitorointipalvelun toimintaa kehitystiimin kehitystyössä. Lisäksi seurataan monitorointipalvelun tuottaman tiedon hyödyllisyyttä kehitystiimille.

Tutkimuksessa perehdytään tutkittavaan aiheeseen keräämällä aiheeseen liittyvää tietoa kirjallisista ja sähköisistä tietolähteistä. Tietoa voidaan kerätä erilaisista tietolähteistä kuten esimerkiksi tutkimuksista, raporteista, teorioista ja malleista, jotka sivuavat tutkittavaa aihetta. Viitekehyksen luominen auttaa ongelman ymmärtämisessä, sekä auttaa ymmärtämään tutkittavaa aihetta. (Mts. 47–48)

Opinnäytetyö perustuu perinteiseen tutkimusmenetelmään, joka koostuu kenttätyön tekemisestä ja sen jälkeisestä kirjoitusvaiheesta (mts. 48). Kenttätyön aikana kerätään tietoa Elastic APM-palvelun dokumentaatiosta sekä Java-instrumentoinnin dokumentaatiosta.

2.5 Tutkimuksen toteutus

Tutkimuksen alussa selvitetään kehitystiimin lähtötilanne tutkimalla tiimin nykyisiä sovellushallintaratkaisuja ja kuinka tiimi hyödyntää niitä työssään. Tutkimuksen osana toteutetaan Proof of Concept (PoC) arvioitavasta palvelusta.

Kerätystä aineistoista on tavoitteena selvittää, millaisia avaintoimintoja ja hyötyjä sovellushallinta ja APM tarjoavat ja kuinka niitä voidaan hyödyntää kehitystiimin työssä. Samalla tutustutaan Elastic APM-palvelun arkkitehtuuriin, sen muodostaviin komponentteihin ja niiden rooleihin APM:n elinkaaressa. Lisäksi käydään läpi teoriaa Javan monitorointiin sopivista mekanismeista.

Tavoitteena on tuoda Elastic APM-palvelu käyttöön eri kohdeympäristöissä ja tutkia APM-palvelun soveltuvuutta kehitystiimin työssä. Kohdeympäristöinä toimivat kehitystiimin kehitys- ja testausympäristöt. Tutkimus tuottaa kuvauksen APM-palvelusta, jota toimeksiantaja pystyy käyttämään APM-ratkaisuiden sopivuuden arviointiin. Lisäksi selvitetään, täyttääkö Elasticin ilmaislisenssi tarvittavat kriteerit APM-palvelun saattamiselle tuotantoympäristöön, vai tarvitaanko sitä varten maksullisen lisenssin tuomia lisäominaisuuksia. Kenttätyön aikana kerätään aineistoa myös APM-palvelun asentamisesta ja pystyttämisestä, sekä sen saattamisesta kehitystiimin käyttöön.

3 APM

Kappaleessa käydään läpi mitä sovellushallinta ja APM ovat. Lisäksi tutkitaan sovellushallinnan historiaa sovellushallinnan kehittymisestä uusien teknologioiden ilmetessä ja selvitetään millaisia toimintoja APM-ratkaisujen tulisi kattaa.

3.1 Mitä on APM?

APM eli Application Performance Monitoring tai Application Performance Management tarkoittaa suomeksi sovelluksen suorituskyvyn hallintaa tai monitorointia. Molempia termejä voidaan käyttää, sillä niiden kattamat työkalut ja toiminnot sisältävät sovellusten saatavuuden ja suorituskyvyn monitorointia ja hallintaa (Watts 2018). Laajimmassa merkityksessä APM:sta voidaan puhua sovellushallintana (application management). Sovellushallinta koostuu sovelluksen monitoroinnista ja hallinnasta, jonka tavoitteena on optimoida suorituskykyä, saatavuutta ja tietoturvallisuutta annetun palvelulupauksen varmistamiseksi. (Sturm 2017)

Jokaisen APM-ratkaisun tehtävänä on ylläpitää odotettua palvelun tasoa. APM keskittyy sovellusten suorituskykyongelmien löytämiseen ja niiden diagnosointiin, jotta ongelmat pystytään tunnistamaan ja korjaamaan ennen kuin ne kasvavat merkittävämmiksi ongelmiksi. Vahvan APM-ratkaisun tulee löytää syy ongelmaan eikä vain korjausta siihen. (Watts 2018)

APM-ratkaisut voivat auttaa muun muassa tunnistamaan yleisiä sovellusongelmia kuten:

- Jäljittää yleistä sovelluksen käyttöä hahmottaakseen piikit tietoliikenteessä
- Löytää hitaus- ja yhteysongelmia sovellusten riippuvaisuuksissa kuten esimerkiksi SQL-yhteyksissä, jonoissa ja välimuistissa.
- Tunnistaa hitaat SQL-kyselyt
- Löytää lukumääräisesti suurimmat ja hitaimmat transaktiot

(Altvater 2015)

3.2 Sovellushallinnan historia

Sovellushallinnan historia ja kehitys on ollut kiinteästi ja erottamattomasti sidottua yleiseen ohjelmistotekniikan kehitykseen. Huolimatta siitä, että sovellushallinta on kiinteä osa infrastruktuurikokonaisuutta, ovat sen innovaatiot seuranneet muun kehityksen perässä. (Sturm 2017)

Taulukko 1. Sovellushallinnan kehittyminen vuosikymmenittäin (Sturm 2017, muokattu)

Aikajakso	**IT-teknologian edistys**	**Sovellushallinta**
1950–1959	Tietokoneiden laaja käyttöönotto	Sovellushallinta tapahtui ilman työkalujen apua
1960–1969	Keskustietokoneet	Virhelokit olivat ensisijainen työkalu sovelluksen hallinnalle ja monitoroinnille
1970–1979	Online, reaaliaikainen prosessointi	Alkeellista informaatiota sovellushallintaa varten alettiin tarjota käyttöjärjestelmien puolesta.
1980–1989	Keskustietokoneiden kehittyminen Pientietokoneiden käyttöönotto	Kasvava määrä ISV hallintatyökaluja keskustietokoneille ja vähemmän pientietokoneille.

	PC (Personal Computer)	PC:lle ei ole sovellushallintatyökaluja.
1990–1999	Internet/LAN/WAN Client/Server-arkkitehtuuri Hajautettu laskenta	Avoimet, standardiin pohjautuvat hallintatyökalut, -laitteet ja -sovellukset. Integroitu, kokonaisvaltainen lähestymistapa sovellushallintaan.
2000–2009	Pilvipalvelut Virtualisointi SaaS (Software as a Service)	Sovellukset eivät pyöri enää yksittäisessä, yhtenäisessä ympäristössä. Hallinnan mahdollisuudet heikentyivät suhteessa uusiin ympäristöihin. SaaS tarjoaa halvempia sovelluksia, mutta estää yritykseltä mahdollisuuden hallita itse sovelluksia.
2010->	SOA-palvelut Mikropalvelut Mobiilisovellukset Sulautetut palvelut (IoT)	Hienostuneita, automatisoituja hallintatyökaluja, jotka ovat luotu hallitsemaan tällaisia muuttuvia ympäristöjä. Mobiilisovellukset ovat haasteellisia sovellus-

		hallinnalle kommunikaation, turvallisuuden ja mittasuhteen kannalta.

3.3 APM:n avaintoiminnot

APM:n tarkoitus on selvittää ”miksi” mahdollisimman nopeasti. Web-sovellusten suorituskyvyn mittaamiseksi on hyvin yksitoikkoista seuloa sovelluslokia, ja selvittää kuinka kauan web-pyyntö kestää. Tämä antaisi vain idean sovelluksen yleisestä suorituskyvystä, ja siitä mitkä pyynnöt ovat hitaita. Tämä ei kuitenkaan kerro miksi pyynnöt ovat hitaita. (Altvater 2015)

Kehittäjille tärkeintä APM:ssä on datan monipuolinen ja runsas kerääminen, sekä mahdollisuus sen analysointiin. Datan lisäksi kehittäjät tarvitsevat vastauksia, joihin löytyy ratkaisu datasta, jolloin he voivat nopeasti päästä sovellusten ongelmien juurisyyhyn. APM-ratkaisut tarjoavat monenlaisia toimintoja eri käyttötarkoituksiin. (Mt.)

Analytiikka ja visualisointi

APM-sovelluksen tulee pystyä monitoroimaan useita erilaisia asioita, kuten esimerkiksi transaktioita, vasteaikoja, muistinkäyttöä ja tietokantakyselyitä. Näitä tietoja pitää pystyä pilkkomaan pienemmiksi osiksi ja jäsentelemään tyypeittäin. APM:n tarjoaman visualisointityökalun pitää pystyä esittämään erilaisia metriikoita. Näitä ovat esimerkiksi suorittimen kuorma, levyn- ja muistinkäyttö, virhetilat ja muut kriittiset muuttujat. Raportointityökalussa on hyvä olla roolipohjainen pääsynhallinta ja roolien oikeudet huomioivat raportointikyvykkyydet. Raportointityökalussa pitäisi olla mahdollista sovittaa sen näyttämät metriikat, mittaukset ja monitorointi käyttäjien tarpeisiin. (Application Performance Management 2019)

Transaktioiden suorituskyvyn seuraaminen

APM-ratkaisun on pystyttävä mittaamaan jokaisen web-pyynnön ja transaktion suorituskykyä sovelluksessa ja luokitella ne esimerkiksi toiminnoittain. Tällöin pystytään tulkitsemaan pyynnöt, joita tehdään eniten, mitkä niistä ovat hitaimpia, ja mitkä niistä vaativat lisäkehitystä. (Altvater 2015)

Palvelun kartoitus ja riippuvaisuuksien hahmotus

Palvelun kartoituksella tarkoitetaan grafiikkaa, joka esittää miten eri komponentit keskustelevat keskenään. Ideaalisesti näiden toimintojen tulisi automaattisesti löytää riippuvaisuudet ja päivittää niitä reaaliajassa, kun ne muuttuvat. Tämä tarjoaa nopean ja visuaalisen tavan ymmärtää sovelluksia ja auttaa selvittämään ongelma-alueita. (Watts 2018)

Kooditason suorituskyvyn profilointi

Hitauden ja virheiden lähteen ymmärtämiseen on APM-ratkaisussa tarpeellista päästä tutkimaan ja seuraamaan suoritusta sovelluskooditasolle asti. Seuraamalla sovelluksen suoritusta kooditasolla voidaan saada lisää tietoa sovelluksesta. Kiinnostavia tietoja ovat muun muassa:

- Mitä keskeisiä metodeja koodissa kutsutaan usein?
- Mitkä metodit ovat hitaita?
- Onko sovellus hidas esimerkiksi muistinsiivoamisen takia?
- Mitä riippuvuuksia kutsutaan koodissa?

(Altvater 2015)

Yksityiskohtainen jäljitys yksittäisille transaktiolle tai web-pyynnöille

Ongelmien selvitys tuotannossa voi olla vaikeaa. Transaktioiden jäljittämisellä pyritään selvittämään mitä sovelluksissa tapahtuu ja miten se vaikuttaa käyttäjiin.

Tracet voivat sisältää muun muassa dataa web-pyyntöjen informaatiosta kuten mikä oli kutsun URL, kuka kutsun käyttäjä oli, mitä riippuvaisuuksia sovellukset tarvitsivat kutsun aikana, lokimerkintöjä, sovellusvirheitä, ja oleellisimpia metodeja sovellusten koodissa. (Mt.)

Real User Monitoring (RUM)

Real User Monitoring eli RUM on client-puolella tapahtuvaa monitorointia. RUM tallentaa ja analysoi jokaisen käyttäjään kohdistuvan transaktion nettisivulla tai sovelluksessa. Sitä käytetään mittaamaan käyttäjäkokemusta, joka sisältää esimerkiksi keskeisiä metriikoita kuten latausajat ja navigaatiopolut. (Altvater 2020)

Nykypäivän sovelluksissa käytetään erittäin paljon muun muassa JavaScript-ohjelmointikieltä, jolloin on tärkeää ymmärtää kuinka kauan selaimella kestää ladata ja piirtää nettisivut. RUM on tärkeä toiminnallisuus projekteille, joissa on käytössä runsaasti eri nettisivuja ja joiden suorituskykyä tulisi monitoroida. (Altvater 2015)

Keskitetty sovellusvirheiden seuranta ja hälytykset

Kehittäjille on oleellista olla tietoisia ympäristöjensä sovelluksen suorituksen virheistä ja jatkuvasti tarkkailla niitä. Virheet ovat tärkeyslistassa ensimmäisenä, kun etsitään sovelluksiin liittyviä ongelmia. Virheenjäljitys, raportointi ja hälytykset ovat kriittisiä toimintoja kehittäjille APM-ratkaisuissa. APM-ratkaisussa on tärkeää olla mahdollisuus konfiguroida hälytyksiä, esimerkiksi asettaa hälytyksiä uusille poikkeuksille, sekä pystyä monitoroimaan virheiden esiintymistiheyttä. Kun tuodaan tuotantoon uusi sovellus, on tarpeellista seu-

rata virheitä virheraportointityökaluilla uusien ongelmien havaitsemiseksi. Tällöin uudet virheet pystytään tunnistamaan ja korjaamaan nopeasti. (Altvater 2015)

Hälytysten konfigurointi on APM-ratkaisulle hyväksi. APM-ratkaisuissa luodaan monia erilaisia hälytyksiä ilmoittamaan virheistä kehitys-, testi- ja tuotantoympäristöissä. Kyseiset hälytykset pitää pystyä konfiguroimaan niin, että ne tiedottavat oikeita vastuuhenkilöitä esimerkiksi sähköpostilla tai tekstiviestillä. Hälytyksiä tulisi pystyä myös konfiguroida suorittamaan valmiiksi määriteltyjä toimintoja, kun hälytykselle kuvattu tilanne, kuten esimerkiksi uusi virhetilanne tapahtuu. (Application Performance Management 2019)

4 Instrumentointi Javassa

Kappaleessa selvitetään mitä on instrumentointi ja miten instrumentointi toteutetaan Java-ohjelmointikielessä. Lisäksi kappaleessa käsitellään Java-agentteja, sen muodostavia komponentteja sekä mekanismeja, joilla Java-agent kiinnitetään käynnistettävään tai käynnissä olevaan Java-sovellukseen.

4.1 Instrumentointi

Java-sovelluksen suorituksesta voidaan kerätä metriikkaa Java-agentin ja instrumentoinnin avulla. Elastic APM-palvelun Java-toteutus nojaa tähän ohjelmointirajapintaan tietueiden muodostamiseksi. Java-agentit ovat osa Java Instrumentation-ohjelmointirajapintaa. Ymmärtääkseen miten Java-agentit toimivat, täytyy ymmärtää mitä instrumentointi on. (Schults 2019)

Sovellusten instrumentointi on tekniikka, jota käytetään laajalti sovelluksen profiloinnissa, suorituskyvyn analysoinnissa, optimoinnissa, testauksessa, virheiden havaitsemisessa ja virtualisoinnissa. Instrumentointi, johon liittyy ylimääräisen koodin lisääminen sovellukseen jonkin ohjelman käyttäytymisen

monitoroimiseksi, voidaan tehdä joko staattisesti (käännöksen aikana) tai dynaamisesti (ajon aikana).

Staattiset instrumentointitekniikat vaihtelevat yksinkertaisista manuaalisista tekniikoista kääntäjä- tai konekielipohjaisiin instrumentointeihin ja linkityksenaikaiseen tai -jälkeiseen ajettavien ohjelmistojen muokkaamiseen.

Dynaamisen instrumentoinnin tekniikat ovat yleensä monimutkaisempia toteuttaa kuin staattiset, mutta ne pystyvät jäljittämään ajonaikaisesti linkitettyjä kirjastoja ja muita haaroja, joita on vaikea käsitellä staattisella implementaatiolla. (Kempf 2008)

4.2 Java-agentin rakenne ja rajapinnat

Java-agent on erityistyyppinen luokka, jotka hyödyntävät Java Instrumentointirajapintaa sovellusten bittikoodin muokkaamiseksi. Java-agenttia voidaan hyödyntää muun muassa ylimääräisen informaation tuottamiseen sovelluksen metodin suorittamisessa, joista voidaan muodostaa esimerkiksi lokitiedosto. (Sathiyakugan 2020)

Java-agentin paketointi

Java-agent otetaan käyttöön JAR-tiedostona. Toteutuksissa, jotka tukevat komentorivirajapintaa, agentti käynnistetään määrittelemällä parametri komentorivillä. Toteutukset voivat myös tukea mekanismia aloittaakseen agentit vasta JVM:n käynnistymisen jälkeen. Esimerkiksi implementaatio voi tarjota mekanismin, joka mahdollistaa työnkalun kiinnittämisen ajossa olevaan sovellukseen ja aloittaa työkalun agentin lataamisen sovellukseen. Yksityiskohdat agentin lataamiselle ovat toteutuksesta riippuvaisia. (Package java.lang.instrument N.d)

Käynnistäminen komentorivirajapinnan avulla

Toteutuksissa, jotka tarjoavat tavan käynnistää agentit komentoriviltä, agentti voidaan käynnistää antamalla parametri komentorivillä:

-javaagent:jarpath[=options]

jarpath on polku agentin JAR-tiedostoon ja **options** on agentin mahdolliset parametrit. Agentteja voidaan samaa komentoa toistamalla luoda monta instanssia. Samaa jarpathia voi käyttää useampi kuin yksi agentti. Agentin JAR-tiedoston täytyy noudattaa JAR-tiedostomäärityksiä.

Agentin JAR-tiedoston manifestin täytyy sisältää attribuutti Premain-Class. Attribuutin arvo on agentin luokan nimi. Agentti-luokan täytyy toteuttaa premain-metodi, joka on periaatteeltaan samanlainen kuin sovelluksen main-metodi. JVM:n käynnistyksen jälkeen jokaista premain-metodia kutsutaan agenttien määritysjärjestyksessä. Tämän jälkeen itse sovelluksen main-metodia kutsutaan. Sovellus käynnistyy, kun jokaisen agentin premain-metodi on suoritettu loppuun.

Toteutetulla premain-metodilla on kaksi mahdollista otsikkoa. JVM pyrkii ensimmäiseksi kutsumaan seuraavaa metodia agenttiluokassa:

public static void premain(String agentArgs, Intrumentation inst);

Jos agenttiluokka ei toteuta tätä metodia, niin JVM kutsuu seuraavaa metodia:

public static void premain(String agentArgs);

Agenttiluokalla voi olla myös agentmain-metodi käytettäväksi tilanteessa, jossa agentti on käynnistetty vasta JVM:n käynnistyksen jälkeen. Kun agentti on käynnistetty käyttämällä komentoriviä, agentmain-metodia ei kutsuta.

Agenttiluokka ladataan järjestelmän luokkalataajan toimesta. Kyseinen luokkalataaja tyypillisesti sama, joka lataa myös sovelluksen main-metodin sisältä-

vän luokan. Tällöin premain-metodit ajetaan samoja turvallisuus- ja luokkalataajasääntöjä noudattaen kuin sovelluksen main-metodi. Agentin premain-metodissa ei ole mallillisia rajoitteita. Kaikki, mitä sovelluksen main-metodi pystyy tekemään, on mahdollista myös premain-metodille.

Jokaiselle agentille annetaan sen agenttikohtaiset asetusparametrit agentArgs parametrin kautta. Agentin asetusparametrit annetaan yhtenä merkkijonona. Agentin tulee itse suorittaa kaikki muu ylimääräinen parsinta.

Jos premain-metodi ei ole sopiva tai luokkaa ei onnistuta lataamaan, JVM:n toiminta lakkaa. Myös, jos agentin premain-metodi palauttaa käsittelemättömän virheen, JVM:n toiminta lakkaa. (Package java.lang.instrument N.d)

Agentin käynnistys JVM:n käynnistyksen jälkeen

Agentin implementaatio voi tarjota mekanismin agentin käynnistämiseen vasta JVM:n käynnistyksen jälkeen. Yksityiskohdat käynnistykselle ovat implementaatiokohtaisia, mutta yleensä sovellus on jo käynnistynyt ja sen main-metodia on jo kutsuttu. Näissä tapauksissa implementaation toimiakseen seuraavat asiat pitävät olla toteutettu:

1. Agentin JAR-tiedoston täytyy sisältää Agent-Class attribuutti. Attribuutin arvo on agenttiluokan nimi.
2. Agent-luokan täytyy toteuttaa public static agentmain-metodi
3. Järjestelmän luokkalataajan täytyy tukea mekanismia lisätäkseen agentin JAR-tiedoston järjestelmän luokkapolkuun.

Agentin JAR-tiedosto lisätään järjestelmän jo käynnissä olevan JVM:n luokkalataajan luokkapolkuun. Tämä on tyypillisesti luokkalataaja, joka lataa luokan, joka sisältää sovelluksen main-metodin. Agentti-luokka ladataan ja JVM kutsuu agentmain-metodia. JVM ensimmäiseksi pyrkii kutsumaan seuraavaa metodia:

public static void agentmain(String agentArgs, Instrumentation inst);

Jos agentti ei implementoi tätä metodia, niin JVM yrittää kutsua seuraavaa metodia:

public static void agentmain(String agentArgs);

Agentti-luokka voi sisältää myös edellä käsitellyn premain-metodin. Kun agentti käynnistetään vasta JVM:n käynnistyksen jälkeen, premain-metodia ei kutsuta.

Toteutetun agentmain-metodin täytyy tehdä kaikki tarpeelliset käynnistystoimenpiteen agentin käynnistämiseksi. Kun käynnistys on onnistunut, metodin tulee palautua. Jos agenttia ei pystytä käynnistämään tai jos agentmain-metodi palauttaa käsittelemättömän virheen, JVM:n toiminta ei pysähdy. (Package java.lang.instrument N.d)

Manifestin attribuutit

Seuraavat manifestin attribuutit on määritelty agentin JAR-tiedostossa:

- Premain-Class – Kun agentti on määritelty JVM:n käynnistyksen aikana, tämä attribuutti määrittelee agent-luokan. Kun agentti käynnistetään JVM:n käynnistyksessä, tämä attribuutti on vaadittu. Jos attribuuttia ei löydy, JVM lopettaa käynnistymisensä.
- Agent-Class – Jos implementaatio tukee mekanismia JVM:n käynnistymisen jälkeiselle agentin käynnistykselle, tämä attribuutti määrittää agent-luokan. Tämä attribuutti on pakollinen ja jos se puuttuu, agenttia ei käynnistetä.
- Boot-Class-Path - Lista polkuja, joita esilatausluokkalataaja tutkii. Polut edustavat kansioita tai kirjastoja. Esilatausluokkalataaja tutkii polkuja, jos alustakohtaiset mekanismit agent-luokan löytämiseksi epäonnistuvat. Polkuja tutkitaan siinä järjestyksessä, kun ne ovat listattu. Listatut

polut ovat erotettu toisestaan yhdellä tai useammalla välilyönnillä. Polun syntaksi on hierarkkisen URI:n mukainen. Polku on absoluuttinen, jos se alkaa ”/”-symbolilla, muuten se on relatiivinen. Relatiivinen polku päätellään agentin JAR-tiedoston absoluuttisesta polusta. Epämuodostuneita ja olemattomia polkuja ei käsitellä. Kun agentti on käynnistynyt vasta JVM-käynnistymisen jälkeen, polkuja, joita ei tunnisteta JAR-tiedostoiksi ei käsitellä. Tämä attribuutti on vaihtoehtoinen.

- Can-Redefine-Classes – Totuusarvo, joka kertoo, onko kyky uudelleen määrittää luokka tarpeellista kyseiselle agentille. Tämä on vaihtoehtoinen attribuutti, joka on oletusarvoisesti arvoltaan ”false”.

- Can-Retransform-Classes – Totuusarvo, joka kertoo, onko kyky uudelleen määrittää luokka tarpeellista kyseiselle agentille. Tämä on vaihtoehtoinen attribuutti, joka on oletusarvoisesti arvoltaan ”false”.

- Can-Set-Native-Method-Prefix - Totuusarvo, joka kertoo, onko kyky asettaa natiivi metodin etuliite tarpeellista kyseiselle agentille. Tämä on vaihtoehtoinen attribuutti, joka on oletusarvoisesti arvoltaan ”false”.

Agentin JAR-tiedostolla voi olla molemmat Premain-Class- ja Agent-Class-attribuutit läsnä manifestissaan. Kun agentti käynnistetään komentoriviltä, Premain-Class attribuutti määrittää agentin nimen ja Agent-Class attribuuttia ei käsitellä. Vastaavasti jos agentti käynnistetään JVM:n käynnistyksen jälkeen, Agent-Class attribuutti määrittää agent-luokan nimen ja Premain-Classia ei käsitellä. (Package java.lang.instrument N.d)

Transformer-luokka

Instrumentoinnille oleellinen osa on Transformer-luokka, eli muuntajaluokka. Muuntajaluokka rekisteröidään instrumentointi-instanssilla, ja sitä kutsutaan aina kun luokkalataaja lataa uuden luokan. Muuntajaluokkien tehtävä on muokata ladattavan luokan bittikoodia. Esimerkiksi muuntajaluokka voi lisätä luokkaan ennen tai jälkeen metodin suorittamista ylimääräisiä rivejä koodia, jotka mittaavat esimerkiksi metodin suorittamisen kestoa. (ks. liite 5)

Muuntajaluokat perivät ClassFileTransformer-luokan, jolla on seuraava metodin allekirjoitus, joka täytyy toteuttaa:

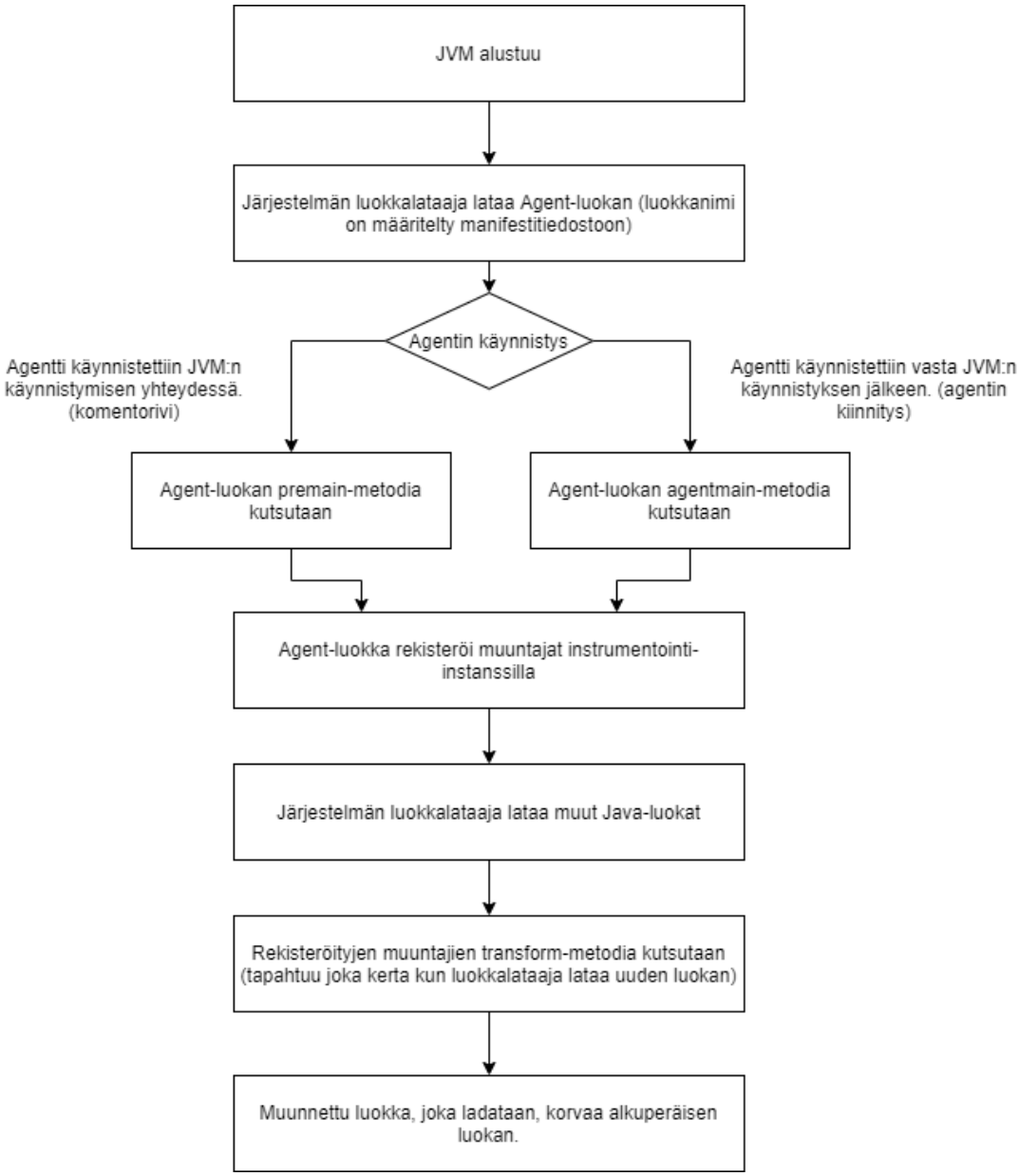
public byte[] transform(ClassLoader loader, String className, Class classBeingRedefined, ProtectionDomain protectionDomain, byte[] classFileBuffer) throws IllegalClassFormatException

(Java Instrumentation 2012)

4.3 Java-instrumentoinnin elinkaari

Java-instrumentoinnin elinkaari alkaa JVM:n alustumisesta. Tämän jälkeen JVM:n luokkalataaja lataa agenttiluokan ja kutsuu riippuen käynnistysmekanismista joko premain-metodia tai agentmain-metodia. Agenttien premain- ja agentmain-metodeilla rekisteröidään instrumentoinnissa käytettävät muuntajaluokat.

Luokkalataaja kutsuu instrumentointi-instanssin rekisteröidyn muuntajaluokan transform-metodia, joka suorittaa ladattavan luokan bittikoodin muokkaamisen ja sen palauttamisen. Luokkalataaja lataa muokatun bittikoodin, joka nyt sisältää muuntajaluokan tekemät muutokset. (ks. Kuvio 1). Liitteistä löytyy yksinkertainen Java-sovelluksen ja Java-agentin lähdekoodi, sekä niiden suoritus, jossa havainnollistetaan instrumentoinnin hyödyntämistä.



Kuvio 1. Java-instrumentoinnin elinkaari (Java Instrumentation 2012, muokattu)

Vaikka instrumentointia hyödynnetään Javan suorituskyvyn profilointiin, voi se myös itse vaikuttaa suorituskykyyn. Esimerkiksi, JVM inlineää pienikokoisia metodeja, jottei metodin kutsuminen olisi tarpeellista lyhyen metodin koodin suorittamiselle. Kääntäjä tekee päätöksen inlineämisestä koodin kokoon perustuen. Pienikokoisia metodeja ei voida välttämättä inlinetä, jos instrumentoinnin lisäämän koodin määrä on liian suuri. (Oaks 2014, s. 54)

5 Elastic APM-palvelu

Kappaleessa käsitellään tutkimukseen valittua Elastic APM-palvelua. Kappale selventää mistä komponenteista Elastic APM koostuu, ja mitkä ovat näiden komponenttien tehtävä tutkittavan datan elinkaaressa. Lisäksi käsitellään Elastic APM:n tietomallit, jossa kerrotaan tietomallien rakenne sekä millaista dataa tietomalleilla esitetään.

5.1 Elastic NV

Elastic APM-palvelun kehittäjänä toimii Elastic NV. Elastic NV on vuonna 2012 perustettu amerikkalaishollantilainen yhtiö, joka on aiemmin tunnettu nimellä Elasticsearch. (Our Story N.d)

Yritys tunnetaan parhaiten avoimen lähdekoodin Elastic Stackista, joka koostuvat seuraavasta kolmesta sovelluskomponentista: Elasticsearch, Kibana ja Logstash. Nämä kolme komponenttia toteuttavat kokonaisuuden, jolla käyttäjät voivat tehokkaasti toteuttaa keskitettyä lokienhallintaa, tutkimista, analysointia ja visualisointia. (Chhajed 2015, s. 5)

5.2 Elastic APM-komponentit

Elastic APM pohjautuu neljään eri komponenttiin, joista kolme toimivat itse APM-palveluna, ja joista yksi komponentti toteuttaa APM metriikan lähettämisen palveluun.

Elasticsearch

Elasticsearch on hajautettu dokumenttisäilö ja hakukone, joka on rakennettu Apache Lucene-hakukonekirjaston pohjalta. Elasticsearch säilöö monimutkaisia datarakenteita, jotka ovat serialisoitu JSON-dokumenteiksi. Jos klusterissa on käynnissä monta Elasticsearch-nodea, säilötyt dokumentit hajautetaan klusterin välillä ja ovat saatavissa miltä tahansa nodelta.

Kun dokumentti on säilötty, se on indeksoitu ja haettavissa Elasticsearchissa lähes reaaliajassa. Elasticsearch käyttää käänteistä indeksiä, joka tukee kokonaisten tekstien nopeaa hakua. Käänteinen indeksi listaa jokaisen uniikin sanan, joka esiintyy missä tahansa dokumentissa ja tunnistaa kaikki dokumentit, jossa jokainen sana esiintyy. Oletusarvoisesti Elasticsearch indeksoi kaiken datan jokaisesta kentästä ja jokaisella indeksoidulla kentällä on oma optimoitu datarakenne.

Tarvittaessa Elasticsearch voi olla myös skeematon, jolloin dokumentit voidaan indeksoida ilman täsmennystä, miten jokainen dokumentissa esiintyvä kenttä käsittellään.

Elasticsearch tarjoaa yksinkertaisen REST API:n klusterin hallintaan, indeksointiin ja datan hakemiselle. Tuotteen REST API tukee rakenteellisia kyselyitä, tekstikyselyitä ja monimutkaisempia kyselyitä, jotka yhdistävät molempia. (Elasticsearch Reference 2020)

Kibana

Kibana on avoimen lähdekoodin analytiikka- ja visualisointialusta, joka on suunniteltu toimimaan Elasticsearchin näkymänä ja käyttöliittymänä. Kibanaa käytetään datan etsimiseen ja näyttämiseen Elasticsearchiin tallennetuista indekseistä.

Kibana toimii selainpohjaisena käyttöliittymänä, jolla voidaan luoda dynaamisia raportointinäkymiä. Kibana toimii myös hallintakäyttöliittymänä muun muassa tietoturva-asetuksien ja käyttäjien roolin hallinnalle. (Kibana Guide 2020)

APM Server

APM Server on avoimen lähdekoodin sovellus, joka ottaa vastaan suorituskykydataa APM agenteilta.

APM Server on suunnitellusti erillinen komponentti, muun muassa seuraavista syistä:

- Se auttaa pitämään agenttien suorituksen rasituksen kevyenä.
- Koska APM Server on oma komponenttinsa, voidaan sitä skaalata riippumattomasti.
- APM Serveristä voidaan kontrolloida datan määrää, mikä virtaa Elasticsearchiin.
- Tilanteessa, kun Elasticsearch ei vastaa kutsuihin, APM Server voi säilyttää dataa väliaikaisesti muistissaan ilman, että APM agenttien rasite lisääntyy.
- APM Server tarjoaa JSON-pohjaisen API:n agenteille käytettäväksi ja näin parantaa yhteensopivuutta eriversioisten agenttien ja Elastic Stackin kanssa.

APM Server validoi ja käsittelee vastaanottamansa tapahtumat, ennen kuin palvelin kääntää datan Elasticsearch-dokumenteiksi ja tallentaa ne vastaavaan Elasticsearch-indeksiin. APM Serveristä paljastetaan portti, johon APM agentit lähettävät keräämänsä datan (APM Server Reference 2020)

APM Java Agent

APM agentti mittaa sovelluksen suorituskykyä ja pitää kirjaa sovelluksen sisäisistä virheistä. Pystyäkseen tähän, agentti hyödyntää JVM:än instrumentointimekanismeja instrumentoimaan luokkien bittikoodia. Agentti automaattisesti instrumentoi tuettuja teknologioita, kuten esimerkiksi Servlet API:a, ja ottaa ylös tapahtumia, kuten esimerkiksi jaksoja tietokantakyselyistä ja transaktioita http-pyynnöistä.

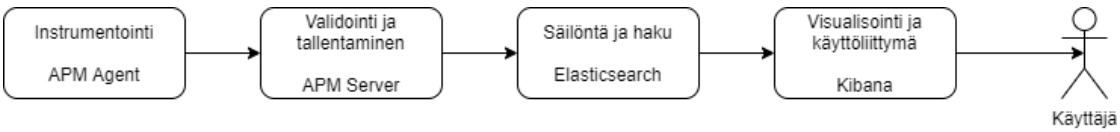
APM-agentit lähettävät tallentamansa tiedon APM serverille käsiteltäväksi ja validoitavaksi, jonka jälkeen dokumentiksi transformoitu data lähetetään Elasticsearchiin vastaavaan indeksiin.

Jos automaattinen instrumentointi ei ole riittävää monitorointitarpeisiin, voidaan sovellukseen lisätä riippuvuudeksi apm-agent-api-kirjasto, jolla voidaan luoda omia transaktioita ja jaksoja sovelluksen sisällä. (APM Java Agent Reference 2020)

Sovellusten riippuvuus ja tehtävät datan elinkaaressa

APM:ssä datan elinkaari alkaa APM Agenteista. APM agenttien tehtävä on instrumentoida sovelluksen koodi, kerätä suorituskykymetriikkaa ja virhetilanteita sovelluksen ajon eli runtimen aikana. Agenttien keräämä data säilyy hetken aikaa välimuistissa, kunnes se lähetetään APM Serverille. APM Server vastaanottaa suorituskyky datan, validoi kyseisen datan. Tämän jälkeen data muodostetaan Elasticsearch-dokumenteiksi ja lähetetään vastaavaan Elasticsearch-indeksiin.

Onnistuneen tallennuksen jälkeen tallennetut tietueet voidaan visualisoida ja analysoida Kibanalla. Kibanaa käytetään Elasticsearchiin tallennettujen tietuiden hakemiseen, näyttämiseen ja käsittelyyn. (Ks. Kuvio 2)



Kuvio 2. Elastic APM-komponentit ja datan elinkaari (APM Overview 2020, muokattu)

5.3 Elastic APM-tietomallit

Elastic APM Agentit instrumentoivat sovelluksien bittikoodin tapahtumia eri tietomalleihin. Tietomallien tyyppi riippuu tapahtuman luonteesta tai tapahtuman työn tasosta. Näitä tietotyyppejä ovat jaksot, transaktiot, virheet, metriikat ja niitä ryhmittelevä trace. (APM Overview 2020)

Jakso

Jaksot (Span) sisältävät informaatiota tietyn koodipolun suorittamisesta. Jaksot mitataan suorituksen alusta loppuun, ja niillä voi olla äiti/lapsi suhde toisten jaksojen kanssa. Agentit instrumentoivat automaattisesti lukuisia kirjastoja jaksojen nappaamiseksi sovelluksen sisällä. On myös mahdollista hyödyntää Elasticin Agent-ohjelmointirajapintoja tiettyjen koodipolkujen itseräätälöityyn instrumentointiin. (Mt.)

Jaksot voivat sisältää seuraavia tietoja:

- transaction.id attribuutin, joka viittaa jakson äititransaktioon.
- parent.id attribuutin, joka viittaa jakson äitijaksoon tai -transaktioon
- jakson aloitus- ja lopetusajan
- nimen
- tyypin, alatyypin ja toiminnon
- vaihtoehtoisen stacktracen

Jaksot säilötään Elasticsearchin jaksoindekseihin, jotka ovat erillään transaktioindekseistä. (Mt.)

Transaktio

Transaktio on erikoistyyppinen jakso, joka sisältää ylimääräisiä attribuutteja liittyen niihin. Transaktiot kuvaavat tapahtumaa, jonka APM Agent on napannut instrumentoidessaan palvelua. Transaktiot ovat palvelun korkeamman tason tehtäviä.

Esimerkkinä transaktio voi olla:

- Pyyntö palvelimelle
- Eräajo

- Tausta-ajo
- Oma määrittelemä transaktio

Transaktio voi sisältää monta jaksoa, tai ei ollenkaan.

Transaktio sisältää:

- Aikaleiman tilanteesta
- Ainutlaatuisen id:n, tyypin ja nimen
- Tietoa ympäristöstä missä tapahtuma on kuvattu:
    - Service – ympäristö, sovelluskehys, ohjelmointikieli.
    - Host – arkkitehtuuri, hostname, IP-osoite.
    - Process – argumentit, PID, PPID.
    - URL – koko URL, domain, portti, kysely.
    - User – sähköposti, ID, käyttäjänimi.

(Mt.)

Virhe

Error eli virhetapahtuma sisältää vähintään tietoa alkuperäisestä virheestä, poikkeuksesta tai lokitiedosta, joka luotiin poikkeuksen tapahtuessa. Virheet esitetään uniikilla tunnisteella.

Virhe sisältää:

- Napatun poikkeuksen ja lokin virheestä, joka sisältää stacktracen, joka helpottaa debuggausta.
- Virheen syyllisen, josta virhe syntyi.

- Virhe voidaan liittää transaktioon, minkä aikana virhe tapahtui transaction.id attribuutin avulla.
- Dataa ympäristöstä, missä tapahtuma tallennettiin.

(Mt.)

Metriikka

APM agentit automaattisesti keräävät isäntäkoneen perusmetriikkaa, jotka sisältävät järjestelmä- ja prosessitason suoritin- ja muistimetriikkaa. Agenttikohtaiset metriikat ovat myös mahdollisia, kuten JVM-metriikat Java-agentille. Infrastruktuuri ja sovellusmetriikat ovat oleellisia informaation lähteitä, kun monitoroidaan tuotantojärjestelmiä. (Mt.)

Trace

Yhdessä transaktiot ja jaksot muodostavat tracen. Tracet eivät ole tapahtumia, vaan ryhmittävät yhteen tapahtumat, joilla on yhteinen juurisyy.

Tracet mahdollistavat mikropalveluarkkitehtuurin kokonaismittaisen analysoinnin yhdessä näkymässä. Esimerkiksi tracesta voidaan hahmottaa koko kutsun elinkaari – mihin palveluihin tehdään kutsuja ja missä järjestyksessä. (Mt.)

6 Tutkimustulokset

Kappaleessa käsitellään tutkimuksen kenttätyön aikaisia tuloksia. Tutkimustulokset ovat jaettu kolmeen osaan. Ensiksi käsitellään Elastic APM-palvelun pystytyksen suunnitelma, sen toteutus ja onnistuminen. APM-palvelun käyttöönoton jälkeen käsitellään kehitystiimin saamia hyötyjä APM-palvelun hyödyntämisestä. Lopuksi evaluoidaan Elastic APM-palvelun validiteetti ja sen laadullisuus tutkimalla ja vertailemalla palvelun toiminnallisuuksia APM:lle asetettuihin avaintoimintoihin.

6.1 Elastic APM-palvelun pystytys

Tutkimuksen tekniset spesifikaatiot ja odotukset

Taulukko 2. Asennetun Elastic APM-palvelun sovellusversiot

APM-palvelun komponentti	**Sovellusversio**
APM Server	7.4.1
Elasticsearch	7.4.1
Kibana	7.4.1
APM Java Agent	1.10

Taulukko 3. Käytössä olevat odotettavasti instrumentoitavat kirjastot

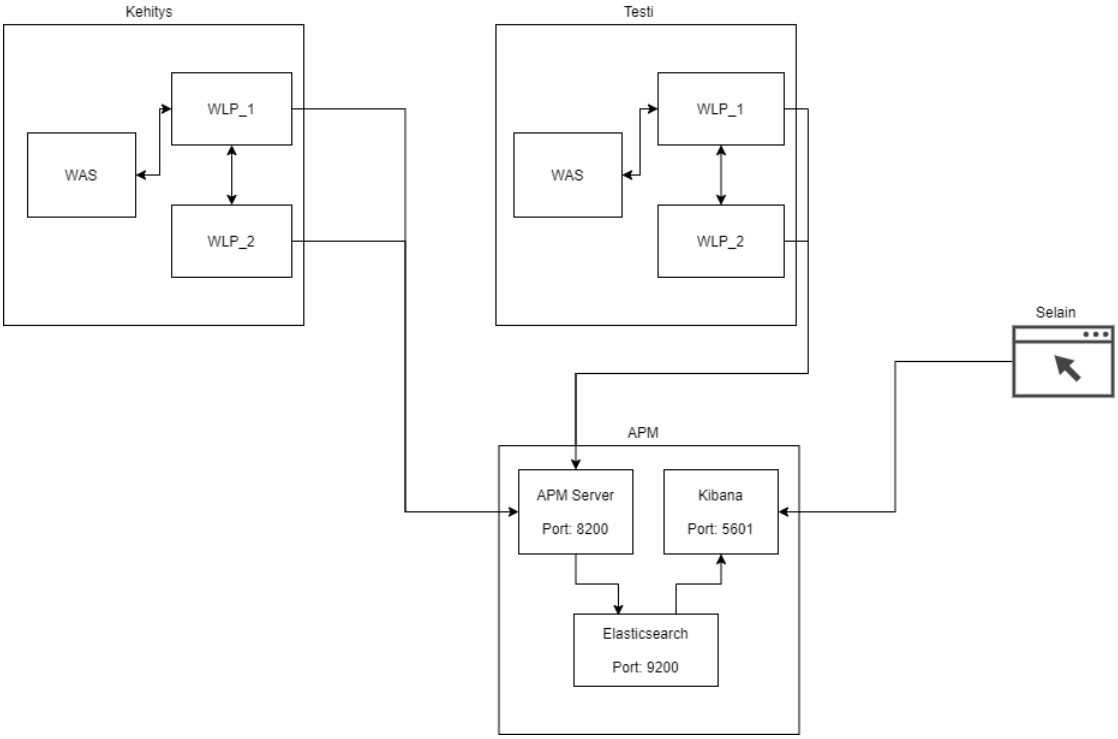
Kirjastot ja JVM:t	**Tuetut versiot**
OpenJDK	8u40+
IBM J9 VM	8 service refresh 5+
Servlet API	3+
Spring Boot	2.x
Websphere Liberty	18+
JDBC	4.1+

Apache HttpClient	4.3+
Spring RestTemplate	4+
OkHttp	2, 3
Scheduling Annotation	Spring Framework riippuvainen
SLF4J	1.4.1+

Elastic APM-palvelu pystytettiin noudattaen taulukon 2 sovellusversioita. Kenttätyön aikainen oletus oli, että taulukon 3 kirjastot pystyttäisiin instrumentoimaan onnistuneesti APM-palvelun käyttöönoton jälkeen. Taulukon 3 kirjastojen tuki perustuu Elastic APM Java-agentin tuettujen teknologioiden dokumentaatioon. (Supported technologies 2020)

Palvelun pystytys ja kohdeympäristöjen instrumentointi

Tutkimuksen aikana kehitystiimin käyttöön pystytettiin palvelinkone, johon asennettiin yksi jokaista APM-palvelun sovellusta. APM-palvelun sisältävältä palvelimelta paljastettiin oletusarvoiset portit, joita APM Server ja Kibana käyttävät. Kehitystiimin kehitys- ja testiympäristön palvelinkoneille tuotiin Elastic APM Java-agent, joka kiinnitettiin palvelinkoneilla sijaitseville sovelluspalvelimille (ks. Kuvio 3).



Kuvio 3. Kehitys-, testi- ja APM-ympäristöjen arkkitehtuurikuva

Palvelimien JVM-argumentteihin lisättiin seuraavat viisi argumenttia.

- -javaagent:<polku APM-agenttiin>
- -Delastic.apm.service_name=<haluttu palvelun nimi>
- -Delastic.apm.environment=<ympäristön nimi (esim. kehitys, testi, integraatio)>
- -Delastic.apm.application_packages=<sovelluspaketit, joita seurataan, esimerkiksi org.example>
- -Delastic.apm.server_urls=<osoite APM Serveriin>

Kun Java-agentin konfigurointi palvelimen JVM-argumentteihin oli suoritettu loppuun, sovelluspalvelin voitiin käynnistää uudelleen, jolloin Java-agentin instrumentointi astui voimaan.

WAS-sovelluspalvelimen instrumentointi

Yksi palveluun kuuluva sovellus oli asennettu WAS eli IBM WebSphere Application Server-sovelluspalvelimelle. Kyseisestä palvelimesta todennettiin, ettei Java-agentin lisääminen palvelimelle ole yhteensopiva ilman syventävää perehtymistä instrumentoinnissa tapahtuvien virheiden selvittelyssä. Java-agentin lisääminen palvelimen JVM-argumentteihin ja palvelimen käynnistäminen aiheuttaa virheen, joka pysäyttää agentin ja palvelimen käynnistymisen. Yhteensopimattomuusongelma oli odotettu, sillä WAS-sovelluspalvelimen tukea ei ole listattu Elasticin tuettujen teknologioiden dokumentaatiossa. (Supported technologies 2020)

WAS-palvelimen yhteensopimattomuus johtaa siihen, ettei WAS-palvelimella olevaa sovellusta pystytä instrumentoimaan. Tällöin sovelluksen rajapinnoista ei pystytä muodostamaan transaktioita, jos sovellukseen kohdistuu HTTP-kutsuja. Jos muut instrumentoidut sovellukset kutsuvat WAS-palvelimella olevaa sovellusta, APM-palveluun muodostuu kutsun alkuperäiseen transaktioon jakso WAS-palvelimen sovellukseen tehdystä HTTP-kutsusta. Jaksosta voidaan päätellä osoite, johon kutsu on lähetetty, sekä kutsuun käytetty aika.

WLP-sovelluspalvelimen instrumentointi

APM-palvelun pystytyksen jälkeen testattiin IBM WebSphere Liberty Profile eli WLP-sovelluspalvelimen sovellusten instrumentoinnin toimivuus. Kaikki palvelimella olevat sovellukset pystyttiin instrumentoimaan onnistuneesti ja palvelin käynnistyi sovelluksineen ilman ongelmia. Tutkimuksen aikana selvisi, että taulukossa 3 listatut kirjastot pystyttiin instrumentoimaan onnistuneesti.

Palvelun yhdestä sovelluksesta löytyi ongelma, jossa sovelluksessa tapahtuvat virheet eivät tallennu APM-palveluun. Tämä ongelma johtui siitä, että

kaikki sovelluksessa tapahtuvat virheet ovat hallittuja, eikä virheitä palauteta sovelluksen Servlet-tasolta. Tämä ongelma pystyttiin kuitenkin ratkaisemaan lisäten virheenhallinnassa Elasticin tarjoamaa apm-agent-api kirjastoa sovelluksen kooditasolla. (ks. Kuvio 4, Kuvio 16).

```
<dependency>
    <groupId>co.elastic.apm</groupId>
    <artifactId>apm-agent-api</artifactId>
    <version>${elastic-apm.version}</version>
</dependency>
```

Kuvio 4. Public API-kirjaston lisääminen Maven Dependencynä.

```
Transaction transaction = ElasticApm.currentTransaction();
transaction.captureException(theException);
```

Kuvio 5. Transaktiossa tapahtuneen virheen poimiminen sovelluksen kooditasolla

Kuvion 4 kirjasto lisättiin lisäinstrumentoitavan sovelluksen riippuvuudeksi, jonka jälkeen kuvion 5 koodi lisättiin sovelluksen virhekäsittelymetodin sisälle. Koodin lisäämisen jälkeen sovelluksessa tapahtuneet virheet pystyttiin poimimaan transaktioon. Kyseisen kirjaston toimivuus testattiin myös poistamalla WLP-sovelluspalvelin instrumentoinnista ja jättämällä lisätty koodi sovelluksen kooditasolla. Testin tarkoituksena oli todentaa, että sovellus ei ajautuisi virhetilanteeseen virheenkäsittelyssä, kun Java-agentin instrumentointi ei ole käynnissä.

Testistä saatiin varmistus, että sovellus toimii normaalisti instrumentoinnin poiston jälkeen. Testistä voitiin vetää johtopäätös, että apm-agent-api kirjasto ja sen koodi voidaan säilyttää sovelluksen lähdekoodissa, ja ylemmät testausympäristöt sekä tuotantoympäristö toimivat tällöinkin normaalisti.

6.2 Miten kehitystiimi voi hyödyntää APM-palvelua kehitystyössään?

Kehitystiimin sovellushallinnan lähtötilanne

Tutkimuksen alussa kehitystiimi hyödynsi sovellushallintaa sovellus- ja käyttölokituksen muodossa. Kehitystiimin ylemmissä testiympäristöissä hyödynnettiin myös keskitettyä vasteaikojen monitorointia, sekä sovelluslokien virheilmoitusten monitorointia ja hallintaa.

Tunnistetuista virheilmoituksista tehdään hälytys kehitystiimille, jolloin tiimi voi reagoida kehitys-, testi- tai tuotantoympäristönsä ongelmiin. Hälytyksen virheviestistä ei välttämättä pystytä selvittämään virheen juurisyytä, jos virhe on tapahtunut sisäisten palvelujen sovelluksissa. Bisneslogiikkavirheissä kehitystiimi pystyy selvittämään palvelun käyttölokilta epäonnistuneen kutsun tiedot, sekä syyn virheen tapahtumiselle. Sisäisten sovellusvirheiden ilmetessä kehitystiimin täytyy selvittää virheen tiedot palvelinkoneella sijaitsevista sovelluslokeista. Sisäisten sovellusvirheiden selvittäminen voi olla työlästä, jos kehitysympäristössä on toteutettu sovellusten kahdennusta ja ruuhkanjakoa, sillä tällöin voidaan joutua jäljittämään sisäisen sovellusvirheen juurisyytä useamman palvelinkoneen sovelluslokeilta.

Kehitystiimin käyttölokisovellus tallentaa vasteaikoja, joita mitataan jokaisen sovelluksen kooditasolla. Kooditasolla uusien vasteaikojen luominen vaatii, että sovelluskehittäjä lisää sovelluksen kooditasolla lisää koodia, joka sisältää uuden vasteajan mittaamisen. Tarkkojen vasteaikojen lisääminen kooditasolle voi olla sovelluskehittäjälle haastavaa. Vasteajat tallentuvat käyttölokisovelluksen kautta tietokantaan, joista ne voidaan hakea visualisoitavaksi vasteaikojen monitorointipalveluun.

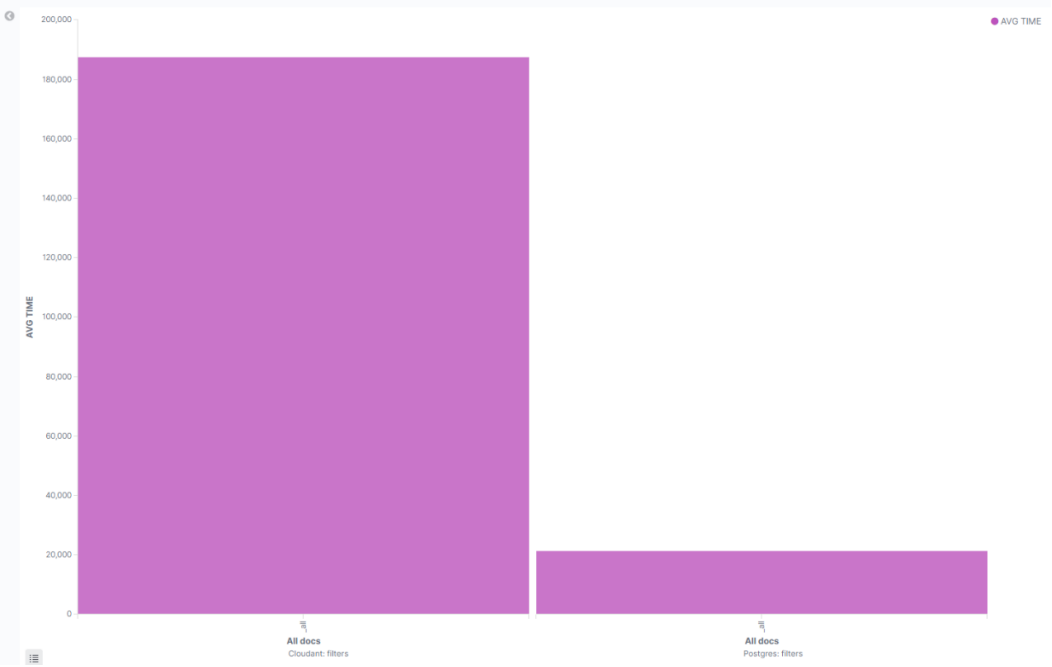
Uusien sovellusversioiden suorituskykyvertailu

APM-palvelua hyödyntäen kehitystiimi pystyi vertailemaan eri ympäristöjensä suorituskykyä. Uusia toiminnallisuuksia toteuttaessa kehitystiimi pystyy testaa-

maan palvelukokonaisuutensa erinäisillä automaatiotesteillä. Automaatiotestejä pystytään myös tällöin hyödyntämään ympäristöjen suorituskyvyn vertailuun.

Kehitystiimi pystyi ajamaan samat automaatiotestit kahteen eri ympäristöön ja tämän jälkeen vertailemaan ympäristöjen välisiä suorituskykyeroja.

Esimerkiksi projektissa kehitettiin uusi tietovarastototeutus, sillä aiempi toteutus oli todistettu liian hitaaksi suorituskykytesteissä. APM-palvelun avulla kehitystiimi pystyi visualisoimaan uuden tuotantokokonaisuuden keskimääräisen keston erot jo ennen suorituskykytesteihin vientiä, lisäämättä sovelluksen kooditasolle ylimääräistä koodia. (ks. Kuvio 6, Kuvio 7 ,Kuvio 8)



Kuvio 6. Tietokantahaun keskimääräisen vasteajan vertailu

Transactions

Name	Avg. duration	95th percentile	Trans. per minute	Impact ↓
PhrResourceServer#doGet	738 ms	2,380 ms	0.7 tpm	
DatabaseGetResourceController#getResourcePaged	1,286 ms	9,698 ms	0.4 tpm	
PhrResourceServer#doPost	50 ms	120 ms	1.3 tpm	
PhrFHIRLogController#save	20 ms	55 ms	2.8 tpm	
PhrResourceServer#doPut	43 ms	95 ms	0.3 tpm	
DatabaseGetResourceController#getResourceWithId	5 ms	6 ms	1.3 tpm	
DatabaseGetResourceController#getResource	14 ms	46 ms	0.5 tpm	
AuthLogController#save	6 ms	9 ms	0.9 tpm	
PhrResourceServer#doDelete	29 ms	61 ms	0.2 tpm	
PagingController#getResources	256 ms	647 ms	0.0 tpm	
DatabaseSaveResourceController#databaseSaveResource	16 ms	22 ms	0.3 tpm	
PagingController#retrieveResultList	222 ms	594 ms	0.0 tpm	
DatabaseDeleteResourceController#deleteResource	27 ms	49 ms	0.1 tpm	
DatabaseUpdateResourceController#databaseUpdateResource	30 ms	40 ms	0.1 tpm	
DatabaseSaveResourceController#databaseConditionalSaveResource	16 ms	21 ms	0.0 tpm	
DatabaseGetResourceController#getResourceVersion	5 ms	6 ms	0.0 tpm	
DatabaseLogResourceController#saveResourceLog	18 ms	18 ms	0.0 tpm	
DatabaseLogResourceController#saveAuthLog	24 ms	24 ms	0.0 tpm	
DatabaseLogResourceController#deleteAuthLog	16 ms	16 ms	0.0 tpm	
DatabaseLogResourceController#deleteResourceLog	14 ms	14 ms	0.0 tpm	

Kuvio 7. Vanhan tietovarastopalvelutoteutuksen transaktiot

Transactions

Name	Avg. duration	95th percentile	Trans. per minute	Impact ↓
PhrFHIRLogController#save	25 ms	72 ms	2.0 tpm	
PhrResourceServer#doPost	56 ms	134 ms	0.8 tpm	
PhrResourceServer#doGet	81 ms	213 ms	0.4 tpm	
DatabaseUpdateResourceController#databaseUpdateResource	62 ms	85 ms	0.1 tpm	
PhrResourceServer#doPut	39 ms	84 ms	0.2 tpm	
ObservationController#getResource	62 ms	145 ms	0.1 tpm	
DatabaseGetResourceController#getResourceWithId	15 ms	22 ms	0.5 tpm	
DatabaseGetResourceController#getResource	51 ms	82 ms	0.1 tpm	
DatabaseGetResourceController#getResourcePaged	109 ms	231 ms	0.1 tpm	
AuthLogController#save	11 ms	17 ms	0.4 tpm	
PhrResourceServer#doDelete	26 ms	42 ms	0.1 tpm	
DatabaseSaveResourceController#databaseSaveResource	46 ms	73 ms	0.1 tpm	
MedicationAdministrationController#getResource	45 ms	83 ms	0.0 tpm	
PatientController#getResourceWithId	6 ms	8 ms	0.3 tpm	
MedicationStatementController#getResource	62 ms	185 ms	0.0 tpm	
ObservationController#databaseConditionalSaveResource	148 ms	156 ms	0.0 tpm	
StructureDefinitionController#getResource	19 ms	36 ms	0.0 tpm	
ObservationController#getResourceWithId	6 ms	9 ms	0.1 tpm	
ValueSetController#getResource	10 ms	12 ms	0.1 tpm	
ObservationController#databaseSaveResource	11 ms	14 ms	0.1 tpm	
CarePlanController#getResource	30 ms	57 ms	0.0 tpm	
CodeSystemController#getResource	11 ms	16 ms	0.0 tpm	
ObservationController#databaseUpdateResource	14 ms	16 ms	0.0 tpm	
StructureDefinitionController#deleteResource	43 ms	285 ms	0.0 tpm	
MedicationAdministrationController#databaseSaveResource	12 ms	17 ms	0.0 tpm	

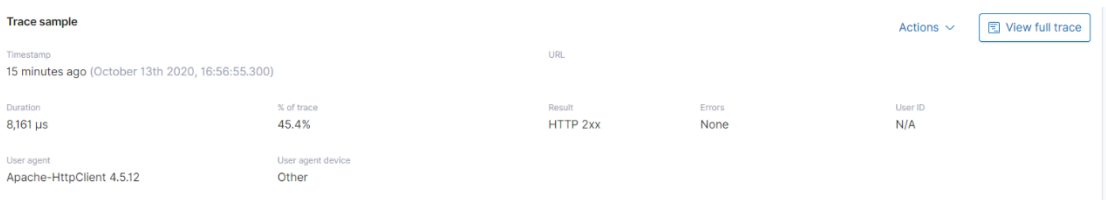
Kuvio 8. Uuden tietovarastopalvelutoteutuksen transaktiot

Kuviot 6, 7 ja 8 esittävät tilannetta, jossa sovelluksen tietovarastoratkaisua on muutettu. Kuviossa 6 verrataan tietovarastoratkaisujen keskimääräistä suoritusaikaa. Datan perusteella tehdystä ympäristöjen suorituskyvyn vertailusta voitiin päätellä, että uudistettu tietovarastototeutus toimi keskimäärin 9 kertaa nopeammin ilman kuormitusta kuin vanha toteutus. Kehitystiimi pystyi varmistumaan suorituskyvyn pullonkaulan poistumisesta tutkimalla APM-palvelusta molempien ympäristöjen tietovarastototeutuksien tietokantakutsujen keskimääräistä ajan kestoa. Lisäksi vertailussa huomattiin, että suuren prosenttipisteen vasteajat laskivat dramaattisesti uuden tietovarastopalvelutoteutuksen myötä. Prosenttipisteiden lasku näkyy kuvioissa 7 ja 8.

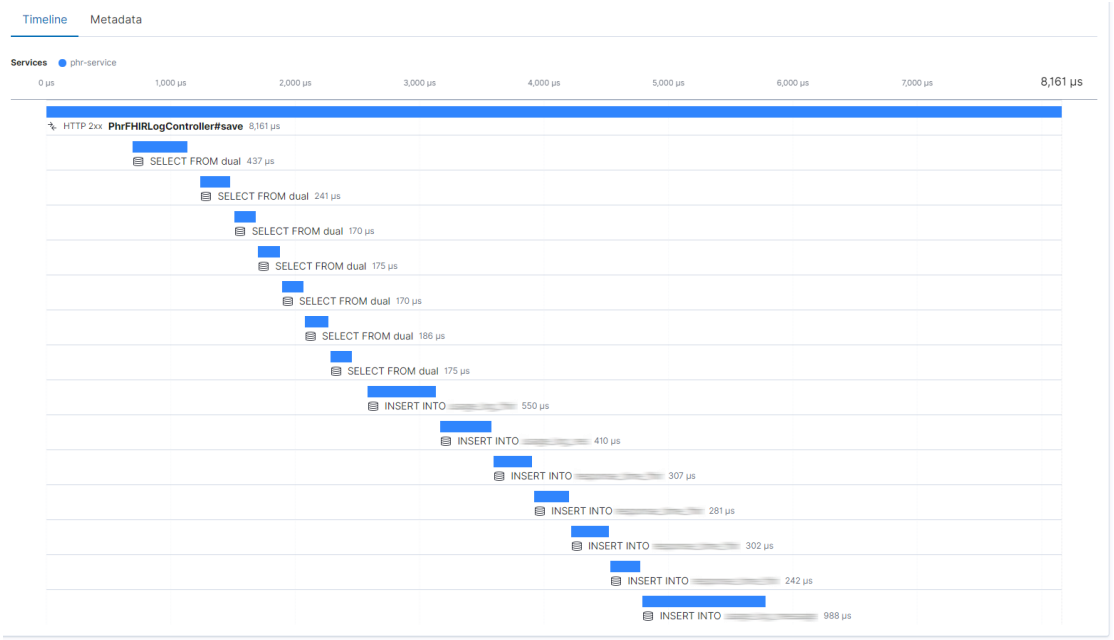
Suorituskyvyn pullonkaulojen selvittäminen

Elastic APM-palvelu toteuttaa APM:n näkymään jokaisen REST-rajapinnan muodostamaksi transaktiotietomalliksi (ks. Kuvio 7, Kuvio 8). Kuvioissa 7 ja 8 data on ryhmitelty transaktioittain. Kuviosta 7 ja 8 transaktioista voidaan päätellä millä rajapinnoilla on suurin vaikutus palveluun. APM:n transaktioiden näkymässä Impact eli vaikutus, kertoo kuinka paljon transaktio vaikuttaa sovelluksen. Kehitystiimi pystyy vaikutusta seuraten nostamaan kehityskehotuksia palvelunsa suorituskyvyn tehostamiseksi.

APM-palvelun transaktioiden näkymästä paljastui, että palvelun käyttölokisovelluksella on suurin vaikutus palvelun suorituskykyyn. Käyttölokisovelluksen hitaimpia transaktioita tarkastelemalla sovelluksessa huomattiin suorituskykyongelmia tietokantakyselyiden toteutuksessa. (ks. Kuvio 9, Kuvio 10).



Kuvio 9. Yksittäisen transaktion yleiset metatiedot



Kuvio 10. Esimerkki käyttölokisovelluksen transaktiosta

Tutkimalla kuvion 9 transaktion metatietoa tarkemmin pystyttiin toteamaan, että käyttölokisovelluksella on prosentuaalisesti suuri vaikutus palvelun kutsujen kestoon. Kuviossa 9 voidaan havaita, että käyttölokipalvelun transaktion kesto on 46,5 prosenttia koko kutsun elinkaaren kestosta. APM-palvelua hyödyntämällä kehitystiimi pystyi paikantamaan sovelluskokonaisuudessaan suorituskyvyn pullonkaulan, joka voidaan korjata optimoimalla koodin tai tietokantahakujen toteutusta. Koska käyttölokipalvelu on JPA-toteutus, sovelluksen lähdekoodista ei ole luettavissa yhtään SQL-lausetta. Kuvion 10 transaktion jaksoista selvisi, että käyttöloki palvelu lukee tietokannasta rivien tunnisteen seuraavan arvon eli sekvenssin arvon ennalta, eikä hyödynnä tietokannan ominaisuuksia tunnisteen generointiin. Kirjoittamalla SQL-lauseet itse, säästyttäisiin tällöin ylimääräisiltä hakulauseilta. Vaihtoehtoisesti nykyistä toteutusta voitaisiin myös optimoida, jolloin ylimääräiset hakulauseet poistuisivat.

Bugiraportointi ja keskitetyt sovellusvirheet

Kehitystiimi on ottanut käyttöön Elastic APM-palvelun selvittäessään kehitys- ja testiympäristön sovellusvirheitä. Virhetilanteeseen aiheutunut transaktio liitetään bugiraporttiin, josta voidaan selvittää virhetilanteen juurisyy.

Kehitystiimin sovelluksien uudet toiminnot ja kirjastojen päivitykset testataan kehitysympäristössä regressiotesteillä, joilla todennetaan, että kaikki vanhat toiminnallisuudet toimivat edelleen. Regressiotesteissä epäonnistuneet testit voidaan selvittää APM-palvelun avulla etsimällä virheitä virhekoosteista, tai etsimällä virhetilanteeseen päätyneen transaktion HTTP-pyynnön uniikilla tunnisteella. APM-palvelun avulla tiimi pystyi nopeuttamaan virhetilanteiden juurisyyn selvittelyä ja täten vapauttamaan resursseja uusien toimintojen toteuttamiseen.

Elastic APM-palvelun soveltuminen tuotantokäyttöön

Toimeksiantajan täytyy noudattaa tuotantoympäristöissään THL:n asettamaa A-luokkaan kuuluvien tietojärjestelmien määräystä. Vahvan roolipohjaisen tunnistautumisen ja audit-lokituksen puuttuminen eivät täytä määräyksen kriteerejä 5 ja 8. (Määräys A-luokkaan kuuluvien sosiaali- ja terveydenhuollon tietojärjestelmien olennaisista tietoturvavaatimuksista 2015, liite 1).

Elasticin ilmaislisenssi ei kata kaikkia toimeksiantajan tarvitsemia toiminnallisuuksia APM-palvelulle. Kehitysympäristöihin Elasticin ilmaislisenssiä voidaan hyödyntää, sillä kehitysympäristöissä käytössä on vain testidataa.

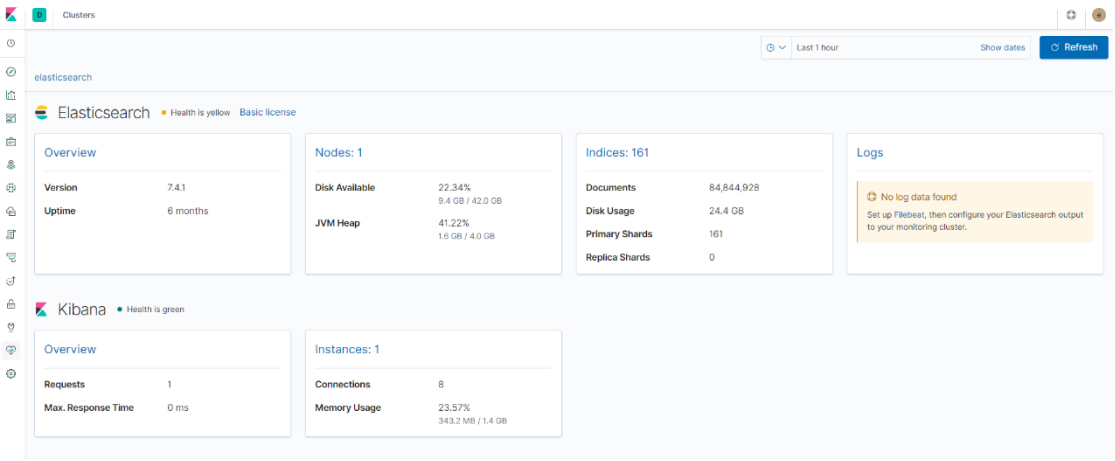
6.3 Miten Elastic APM-palvelu toteuttaa APM:n avaintoiminnot?

Luvussa 3.3 esiteltiin APM:n avaintoimintoja. Tässä luvussa käsitellään, mitkä APM:n avaintoiminnot Elastic APM-palvelu toteuttaa. Pystytetyn Elastic APM-palvelun avulla on mahdollista kokeilla ja arvioida aiemmin mainittujen avaintoimintojen toteutuksen laatua.

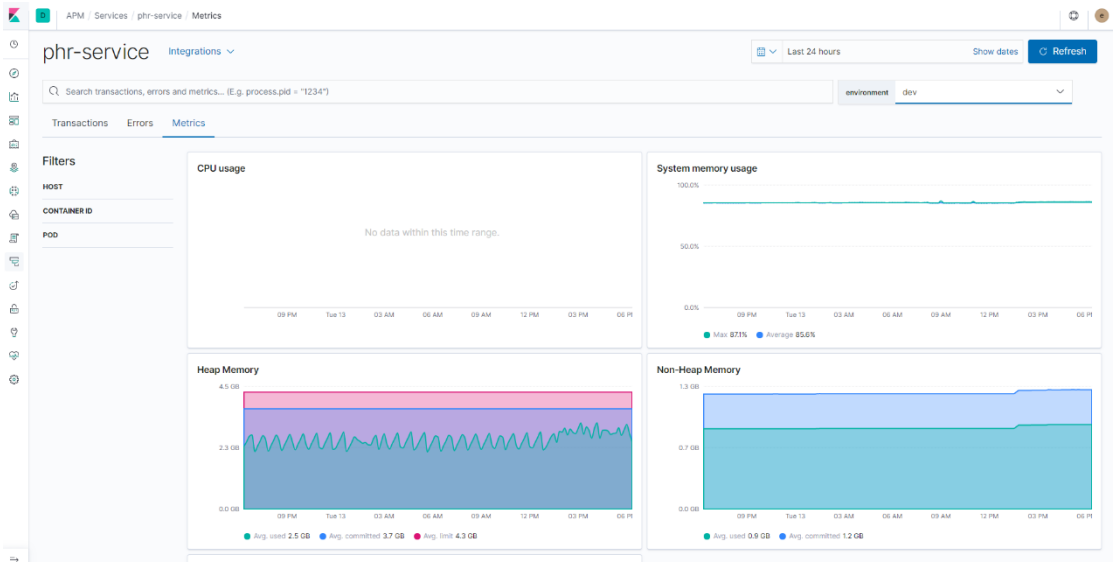
Analytiikka ja visualisointi

Elastic APM-palvelun visualisoinnin ja analysoinnin työkaluna toimii Kibana. Kibana mahdollistaa ympäristökohtaisen metriikoiden seuraamisen. Metriikat vaihtelevat ympäristöjen transaktioista ympäristöjen omiin järjestelmän tilasto-tietoihin.

Elastic APM-palvelu tarjoaa palvelun ”health checkin”, joka kertoo muun muassa kuinka paljon Elastic APM-palvelun komponentit vievät resursseja palvelinkoneelta tai -klusterista ja kuinka paljon kyseisiä resursseja on vielä avoinna. Tällaisia resursseja ovat muun muassa palvelun väli- ja levymuistinkäyttö (ks. Kuvio 11). Lisäksi APM-palvelu tarjoaa instrumentoitujen palveluiden metriikoiden monitoroinnin. Tässä tapauksessa palvelun metriikat ovat palvelinkoneen välimuistin käyttö sekä JVM-metriikat. (Ks. Kuvio 12)

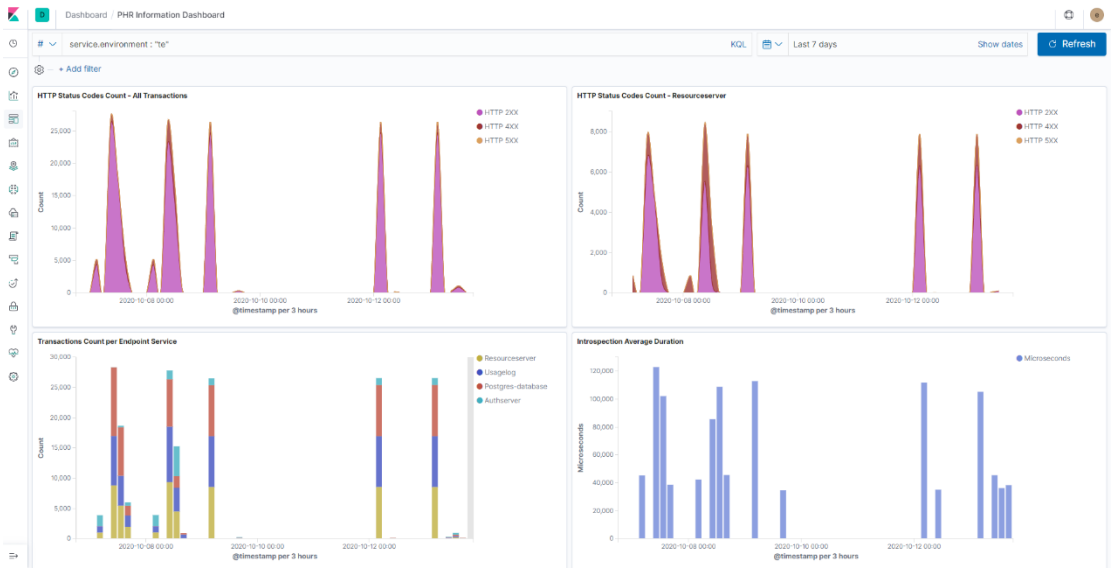


Kuvio 11. Elastic-APM-palvelun "health check"

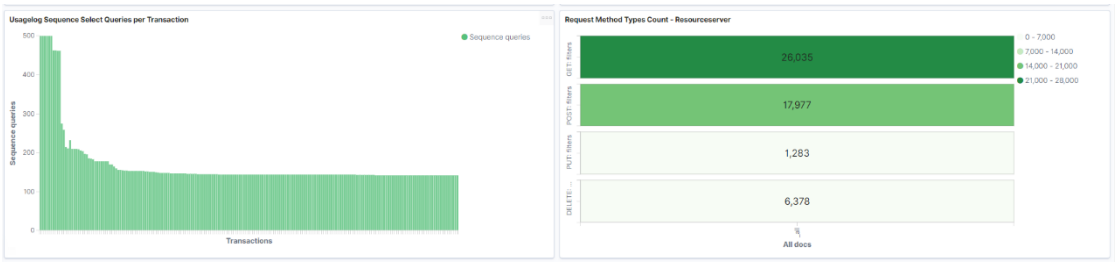


Kuvio 12. Kehitysympäristön metriikat päivän aikana

Kibanalla on mahdollista myös rakentaa dashboardeja, joilla voidaan havainnollistaa ja aggregoida kehitystiimiä kiinnostavaa dataa sovelluskokonaisuuksista. Kyseisiä tietoja voivat olla muun muassa transaktioiden eli palveluun tulevien kutsujen määrä sekä kutsujen keskimääräinen kesto. (ks. Kuvio 13, Kuvio 14)



Kuvio 13. Testiympäristön informaatiodashboard viimeisen viikon ajalta



Kuvio 14. Palvelun informaatiodashboardin jatko

Kuvion 13 piikit kuvastavat palvelun päivittäin ajettavia regressiotestejä. Kuvion 14 vasemmalla rakennetusta visualisoinnista voidaan havaita käyttölokipalvelun suurimmat sekvenssien hakulauseiden lukumäärät per palvelun transaktio.

Elastic APM-palvelun ilmaislisenssi tukee myös Machine Learning, eli koneoppimisalgoritmeja Elasticsearchissa sijaitsevien indeksien visualisoimiseksi. Datan visualisointia voidaan hyödyntää esimerkiksi yleisten APM-tietojen selvittämiseen. (Ks. Kuvio 15)

Kuvio 15. Koneoppimisalgoritmin hyödyntäminen APM-tietueiden indeksin visualisoimiseksi

Transaktioiden suorituskyvyn seuraaminen

Elastic APM-palvelun toteuttama transaktioiden suorituskyvyn seuraaminen toimii APM-ratkaisuille määritellyn toiminnallisuuden mukaisesti. Transaktioista pystytään hahmottamaan ongelmalliset pyynnöt, ja niihin keskittymällä pystytään hahmottamaan suorituskykyyn vaikuttavat syyt.

Tutkimuksessa löydettiin suorituskykyä seuraamalla yksi palvelun suorituskykyyn vaikuttava pullonkaula, sekä pystyttiin vertailemaan kahden eri tietovarastopalvelun suorituskyvyn eroa.

Palvelun kartoitus ja riippuvaisuuksien hahmotus

Elastic APM-palvelu ei tarjoa palvelukarttojen generoimista. Elastic siteeraa 13.10.2020 viitatussa dokumentaatiossa (Service maps 2020), että Elastic tukee nykyään palvelukarttojen generointia. Toiminto ei ole tällä hetkellä tuotantovaiheessa ja on altis muutoksille.

Riippuvaisuuksia voidaan hahmottaa ilman palvelukarttaa tutkimalla palvelun kokonaisia traceja, joista selviää, mitä riippuvaisuuksia sovellukset kutsuvat transaktion aikana.

Kooditason suorituskyvyn profilointi

Elastic APM-palvelun kooditason profilointi ei ole suunniteltu kooditason intensiiviseen profilointiin. Palvelun tietueet keräävä Java-agent seuraa ensisijaisesti vain tunnettuja kirjastoja ja niiden metodeja (Supported technologies 2020). Tällöin itse kirjoitettu koodi ei tule näkymään jaksona APM-palvelussa. Myöskään Javan suoritukseen liittyvää kooditasoa, esim. muistinsiivoamista ja JiT-kääntämistä, ei seurata automaattisesti. APM-palvelun tietueista voidaan kuitenkin hahmottaa riippuvaisuuksia ja niiden suorituskykyä. Kyseisiä riippuvaisuuksia ovat esimerkiksi HTTP-pyynnöt toiseen palveluun ja SQL-kyselyt.

Omatoimista profilointia voidaan kuitenkin suorittaa Elastic APM-palvelulle sovelluksen kooditasolla toimivalla elastic-agent-api-kirjastolla (ks. Kuvio 16). Kirjaston avulla pystyttiin muun muassa lisäämään sovellusten kooditasolla uusia jaksoja, jolloin metodin suoritukseen liittyvät tietueet tallentuvat APM-palveluun.

```
@CaptureSpan
private void validate(String requestText) {
```

Kuvio 16. Oman jakson luominen Annotation API:a hyödyntäen

Koska kyseessä on avoimen lähteen koodi, Elastic APM-palvelun Java-agenttiin on myös mahdollista lisätä oma instrumentointi Elasticin Java-agentin liitännäiskirjastoon, joka sisältää myös tuettujen teknologioiden instrumentointikomennot.

Yksityiskohtainen jäljitys yksittäisille transaktiolle tai web-pyynnöille

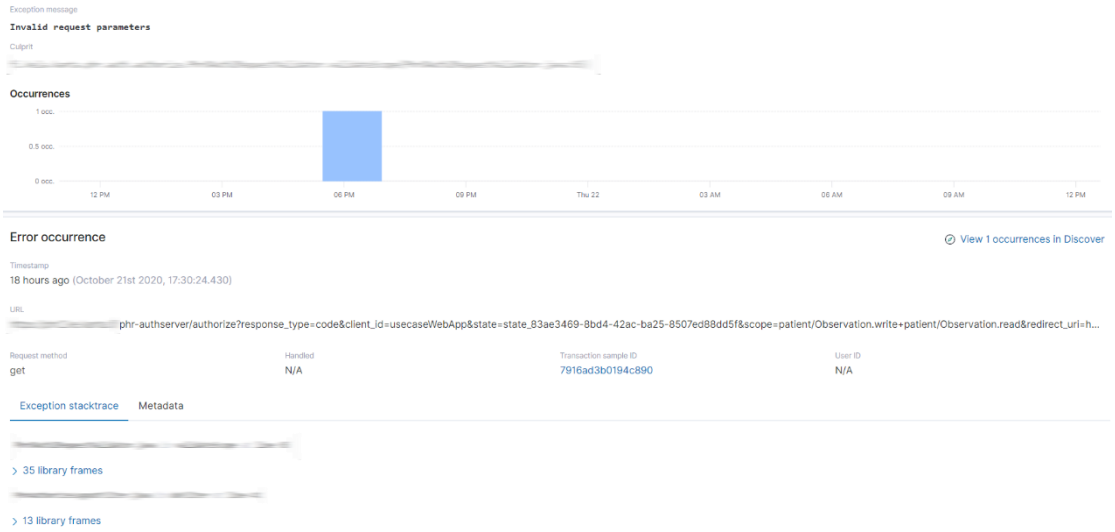
Yksityiskohtainen jäljitys eli tracing toteutuu Elastic APM-palvelun toiminnallisuutena. Transaktioiden ja jaksojen summasta muodostuu trace, joka sisältää koko transaktion elinkaareen liittyviä tietoja, sekä jaksoja, jotka paljastavat mitä riippuvaisuuksia kutsuttiin transaktion aikana. Tiedoista pystytään rakentamaan profiili, joka kertoo esimerkiksi, mistä kutsu saapui palveluun, tai kuka on kutsuun liittyvä käyttäjä.

Real User Monitoring (RUM)

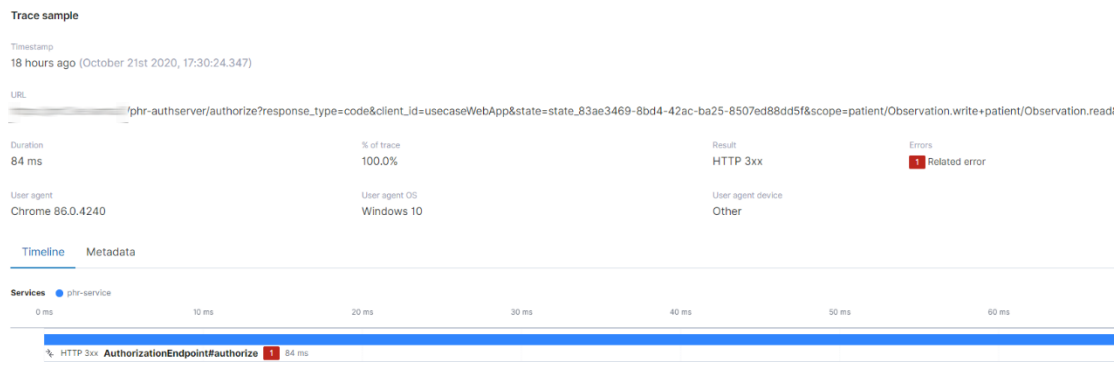
Elastic APM-palvelu toteuttaa Real User Monitoring-toiminnon, mutta kyseistä toimintoa ei hyödynnetty tutkimuksessa. Perusteena tähän on, ettei kehitystiimillä ollut tarvetta monitoroida käyttöliittymien suorituskykyä.

Keskitetty sovellusvirheiden seuranta ja hälytykset

Tiimin arvostetuin toiminto Elastic APM-palvelun toiminnoista oli keskitetty sovellusvirheiden hallinta. Virheiden tallentuminen Elasticsearchiin mahdollisti tiimille tavan selata ympäristössä tapahtuneita virheitä, joko listaamalla kaikki ympäristössä tapahtuneet virheet, tai hakemalla virhetilanteita erilaisilla tunnisteilla. Tunnisteita ovat muun muassa virhetilanteeseen ajautuneen pyynnön URL tai vastauksessa palautunut tunnisteotsikko.



Kuvio 17. Esimerkki virhetiedosta Kibanan yksittäisen virheen näkymässä



Kuvio 18. Esimerkki virheen sisältävästä transaktiosta

Kuvion 17 virhetiedosta voidaan selvittää tapahtuneen virheen stacktrace. Valitsemalla virhenäkymästä ”Transaction Sample Id”, näkymä ohjautuu kuvion 18 transaktion näkymään, jossa kyseinen sovellusvirhe on tapahtunut.

Ilmaislisenssi tarjoaa myös hälytystoiminnot, jotka on sidottu tapahtuneiden virheiden lukumäärään. Kyseisestä toiminnosta ei ole kehitysympäristöissä hyötyä, sillä jokapäiväiset regressiotestit aiheuttavat tarkoituksella virhetilanteita, jolloin hälytysten määrä tulisi asettaa hälytykselle erittäin suureksi, ennen kuin virhetilanteesta hälytettäisiin.

7 Johtopäätökset

Tässä luvussa käsitellään tutkimuksen tutkimustuloksista johdettuja johtopäätöksiä. Johtopäätökset on esitetty tutkimustulosten mukaisessa järjestyksessä. Johtopäätöksien tarkoituksena on käsitellä miten Elastic APM-palvelun pystytys onnistui, ja mitä mahdollista lisäarvoa palvelu on tuonut kehitystiimille.

Elastic APM-palvelun pystytys

Elastic APM-palvelun pystytys ja käyttöönotto kehitystiimin käyttöön toteutettiin onnistuneesti lukuun ottamatta WAS-sovelluspalvelimen yhteensopivuusongelmaa. Kehitystiimi on myöhemmin poistanut WAS-sovelluspalvelimen käytöstä, ja kaikkien palvelun sovellusten instrumentointi onnistuu nyt APM-palvelulla.

APM-palvelun resurssit yhdellä palvelinkoneella todennettiin riittäväksi kehitystiimin käyttöön, eikä APM-palvelun komponentteja tarvittu tällöin sijoittaa useampaan palvelinkoneeseen. APM-palvelun tietomäärän kasvaessa useampien ympäristöjen instrumentoinnin takia palvelu tulisi sijoittaa useammalle palvelinkoneelle, sekä sovellukset olisi syytä kahdentaa tietoliikenteen ruuhkan tasoittamiseksi.

Elastic APM-palvelusta todennettiin myös, ettei palvelun ilmaislisenssi tarjoamat toiminnallisuudet kata kaikkia kriteerejä, joita toimeksiantajan tulee noudattaa tuotantoympäristöissään. Tuotantoympäristöön soveltuvuuden varmistamiseksi tulisi tutkimuksessa hyödyntää Elasticin maksullista lisenssiä.

Miten kehitystiimi pystyy hyödyntämään Elastic APM-palvelua kehitystyössään?

Kehitystiimin alkupisteessä vasteaikojen ja mahdollisten suorituskyvyn pullonkaulojen selvitys vaati sovelluskehittäjiltä manuaalista vasteaikojen lisäämistä sovellusten kooditasolle. Lisäksi sisäisten virhetilanteiden tapahtuessa kehitystiimin täytyi selvittää virhetilanteen juurisyy palvelinkoneilla sijaitsevilta sovelluslokeilta.

Kehitystiimi on integroinut työssään APM-palvelun hyödyntämisen kehitys- ja testiympäristössään ja on pystynyt nopeuttamaan kehitystyötänsä hyödyntämällä APM-palvelun toiminnallisuuksia. Kehitystiimi on pystynyt paikantamaan sovelluskokonaisuuksissaan suorituskykyongelmia sekä löytänyt kyseisiin ongelmiin myös juurisyyn. Kehitystiimi on todennut, että APM-palvelu on ollut hyödyllinen työkalu sovelluskokonaisuuden kehittämisessä ja ylläpidossa.

Ympäristörajoitusten vuoksi ei pystytty instrumentoimaan kuin alimmat kehitys- ja testiympäristöt. Tämän takia APM-palvelun hyödyt koostuivat suurimmaksi osaksi kehitystyön edistämiseen kohdistuvista sovellusten suorituskyvyn seurannasta, virhetilanteiden selvittämisestä, sekä sovelluskokonaisuuksien suorituskyvyn vertailusta ympäristöittäin. Jos APM-palvelu otettaisiin käyttöön ylemmissä ympäristöissä, monitorillisten hyötyjen havainnointi olisi mahdollista.

Tutkimuksen kenttätyön jälkeen suoritettiin kysely, jossa haastateltiin kehitystiimiä APM-palvelun hyödyllisyydestä. Kyselyn kysymykset löytyvät liitteestä 7. Kyselyn tuloksista voitiin tulkita, että APM-palvelusta oli hyötyä kehitystyössä. Jokainen kyselyyn osallistunut vastasi myönteisesti APM-palvelun hyödyllisyyteen. Kehitystiimi nimesi APM-palvelun suurimmaksi hyödyksi virhetilanteiden

selvittelyn mahdollistavat toiminnallisuudet. Kehitystiimi suosittelisi APM-palvelun käyttöönottoa muillekin kehitystiimeille.

Miten Elastic APM-palvelu toteuttaa APM:n avaintoiminnot?

Tutkimustuloksista voidaan todeta, että Elastic APM-palvelu toteuttaa lähes kaikki APM:lle asetetut avaintoiminnot. Tutkimustulokset on johdettu tutkimalla Elasticin laatimaa dokumentaatiota ja todentamalla kenttätyön aikana avaintoimintojen toteutumisen validiteetti.

Tutkimuksen kenttätyön aikana on todennettu, että APM-palvelun tarjoamat toiminnallisuudet ovat luotettavia ja Elasticin dokumentaation lupauksen mukaisia. Avaintoiminnoista suurimmasta osasta oli kehitystiimille välitöntä hyötyä.

8 Pohdinta

Tutkimuksen tavoitteena oli tutkia Elastic APM-palvelua ja kehitystiimin saamia välittömiä hyötyjä APM-palvelun käyttöönoton jälkeen. Lisäksi tavoitteena oli todentaa Elastic APM-palvelun toiminnallisuus APM-avaintoimintoihin verraten.

Tavoitteet saavutettiin suhteellisen hyvin. Kehitystiimi pystyi tehostamaan toimintaansa APM-palvelun käyttöönoton jälkeen. Huomattavia hyötyjä APM-palvelussa olivat uusien suorituskyvyn pullonkaulojen paikantaminen, sisäisten virhetilanteiden selvittäminen bugiraporteista, sekä uusien toiminnallisuuksien suorituskyvyn vertailu. Tutkimuksen tuloksia pystyttäisiin parantamaan suorittamalla laajempi kenttätyö, jossa APM-palvelu olisi otettu käyttöön ylemmissä testiympäristöissä. Alemmissa kehitys- ja testiympäristöissä APM-palvelu avustaa kehitystiimiä sovelluskokonaisuuksiensa suorituskyvyn tehostamiseksi.

Elastic APM-palvelun toteuttamien avaintoimintojen validiteetti pystyttiin todentamaan varsin hyvin. Tutkimustulokset ovat tyydyttävät, sillä Elastic APM-palvelu toteuttaa lähes kaikki yleisesti tunnustetut avaintoiminnot ilman maksullista lisenssiä.

Toimeksiantajan on mahdollista hyödyntää tutkimuksen aikana syntynyttä APM-palvelua muissakin kehitystiimien palveluissa, sekä käyttää APM-palvelua vertailun pisteenä APM-hankkeessaan. Vaatimukset APM-palvelun käyttöönotolle ovat, että kohdepalvelun käyttämät yleiset teknologiat ovat tuettuja Elastic APM Java-agentissa. Muiden kehitystiimien sovelluskokonaisuudet voidaan tarvittaessa liittää samaan APM-palveluun, kunhan APM-palvelun resursseja skaalataan tarpeiden mukaan suuremmiksi. Roolipohjaisen pääsynhallinnan ja metriikoiden ymmärtämisen helpottamiseksi olisi hyödyllisempää pystyttää jokaiselle palvelulle oma APM-palvelu tai hyödyntää maksullisen lisenssin tuomia pääsynhallintatoimintoja.

Tutkimuksen luotettavuus

APM on itsessään vielä hyvin uusi käsite sovellushallinnan työkalusalkussa. APM:stä ei löydy kattavasti kirjallisia lähteitä ja tämän vuoksi on jouduttu hyödyntämään enemmän verkosta löytyviä lähteitä.

APM:n avaintoimintojen selvittämiseksi tutkimuksessa monen APM-ratkaisun tarjoajan verkkoartikkeleista kerättiin ylös yleisesti tunnistetut eli usein esiintyvät avaintoiminnot.

Tutkimuksen luotettavuutta edistää tutkimuksessa suoritettu kenttätyö, jossa Elastic APM-palvelu on pystytetty ja otettu käyttöön oikeassa aktiivisessa kehitystiimissä.

Jatkokehitys ja uudet tutkimusaihiot

Tutkimuksen rajoitteet ja rajoitukset vaikuttivat tutkimusaihion alueisiin. Tutkimusta voitaisiin viedä jatkokehitykseen tai APM:stä voitaisiin nostaa uusia tutkimusaihioita tutkittavaksi.

Tutkimus toteutettiin vain kahdessa alimmassa ympäristössä – testi- ja kehitysympäristössä. Tällöin APM-palvelun hyödyt rajautuvat kehitystiimin välittömiin hyötyihin palvelukokonaisuuden kehittämisessä, mutta APM-palvelun monitoroinnillisia ja ylläpidollisia hyötyjen validiteettia ei pystytä todentamaan tehokkaasti. APM:n toimintaa voitaisiin tutkia myös ylemmissä testiympäristöissä, jonka sovellusarkkitehtuuri on laajennettu useammalle palvelinkoneelle. Ylemmissä ympäristöissä tehdään myös hyväksymis-, suorituskyky- ja integraatiotestaamista, jotka ovat kehitystiimin ulkopuolella toteutettua testaamista. APM-palvelun käyttöönotto ylemmissä ympäristöissä mahdollistaisi uusien käyttötapausten tunnistamisen, josta saataisiin lisäarvoa APM-palvelun käyttöön. Lisäksi Elasticin maksullisen lisenssin toiminnallisuuksien todentaminen toisi tutkimukselle lisäarvoa. Tutkimusta olisi myös mahdollista laajentaa tutkimalla ja vertaamalla useamman APM-ratkaisujen toimintojen laatua toisiinsa.

Tutkimuksessa ei tutkittu liiketoiminnallisen yksikön saamia hyötyjä APM-palvelusta. APM-ratkaisujen tarjoajat listaavat useita liiketoiminnallisia hyötyjä, joita APM-palvelu tuo organisaatiolle. Kyseisten liiketoiminnallisten hyötyjen validiteetista voisi suorittaa erillisen tutkimuksen. Myös sovelluksen elinkaaren aikaisen käytön roolien kuten esimerkiksi ylläpidollisten roolien näkökulmasta voitaisiin tutkia APM-palvelun käyttöä.

Lähteet

Altvater, A. 2015. What is APM? Overview, Common Terms, and 10 Critical APM Features. Artikkeli Stackifyn sivustolla 14.9.2015. Viitattu 3.9.2020. https://stackify.com/what-is-apm/

Altvater, A. 2020.What is Real User Monitoring? How It Works, Examples, Best Practices, and More. Artikkeli Stackifyn sivustolla 29.1.2020. Viitattu 5.9.2020. https://stackify.com/what-is-real-user-monitoring/

Application Performance Management. 2019. Artikkeli IBM:n nettisivuilla 10.6.2019. Viitattu 3.9.2020. https://www.ibm.com/cloud/learn/application-performance-management

APM Java Agent Reference. 2020. Tekninen spesifikaatio. Viitattu 3.6.2020 https://www.elastic.co/guide/en/apm/agent/java/current/index.html

APM Overview. 2020. Tekninen spesifikaatio. Viitattu 3.6.2020 https://www.elastic.co/guide/en/apm/get-started/current/index.html

APM Server Reference. 2020. Tekninen spesifikaatio. Viitattu 3.6.2020. https://www.elastic.co/guide/en/apm/server/current/index.html

Chhajed, S. 2015. Learning ELK Stack. Birmingham, Yhdistynyt kuningaskunta. Packt Publishing Limited.

Elasticsearch Reference. 2020. Tekninen spesifikaation. Viitattu 3.6.2020. https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

Introduction to the Java Persistence API. N.d. Tekninen spesifikaatio. Viitattu 26.10.2020. https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html

IT-palvelut. 2020. Artikkeli Kelan nettisivuilla. Viitattu 22.9.2020. https://www.kela.fi/it-palvelut1

JAR File Specification. N.d. Tekninen spesifikaatio. Viitattu 21.10.2020. https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jar.html

Java EE Servers. N.d. Tekninen spesifikaatio. Viitattu 21.10.2020. https://docs.oracle.com/javaee/6/firstcup/doc/gcrkq.html

Java Instrumentation. 2014. Nettiartikkeli Javapapersin sivuilla. Viitattu 14.9.2020. http://javapapers.com/core-java/java-instrumentation/

Kananen J. 2012. Kehittämistutkimus opinnäytetyönä. Jyväskylän ammattikorkeakoulu.

Kempf, T. & Karuri, K. & Gao, L. 2008. Software instrumentation. Wiley Encyclopedia of Computer Science and Engineering. Viitattu 24.8.2020. https://doi.org/10.1002/9780470050118.ecse386

Keskeiset käsitteet. N.d. Nettiartikkeli Findatan sivuilla. Viitattu 21.10.2020. https://www.findata.fi/tietoa-meista/keskeiset-kasitteet

Kibana Guide. 2020. Tekninen spesifikaatio. Viitattu 3.6.2020. https://www.elastic.co/guide/en/kibana/current/index.html

Korhonen, P. 2018. Pieni API-sanakirja. Nettiartikkeli CGI:n sivuilla. Viitattu 21.10.2020. https://www.cgi.fi/fi/blogi/pieni-api-sanakirja

Määräys A-luokkaan kuuluvien sosiaali- ja terveydenhuollon tietojärjestelmien olennaisista tietoturvavaatimuksista. 2015. THL Määräys 1/2015. THL/1305/4.09.00/2014. Viitattu 9.10.2020. https://thl.fi/documents/920442/1449818/Allekirjoitettu_THL_M%c3%a4%c3%a4r%c3%a4ys_1_2015_Tietoturvavaatimukset_20150130-b.pdf/bcbc0d70-1749-488d-8e09-54f1ebd46484

Niemeyer, P. & Leuck, D. 2013. Learning Java. Sebastopol, California: O'Reilly Media.

Oaks, S. 2014. Java Performance: The Definitive Guide. Sebastopol, California: O'Reilly Media.

Our Story. N.d. Nettiartikkeli Elasticin sivuilla. Viitattu 22.10.2020. https://www.elastic.co/about/history-of-elasticsearch

Package java.lang.instrument. N.d. Tekninen spesifikaatio. Viitattu 3.6.2020. https://docs.oracle.com/javase/8/docs/api/java/lang/instrument/package-summary.html

Richardson, C. N.d. Microservices Architecture. Nettiartikkeli. Viitattu 21.10.2020. https://microservices.io/

Sathiyakugan, B. 2020. Understanding Java Agents. Nettiartikkeli DZonen sivuilla. Viitattu 22.10.2020. https://dzone.com/articles/java-agent-1

Schults, C. 2019. What Are Java Agents and How to Profile With Them. Viitattu 24.8.2020. https://stackify.com/what-are-java-agents-and-how-to-profile-with-them/

Service maps. 2020. Tekninen spesifikaatio. Viitattu 13.10.2020. https://www.elastic.co/guide/en/kibana/current/service-maps.html

Sturm, R. & Pollard, C. & Craig, J. 2017. Application Performance Management (APM) in the digital enterprise: managing applications for cloud, mobile, IoT and eBusiness. Cambridge, MA: Morgan Kaufmann Publishing.

Supported technologies. 2020. Tekninen spesifikaatio. Viitattu 3.6.2020. https://www.elastic.co/guide/en/apm/agent/java/current/supported-technologies-details.html

The __inline Keyword for Inline Functions. N.d. Tekninen spesifikaatio. Viitattu 21.10.2020. https://support.sas.com/documentation/onlinedoc/ccompiler/doc750/html/clug/zcoptinl.htm

Watts, S. 2018. What is Application Performance Management? APM Explained. Viitattu 3.9.2020. https://itsm.tools/what-is-application-performance-management-apm-explained/

What is HTTP? N.d. Nettiartikkeli W3Schoolsin sivuilla. Viitattu 21.10.2020. https://www.w3schools.com/whatis/whatis_http.asp

Liitteet

Liite 1. Java-agentin ja sovelluksen maven pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>oma.koulu.testi</groupId>
        <artifactId>instrumentation-root</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </parent>
    <artifactId>instrumentation-agent</artifactId>
    <dependencies>
        <dependency>
            <groupId>javassist</groupId>
            <artifactId>javassist</artifactId>
            <version>3.12.1.GA</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.2.3</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <shadedArtifactAttached>true</shadedArtifactAttached>
                            <transformers>
                                <transformer
                                    implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                                    <mainClass>oma.koulu.testi.instrumentation.main.TestApplication</mainClass>
                                    <manifestEntries>
                                        <Premain-Class>oma.koulu.testi.instrumentation.agent.TestAgent</Premain-Class>
                                        <Can-Redefine-Classes>true</Can-Redefine-Classes>
                                        <Can-Retransform-Classes>true</Can-Retransform-Classes>
                                        <Can-Set-Native-Method-Prefix>true</Can-Set-Native-Method-Prefix>
                                    </manifestEntries>
                                </transformer>
                            </transformers>
                            <finalName>instrumentation-agent</finalName>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

Liite 2. Sovelluksen Main-luokka

```java
package oma.koulu.testi.instrumentation.main;

import oma.koulu.testi.instrumentation.model.TestClass;

public class TestApplication {
    public static void main(String args[]) throws InterruptedException {
        TestClass test = new TestClass();
        test.runClass();
    }
}
```

Liite 3. Agent-luokka, joka sisältää premain-metodin

```java
package oma.koulu.testi.instrumentation.agent;

import java.lang.instrument.Instrumentation;

import oma.koulu.testi.instrumentation.transformer.CodeExecutionDurationTransformer;

public class TestAgent {

    public static void premain(String agentArgs, Instrumentation inst) {

        inst.addTransformer(new CodeExecutionDurationTransformer());
    }
}
```

Liite 4. Instrumentoitava Java-luokka

```java
package oma.koulu.testi.instrumentation.model;

public class TestClass {
    public void runClass() throws InterruptedException {
        System.out.println("TestClass is about to run........");
        Thread.sleep(2000L);
    }
}
```

Liite 5. Transformer- eli muuntajaluokka

```java
package oma.koulu.testi.instrumentation.transformer;

import java.io.ByteArrayInputStream;
import java.lang.instrument.ClassFileTransformer;
import java.lang.instrument.IllegalClassFormatException;
import java.security.ProtectionDomain;

import javassist.ClassPool;
import javassist.CtClass;
import javassist.CtMethod;

public class CodeExecutionDurationTransformer implements ClassFileTransformer{

    public byte[] transform(ClassLoader loader, String className, Class<?> classBeingRedefined,
            ProtectionDomain protectionDomain, byte[] classfileBuffer) throws IllegalClassFormatException {


        byte[] byteCode = classfileBuffer;

        //check for TestClass-className
        if (className.equals("oma/koulu/testi/instrumentation/model/TestClass")) {
            System.out.println("Instrumenting......");
            try {
                ClassPool classPool = ClassPool.getDefault();
                CtClass ctClass = classPool.makeClass(new ByteArrayInputStream(
                        classfileBuffer));
                CtMethod[] methods = ctClass.getDeclaredMethods();
                for (CtMethod method : methods) {
                    method.addLocalVariable("startTime", CtClass.longType);
                    method.insertBefore("startTime = System.nanoTime();");
                    method.insertAfter("System.out.println(\"Execution Duration "
                            + "(nano sec): \"+ (System.nanoTime() - startTime) );");
                }
                byteCode = ctClass.toBytecode();
                ctClass.detach();
                System.out.println("Instrumentation complete.");
            } catch (Throwable ex) {
                System.out.println("Exception: " + ex);
                ex.printStackTrace();
            }
        }
        return byteCode;
    }

}
```

Liite 6. Java-ohjelman instrumentointi komentorivillä

```
D:\>java -javaagent:./instrumentation-agent.jar -jar instrumentation-agent.jar
Executing premain...
Executing main...
Instrumenting......
Instrumentation complete.
TestClass is about to run........
Execution Duration (nano sec): 2004191900
```

Liite 7. Haastattelun kysymykset kehitystiimille

Onko APM-palvelu ollut hyödyllinen käyttöönoton jälkeen? Jos on, mihin olet käyttänyt APM-palvelua ja mistä toiminnoista oli hyötyä?

Näetkö APM-palvelun olevan hyödyllinen ylemmissä testausympäristöissä? Jos on, mitä hyötyjä APM-palvelu toisi?

Suosittelisitko APM-palvelun käyttöönottoa muille kehitystiimeille?