jamk.fi

Verkkosovellusten monitorointi

Perttu Savolainen

Opinnäytetyö
Toukokuu 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), mediatekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Savolainen, Perttu	Opinnäytetyö, AMK	22.5.2018
	Sivumäärä 75	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Verkkosovellusten monitorointi

Tutkinto-ohjelma
Mediatekniikan koulutusohjelma

Työn ohjaaja(t)
Pasi Manninen

Toimeksiantaja(t)
Nordcloud Solutions

Tiivistelmä

Opinnäytetyön toimeksiantajana toimi Nordcloud Solutions. Opinnäytetyön tavoitteena oli toteuttaa soveltuvuustutkimus sovellustason monitoroinnista pilvipalveluympäristöissä.

Tutkimuksen tavoitteena oli selvittää, onko systemaattinen sovellusten monitorointi ylläpitotiimin sovellusten kohdalla mahdollista toteuttaa ja millaisilla työkaluilla se onnistuisi.

Tutkimuksessa kartoitettiin ja vertailtiin eri monitorointityökaluja maksullisista palvelukokonaisuuksista vapaan lähdekoodin työkaluihin. Lisäksi haluttiin määrittää myös sovellusten monitoroinnin konfiguraatiovaatimukset, joita voitaisiin käyttää hyödyksi uusien ylläpitoon tulevien sovellusten monitoroinnin konfiguroinnissa. Määriteltyjen vaatimusten mukainen monitorointi toteutettiin yhteen ylläpidossa olevista sovelluksista, jolla validoitiin tutkimuksen tulokset.

Tuloksena saatiin kokonaisvaltainen käsitys monitoroinnista ja sen haasteista sekä mahdollisuuksista pilvipalveluympäristöissä. Määriteltyjä konfiguraatiovaatimuksia voitaisiin käyttää uuden sovelluksen monitoroinnin konfiguroimisessa ylläpidon alkaessa ja monitorointisovellusten vaatimusmäärittelyä sekä toteutettua vertailua voitaisiin käyttää tulevaisuudessa uusien monitorointisovellusten arvioinnissa. Edellä mainitut tulokset ovat niin sanottuja eläviä dokumentteja, joita voidaan päivittää tarpeen tullen.

Tulosten valossa voitiin todeta, että systemaattinen monitorointi on mahdollista toteuttaa ylläpidossa oleviin sovelluksiin.

Avainsanat (asiasanat)
Sovellusten monitorointi, pilvipalvelut, AWS, modernit verkkosovellukset

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Savolainen, Perttu	Bachelor's thesis	22.5.2018
		Language of publication: Finnish
	Number of pages: 75	Permission for web publication: x

Title of publication
Monitoring for web applications

Degree programme
Media Engineering

Supervisor(s)
Manninen, Pasi

Assigned by
Nordcloud Solutions

Abstract

The bachelor's thesis was assigned by Nordcloud Solutions. The goal was to conduct a feasibility research on application monitoring in cloud environments.

The goal of the research was to examine, whether application monitoring was possible to implement on applications, which were maintained by the assignor's application management team.

The research included defining and comparing different monitoring tools from full services to open source tools. On top of that, there was a need to define the baseline for application monitoring. The defined baseline would be used when the new application monitoring was to be set up. The research was validated by using the newly defined monitoring baseline in one of the currently maintained applications.

The results of the thesis were overall knowledge of monitoring in general and understanding its challenges and possibilities in different cloud environments. The defined baseline could be used when configuring monitoring for a new application, while the analysis and comparison of the monitoring tool requirements itself could be used during the evaluation of new monitoring tools in the future. The previously mentioned results are so-called living documents, which can be updated when necessary.

In the light of the obtained results, it was only natural to point out that it is possible to implement application monitoring in maintained applications.

Keywords/tags (subjects)
Application monitoring, cloud services, AWS, modern web applications

Miscellaneous

Sisältö

Kuviot

Taulukot

Sanasto

API	Ohjelmointirajapinta (Application Programming Interface)
APM	Kontekstista riippuen joko sovellussuorituskyvyn ylläpito (Application Performance Management) tai sovellussuorituskyvyn monitorointi (Application Performance Monitoring)
AWS	Amazon Web Services
Backend	Sovelluksen tiedonhallintataso, palvelintaso
Baseline	Pohjakonfiguraatio
CD	Jatkuva toimitus (Continuous Delivery)
CI	Jatkuva integraatio (Continuous Integration)
CW	AWS:n tarjoama monitorointipalvelu (CloudWatch)
Devops	Kehitys- ja tuotantotoimintojen edistämiseen pyrkivä ketterä toimintamalli
FaaS	Funktio palveluna (Function as a Service)
Frontend	Sovelluksen esitystaso, käyttäjälle näkyvä osa
GDPR	EU:n laajuinen tietosuoja-asetus (General Data Protection Regulation)
IaaS	Infrastruktuuri palveluna (Infrastructure as a Service)
ITIL	Kokoelma käytäntöjä IT-palveluiden hallintaan ja johtamiseen (Information Technology Infrastructure Library)
KPI	Suorituskykyindikaattori (Key Performance Indicator)
Lambda	AWS:n Serverless computing -palvelu, jossa palveluntarjoaja dynaamisesti hallitsee ja jakaa tietokoneresursseja.
MFA	Moniosainen todentamismenetelmä (Multi-factor Authentication)
PaaS	Alusta palveluna (Platform as a Service)
RDS	AWS:n tarjoama relaatiotietokantapalvelu (Relational Database Service)
S3	AWS:n tarjoama tietovarasto (Simple Storage Service)
SaaS	Ohjelmisto palveluna (Software as a Service)
SLA	Palvelutasosopimus (Service Level Agreement)
SLO	Palvelutasosopimuksen tavoite (Service Level Objective)
SNS	AWS:n ilmoituspalvelu (Simple Notification Service)
SRE	Ohjelmistokehitystä ja IT-operaatioita yhdistävä tieteenhaara (Site Reliability Engineering)

1 Lähtökohdat

1.1 Tilaaja

Opinnäytetyön tilaajana toimi Nordcloud Solutions (ent. SC5 Online Oy). Tilaaja on pilvinatiivien sovellusten kehittämiseen ja suunnitteluun erikoistunut yritys. Toimipisteitä yrityksellä on Jyväskylässä ja Helsingissä. (Leader in native cloud applications 2017; Nordcloudista ja SC5:stä Euroopan -- 2017.)

Sisaryhtiöt SC5 Online Oy ja Nordcloud Oy ilmoittivat syyskuussa 2017 yhdistyvänsä, minkä myötä SC5:n nimi muutettiin vuoden 2018 alusta alkaen muotoon Nordcloud Solutions. Yhdistymisen jälkeen yrityksillä on n. 250 pilviasiantuntijaa ympäri Eurooppaa, jolla he tarjoavat entistä monipuolisempia palveluja asiakkailleen. (Nordcloudista ja SC5:stä Euroopan -- 2017.)

”Nordcloud on Euroopan johtavia pilvipalveluiden asiantuntijoita. Tarjoamme yritysasiakkaillemme pilvipalveluiden suunnittelu-, migraatio-, automaatio- ja managed services -palveluita sekä pilvikapasiteettia ja -koulutusta.” (Nordcloudista ja SC5:stä Euroopan -- 2017.)

1.2 Tavoitteet

Opinnäytetyön tavoitteena oli toteuttaa soveltuvuustutkimus sovellustason monitoroinnista. Tutkimuksen tavoitteena oli selvittää, onko systemaattinen sovellusten monitorointi ylläpitotiimin sovellusten kohdalla mahdollista toteuttaa ja millaisilla työkaluilla se onnistuisi. Tutkimuksessa kartoitettiin ja vertailtiin eri monitorointityökaluja maksullisista palvelukokonaisuuksista avoimen lähdekoodin työkaluihin. Tilaajan ylläpidossa olevat sovellukset ovat luonteeltaan pilvinatiiveja verkkosovelluksia ja -palveluita.

Tutkimuksen yhteydessä haluttiin määrittää myös sovellusten monitoroinnin pohjakonfiguraatio (baseline), jossa esitetään sovellustason monitoroinnin konfiguraatiovaatimukset. Määritettyä konfiguraatiota voitaisiin käyttää hyödyksi uusien sovellusten toteuttamisvaiheessa tai sovelluksen siirtyessä ylläpidettäväksi.

Pohjakonfiguraation mukainen monitorointi päätettiin samalla toteuttaa yhteen ylläpidossa olevista sovelluksista, jolla validoidaan tutkimuksen tulokset.

Myös käytäntöjä haluttiin yhtenäistää Nordcloudin käyttämän pilvi-infrastuktuuritason ja Nordcloud Solutionsin sovellustason monitoroinnin välillä. Nordcloudin jo olemassa olevasta pilvi-infrastruktuurin monitoroinnista saatiin vinkkejä sovellustason toteutuksen tueksi.

2 Tutkimusaineisto ja -menetelmät

2.1 Tutkimusmenetelmät

Opinnäytetyö on muodoltaan soveltuvuustutkimus, jonka tavoitteena oli selvittää sovellusmonitoroinnin soveltuvuus tilaajan ylläpitoympäristössä.

Opinnäytetyön toteuttamisessa käytettiin soveltavaa tutkimusmenetelmää. ”Soveltava tutkimus tarkoittaa toimintaa uuden tiedon saavuttamiseksi, joka ensisijaisesti tähtää tiettyyn käytännön sovellutukseen.” (Soveltavat tutkimusmenetelmät n.d.) Tutkimuksessa yhdistettiin teoreettisen tutkimuksen käytänteitä empiiriseen eli kokeelliseen tutkimukseen.

2.2 Tutkimuksen rajaus

Tutkimus jaettiin kahteen osaan: teoreettiseen ja empiiriseen osuuteen. Teoreettisessa osuudessa perehdyttiin tutkimuskohteeseen eli sovellusmonitorointiin yleisesti. Osuuden kautta haluttiin perusymmärrys aiheesta, jonka saavuttamiseksi tutustuttiin aiheesta julkaistuihin tutkimuksiin, artikkeleihin, blogeihin sekä kirjallisuuteen.

Empiirisessä osuudessa vaatimusmäärittelyn perusteella valittujen monitorointityökalujen ja määritellyn pohjakonfiguraation soveltuvuutta testattiin käytännössä yksittäiseen ylläpitotiimin ylläpitämään verkkosovellukseen.

2.3 Tiedonkeruu- ja analyysimenetelmät

Opinnäytetyön tiedonkeruumenetelmänä teoreettisessa osuudessa käytettiin jo olemassa olevien tutkimusten ja aiheeseen liittyvien artikkelien hyödyntämistä. Empiirissä osuudessa tiedonkeruumenetelminä käytettiin kokeelliselle tutkimukselle sopivia menetelmiä, havainnointia sekä tulosten mittausta ja tarkastelua.

3 Teoreettiset lähtökohdat

3.1 Sovellusmonitorointi yleisesti

“Monitorointi on metodologinen lähestymistapa, joka pyrkii selvittämään tapahtumisen prosesseja ja kehityskulkuja – trendejä – juuri silloin, kun ne tapahtuvat tai heti sen jälkeen, kun jotain niiden aiheuttamaa on tapahtunut.” (Rubin n.d.)

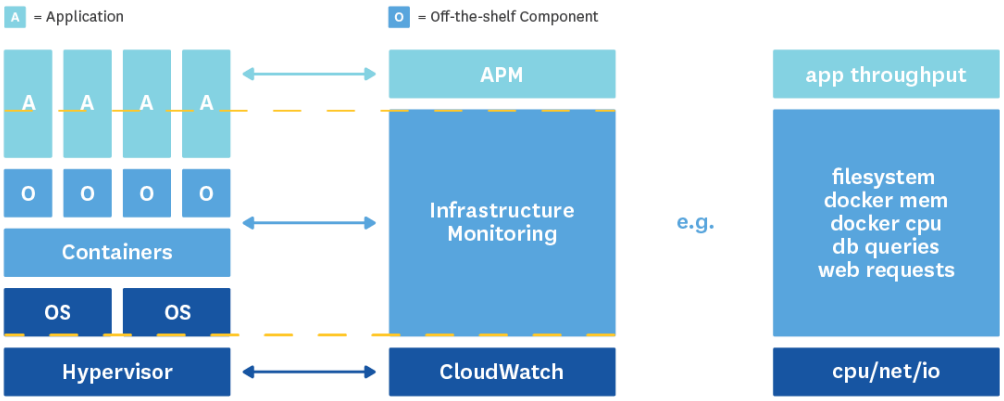
Teknologisessa ympäristössä monitorointi tarkoittaa prosesseja ja työkaluja, joilla mitataan ja ylläpidetään IT-järjestelmiä. Sen tarkoitus on tukea järjestelmän suorituskykyä ja sen myötä tarjota järjestelmän käyttäjälle mahdollisimman laadukas käyttäjäkokemus. Monitorointijärjestelmä kääntää metriikan mitattavaksi liikearvoksi eli käyttäjäkokemukseksi, josta saadaan tietoa järjestelmän toiminnasta sekä palautetta mitä käyttäjät siitä haluavat. Saadulla palautteella mahdollistetaan hyvän tuotteen toteuttaminen ja hyvien investointiratkaisujen tekeminen. (Rouse n.d.; Turnbull 2016, luku 1.2.)

Sovellukset eivät ole enää useinkaan monoliittisia itsenäisiä kokonaisuuksia. Modernit sovellukset ovat tietokannoista, verkkopalveluista ja muista komponenteista koostuvia kokonaisuuksia, joiden tilan määrittäminen ei onnistu vain sovellusta itseään seuraamalla. Ylläpitoa varten on hyvä seurata ja hallita seuraavia sovelluksessa tapahtuvia asioita: saatavuutta (sovittu SLA), liikennettä ja käyttöä, määritettyjä KPI:tä sekä tapahtuneita virheitä. (Jones 2013; Watson 2017.)

Monitoroimattomasta palvelusta ei voida tietää, onko se edes toiminnassa. Monitorointi mahdollistaa palvelussa tapahtuneiden ongelmatilanteiden tiedottamisen jo

ennen, kuin käyttäjät ilmoittavat niistä. Lisäksi hyödyntämällä poikkeaman- ja hahmontunnistusta mahdollistetaan ongelmien havaitseminen ennen kuin ongelma tai virhe edes tapahtuu. (Part III. Practices. n.d; Turnbull 2016, luku 2.3.)

On suositeltavaa monitoroida koko sovelluskokonaisuutta aina itse sovelluksesta käyttöjärjestelmätasolle. Jos käytetään esimerkiksi EC2-palvelua, voidaan hyödyntää CloudWatch-palvelua virtuaalikoneen monitorointiin sekä infrastruktuurimonitorointia ja sovellussuorituskykymonitorointia ongelma-alueiden paikantamiseen (ks. kuvio 1). (Matson & Young n.d., luku 4.)



Kuvio 1. Sovellusmonitoroinnin eri tasot (Young 2015)

Sovellusmonitorointi jaetaan kahteen eri monitorointityyppiin: saatavuus- ja suorituskykymonitorointiin. Näiden monitoroinnista ja hallinnasta käytetään myös nimitystä sovellussuorituskyvyn ylläpito. (Jones 2013; Kowall 2014.)

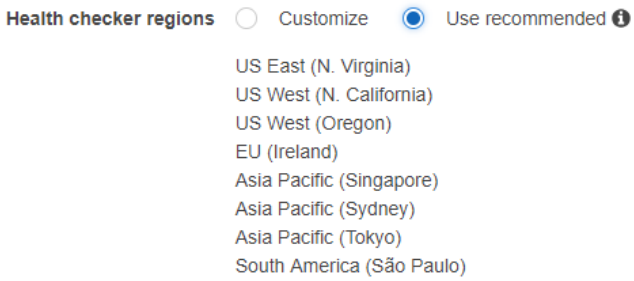
3.2 Saatavuusmonitorointi

3.2.1 Saatavuusmonitorointi yleisesti

Saatavuusmonitorointia käytetään yleisesti palveluiden tai resurssien tilan monitorointiin. Suosittu toteutustapa on tehdä sovellukselle sen tilan tarkastus (health check), jossa tiettyä osan tilaa tiedustellaan, esim. verkkosovelluksen kohdalla tavallisella HTTP-pyynnöllä. Osa voi olla sovellus itse tai vaikkapa sen ohjelmointirajapinta. Esimerkiksi voidaan tiedustella sovellukselta, palauttaako se HTTP-tilakoodia 200. Tällaista toteutusta käytetään tarkistamaan, onko sovellus ”pystyssä” monitoroimalla

sen ulkoista pintaa. Vaihtoehtoisesti tilanne voi olla sellainen, ettei sisäistä sovelluksen monitorointia voida toteuttaa ja joudutaan tyytymään ulkoiseen monitorointiin. (Creating Amazon Route -- n.d; Turnbull 2016, luku 9.4.)

Esimerkiksi Amazon Route 53 -palvelu tarjoaa Health Check -ominaisuutta, jolla voidaan monitoroida resurssien saatavuutta seuraamalla sen tilaa ja suorituskykyä (esim. viive). Palvelussa voidaan myös määrittää halutut alueet, joilta resurssin tilaa ja suoriutumista tiedustellaan (ks. kuvio 2). (Values That You -- n.d.)



Kuvio 2. Health Check -alueiden määrittäminen Amazon Route 53 -palvelussa

3.2.2 Loppukäyttäjän monitorointi

Saatavuusmonitorointi voidaan jakaa kahteen kategoriaan: passiiviseen ja aktiiviseen monitorointiin. Loppukäyttäjän monitorointi on passiivisen monitoroinnin muoto. Se luottaa palveluihin, jotka seuraavat järjestelmän saatavuutta, toimivuutta ja reagointikykyä. Loppukäyttäjän monitorointi ei lepää koskaan, sillä se kerää käyttäjien jokaisen pyynnön. Synteettinen monitorointi taas on aktiivisen monitoroinnin muoto, jolla simuloidaan loppukäyttäjän käyttäytymistä. Kyseistä tekniikkaa käytetään mm. palveluiden testausvaiheessa ennen julkaisua. Synteettinen monitorointi soveltuu myös esim. rasitustestaukseen (kuinka palvelu käyttäytyy suuren kävijäpiikin aikana) tai uuden ominaisuuden testaukseen ennen sen julkaisua. (Gasparyan 2017; What is Real User Monitoring -- 2017.)

Loppukäyttäjän monitorointia voidaan käyttää ongelmien havaitsemiseen, joita käytetyillä testausmetodeilla ei olla löydetty. Esimerkkeinä tällaisille tapauksille: käyttäjämääristä johtuvat latausajan muutokset ja aikakatkaisut tai testaamattomista laiteympäristöistä johtuvat ongelmat (mm. mobiililaitteet). Loppukäyttäjän monitorointi soveltuu myös SLA:n toteutumisen mittaamiseen. Monitoroinnilla saadaan

tarkkaa dataa käyttäjien käyttäytymisestä, jonka avulla voidaan määrittää palvelun tärkeimmät käyttötapaukset. (What is Real User Monitoring -- 2017.)

Moni on voinut käyttää loppukäyttäjän monitorointituotteita muistuttavia palveluita huomaamattaan, esim. Google Analyticsiä. Se tarjoaa korkean tason dataa mm. käyttäjän polusta, käytetyistä selainversioista ja sivujen latauksista. Usein pelkästään Analyticsin seuraamat ominaisuudet eivät riitä, kun halutaan ammattimaisempaa tietoa suorituskyvystä ja loppukäyttäjän toiminnasta. Tärkeitä monitoroitavia ominaisuuksia ovat mm. latausprosessit (esim. nimipalvelukyselyyn käytetty aika, TCP:n yhdistämisaika, TLS-kättely), sivun renderöintiaika ja loppukäyttäjän miettimisaika. (Gasparyan 2017.)

Aika ajoin parhaissakin sivustoissa tulee negatiivia käyttäjäkokemuksia. On tärkeää päästä käsiksi näiden kokemuksien suorituskykydataan, josta voidaan nähdä käyttäjän reaktiot. Jos sivusto latautuu hitaasti, kuinka käyttäjä käyttäytyy? Kuinka kauan käyttäjä jaksaa odottaa sivun latausta, ennen kuin hylkää sivun? (SOASTA Real User Monitoring Best Practices n.d.)

Synteettisessä monitoroinnissa huonona puolena on erilaisten variaatioiden puute. Esimerkiksi lataushetki, käyttäjän sijainti, laiteympäristö ja latausnopeus voivat johtaa erinäisiin tilanteisiin, joita synteettisellä monitoroinnilla ei voida testata. Lisäksi oikea käyttäjä voi käyttää erilaisia reittejä päästäkseen haluamaansa lopputulokseen. (SOASTA Real User Monitoring Best Practices n.d.)

3.3 Sovellussuorituskyvyn monitorointi

3.3.1 APM yleisesti

Yleisesti sovellussuorituskyvyn monitorointi eli APM mittaa järjestelmän suoriutumista määritellyissä käytännön tehtävissä. Edellisen lisäksi hyvä APM-toteutus mittaa komponenttikohtaisesti suorituskykyä ja niiden tilaa, jolla voidaan paikallistaa virheen aiheuttaja. Hyvä APM-toteutus keskittyy aina hyvän loppukäyttäjäkokemuksen ylläpitämiseen ensisijaisena tavoitteena. Muita seurattuja mittareita ja tavoitteita hyödynnetään ongelmien ratkaisemisessa ja niillä tuetaan ensisijaisen tavoitteen täyttymistä. (Jones 2013.)

APM voi tarkoittaa eri lähteestä riippuen sovellussuorituskyvyn ylläpitoa tai sen monitorointia. Se on joka tapauksessa välttämätön työkalu sovelluksen suorituskyvyn optimoinnissa ja monitoroinnissa, jonka päätavoite on auttaa selvittämään, miksi jokin asia tapahtui mahdollisimman nopeasti. (What is Application -- 2015.)

Sovellussuorituskyvyn monitorointiin käytettyjä työkaluja on eri tyyppisiä: sovellusmetriikkapohjaisia, tietoverkkopohjaisia ja kooditason suorituskykypohjaisia, jotka pohjautuvat koodiprofilointiin ja transaktiojälkeen. Monet työkalut tarjoavat koodintason suorituskyvyn mittausta. Tällaista mittausta voidaan käyttää esim. hitaiden SQL-kyselyiden tai hitaiden kutsujen selvittämisessä. Kooditason suorituskyvyn mittaus mahdollistaa myös saadun datan visualisoinnin, joka auttaa ongelman ratkaisussa tarjoamalla matalan tason yksityiskohtia koodin käyttäytymisestä. (Watson 2017; What is Application -- 2015.)

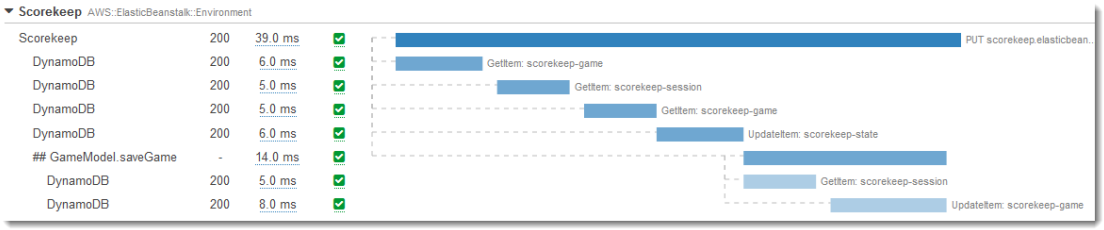
3.3.2 Koodiprofilointi

Koodiprofilointi on kehittäjien käyttämä analyysimuoto, jolla pyritään havaitsemaan koodin suorituskykyongelmia ja sen myötä parantamaan suorituskykyä. Profilointiin käytetään yleensä apuna niihin tarkoitettuja työkaluja eli profiloijia. Profiloijaa käyttämällä voidaan muun muassa selvittää, kuinka monesti kukin metodi suoritetaan ja kuinka kauan suorittaminen kestää. Profiloijat ovat omiaan muistivuotojen löytämiseen hyvissä ajoin, sillä ne mahdollistavat muistin allokoinnin ja roskankerääjän käyttäytymisen seuraamisen. Koodiprofilointiominaisuutta tarjoaa paljon eri sovelluksia, mm. Retrace, New Relic ja Microsoft Azuren Application Insights. Yksinomaan profilointia tarjoavia työkaluja ovat mm. JetBrainsin dotTrace, Redgate ANTS Performance Profiler ja Microsoftin Xamarin Profiler. (Dorsey 2015.; What is code profiling -- 2016.)

3.3.3 Jäljitys

Jäljityksen (tracing) ideana on tallentaa järjestelmän käyttäytymistä, jota voidaan jälkikäteen analysoida. Jäljitys on hyödyllisin suorituskykyongelmien selvittämisessä ja viiveen selvittämisessä hajautetuissa järjestelmissä (distributed systems) tai mikropalveluarkkitehtuureissa. Sillä voidaan löytää epäsäännöllisyyksiä kauan ennen kuin niihin liittyvät bugit tulee ilmi asiakaskäytössä. (Kraft 2017; Turnbull 2016, luku 9.6.)

Esimerkiksi AWS tarjoaa pilvinatiiveille järjestelmilleen X-Ray -jäljityspalvelua. Se mahdollistaa ongelmien identifioinnin ja tarjoaa optimointimahdollisuuksia esittämällä tarkempaa statistiikkaa seuratuista kyselyistä. Kysely- ja vastausinformaation lisäksi se kykenee näyttämään resurssiin liittyvien AWS-resurssien (esim. mikropalvelut, tietokannat, HTTP Web API:t) transaktiojäljen (ks. kuvio 3). X-Ray:n taustaprosessi löytyy esiasennettuna Elastic Beanstalk ja Lambda -alustoista. (What is AWS X-Ray n.d.)

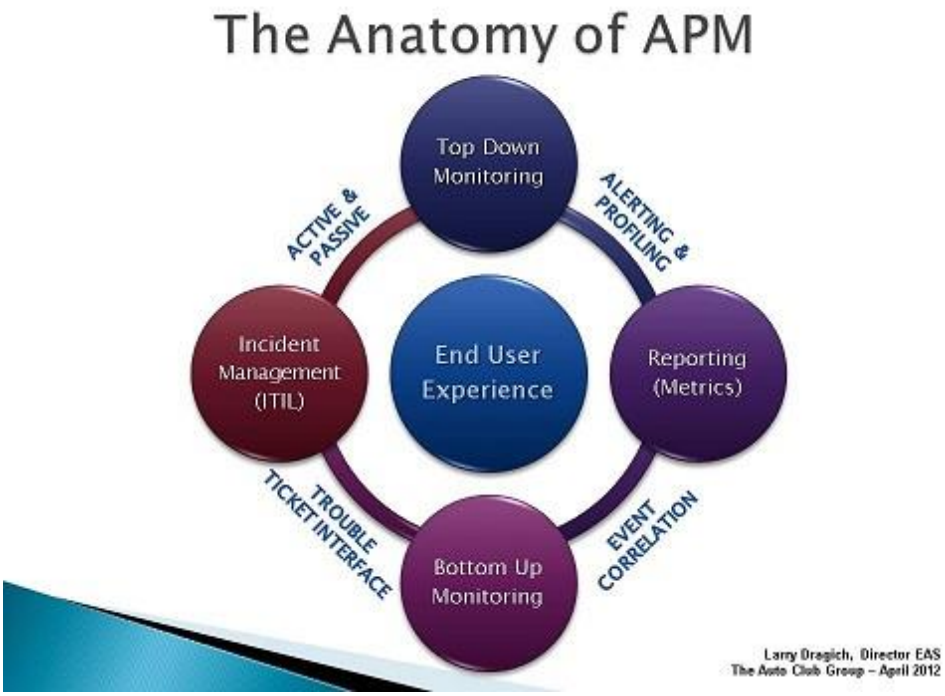


Kuvio 3. EBS-palvelusta kerätty informaatio (What is AWS X-Ray n.d)

3.4 Sovellussuorituskyvyn ylläpito

3.4.1 Yleistä

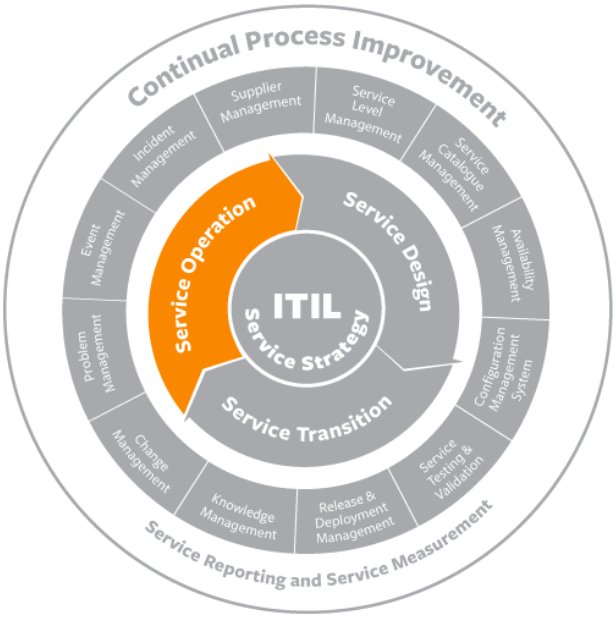
Sovellussuorituskyvyn ylläpito jaetaan karkeasti neljään osioon: loppukäyttäjämonitorointiin (top down monitoring), järjestelmän monitorointiin (bottom up monitoring), häiriöiden hallintaprosessiin ja raportointiin (ks. kuvio 4). (Dragich 2012.)



Kuvio 4. Sovellussuorituskyvyn ylläpidon anatomia (Dragich 2012)

3.4.2 Häiriöiden hallintaprosessi

Häiriöt ovat suunnittelemattomia tapahtumia, jotka häiritsevät, laskevat palvelun laatua tai uhkaavat tehdä niin. Häiriöiden hallintaprosessi auttaa palvelun toimintaa määrittämällä yleisiä käytänteitä. ITIL on määrittänyt suosituksia häiriöiden hallintaprosessiin liittyen, jotka ovat osa jatkuvan prosessin elinkaarimallia (ks. kuvio 5). Toimivan prosessin takaamiseksi on syytä määrittää menetelmät häiriöiden tunnistamiseen ja kirjaamiseen, kategorioimiseen ja priorisoimiseen sekä niihin reagoimiseen. (ITIL incident management n.d.; Wright n.d.)



Kuvio 5. ITIL-elinkaarimallin rakenne (ITIL incident management n.d)

Avain hyvään häiriöiden hallintaan on määritellä prosessi, ja kurinalaisesti noudattaa sitä. On suotavaa käyttää olemassa olevia prosesseja hyödyksi, sillä pyörää ei tarvitse keksiä uudelleen. (Wright n.d.)

Ensimmäinen asia on määrittää häiriö ja kirjata se. Häiriö voi tulla mistä vain, se voi tulla puhelimitse tai vaikka kirjaimellisesti pudota syliin. Kirjattu häiriö sisältää yleensä raportoijan nimen, päivämäärän ja ajan, lyhyen kuvauksen ja uniikin numeron seurantaa varten. (Mt.)

Häiriöiden kategorioiminen ja priorisoiminen ovat tärkeitä vaiheita jotka helposti jätetään huomiotta. Kategorioiminen mahdollistaa häiriöiden analysoinnin, jolla voi seurata niiden trendejä ja toistuvuuksia ja näin on tärkeässä osassa hallintaprosessia ja tulevien häiriöiden ennaltaehkäisyssä. Priorisoinnissa kannattaa huomioida kuinka moneen ihmiseen häiriö vaikuttaa, unohtamatta taloudellista riskiä ja turvallisuutta – kuinka kiireellinen se on ratkaista liiketoiminnan kannalta. Monet organisaatiot määrittävät selkeät palvelusopimukset jokaisen määritetyn prioriteetin ympärille, joten asiakkaat tietävät kuinka nopeasti vastaus ja ratkaisu ovat odotettavissa. (Mt.)

Häiriöön reagoiminen on vuorossa, kun se on määritetty, kategorioitu ja priorisoitu. Reagoiminen on terminä laaja, joten se on syytä pilkkoa pienempiin osiin. Se voidaan

jakaa alustavaan diagnosointiin, eskaloimiseen ja tutkinta- sekä diagnosointivaiheeseen. Alustavassa diagnoosivaiheessa häiriö luokitellaan. Luokittelussa tehdään nopea hypoteesi mikä on pielessä, jolla määritetään seuraavat askeleet häiriön ratkaisemiseksi. Eskalointivaiheessa häiriö välitetään eteenpäin, jos ratkaisu vaatii jonkun muun tahon toimia. Tehtävänä on kerätä oikeaa informaatiota seuraavalle, jotta ongelman ratkaiseminen olisi mahdollisimman jouhevaa. Tutkinta- ja diagnosointivaihe tapahtuu osana häiriön elinkaarta, johon häiriön tiedonkerääminen jo kuuluu. (Mt.)

Viimein ja ideaalissa tilanteessa sovitun SLA:n merkeissä päädytään diagnoosiin ja tehdään tarvittavat vaiheet häiriön ratkaisemiseksi. Palautumisella mitataan aikaa, kun häiriöstä on täysin toivuttu (esim. bugin paikkaaminen, joka vaatii testausta ja käyttöönoton) vaikka oikea ratkaisu olisikin jo määritetty. (Mt.)

3.4.3 Raportointi

Raa'an datan kerääminen analysointia varten on välttämätöntä. On syytä määrittää seurattavat metriikat ja standardoida näkymät reaaliaikaisen datan esittämiseksi. Hyvä lähtökohta on esimerkiksi käyttää 5 minuutin keskiarvoja reaaliaikaisen suorituskyvyn hälytyksissä ja prosentteja kokonaiskuvan profiloinnissa sekä palvelutason hallinnassa. (Dragich, 2012.)

Datan kerääminen on halpaa, mutta sen puuttuminen sitä tarvittaessa voi tulla kalliiksi. Siksi tulisikin mitata ja kerätä kaikki hyödyllinen data niin tarkasti kuin mahdollista. Hyvän datan tulisikin olla helposti tulkittavissa, laajalti kerätty (myös järjestelmän normaalitilanteista) ja tulla säilytetyksi pitkältä ajalta, joka helpottaa ymmärtämään mikä on järjestelmälle normaalia. (Matson & Young n.d., luku 2.)

Hyvin havaittavissa olevan järjestelmän toteuttaminen ei ole pelkästään monitorointidatan hankkimista. Datan tulee olla myös helposti saatavilla ja tulkittavissa. Metriikkaa ja sen visualisoitua vastinetta on usein hankala tulkita. Ihmisillä on tapana etsiä merkityksellisiä toistuvuuksia jopa satunnaisesta datasta. Tilannetta voi pahentaa rajoitettu piirtotarkkuus, mittakaava ja kuinka päätämme esittää tiedon. Siksi on syytä ottaa huomioon datan esittämisen selkeys, johtaa käyttäjä ajattelemaan pääsisältöä eikä visualisointeja, välttää vääristämästä dataa ja muistaa suurten datamäärien sointuvuus. (Matson 2016; Turnbull 2016, luku 2.5.)

Metriikka

Metriikka tallentaa järjestelmän arvon tietyllä ajan hetkellä, esimerkiksi järjestelmän sen hetkisen kirjautuneiden käyttäjien määrän. Yleisesti halutaan seurata kahdenlaista metriikkaa: sovellus- ja liiketoimintametriikkaa (Matson & Young n.d., luku 2.; Turnbull 2016, luku 9.2).

Sovellusmetriikka mittaa yleisesti sovelluksen tilaa ja suorituskykyä. Se ilmaisee loppukäyttäjän kokemusta sovelluksesta, kuten viivettä ja vasteaikoja. Taustalla mitataan myös sovelluksen suorituskykyä: kyselyjä, niiden määrää, transaktioita ja transaktioiden ajoituksia. Lisäksi seurataan myös sovelluksen toiminnallisuutta ja tilaa. Hyvinä esimerkkeinä onnistuneet ja epäonnistuneet kirjautumiset ja virheet yleisesti. Lisäksi voidaan mitata aktiviteettien (esim. sähköpostit ja muut asynkroniset tehtävät) määrää ja suorituskykyä. (Turnbull 2016, luku 9.2.1.)

Liiketoimintametriikalla mitataan sovelluksen arvo (esim. verkkokaupan tapauksessa kuinka monta myyntiä tehty). Muina esimerkkeinä uusien asiakkaiden määrä, myynti alueittain tai mikä tahansa joka auttaa mittaamaan liiketoiminnan ”tilaa”. (Turnbull 2016, luku 9.2.2.)

Tapahtumat

Metriikan lisäksi, monitorointityökalut seuraavat yleensä tapahtumia (events), jotka tarjoavat tärkeitä asiayhteyksiä järjestelmän käyttäytymisen tueksi. Näistä esimerkkeinä koodimuutokset, järjestelmän ajastettujen varmuuskopioinnin epäonnistuminen ja hälytykset. (Matson & Young n.d., luku 2.)

3.5 Hälytykset

3.5.1 Hälytykset yleisesti

Yleinen tapa on seurata määriteltyjä raja-arvoja tai tilanteita, jonka toteutuessa lähetetään hälytys. Ratkaisu on usein tehoton, sillä ihminen tarvitaan arvioimaan hälytyksen aiheellisuus. Hyvän monitorointijärjestelmän tulisi itse tehdä johtopäätös ja ilmoittaa ainoastaan tilanteista, jotka vaativat ihmisen toimintaa. Hälytysten tulisi olla

selkeitä, tarkkoja, tiiviitä ja vaatia toimintaa. (Sloss & Beyer n.d.; Turnbull 2016, luku 2.4.)

Jos hälytyksiä tulee paljon ja ne ovat usein virheellisiä, niihin voi uupua. Tilanne voi johtaa siihen, että hälytyksiin vastataan viiveellä tai jätetään puuttumatta kokonaan. (Holmwood 2014.)

Hyvän hälytysjärjestelmän luomiseksi on syytä määrittää seuraavat asiat: kenelle, kuinka ja kuinka usein ilmoittaa ongelmasta sekä päättää jatkotoimenpiteistä (esim. milloin lopettaa vastuuhenkilölle ilmoittaminen ja eskaloida jollekin muulle). Hälytyksen tulisi ilmoittaa ennemmin häiriön oireista kuin syistä. (Matson & Young n.d., luku 3; Turnbull 2016, luku 2.4.)

Automatisoidut hälytykset ovat monitoroinnin elinehto. Ne mahdollistavat ongelmien havaitsemisen, jotta voidaan nopeasti määrittää syy ja minimoida palvelun laadun heikentyminen ja häiriön kesto. (Matson & Young n.d., luku 3.)

3.5.2 Hälytysten priorisointi

Monitorointi on kehittyvä prosessi, sillä hälytys joka oli joskus hyödyllinen, voi olla myöhemmin turha ja hyvinkin poistettavissa. Kaikki hälytykset eivät ole siis yhtä kiireellisiä. Vain osa vaatii ihmisen välitöntä huomiota, osa taas huomiota jossain vaiheessa ja osa mahdollisesti huomiota tulevaisuudessa. Kaikki hälytykset tulisi olla vähintään keskitetyssä paikassa, josta ne ovat helposti saatavilla metriikan ja tapahtumien välittömässä läheisyydessä. (Matson & Young n.d., luku 3; Turnbull 2016, luku 10.8.)

Hälytykset kirjauksina

Monet hälytykset eivät ole osallisina palvelun ongelmissa, joten ihminen ei välttämättä edes ole tietoinen niistä. Esimerkiksi tietokannan hitaus, joka ei ole kuitenkaan niin hidas vaikuttaakseen huomattavasti loppukäyttäjän kokemukseen, voidaan ilmoittaa matalan tason hälytyksenä ja kirjata monitorointijärjestelmään tulevaisuutta varten. (Matson & Young n.d., luku 3.)

Hälytykset ilmoituksina

Ilmoitustason hälytykset vaativat toimintaa, muttei välitöntä sellaista. Esimerkiksi levytila alkaa olla vähissä ja vaatii ratkaisua lähipäivinä. Ilmoitustason hälytykset voidaan ilmoittaa sähköpostina tai ilmoituksena esim. Slack-palvelussa, muttei tulisi herättää päivystäjää keskellä yötä. (Mt.)

Hälytykset kuulutuksina

Kaikista kiireellisimmät hälytykset tulisi saada erityiskohtelua ja niiden tulisi eskaloitua vastuuhenkilön välitöntä huomiota varten. Esimerkiksi määritetyn SLA:n ylityksestä tulisi hälyttää vastuuhenkilöä kellonajasta riippumatta. Kuulutustyyppiset hälytykset ovat erityisen tehokkaita informaation välittämisessä, mutta liikaa käytettynä voivat olla hyvin häiritseviä. (Mt.)

3.5.3 Hälytyksen määrittäminen

Hälytyksen määrittämiseen on suositeltavaa vastata seuraaviin kysymyksiin:

- Onko ongelma todellinen? Huonosti määritelty hälytys voi aiheuttaa paljon turhia hälytyksiä ja täten uupumusta.
- Vaatiiko ongelma huomiota?
- Onko ongelma kiireellinen?

Hälytykset tulisi määrittää ennemmin oireisiin kuin syihin. Oireet paljastavat usein käyttäjälle näkyvät ongelmat eivätkä hypoteettisia tai sisäisiä ongelmia. Lisäksi oirepohjaiset hälytykset ovat ”kestäviä”, eli ne eivät riipu taustajärjestelmästä ja sen arkkitehtuurista. Jos järjestelmän toiminta ei ole vaaditulla tasolla, saadaan hälytys ilman hälytyksen uudelleenmäärittämistä. (Matson & Young n.d., luku 3.)

3.6 Hälytysten ja suorituskykyongelmien tutkiminen

3.6.1 Tutkiminen yleisesti

Kuten elämässä yleensä, ongelman määrittäminen on yleensä melko helppoa. Ongelman ratkaiseminen sen sijaan vaatii suurempaa työtä. Monitoroinnin kohdalla saata-

villa olevat työkalut voivat auttaa tässä ongelmassa. Lisäksi noudattamalla määriteltyä ja sovittua monitorointiviitekehyksestä mahdollistetaan systemaattisempi ongelmien tutkiminen. (Watson 2017; Matson & Young n.d., luku 4.)

3.6.2 Monitorointijärjestelmät

Monitorointijärjestelmät ovat joko pull- tai push-pohjaisia. Pull-pohjaisissa (esim. Nagios) järjestelmä tiedustelee monitoroitavilta komponenteilta (esim. palvelut, sovellukset) niiden tilaa. Monitoroitavien komponenttien määrän kasvaessa myös tarvittavien kyselyiden ja prosessien määrä kasvaa. Tämä johtaa järjestelmän vertikaaliseen skaalaukseen. Push-pohjaisissa järjestelmissä (esim. Riemann) komponentit ovat lähettimiä (emitter), jotka lähettävät dataa keskitetylle kerääjälle (central collector). Datan keräys hoidetaan komponenteissa, joten monitorointijärjestelmä ei vaadi suuria resursseja komponenttien määrän kasvaessa. (Turnbull 2016, luku 2.1.)

Modernien monitorointijärjestelmien tärkeimpiä perusominaisuuksia ovat sisäänrakennettu koostaminen (built-in aggregation), kattavuus (comprehensive coverage), skaalautuvuus, hienostunut hälyttäminen ja mahdollisuus yhteistyöhön (komponenttien jakaminen, kommentointi). (Matson & Young n.d., luku 1.)

3.6.3 Juurisyyanalyysi

Monitorointijärjestelmän velvollisuudet eivät rajoitu oireiden havaitsemiseen. Oireen havaitsemisen jälkeen sen tehtävä on auttaa juurisyyn analysoinnissa. Analysointi on usein monitoroinnin vähiten jäsennelty vaihe, joka usein pohjautuu aavistuksiin, arvauksiin ja tarkistuksiin. (Matson & Young n.d., luku 4.)

Juurisyyn tutkimista avustamassa ovat monitorointidatan 3 päätyyppiä: suoritusmetriikka (work metrics), resurssimetriikka (resource metrics) ja tapahtumat (events). On suotavaa aloittaa suoritusmetriikasta ja määritellä ongelma. On helppoa eksyä aiheeseen syventyessä, mikäli ongelmaa ei ole määritelty. Seuraavaksi voidaan aloittaa mitatun datan tutkiminen korkeimmalta tasolta, jossa ongelma ilmenee. Tämä usein määrittää ongelman lähteen tai ainakin osoittaa suunnan jatkotutkimukselle. Mikäli edellinen ei määrittänyt ongelman syytä, seuraavaksi voidaan syventyä resurssimetriikkaan. Luodut raportointinäkymät (dashboard) auttavat nopeuttamaan oleellisesti

resurssimetriikoiden tutkimista. Ovatko resurssit kuormitettuja tai saavuttamattomissa? Mikäli näin on, syvennytään ongelmallisiin resursseihin ja aloitetaan niiden tutkiminen yksitellen. (Matson & Young n.d., luku 4.)

Seuraavaksi voidaan huomioida hälytyksiä ja muita tapahtumia jotka ovat voineet korreloida mittausdatan kanssa. Jos koodimuutos, sisäinen hälytys tai jokin muu tapahtuma rekisteröitiin juuri ennen ongelmien alkamista, kannattaa tutkia niiden yhteyttä. (Matson & Young n.d., luku 4.)

Kun ongelman aiheuttaja on määritetty, se korjataan. Tutkimus on valmis, kun oireet häviävät – seuraavaksi voidaan miettiä, kuinka muuttaa järjestelmää, jotta sama ongelma ei tapahdu uudestaan. (Matson & Young n.d., luku 4.)

3.6.4 Raportointinäkymät

Raportointinäkymät kannattaa rakentaa ennen kuin niitä tarvitaan, sillä katkoksen aikana jokainen minuutti on tärkeä. Näkymät voivat nopeuttaa tutkimusta ja pitävät keskittymisen itse tehtävässä. Sovellukselle kannattaa määrittää yksi näkymä korkean tason metriikoita varten, jonka lisäksi yksi raportointinäkymä kutakin alijärjestelmää kohden. Jokaisen näkymän tulisi esittää kyseisen järjestelmän suoritus- ja resurssimetriikka sekä kyseisen järjestelmän alijärjestelmien avainasemassa oleva metriikka. Jos tapahtumadata on saatavilla, kerrostetaan relevantit tapahtumat graafeihin korrelaatioanalyysiä varten. (Matson & Young n.d., luku 4.)

3.7 Monitoroinnin toteuttaminen

3.7.1 Toteuttamisen lähtökohdat

Hyvän sovelluskehittämismetodologian mukaisesti on hyvä idea määrittää mitä halutaan rakentaa ennen sen rakentamista. Monitorointi ei poikkea tästä ajatuksesta. Harmillisesti, monitoroinnin rakentamisessa on virheellisesti ymmärrettyjä suunnittelumalleja (anti-pattern). Yleinen virhe on, että kuvitellaan monitoroinnin olevan arvonlisäämiskomponentti eikä sovelluksen ydintoiminnallisuutta. Joten, jos ollaan luomassa sovelluksen määrittelyä tai sen käyttäjätarinoita, on suositeltavaa lisätä moni-

torointi jokaiseen sovelluksen komponenttiin. On jopa suositeltavaa yli-instrumentoida sovelluksia. Monitoroinnin poisjättäminen on vakava asia ja voi johtaa kyvyttömyyteen virheiden diagnosoinnissa ja identifioinnissa, kyvyttömyyteen mitata sovelluksen suoriutumista ja liiketoiminnan suorituskykyä sekä sovelluksen menestymistä. (Turnbull 2016, luku 9.1.)

Monen ympäristön (kehitys-, tuotanto- yms) tapauksessa on hyvä identifioida ympäristöt selvästi. Sovelluksen instrumentointi kannattaa aloittaa sisään- ja uloskäynneistä. Esimerkiksi kirjataan ohjelmointirajapinnan pyynnöt ja vastaukset. Jos ollaan luomassa instrumentointia olemassa olevaan sovellukseen, voidaan tehdä priorisoitu lista sivuista ja rajapinnoista, jotka instrumentoidaan tärkeysjärjestyksessä. Lisäksi kirjataan myös kaikki ulkoiset pyynnöt ja vastaukset (tietokanta, välimuisti jne) sekä kolmannen osapuolen palvelut (esim. maksurajapinta). Edellisten lisäksi mitataan ja kirjataan tehtävien ajoitus, suoritus ja muut ajoitetut palvelut kuten cron jobit. Kannattaa mitata tärkeimmät liiketoimintatapahtumat ja muut toiminnalliset tapahtumat. Esimerkiksi käyttäjien luonti tai transaktiot, kuten maksut ja myynnit. Suositeltavaa on mitata myös metodit ja funktiot, joilla luetaan ja kirjoitetaan tietokannasta sekä välimuistista. (Turnbull 2016, luku 9.1.1.)

3.7.2 Monitoiroinnin suunnittelumallit

Kun tiedetään mitä halutaan monitoroida ja mitata, seuraava vaihe on päättää mihin monitorointi toteutetaan. Lähes aina paras paikka on kooditasolla mahdollisimman lähellä toimintoa, jota halutaan mitata. (Turnbull 2016, luku 9.2.)

”Utility pattern”

Konfiguraatiota ei haluta sijoittaa minne sattuu, vaan on suotavaa luoda keskitetty kirjasto, jossa metriikan luonti käsitellään. Toteutusta kutsutaan joskus ”utility patterniksi”, jossa metriikan kirjaaminen on toteutettu luokkana. Luokka ei vaadi asennusta ja sisältää vain staattisia metodeja. (Turnbull 2016, luku 9.2.3.)

”External pattern”

On tilanteita, joissa ei itse hallita koodipohjaa eikä voida lisätä monitorointilogiikkaa koodiin tai vanhentunutta sovellusta ei voida muuttaa tai päivittää. Tällaisissa tilanteissa joudutaan etsimään seuraavaksi lähin paikka monitoroinnille. Ilmiselvimmät paikat ovat ulostulot ja ulkoiset järjestelmän osat sovelluksen välittömässä läheisyydessä. (Turnbull 2016, luku 9.2.5.)

3.7.3 Lokitapahtumat

Metriikan lisäksi on hyvä lisätä myös tuki lokitapahtumien kirjaamiseen sovelluksessa. Metriikka on hyödyllistä suorituskyvyn tai sovelluksen tilan selvittämisessä, mutta lokitapahtumat ovat usein paljon kuvaavampia. Niillä saadaan lisätietoa kontekstista tai lisäinformaatiota tapahtuneesta, esim. virheestä saatu stack trace. Sovelluksen kuorman minimoimiseksi kirjaustapahtumat kannattaa toteuttaa asynkronisesti. (Turnbull 2016, luku 9.1.3; Turnbull 2016, luku 9.3.)

3.7.4 Käyttöönotto ja päivitykset

Yksi monitoroinnin avainasioista on seurata sovellusten muutosta, tarkemmin sanottuna seurata muutosten käyttöönottoa. Mikäli metriikkaa ja tapahtumia seurataan, niin on tärkeää tietää tapahtumista, jotka voivat muuttaa metriikoita ja lokitapahtumia. Kun päivityksiä julkaistaan monitoroitaviin sovelluksiin, halutaan siitä myös tulla informoiduksi. (Turnbull 2016, luku 9.5.)

3.8 Kultaiset signaalit

Järjestelmän monitoroinnin tärkeimpiä mitattavia metriikoita kutsutaan kultaisiksi signaaleiksi. Kultaisista signaaleista on erilaisia menetelmiä, joista kolme suosittua ovat Googlen SRE -ympäristössä käytetyt (viive, liikenne, virheet ja saturaatio), USE (käyttö, saturaatio ja virheet) ja RED (määrä, virheet ja kestoaika) -menetelmät. (Ewaschuk & Beyer n.d.; Mushero 2017.)

Näistä saturaatio (kuinka kuormittunut järjestelmä on) ja käyttö (kuinka kiireinen järjestelmä on) ovat usein vaikeimpia mitattavia ja ovat täynnä olettamuksia, aukkoja ja ovat matemaattisesti monimutkaisia, joten näitä tulisi käsitellä korkeintaan suuntaa

antavina arvoina. Lisäksi on myös hienostuneempia tapoja mitata asioita. Esimerkiksi tapahtuneiden virheiden viive on yleensä pienempi kuin onnistuneiden pyyntöjen, joten viivettä laskiessa kannattaa rajata virheet pois, mikäli se on mahdollista. (Mushero 2017.)

Edellä mainittuja signaaleja kutsutaan ”kultaisiksi”, sillä ne yrittävät mitata asioita, jotka vaikuttavat suoraan loppukäyttäjään ja järjestelmän tärkeisiin osiin. Ne ovat yleisesti parempia kuin muut epäsuoremmat mittarit, kuten CPU, RAM, verkkoliikenne jne. Kultaisia signaaleja kerätään hälytyksiä, vianetsintää, hienosäätöä ja kapasiteettisuunnittelua varten. (Mt.)

Kultaisista signaaleista ilmenneet hälytykset ovat piilevien ongelmien oireita eivätkä sen vuoksi useinkaan ole selvitettävissä suoraan hälytyksestä. Sen lisäksi ne voivat olla hyvin abstrakteja ja niitä voi tulla helposti paljon. Tällainen vaatii hyvää järjestelmän tuntemusta ja ongelmanratkaisukykyä, mutta jokaisen järjestelmän osan kultaisten signaalien kerääminen auttaa määrittämään todennäköisen ongelman lähteen ja selvittämään mihin kannattaa keskittyä. (Mt.)

3.9 SLA

3.9.1 Määritelmä

”SLA (Service Level Agreement) eli palvelutasosopimus on asiakkaan ja palveluntarjoajan välinen sopimus, jossa määritellään asiakkaalle tarjotun palvelun palvelutaso. SLA:lla pyritään takamaan tarjotun palvelun laatu.” (Kautto 2009.)

SLA määritellään korkealla ja abstraktilla tasolla, joten tarkemmat vaatimukset määritellään SLO:ita käyttäen. Toteutunutta tasoa ja sovittuja tavoitteita vertaamalla saadaan selville, onko määritetty SLA toteutunut. (Kautto 2009.)

3.9.2 SLO

Palvelutasotavoitteet eli SLO:t ovat SLA:n pohjalta luotuja tavoitearvoja, jotka määrittävät palvelun tasot. Tavoitteita mitataan yleensä yhdellä tai useammalla palvelutasoindikaattorilla eli SLI:lla. (Kautto 2009; Chang 2017.)

3.9.3 SLI

Palvelutasoindikaattorit ovat itse metriikoita, joista yleisiä käytössä olevia ovat saatavuus, viive, järjestelmän suoritusteho ja virheiden osuus. Indikaattorit ovat usein kokonaisuuksia eli aikaikkunan aikana kerättyjen arvojen keskiarvoja, prosenttiosuuksia tai suhteita. (Chang 2017.; Jones, Wilkes, Murphy, Smith & Beyer n.d.)

3.10 KPI

3.10.1 Määritelmä

KPI on mitattava arvo, jolla voidaan todistaa kuinka tehokkaasti organisaatio saavuttaa määriteltyjä tavoitteita. Tehokkaan ja yleispätevän KPI:n määrittämiseen ei ole ratkaisua, vaan jokaisen organisaation tulisi määrittää omat arvot. Arvoja määrittäessä tulisi ymmärtää oman organisaation tavoitteet, kuinka valmistautua niiden saavuttamiseksi ja kuka voi reagoida tähän informaatioon. (What is a KPI n.d.)

KPI-arvojen tuloksen seuraaminen vaatii usein tarkempaa analyysiä. Sillä täydennetään trendien taustojen ymmärrystä, jotka johtivat saavutettuun KPI:n tulokseen. (Key Performance Indicators n.d.)

3.10.2 KPI:n määrittäminen

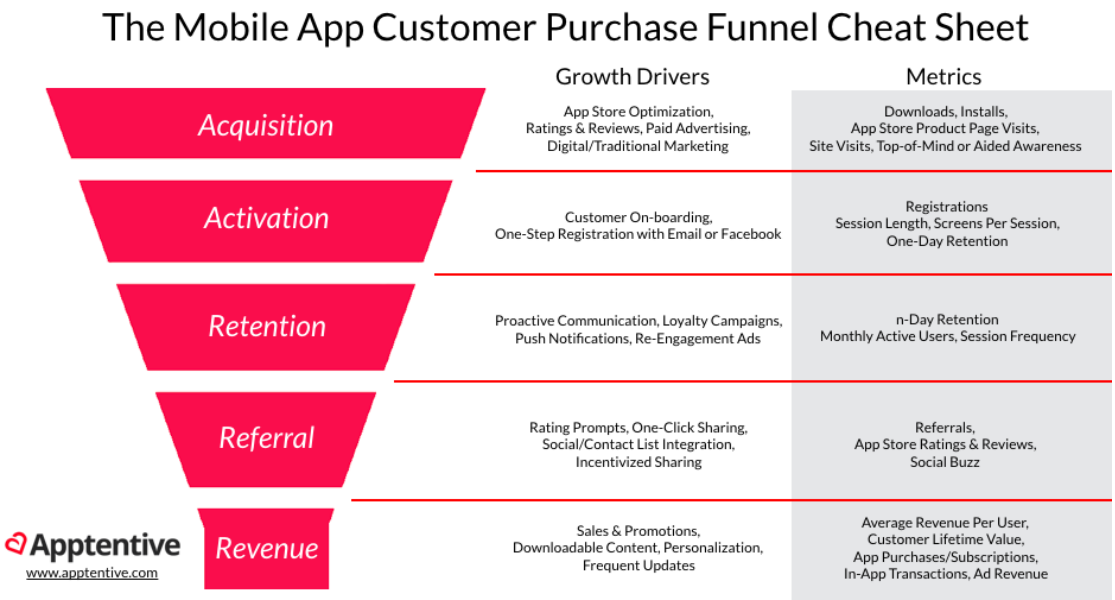
KPI:n määrittäminen voi olla haastavaa, sillä jokainen KPI tulisi liittyä tiettyyn liiketoimintalopputulokseen. KPI:t tulee siis määrittää kriittisten liiketoimintatavoitteiden mukaan. (What is a KPI n.d.)

KPI:n määrittämisessä kannattaa vastata seuraaviin kysymyksiin:

- Mikä on määritelty tavoite?
- Miksi tavoitteella on väliä?
- Kuinka mittaat tavoitteen edistymistä?
- Kuinka vaikutat tavoitteen lopputulokseen?
- Kuka on vastuussa tavoitteen toteutumisesta?
- Kuinka tiedät saavuttaneesi tavoitteesi?
- Kuinka usein seuraat tavoitteesi edistymistä? (Mt.)

Sovelluksen suoriutumisen mittaamisessa voidaan hyödyntää GQM-menettelytapaa. GQM on menettelytapa, jota on käytetty sovelluskehityksessä toimintakykyisten ja mitattavien sovellusten määrittämiseen. GQM on akronyymi, joka koostuu käsitteellisestä (Goal), operatiivisesta (Question) ja määrällisestä (Metric) tasosta. (Basili, Caldiera & Rombach n.d.)

Sovellusmarkkinointimetriikoita varten voidaan hyödyntää AARRR-mallia. Se on akronyymi, joka tulee mallin tasoista (ks. kuvio 6) hankinta (Acquisition), aktivointi (Activation), säilyttäminen (Retention), lähete (Referral) ja tulo (Revenue). (Walz 2016.)



Kuvio 6. AARRR-mallin tasot (Walz 2016)

3.10.3 SMART- ja SMARTER-kriteerit

Yksi tapa arvioida määritellyn KPI:n olennaisuus on käyttää SMART-kriteeriä. Lyhenne tulee sanoista Specific (erityinen), Measurable (mitattava), Attainable (saavutettava), Relevant (oleellinen) ja Time-bound (ajallinen). Lyhenteestä on paljon variaatioita, joiden myötä kirjainten merkitykset muuttuvat. Erityyppiset variaatiot sopivat eri käyttötapauksiin, mutta edellä mainitut sopivat hyvin KPI:n määrittämiseen (ks. Taulukko 1). (What is a KPI n.d.)

Taulukko 1. SMART-kriteerin selitykset (Mt.)

Specific	Onko tavoitteesi selkeä?
Measurable	Voiko edistymistä mitata?
Attainable	Onko tavoitteesi saavutettavissa?
Relevant	Kuinka oleellinen tavoitteesi on organisaatiolle?
Time-bound	Millainen aikaraja tavoitteesi valmistumisella on?

SMART-kriteeriä voidaan laajentaa SMARTER-muotoon, joka lisää Evaluate (arviointi) ja Reevaluate (uudelleenarviointi) -vaiheet. Vaiheet ovat erityisen tärkeitä, sillä niillä jatkuvasti määritellään laadittuja KPI:ta uudelleen ja arvioidaan niiden merkitystä liiketoimintaan. (Mt.)

4 Pilvipalvelut

4.1 Määritelmä

”Pilvipalveluilla tarkoitetaan internetin kautta tarjottavia ohjelmistopalveluita ja tietotekniikkaresursseja, jotka saa käyttöönsä eri päätelaitteille, itsepalveluna ja joista maksetaan käytön mukaan.” (Laaksonen 2015).

Pilvipalvelut mahdollistavat organisaation koosta riippumatta laajan teknologiakirjon käyttämisen, joita ennen vain suurimmat yritykset ovat käyttäneet. Siirtyminen pilveen on muuttanut keskeisesti myös ylläpitäjien ja sovelluskehittäjien toimintaa, joten infrastruktuurin jatkuva muutos vaatii uusia työkaluja ja ratkaisuja. Sovellusten kehitys ja ylläpito on mennyt Devops-suuntaan, painottaen CI- ja CD-käytänteiden hyödyntämistä. (Matson & Young n.d., luku 1.)

4.2 Jakelumallit

Pilvipalveluilla on erilaisia jakelumalleja, joita ovat julkinen ja yksityinen pilvi sekä yhteisö- ja hybridipilvi (Laaksonen 2015). Monitoroitavien sovellusten infrastruktuurina

toimii julkisia pilvipalveluita, joiden tarjoajia ovat mm. AWS, Microsoft Azure sekä Google Cloud Platform.

4.3 Pilvipalvelut ja Nordcloud Solutions

4.3.1 Kumppanit

Nordcloudilla on laaja osaaminen johtavien pilvipalvelualustojen ympäristöissä, jonka myötä yhteistyökumppanuuksia on AWS:n, Google Cloud Platformin ja Microsoft Azuren kanssa. Lisäksi on monia teknologiakumppaneita, joihin kuuluvat mm. Datadog ja Sumo Logic. (Partners n.d.)

4.3.2 AWS

AWS on Amazonin vuodesta 2006 lähtien tarjoama julkinen pilvipalvelualusta. Se tarjoaa erittäin luotettavaa, skaalautuvaa ja edullista pilvi-infrastruktuurialustaa, jota käyttävät sadat tuhannet yritykset ympäri maailmaa. (Mathew 2017.)

4.3.3 Microsoft Azure

Azure on Microsoftin tarjoama laaja kokoelma pilvipalveluita, joita kehittäjät ja IT-ammattilaiset käyttävät rakentaakseen, julkaistakseen ja hallitakseen sovelluksiaan. Julkisen pilvialustan lisäksi Azure tarjoaa hybridipilviratkaisuja ja tukee laajaa kirjoa käyttöjärjestelmiä, ohjelmointikieliä, sovelluskehyksiä, tiekantoja sekä laitteita. (What is Azure n.d.)

4.3.4 Google Cloud Platform

Googlen tarjoama julkinen pilvipalvelualusta. Google julkaisi Google App Enginen (PaaS) vuonna 2008 ja on siitä lähtien jatkanut uusien pilvityökalujen sekä -palveluiden julkaisemista. Kokonaisuus sai myöhemmin nimen Google Cloud Platform. (Forrest 2017.)

5 Monitoroitavien sovellusten tilannekuva

5.1 Sovellusten rakenne

Ylläpidossa olevat sovellukset ovat pääosin rakenteeltaan moderneja verkkosovelluksia, jotka ovat jaettu staattisesti jaettaviin, mutta kuitenkin dynaamiseen käyttöliittymiin sekä ohjelmointirajapintoihin. Kunkin sovelluksen käyttöliittymä käyttää ohjelmointirajapintojaan sisällön hakemiseen, jota erinäisin menetelmin esitetään käyttöliittymässä. Sovellukset sijaitsevat AWS:n ja Azuren julkisissa pilviympäristöissä.

Sovellusten frontend on pääasiassa toteutettu JavaScript-sovelluskehyksiä hyödyntäen (React eri versioineen ja AngularJS). Tuetut selainversiot ovat lähinnä ns. evergreen-selaimet (Firefox, Chrome, Safari, Edge) ja Internet Explorer versio 11.

Vastaavasti backend on hiukan kirjavampaa, sillä toteutuksia löytyy niin C#-, JavaScript- kuin Golang-ohjelmointikielillä. Suurinta osaa toteutetuista ohjelmointirajapinnoista yhdistää kuitenkin FaaS-luonne.

Sovellusten versionhallintaan käytetään sovelluksesta riippuen Github, Bitbucket tai Visual Studio Team Services -palveluita. Valtaosassa sovelluksista muutosten julkaisut ovat automatisoituja, jossa versionhallinnan haaraan lisätyt muutokset julkaistaan kehitys- tai tuotantoversioon. Automatisoinnit ovat toteutettu Bitbucket Pipelinesilla, CircleCI -työkalulla tai Visual Studio Team Servicesillä.

5.2 Käytössä olevat monitorointipalvelut ja -työkalut

5.2.1 Työkalut yleisesti

Käytössä olevat työkalut ovat yleisesti ympäristöjen käyttämien alustoiden tarjoamia monitorointityökaluja. Työkalut monitoroivat mm. virhetilanteita, joiden hälyttäessä virhe eskaloidaan ylläpitotiimin sähköpostiin. Sähköpostista hälytykset ohjataan automaattisesti Trello-palveluun. Joissain ympäristöissä seurataan myös tapahtumia, esim. koodimuutosten julkaisut ja CI/CD-putken tilat, joista ilmoitetaan ympäristöjen Slack-kanavilla.

5.2.2 CloudWatch

Amazonin oma monitorointipalvelu CloudWatch on käytössä AWS-ympäristössä sijaitsevissa sovelluksissa. Se seuraa mm. Lambda-ympäristön virheitä, joka lähettää SNS-palvelun kautta sähköpostiviestin sallitun virherajan ylittyessä.

5.2.3 Application Insights

Azuren monitorointipalvelu Application Insights mahdollistaa frontendin ja backendin monitoroinnin. Application Insights on käytössä Azuressa sijaitsevien sovellusten monitoroinnissa.

5.2.4 Datadog

Datadog on kolmannen osapuolen SaaS-pohjainen monitorointipalvelu, joka on käytössä osassa sovelluksista. Datadog seuraa näiden sovellusten backendien tilaa ja ilmoittaa poikkeamista Slack-palvelun ja sähköpostin välityksellä.

6 Työn toteutus

6.1 Infrastruktuuritason monitoroinnin hyödyntäminen

Nordcloudin pilvi-infrastruktuuritasolla monitorointia on harjoitettu jo pidemmän aikaa. Infrastruktuuritasolla on määritelty mm. hälytysten eskalointiprosessi ja monitoroinnin pohjakonfiguraatioita. Infrastruktuuritason toteutuksista saatiin vinkkejä, kuinka sovellustason monitoroinnin toteuttaminen kannattaa toteuttaa ja mitä työkaluja käyttää.

6.2 Monitorointityökalujen vertailu

6.2.1 Vaatimusmäärittely

Monitorointityökalujen vertailua varten laadittiin vaatimusmäärittely. Määriteltyjä vaatimuksia käytettiin työkalujen vertailussa, jonka perusteella rajattiin monitorointitilanteisiin sopivimmat kandidaatit.

Vaatimuksista tehtiin selkeitä, tarkkoja sekä helposti mitattavissa olevia. Vaatimusmäärittelystä päätettiin tehdä dokumentti, jota voidaan jalostaa tarpeen tullen. Toteutuksen aikana päätetyt vaatimukset ovat listattuna alla tärkeysjärjestyksessä.

Turvallisuus

Turvallisuus on tärkeässä osassa valittavaa työkalua. On esimerkiksi erityisen tärkeää, että ylläpitotiimin tunnuksia käyttää oikeasti tiimin jäsen. Vaatimuksena tämän vuoksi on MFA, jolla todennetaan tunnusten käyttäjä, sillä työkalun kautta kulkee paljon sovelluksista tuotavaa dataa. Lisäksi nähtiin, että työkalun tulee toimia push-pohjaisesti, eli esim. lähettää lokidataa työkalun rajapinnan kautta. Tällä rajataan työkalun pääsy sovellusympäristöön (esim. lokiryhmäoikeuksien jakaminen AWS:ssä), jota voi paikoin olla vaikea myydä loppuasiakkaalle. Turvallisuudesta ei myöskään pidä unohtaa EU:n GDPR tietosuoja-asetuksia, joka kattaa käyttäjiin liittyvän datan turvallisen käsittelyn.

Pilvivalmius

Työkalun täytyy olla hyödynnettävissä julkisten pilvipalvelutarjoajien ympäristöissä. Julkisiksi pilvipalvelutarjoajiksi vaatimuksessa rajattiin AWS ja Azure. Työkalun tulee tarjota ylläpidettyjä sovelluskehityspaketteja- tai kirjastoja, jottei käyttöönotossa tarvitse turvautua kolmannen osapuolten tai omiin ratkaisuihin. Tällä vähennetään mahdollisten tietovuoto-ongelmien riskejä sekä taataan kirjastojen kehitys ja ylläpito tulevaisuudessa.

Lisäksi työkalun tulee tukea serverless-ratkaisuja, eli AWS:n tapauksessa ainakin Lambda-, RDS- ja DynamoDB-palveluja. Azuren kohdalla vaadittuja palveluita ovat Azure Functions, Azure SQL ja Azure Table Storage.

Monitorointityypit ja -tilanteet

Työkalun vaatimukset eri monitorointityyppeihin ja -tilanteisiin olivat suppeaksi tarkoituksella määritelty. Pakollisina vaatimuksina työkaluille nähtiin poikkeusten seuranta ja saatavuuden monitorointi.

Ei pakollisena, mutta hyvänä lisänä nähtiin infrastruktuuritason monitorointi, loppukäyttäjän kokemuksen seuranta ja tapahtumien profilointi, komponenttitason monitorointi, raportointi sekä analytiikka.

Käytettävyys

Työkalun pakollisena vaatimuksena nähtiin työkalun integroimisen helppous ylläpidossa oleviin sovelluksiin, tai ylipäätään tavanomaiseen serverless-sovellukseen. Hyvänä lisänä nähtiin, että kaikki ylläpidossa olevat sovellukset olisivat tarkasteltavissa saman palvelun tai työkalun kautta. Lisäksi käytön tulisi olla helppoa, eikä vaatia suurta panostusta ja opettelua käyttöönottovaiheessa ylläpitotiimin toimesta.

Hälytykset

Työkalun tulisi pystyä poikkeustilanteen tapahtuessa lähettämään hälytys sähköpostitse haluttuun sähköpostiosoitteeseen. Työkalun tulisi mahdollistaa myös muiden hälytyksiin liittyvien palveluiden (esim. Pagerduty) integroimisen.

Kuorma

Työkalun ei tulisi tuottaa huomattavaa kuormaa monitoroitavalle sovellukselle.

Hinnoittelu

Työkalusta maksettava hinta tulisi olla kohtuullinen ominaisuuksiin nähden.

6.2.2 Vertailtavat monitorointiratkaisut

Vertailtaviksi ratkaisuiksi valittiin monitoroinnissa suosittuja työkaluja ja Nordcloudin yhteistyökumppaneita. Lisäksi koitettiin valita hieman erityyppisiä ratkaisuja, joilla nähtäisiin kokonaiskuva työvälineiden hintaeroista sekä työmäärästä, mikäli päädyttäisiin toteuttamaan osa monitoroinnista itse (esim. Prometheus-työkalua hyödyntämällä). Vertailtavaksi valituista ratkaisuista osa on valmiita SaaS-palveluita ja osa taas avoimen lähdekoodin ratkaisuja ja niiden yhdistelmiä (ks. taulukko 2).

Taulukko 2. Vaatimusmäärittelytasolla vertailtavat työkalut

Palvelu	**Lyhyt kuvaus**
Sumo Logic	Pilvinatiivi lokitapahtumien ja metriikoiden analysointityökalu (SaaS).
Datadog	Moderni monitorointi- ja analytiikkatyökalu (SaaS).
New Relic	Enemmän suorituskykymonitorointiin keskittynyt työkalu (SaaS).
Elastic Stack	Kokoelma avoimen lähdekoodin tuotteita, jotka kokonaisuutena luovat suositun analytiikkatyökalusetin.
Elastic Cloud	Elasticin ylläpitämä Elastic Stack valmiina SaaS-palveluna.
Dashbird	Serverless-monitorointia tarjoava SaaS -palvelu
Riemann + Graphite	Avoimen lähdekoodin monitorointityökalu.
Prometheus	Avoimen lähdekoodin monitorointityökalu.
GrafanaCloud	Grafana valmiina SaaS-palveluna.

Vertailluista ratkaisuista valittiin vaatimusmäärittelyä hyödyntäen potentiaalisimmat palvelut. Tarkoitus oli vertailla valittuja työkalukandidaatteja testaamalla käyttöönottoa ja suoriutumista ylläpidossa olevaan yksittäiseen sovellukseen. Testattavaksi ympäristöksi valittaisiin sovellus, jonka monitoroinnin onnistumista pidettiin epävarmimpana. Näin pystyttäisiin todentamaan myös muiden sovellusten monitoroinnin instrumentoinnin onnistuminen. Työkalujen vertailun aikana kuitenkin todettiin, että käytetyt työkalut vaativat suhteellisen vapaita oikeuksia toimiakseen pilviympäristössä, joka olisi asiakkaan ympäristön tapauksessa tarkoittanut luvan pyytämistä vertailtavien sovellusten testaamiseen. Tämän myötä päädyttiin testaamaan käyttöönottoa erillisessä serverless-testiympäristössä, jonka käytössä oli pitkälti samat palvelut.

6.3 Monitoroinnin pohjakonfiguraatio

6.3.1 Tavoite

Sovellusten monitorointia varten haluttiin pohjakonfiguraatio, jota voitaisiin hyödyntää sovelluskohtaisen monitoroinnin konfiguroinnissa. Saatu tulos olisi siis dokumentti, joka sisältäisi ohjeita ja esimerkkitapauksia pilvipalvelukohtaisista konfiguraatioasetuksista. Tapaukset määrittelisivät tilanteita, joissa määriteltyjen arvojen alittaessa tai ylittyessä laukaistaisiin hälytys. Pohjakonfiguraatioiden olisi tarkoitus olla yleispäteviä, jotta ne saataisiin toteutettua pienellä vaivalla pilviympäristöstä riippumatta.

6.3.2 Määrittely

Konfiguraation määrittely aloitettiin päättämällä yleismääritykset metriikoille, joita palveluista kerätään. Yleismääritysten kohdalla valittiin ns. kultaisten signaalien kokonaisuus USE- ja RED-määritelmistä ja Googlen SRE-ympäristössä käytetyistä signaaleista.

Määritelmissä on päällekkäisyyttä, joten ylijoukko (superset) määritelmien signaaleista ovat määrä virheet, viive, saturaatio ja käyttö (Mushero 2017). Valituiksi metriikoiksi päädyttiin käyttämään edellä mainittuja metriikoita (ks. taulukko 3). Määrittelyn aikana huomioitiin myös se, ettei jokaisesta palvelusta välttämättä ole jokaista määriteltyä arvoa saatavilla.

Taulukko 3. Valitut metriikat

Metriikka	**Selite**
Määrä (rate)	Kyselyjen määrä (määrä / sekunti)
Virheet (errors)	Virheiden määrä (määrä / sekunti)
Viive (latency)	Vastausaika kokonaisuudessaan, sis. odotus- ja suoritusajan (millisekunneissa)
Saturaatio (Saturation)	Kuinka kuormittunut resurssi on (mitattu esim. jonon pituus tai samanaikaisten suoritusten määrä).
Käyttö (Utilization)	Kuinka kiireinen resurssi on. Usein mitattu prosenttiarvona ja hyödyllisin ennakoimiseen.

Esimerkkitapaukset listattiin palvelukohtaisesti eri monitorointialueen mukaan. Metriikat jaettiin informatiivisiin ja hälytyksiin, joista jälkimmäiseen määriteltiin vakavuusasteet. Vakavuusasteiksi päädyttiin käyttämään yhtenäisiä varoitus ja kriittinen -asteita, jotka ovat infrastruktuurimonitoroinnin käytössä sellaisenaan. Esimerkiksi viiveestä määriteltiin informatiivinen tieto ja varoitus- sekä kriittinen-tason hälytykset (ks. taulukko 4). Kaikki määrittelyt sisältävä konfiguraatiotiedosto löytyy liitteenä (ks. liite 2).

Taulukko 4. AWS Lambdan viiveen esimerkkikonfiguraatio

Metric	**Trigger**	**Severity**	**Action**	**Type**
Latency	Duration (in ms)	-	-	Informative
Latency	Duration exceeds 2000 ms	Warning	Check if the downstream service is making the latency. If the lambda function is the source, check if memory or other setting needs tweaking.	Alert
Latency	Duration exceeds 5000 ms	Critical	Check if the downstream service is making the latency. If the lambda function is the source, check if memory or other setting needs tweaking.	Alert

6.3.3 Normaalitilan määrittäminen

Jotta monitorointi olisi mahdollista, palvelusta on määriteltävä sen normaali käyttäytyminen. Se onnistuu palvelun menneisyydessä mitattua monitorointidataa hyödyntämällä tai mittaamalla palvelun suoritusta useaan kertaan erilaisilla kuormilla. Kokonaisuutena saadaan käsitys palvelun normaalista käyttäytymisestä, jota hyödyntämällä voidaan määrittää halutut asetukset. (Mushero 2017.)

6.4 Pohjakonfiguraation validointi

6.4.1 Validointiympäristö

Validointiympäristöksi päädyttiin käyttämään sovellusta, joka sijaitsee AWS-pilviympäristössä. Sovelluksen tärkeimmissä osissa olevat osat sijaitsevat Lambda- ja RDS-palveluissa, joiden monitorointikonfigurointiin keskityttiin. Validointi rajautui siis AWS-ympäristöön, joten Azure-ympäristön konfiguraation luontia ei nähty tämän toteutuksen puitteissa tarpeellisena. Toteutusta pystyttäisiin toki hyödyntämään Azuren vastaavan konfiguraation luontivaiheessa.

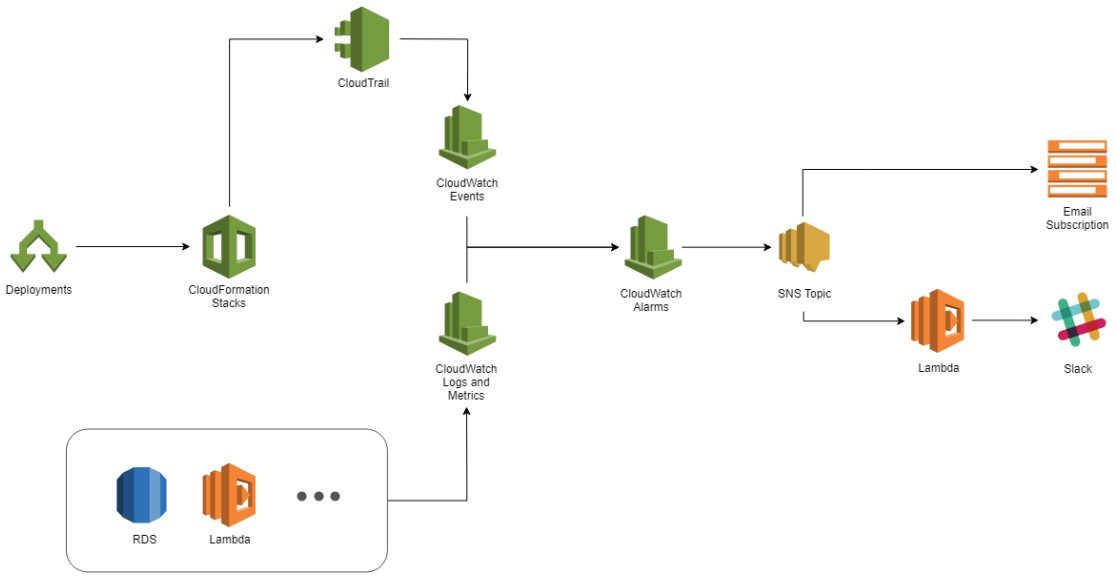
Lambda-komponentit ovat toteutettu Node.js:llä (versio 4.3) ja RDS:n kohdalla käytössä on PostgreSQL-tietokanta. Lambdojen luonnin, konfiguroinnin ja julkaisemisen apuna käytössä on Serverless-sovelluskehys.

AWS-ympäristön vuoksi konfiguraatiot päätettiin tehdä jo valmiiksi käytössä olevaan CloudWatch-monitorointipalveluun. Sovelluksen konfiguraatiot sijaitsevat Serverless-sovelluskehyksen konfiguraatiotiedostossa (serverless.yml), joten monitorointikonfiguraatio oli luonnollista lisätä siihen. Hälytysten luominen konfiguraatiotiedostoon mahdollistaa konfiguraatioiden uudelleenkäytön.

Hälytysten raja-arvojen määrittämiseksi tarvitaan määrittää funktioiden normaalitila. Normaalitila selvitettiin historiadatasta, joka oli saatavilla CloudWatchin Metrics-osiossa. Funktiokohtaista historiadataa katsottiin edeltävältä kahden viikon ajalta. Pidemmän ajanjakson seuraamisella voisi olla negatiivinen vaikutus, sillä ympäristö on jatkuvan muutoksen alla, jotka voivat muuttaa metriikoiden arvoja suurestikin.

6.4.2 Hälytysten rakenne

Cloudwatchin hälytykset rakennettiin seuraamalla palveluista kerättyjä metriikoita, lokitapahtumia ja tapahtumia (ks. kuvio 7).



Kuvio 7. Hälytysten kulku

Metriikoiden (metrics) kohdalla seurattiin Lambda- ja RDS-palveluista tallennettavia oletusmetriikoita ja itse luotuja metriikoita. Metriikoihin yhdistettiin hälytyksiä, jotka hälyttävät metriikoiden ylittäessä asetettuja raja-arvoja.

Monitoroimalla lokitapahtumia (logs) mahdollistetaan Lambdoissa tapahtuvien virheiden seuraaminen. Seuraaminen toteutettiin metric filtereillä, jotka seuraavat määritetyn kaavan toistuvuutta valitun lokiryhmän lokivirrasta. Metric filter luo määritetystä kaavasta metriikan, jonka määrää kasvatetaan kaavan toteutuessa. Virheiden tapahtumista seurattiin kaavalla ”Error”, joka kerää kaikki lokitapahtumiin kirjoitettavat virheet.

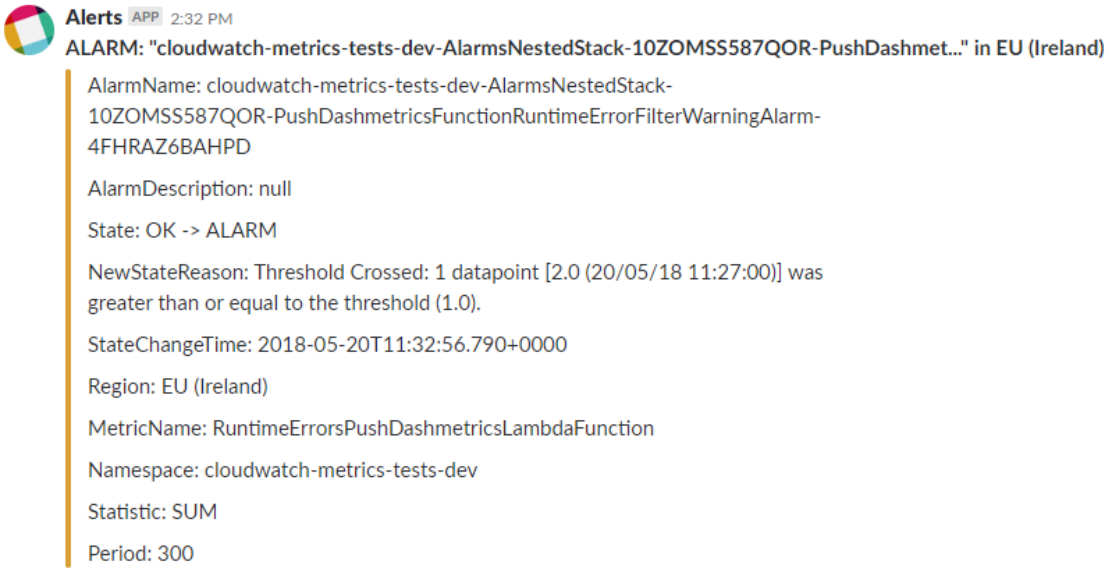
Tapahtumilla (events) voidaan seurata käytetyissä resursseissa tapahtuvia muutoksia. Järjestelmän muutosten julkaisusta (deployment) CloudFormation-palvelun kautta luotiin oma sääntö, joka seuraa ja ilmoittaa StackUpdate-tapahtuman toteutuessa. Säännön luominen lisää myös CloudWatchiin metriikan, jota voidaan hyödyntää esim. ongelmatilanteiden selvittämisessä (korreloiko tutkittavan virheen ajanhetki tehdyn julkaisun vastaavaan).

6.4.3 Hälytysten integrointi

Hälytyksistä luotiin pohjakonfiguraation mukaisesti vakavuusasteet (varoitus ja kriittinen). Vakavuusasteet toteutettiin yhdellä SNS Topicilla, jota seurattiin luoduilla SNS

Subscriptioneilla. Subscriptionit lähettävät SNS Topicin vastaanottaman viestin ylläpitotiimin sähköpostiin ja Lambda-funktion käsiteltäväksi, jossa se lähetetään Slack-palvelun webhookin kautta määritetylle Slack-kanavalle. Sähköpostin lähettäminen rajoitetaan SNS Topicin Subscription Filterillä, joka lähettää viestin ainoastaan kriittinen-tasoisesta hälytyksestä.

Käytössä oleva Lambda-funktio seuraa sille lähetetyn viestin sisältöä, jonka mukaan se luo kanavalle varoitus- tai kriittinen-tasoisen viestin (ks. kuvio 8). Viesti lähetetään webhookin kautta HTTP POST -pyynnöllä.



Kuvio 8. Varoitustason hälytys Slack-ilmoituksena

6.4.4 Itse luodut metriikat

CloudWatchin oletusmetriikoiden lisäksi haluttiin kerätä myös omia metriikoita, joita varten vaadittiin laskutoimituksia ja esim. lambdan konfiguroinnista saatavilla olevia asetuksia. Esimerkiksi yhtä aikaa suoritettavien funktioiden määrää haluttiin verrata määritettyyn maksimiarvoon, josta nähdään sen hetkisen kuormituksen määrä.

Itse luotavien metriikoiden keräämiseksi joudutaan toteuttamaan oma Lambda-funktio, joka kysyy aika ajoin palvelukohtaisia tietoja CloudWatchilta, tekee tarvittavat laskutoimitukset ja lähettää itse määritetyn metriikan takaisin CloudWatchille.

6.4.5 AWS Lambda

Lambda-metriikoiden ja -hälytysten konfiguroinnissa päädyttiin käyttämään Serverless-sovelluskehyksen lisäosaa nimeltä serverless-plugin-aws-alerts.

Lisäosa mahdollistaa yleishälytysten määrittämisen, joka luo jokaiselle Lambda-funktiolle omat hälytykset. Esimerkiksi määrittämällä suoritusaikahälytyksen (ks. kuvio 9), lisäosa luo jokaiselle funktiolle oman versionsa hälytyksestä, joita voidaan tarvittaessa ylikirjoittaa funktiokohtaisesti. Ratkaisu on siinä mielessä tehokas, että hälytyksen ilmoittaessa tiedetään heti mistä komponentista on kyse.

```
customAlarm:
  description: 'My custom alarm'
  namespace: 'AWS/Lambda'
  metric: Duration
  threshold: 200
  statistic: Average
  period: 300
  evaluationPeriods: 1
  comparisonOperator: GreaterThanOrEqualToThreshold
  treatMissingData: notBreaching
```

Kuvio 9. Esimerkki hälytyksen konfiguraatiosta serverless.yml-tiedossa

Lambda-kohtaisten hälytysten lisäksi määritettiin hälytyksiä metriikoille, jotka jaetaan kaikkien alueen (region) funktioiden kesken. Seuraamalla ”UnreservedConcurrentExecutions”-metriikkaa nähdään koko alueen yhtäaikaisten suoritusten määrä. Jotta saataisiin hyvä käsitys kuormasta, luotiin oma metriikka, joka laskee yhtäaikaisten suoritusten määrän suhteessa sen hetkiseen varaamattomien suoritusten maksimimäärään. Mikäli yksittäiselle funktiolle määritetään suoritusten maksimimäärä, varaamattomien yhtäaikaisten suoritusten maksimimäärä muuttuu. Sen vuoksi nähtiin, että on hyvä luoda tästä oma metriikka ja laskea arvo. Yleismetriikoille luotavat hälytykset lisättiin Cloud Formation Template -muodossa, joita serverless.yml-tiedostoon voidaan liittää (ks. kuvio 10).

```
LambdaUnreservedConcurrentExecutionsCritical:
    Type: AWS::CloudWatch::Alarm
    Properties:
        AlarmDescription: UnreservedConcurrentExecutions usage over 90%
        AlarmActions:
          -
            Fn::Join:
              - ""
              - - "arn:aws:sns:"
                - Ref: "AWS::Region"
                - ":"
                - Ref: "AWS::AccountId"
                - ":"
                - cloudwatch-metrics-tests-dev-alerts-alarm
        MetricName: UnreservedConcurrentExecutionsUsage
        Namespace: Custom/Lambda
        Unit: Percent
        Statistic: Maximum
        Period: 60
        EvaluationPeriods: 10
        Threshold: 90
        ComparisonOperator: GreaterThanThreshold
```

Kuvio 10. Hälytyksen määrittäminen Cloud Formation Template -muodossa

Konfiguraation toteutuksen yhteydessä tuli myös toive Source Mapien tuen lisäämiselle, jolla mahdollistettaisiin pakatun tiedoston sijainnin sijoittaminen alkuperäiseen pakkaamattomaan tiedostoon nähden. Source Mapien tuesta olisi hyötyä erityisesti virhetilanteiden selvittämisessä, jolla hahmotettaisiin missä kohdassa pakkaamattomasta tiedostosta virhe on tapahtunut.

Sovelluksessa on käytössä Babel-työkalu, joka pakkaa kaikki käytössä olevat moduulit samaan tiedostoon, joka Lambda-palvelussa suoritetaan. Source Mapien tuki päätettiin toteuttaa Babel-paketin lisäosalla ”babel-plugin-source-map-support”, joka vaati myös ”source-map-support”-paketin asentamisen.

6.4.6 AWS RDS

Serverless-sovelluskehyksen konfiguraatiotiedosto ei suoraan tue RDS:ään liittyvien asetusten toteuttamista, mutta mahdollistaa resources-osion alla CloudFormation Template -asetusten määrittämisen. Templateilla voidaan luoda RDS:stä seurattujen metriikoiden hälytykset.

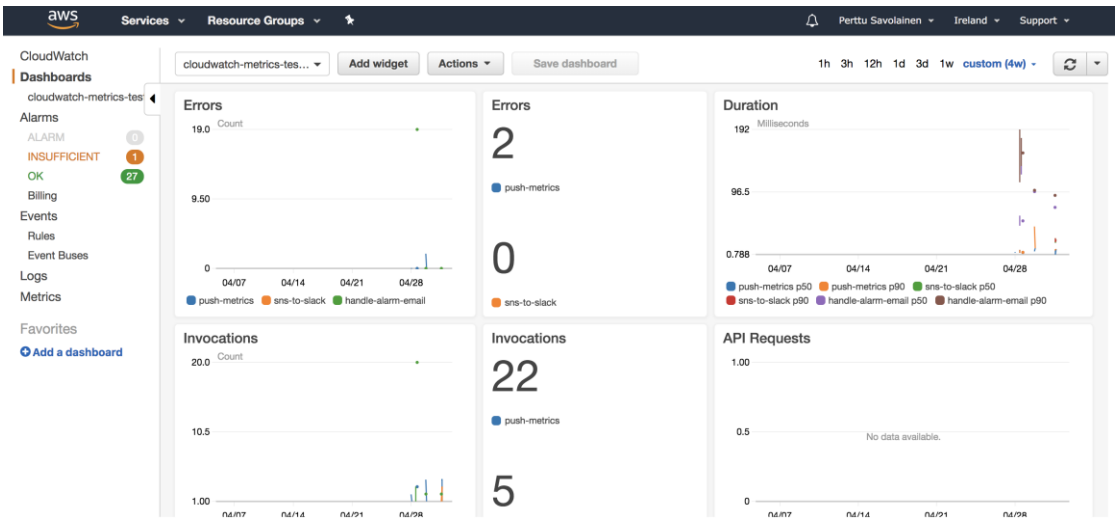
6.4.7 Raportointinäkymät

Määriteltyjen metriikoiden seuraamisen helpottamiseksi päädyttiin tekemään raportointinäkymiä, jotka koostavat palvelukohtaiset metriikat samaan näkymään. Näkymien luonti toteutettiin myös Cloud Formation Template -asetuksilla (ks. kuvio 11).

```
MetricsDashboard:
  Type: AWS::CloudWatch::Dashboard
  Properties:
    DashboardName: example-metrics-dashboard
    DashboardBody:
      '{
       "widgets":[
         {
            "type":"metric",
            "x":0,
            "y":0,
            "Width":24,
            "height":3,
            "properties":{
               "metrics": [
                  [
                    "AWS/Lambda",
                    "Errors"
                  ],
                  [
                    "AWS/Lambda",
                    "RuntimeErrors"
                  ],
```

Kuvio 11. Raportointinäkymän esimerkkikonfiguraatio

Määritellyt näkymät ovat saatavilla AWS:n konsolissa CloudWatch-palvelun Dashboard-osion alla (ks. kuvio 12).



Kuvio 12. Esimerkki luodusta raportointinäkymästä AWS:n konsolissa

7 Tulokset

7.1 Pohjakonfiguraation validointi

Pohjakonfiguraation validointi saatiin toteutettua valittuun ympäristöön. Ympäristön ohjelmointirajapinnat on jaettu toiminnallisuuden mukaan omiin moduuleihin (esim. käyttäjään liittyvät rajapinnat sijaitsevat omassa osiossaan), jonka myötä raportointinäkymät saatiin asetettua myös moduulitasolla. Validointi rajoittui ainoastaan yhteen moduuliin ajankäytöllisten haasteiden vuoksi. Yhdellä moduulilla saatiin kuitenkin pohjakonfiguraation validointi todistettua.

Funktioiden julkaisuvaiheessa kehitysympäristöön huomattiin, että CloudFormationin Stackissä olevien resurssien raja (200 resurssia stackiä kohden) tulee äkkiä vastaan. Jokainen määritelty hälytys käsitellään omana resurssina, joka johtaa helposti satoihin resursseihin. Tähän ratkaisuna oli käyttää Serverless-sovelluskehyksen lisäosaa serverless-plugin-split-stacks. Lisäosa nimensä mukaisesti pilkkoo resurssit pienempiin ryhmiin, jota onkin käytetty usein juuri hälytysten asettamisen yhteydessä.

Omia metriikoita luovan funktion pohja sekä Slack-integraatiofunktio saatiin toteutettua, mutta ne jäivät odottamaan asiakkaan hyväksyntää, sillä ne aiheuttavat pieniä kuluja asiakkaalle. Muihin metriikoihin liittyvät hälytykset saatiin määriteltyä niiden normaalitilojen perusteella. RDS-ympäristössä määritetyt hälytykset toteutettiin staattisilla raja-arvoilla, vaikka alkuperäinen ajatus oli toteuttaa osa itselasketuilla metriikoilla.

Normaalitilat määritettiin, mutta raja-arvojen asettamisen onnistuminen jäi validoimatta. Tällä ei ole toisaalta nähty niin suurta arvoa, sillä tuotantoympäristöön siirryttäessä raja-arvojen kanssa pitää tehdä vielä konfigurointia. Tärkeämpänä nähtiin ymmärtää normaalitilan määrittämisen prosessi.

Ympäristö käyttää paikoin vanhoja paketteja, joka aiheutti työtä Serverless-sovelluskehyksen liitännäisten asentamisessa. Monen paketin päivittäminen johtaa tilanteeseen, jolloin palvelun toiminnallisuus saattaa hajota tai muuttua. Tämä on syytä huomioida ennen tuotantoympäristöön siirtoa.

Source mapien lisääminen ja niiden toimivuuden testaus varmistettiin luomalla virhe, josta nähtiin, toimiiko asennettu paketti halutulla tavalla.

7.2 Vertaillut työkalut

7.2.1 Yleistä

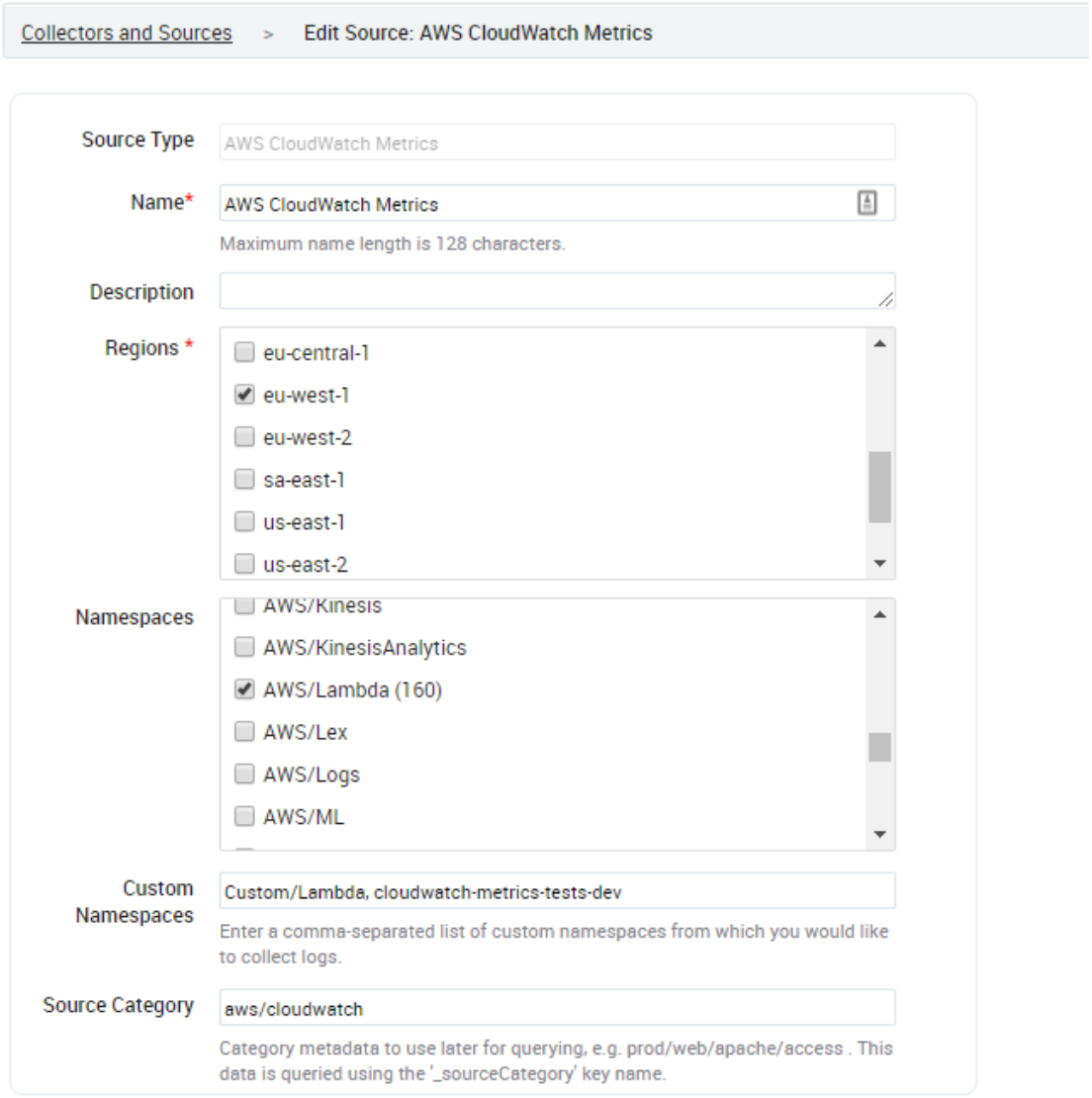
Työkalujen validointi toteutettiin testiympäristöön, joka sisälsi komponentteja Lambda- ja RDS-palveluista. Vertailussa potentiaalisimmiksi työkaluiksi nousivat Sumo Logic, Datadog, New Relic ja GrafanaCloud. Näistä valittiin Sumo Logic, New Relic ja GrafanaCloud tarkempaan testaukseen. Datadog päätettiin jättää vertailusta, sillä se on jo ylläpitotiimin käytössä.

Jokainen tarkemmassa vertailussa ollut sovellus oli SaaS-palvelu. Ne nojasivat metriikoiden keräämiseen tietyn ajan välein (poll-tyyppinen), joka oli vaatimusmäärittelyn vaatimuksen vastaista. Metriikoiden kerääminen vaati oikeuksien luomista AWS-ympäristöön. Tarkemmat vaatimuskohtaiset tulokset liitteenä (ks. liite 1).

7.2.2 Sumo Logic

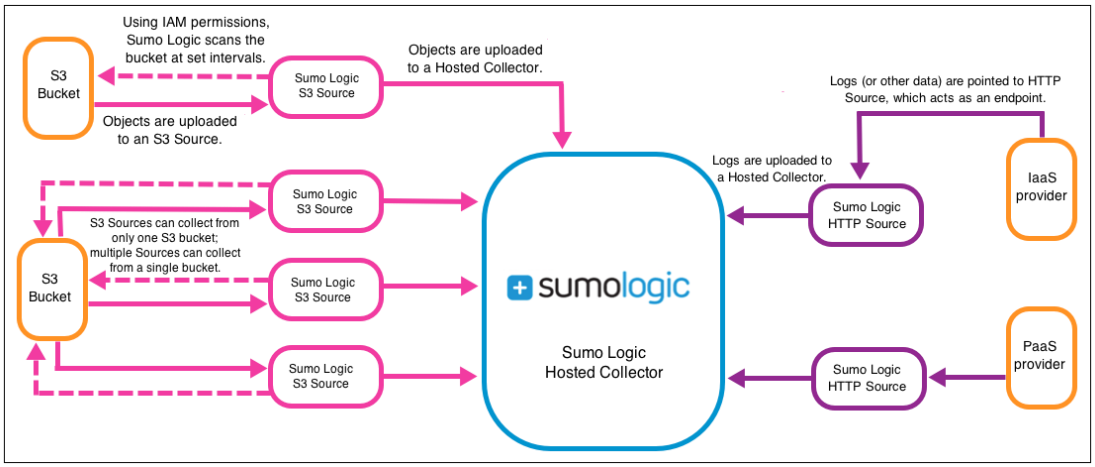
Käyttöönotto

Sumo Logicin käyttöönotto aloitettiin rekisteröitymällä palveluun. Seuraavaksi määritettiin datalähteet metriikoille ja lokidatalle. Metriikoiden seuraaminen toteutetaan Sumo Logicissa siten, että se kerää määritetyn ajan välein dataa asetetusta lähteestä (oletuksena 5 minuutin välein). Metriikoita varten määritettiin seurattavat palvelut ja omat nimiavaruudet (custom namespaces), joilla saadaan oletusmetriikoiden lisäksi omat metriikat talteen (ks. kuvio 13).



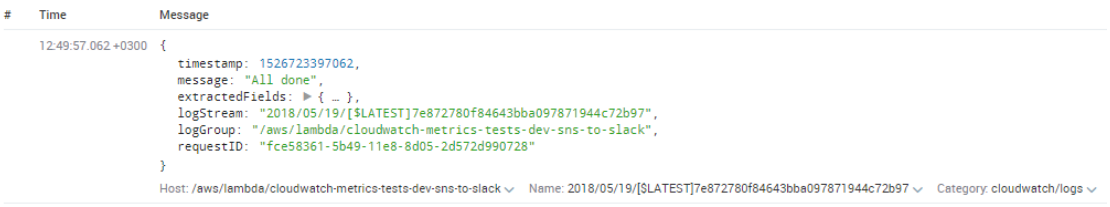
Kuvio 13. AWS CloudWatch metriikoiden määrittäminen Sumo Logiciin

Seuraavaksi määritettiin lokitapahtumien seuraaminen. Sumo Logic suosittelee lokitapahtumien seuraamiseen heidän omaa Lambda-funktiota, joka muuttaa CloudWatch lokitapahtumat Sumo Logicin ymmärtämään muotoon ja lähettää ne määritetyn HTTP-rajapinnan kautta. Datan lähettämistä varten täytyy määritellä kerääjä (Hosted Collector) ja HTTP-lähde (HTTP Source) (ks. kuvio 14).



Kuvio 14. Kerääjän ja lähteen toiminta (Configure a hosted -- 2018)

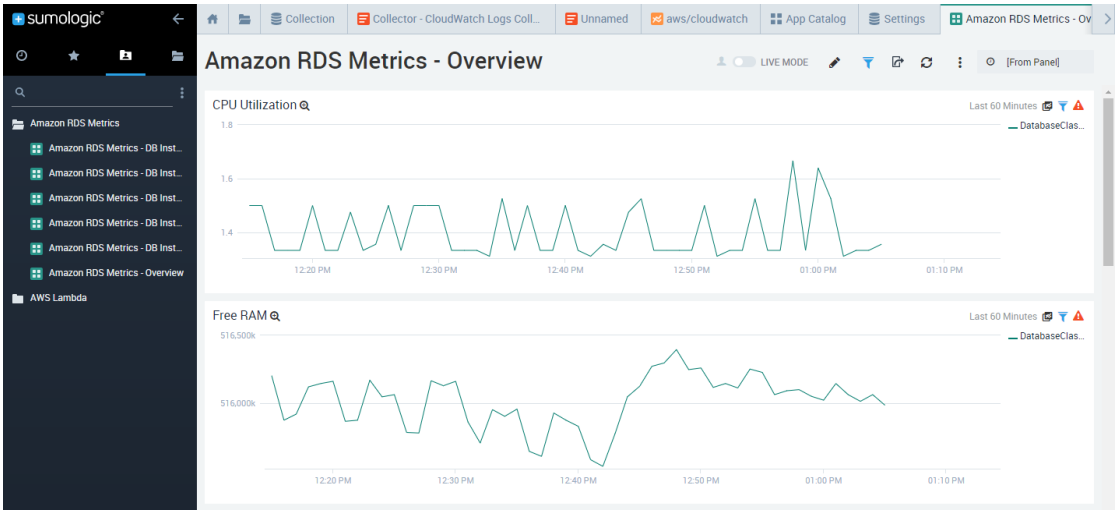
Lambda-funktio määriteltiin Sumo Logicin tarjoamalla CloudFormation Templatella. Template lisättiin CloudFormation-palvelun kautta luomalla uusi CloudFormation Stack ja määrittämällä sille aiemmin luotu HTTP-lähde. Luotuun funktioon yhdistettiin halutut lokiryhmät, jotka haluttiin välittää Sumo Logiciin. Vaiheeseen olisi ollut ohjeet kaikkien lokiryhmien yhdistämisen automatisoimiseksi, mutta testausvaiheessa nähtiin riittäväksi yhdistää ryhmät käsin. Yhdistämisen jälkeen funktio käsittelee ja lähettää sille välitetyt lokitapahtumat Sumo Logiciin (ks. kuvio 15).



Kuvio 15. Lokitapahtuma Sumo Logicissa

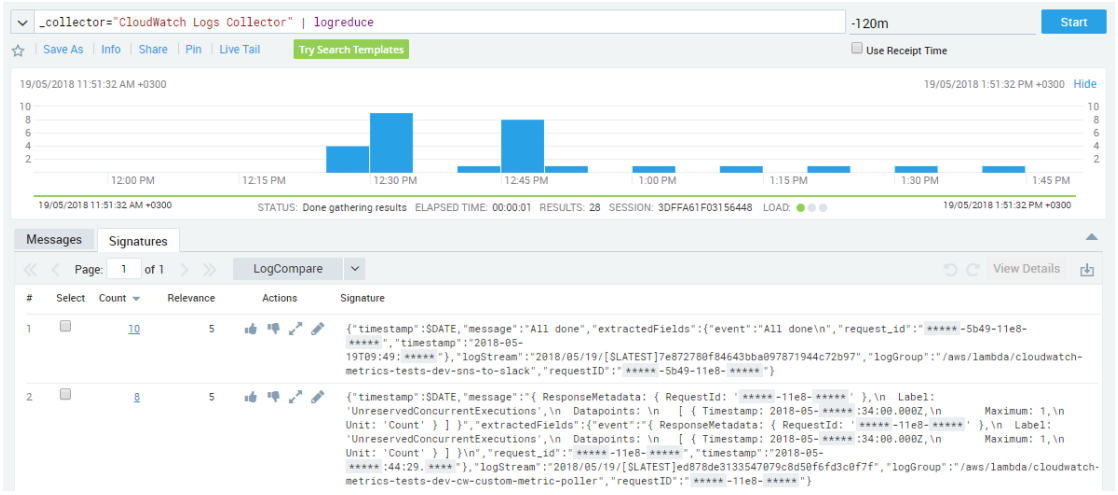
Käyttö

Sumo Logic tarjoaa verkkoselaimella käytettävää käyttöliittymää. Käyttöliittymä on itsessään melko selkeä ja moderni, mutta tarjotessaan paljon ominaisuuksia, voi olla ainakin aloittelijalle paikoin monimutkainen. Näkymät avautuvat omiin välilehtiin, joiden välillä voi liikkua kätevästi. Raportointinäkymiä voidaan luoda joko itse tai käyttää valmiita. Palvelusta löytyy ”App Catalog”, josta löytää valmiita palvelukohtaisia kirjastoja. Kirjastot sisältävät Sumo Logicin komponentteja esim. asentamalla AWS RDS:n kirjaston saadaan RDS-kohtaisia raportointinäkymiä (ks. kuvio 16).



Kuvio 16. Amazon RDS -raportointinäkymä

Lokien ja metriikoiden seuraaminen tapahtuu omalla hakukielellä. Kielellä voidaan ketjuttaa kyselyjä ja käsitellä palautunutta dataa. Mahdollista on myös seurata lähes reaaliaikaisesti metriikoita ja lokitapahtumia Live Tail -tilassa. Log Reduce -ominaisuutta voidaan käyttää lokitapahtumien vertailemiseen ja ryhmittelemiseen (ks. kuvio 17).



Kuvio 17. Sumo Logicin hakutulosten ryhmittely Log Reduce -ominaisuudella

Komponenttien ja hakujen jakaminen onnistuu samaan tiimiin kuuluvien kesken. Lisäksi komponenteista voidaan tehdä myös julkisia, jolloin ne ovat kaikkien nähtävillä.

Hälytykset toteutetaan Sumo Logicissa hakujen perusteella. Määritelty haku voidaan määrittää ajoitetuksi hauksi, jossa voidaan määrittää haku toistumaan esim. jatkuvasti tai vaikka 15 minuutin välein. Määritetyn ehdon toteutuessa voidaan lähettää

hälytys esim. sähköpostin tai webhookin välityksellä Slack-palveluun (ks. kuvio 18). Metriikoiden hälytyksissä voidaan lisäksi määrittää raja-arvot eri vakavuusasteille (kriittinen ja varoitus).

Kuvio 18. Sumo Logicin luoma hälytys Slack-palvelussa

Arvio

Turvallisuuden puolesta Sumo Logic on onnistunut hyvin, sillä se mahdollistaa MFA:n käytön ilman ylimääräistä konfigurointia. Sen lisäksi on mahdollista käyttää push-pohjaista datan siirtoa HTTP API:n kautta, joten ei ole välttämätöntä antaa lukuoikeuksia AWS-ympäristöön. Lokitapahtumien lähettäminen tapahtui määritetyn Lambda-funktion avulla juurikin kyseisen API:n kautta.

Käyttöliittymä on paikoin raskas, varsinkin kun välilehtiä alkaa olla monta. Haettavaa ja käsiteltävää dataa on paikoin paljon, joka ilmenee pitkinä latausaikoina. Monipuolisempia kyselyjä varten on opeteltava datan kyselyihin ja käsittelyyn käytetty oma kieli.

Sumo Logic vertailun potentiaalisin työkalu, Log Reducen ja muiden modernien ominaisuuksien vuoksi.

7.2.3 New Relic

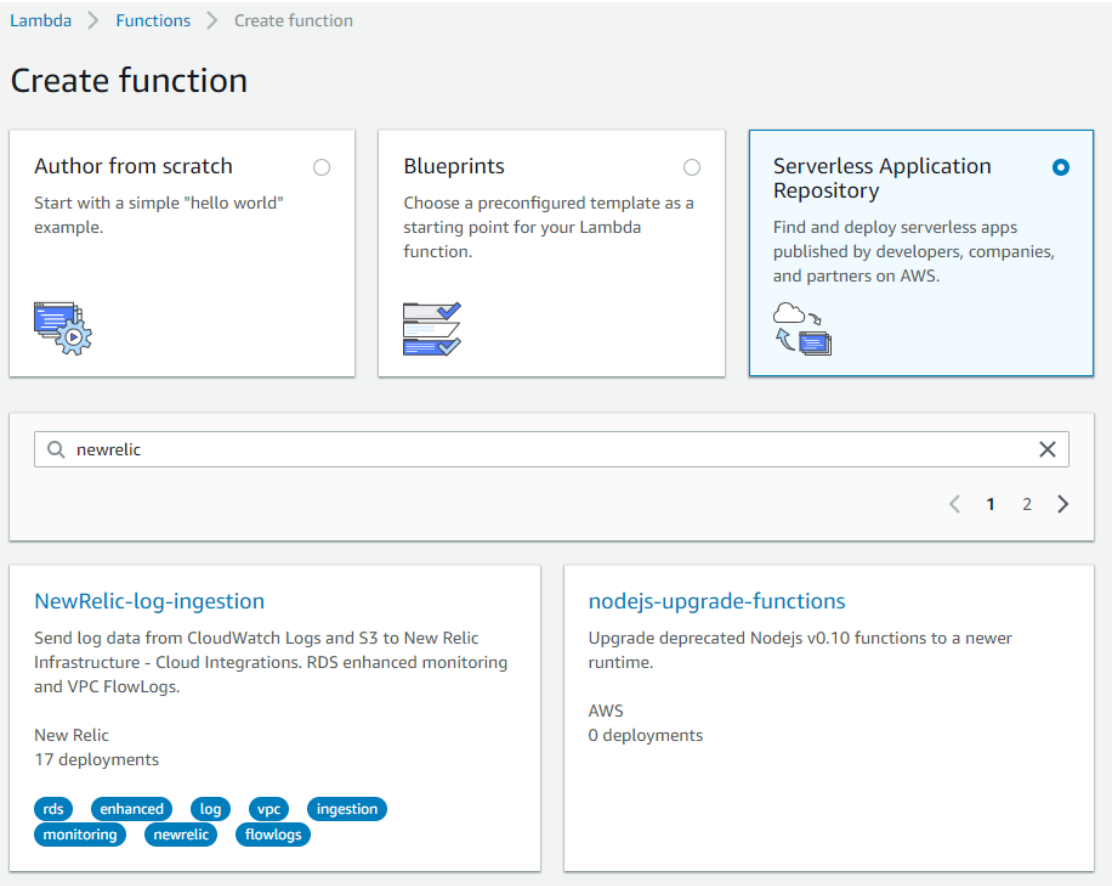
Käyttöönotto

New Relicillä on laaja kirjo eri tilanteisiin sopivia palveluita, mutta serverless-ympäristön monitorointiin tarvittiin New Relic Infrastructure -palvelua.

Infrastruktuuripalvelun kokeilujakson aloittamiseksi oli pakollista asentaa agentti, joka lähettää dataa ympäristön toiminnasta New Relicille. Vaikka tarkoituksena olikin kokeilla käyttöä AWS-ympäristössä, jouduttiin agentti asentamaan. Asennus tehtiin hetkellisesti käyttöön otetulle EC2-instanssille, jonka jälkeen palvelun testaaminen saatiin aloitettua.

Kokeilujakson aktivoimisen jälkeen AWS-palvelun yhdistämiseksi luotiin uusi rooli AWS IAM-palvelussa. New Relic tarvitsee lukuoikeudet monitoroitavien palveluiden metriikoiden ja tapahtumien seuraamiseen. Testausmielessä sallittiin lukuoikeus kaikkiin AWS-palveluihin New Relicin ohjeen mukaisesti. Tämä mahdollistaa esim. Amazon Billing -palvelun hyödyntämisen kulujen seuraamiseen ja AWS CloudTrail -palvelun hyödyntämisen mm. tilikohtaisen tapahtumien tarkkailussa. New Relic hakee oletuksena 5 min välein tietoa AWS-ympäristöstä.

Lokitapahtumat eivät oletuksena ole New Relicissä saatavilla. Käsittelyä varten täytyy tehdä oma Lambda-funktio, joka lähettää lokitapahtumat New Relicille. Funktion pohja löytyy valmiina Lambda-palvelun Serverless Application Repositorystä, joka listaa kehittäjien, yritysten ja kumppaneiden julkaisemia sovelluspohjia (ks. kuvio 19).

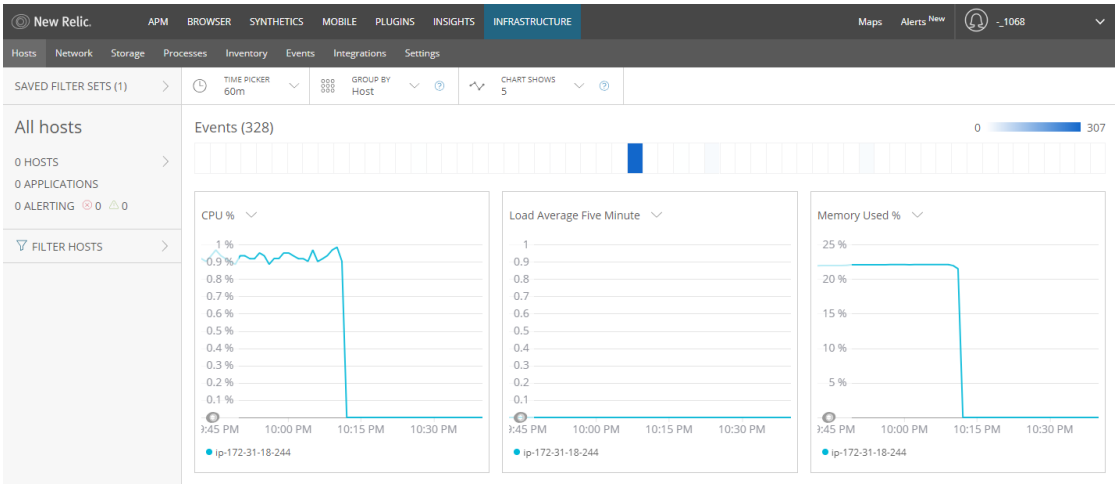


Kuvio 19. Lambda-palvelun Serverless Application Repository

Lambda-funktion käyttöönotossa sille määritettiin vielä New Relicin lisenssi. Lisenssi salattiin AWS Key Management Service -palvelun salausavaimella. Funktiolle piti asettaa salausavaimen käyttöoikeus, jotta salauksen purku onnistuu ohjelmallisesti. Tämän jälkeen lokitapahtumien siirtäminen onnistui AWS:stä New Reliciin.

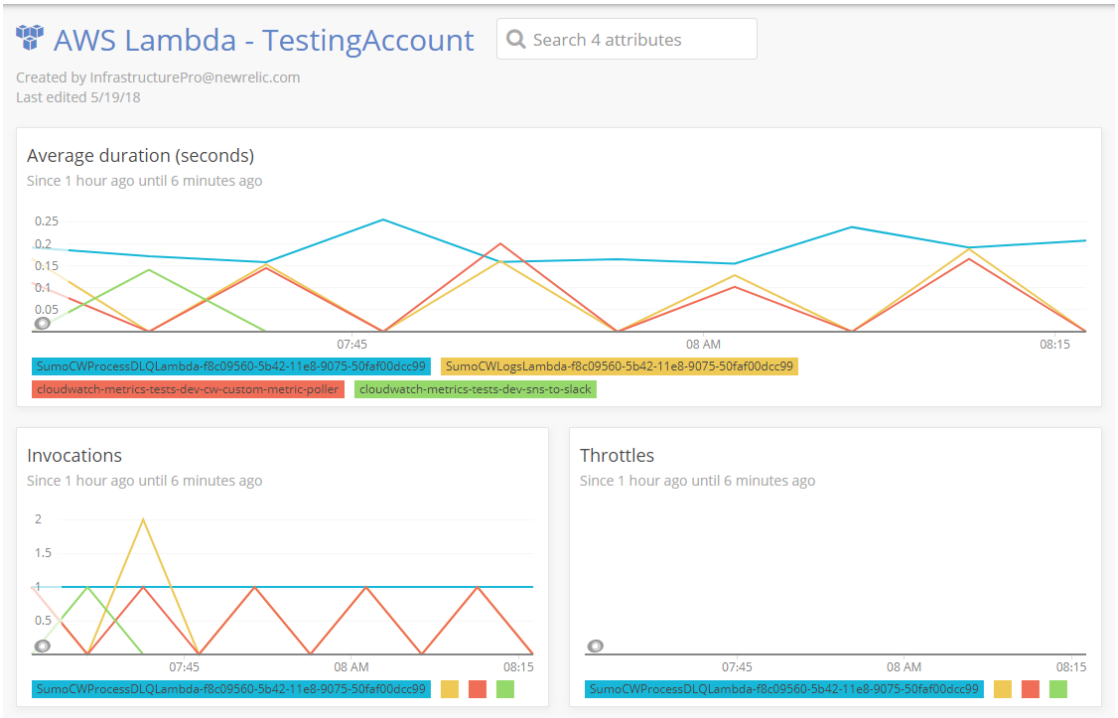
Käyttö

New Relic on SaaS-palvelu, joka tarjoaa käyttöliittymän selaimen kautta. Käyttöliittymä on selkeä ja tarjolla olevat palvelut ovat jaettu omien alasivujen alle (ks. kuvio 20). AWS-infrastruktuurin monitorointia varten käytetään Infrastructure-, Insights- ja Alerts-palveluita.



Kuvio 20. New Relic Infrastructuren käyttöliittymä

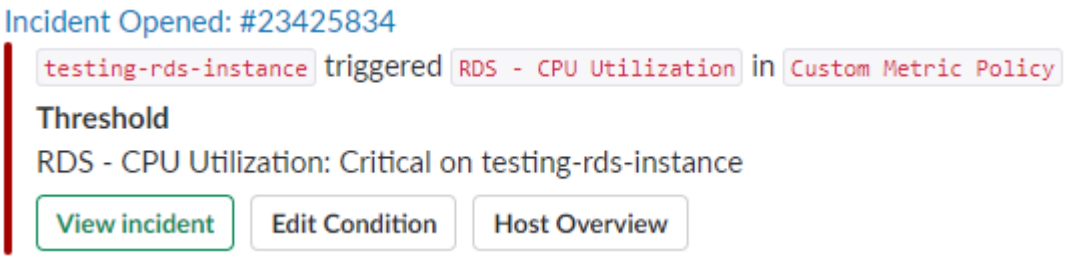
Infrastructure-palvelu näyttää määritettyjen isäntien (host) metriikoita ja tapahtumia (events). Palvelu käyttää asennettavia agentteja, jotka seuraavat monitoroitavien järjestelmien tilaa. AWS-palveluita seuratessa on käytettävä Insights-palvelua. AWS-integraation jälkeen jokaiselle palvelulle on saatavilla oletusraportointinäkymät (ks. kuvio 21) Insights-palvelussa, joita voidaan luoda myös itse lisää. Insights-palvelu sisältää raportointinäkymiä ja hakuominaisuuden, jolla voi tehdä datahakuja New Relicin omalla hakukielellä. Alerts-palvelua puolestaan käytetään hälytysten määrittämiseen ja seuraamiseen.



Kuvio 21. New Relic Insights -raportointinäkymä

Käyttäjien välinen hakutulosten ja niistä luotujen kaavioiden jakaminen tapahtuu ikilinkkien (permalink) avulla. Käyttäjillä täytyy olla pääsy samaan ympäristöön, jotta ikilinkki toimisi.

Hälytysten integrointi Slack-palveluun onnistuu määrittämällä uuden Slack-tyyppisen kanavan (Notification Channel), jonka jälkeen kanavan voi lisätä haluttuihin hälytyksiin. Oletuksena se tarjoaa selkeitä ja linkeillä varustettuja ilmoituksia (ks. kuvio 22).



Kuvio 22. New Relicin luoma hälytys Slack-palvelussa

Arvio

Agentin pakollinen asentaminen oli aika vanhanaikaisen oloinen ratkaisu, mutta se onneksi onnistuttiin kiertämään. Lokitapahtumien käsittelyä varten asennettava funktio oli nopea ja helppo ottaa käyttöön. Ohjeistukset olivat perusteelliset ja niitä oli helppo seurata.

Metriikoiden ja tapahtumien näyttäminen päällekkäin samassa näkymässä on todella hyvä ominaisuus, joka lyhentää MTTR:n aikaa merkittävästi. Harmillisesti ei ole saatavilla Insights-palvelussa, jossa AWS-palvelun kaikki data on nähtävillä. Lisäksi myös New Relicissä monipuolisempia kyselyjä varten on opeteltava datan kyselyihin ja käsittelyyn käytetty oma kieli.

Slack-viestit olivat oletuksena hyvin konfiguroitu. Kokonaisuudessaan New Relic on potentiaalinen kokonaisuus, mutta jää selvästi Sumo Logicin ominaisuuksien taakse.

7.2.4 GrafanaCloud

Käyttöönotto

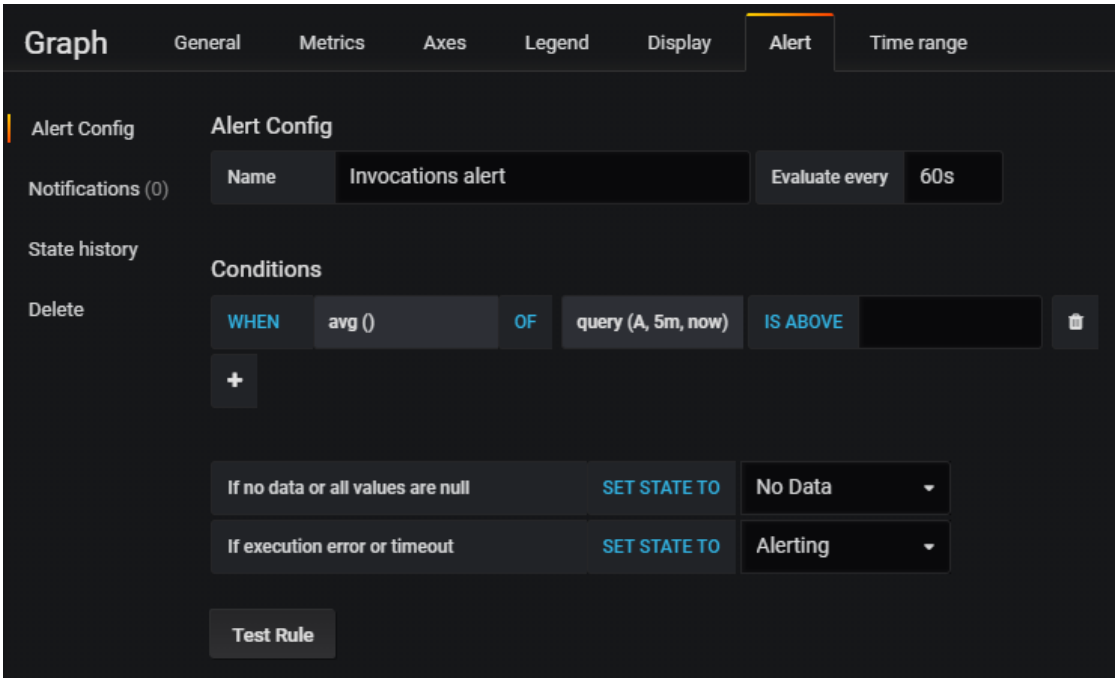
Rekisteröitymisen jälkeen GrafanaCloudin käyttöönotto aloitettiin määrittämällä datalähteet. Datalähdettä varten luotiin käyttäjä ja lisättiin sille rajoitettu lukuoikeus CloudWatch-datalle AWS IAM-palvelussa. Luotu käyttäjä lisättiin GrafanaCloudin datalähteeseen. Seuraavaksi konfiguroitiin raportointinäkymä. GrafanaCloud mahdollistaa yhteisön jakamien dashboardien käytön, josta löytyi pohjat Lambda- ja RDS-palveluille.

Käyttö

GrafanaCloud on yleisnäkymältään ja käyttöliittymältään selkeä, sillä tarjottavien ominaisuuksien määrä on aika vähäinen (ks. kuvio 23). Lokistriimien seuraaminen ei ole tällä hetkellä mahdollista rajoittaen käytön ainoastaan metriikoiden seuraamiseen. Grafana mahdollistaa tiimien ja käyttäjien luonnin sekä näkymien suorien linkkien jakamisen käyttäjien kesken tai jopa julkisesti.

Kuvio 23. Grafanan raportointinäkymä

Hälytykset määritetään Grafanassa raportointinäkymistä löytyvien kaavioiden pohjalta. Kaaviot käyttävät tiettyä kyselyä, johon voidaan määrittää hälytyksen raja-arvot (ks. kuvio 24).



Kuvio 24. Grafanan hälytysten asettaminen

Arvio

GrafanaCloudin kohdalla lokitapahtumien seuraamisen puuttuminen oli suuri puute, sillä yleensä halutaan metriikoiden, hälytysten ja lokitapahtumien olevan saatavilla lähellä toisiaan. Lisäksi hälytysten luonnissa huomattiin, ettei se tue muuttujia hälytysten kyselyissä. Tämä johtaa siihen, ettei hälytyksiä voida luoda dynaamisesti vaan jouduttaisiin luomaan esim. jokaiselle lambda-funktiolle omat hälytykset käsin. Edellisten puutteiden vuoksi nähtiin ettei GrafanaCloudia voida hyödyntää ylläpitotiimin käytössä.

7.3 Jatkokehitys

Jatkokehittämistä jäi melko paljon. Omien metriikoiden käsittely jäi RDS-ympäristön osalta toteuttamatta, mutta pohjan ollessa jo olemassa, ei vaatisi suurta panostusta toteuttaa sitä loppuun.

Ylläpidossa olevissa sovelluksissa on käytössä Route 53 Health Checkit, jotka voitaisiin lisätä Serverless-konfiguraatioon Cloud Formation Templaten muodossa. Lisäksi Cloud Formationin tapahtumien (esim. StackUpdate) ilmoittamisen toteutus jäi validoimatta.

Hälytysten eskaloimista sähköpostitse pitäisi määrittää SNS Subscription Filterillä siten, että ainoastaan kriittisestä hälytyksestä lähetetään sähköposti-ilmoitus. Lisäksi kriittisen rajan ylittyessä tulisi lähettää ainoastaan yksi ilmoitus, nykyisellä toteutuksella tilanne laukaisee molemmat hälytykset (kriittinen ja varoitus).

Lisäksi jo aiemmin mainittu pohjakonfiguraation validointi yhteen moduuliin pitäisi toteuttaa kokonaisvaltaisesti kaikkiin moduuleihin. Toisaalta tämäkään ei ole suuri työ, mutta ympäristön muiden pakettien päivitysten vuoksi vaatii toimivuuden todentamista ennen tuotantoympäristöön julkaisemista.

8 Johtopäätökset ja pohdinta

8.1 Haasteet

Tutkimuksen suurimpana haasteena oli sisällön rajaaminen, sillä monitorointi on aiheena valtava. Tämä johti ajallisiin ongelmiin, sillä aiheeseen tutustumiseen saattoi käyttää paljonkin aikaa lopulta todeten, että se on rajattava tutkimuksen ulkopuolelle. Lisäksi tavoitteiden asettaminen kahteen osaan (työkalujen vertailu ja pohjakonfiguraation luonti) vaikutti asiaan keskittymiseen. Näistä toisen valitseminen ja sen toteuttaminen olisi itsessään riittänyt opinnäytetyön aiheeksi. Toisaalta tutkimuksen alussa ei oikein osannut arvioida, kuinka paljon aikaa kukin asia voikaan viedä.

Aiheen teoriaosuudessa haasteita aiheutti monitorointiin liittyvä sanasto. Sanastolla ei ollut juuri yhteistä linjaa, joten jokainen palveluntarjoaja tai aiheeseen liittyvä artikkeli käytti omaa sanastoaan. Esimerkiksi APM saattoi lähteestä riippuen tarkoittaa sovellussuorituskyvyn ylläpitoa tai sen monitorointia.

Työkalujen vertailusta saatujen tulosten luotettavuus on sinällään oma haasteensa, sillä empiirinen osuus pohjautui tutkimuksen tekijän omaan arviointiin ja mielipiteeseen.

Haasteena työkalujen validoinnille toi testausympäristö ja oikeudet. Testausympäristö ei vastannut datamäärältään yleisten sovellusten tuotantoversioita, joten työkalujen testaaminen oli paikoin rajoittunutta. Lisäksi vaatimusmäärittelyssä määritelty oikeuksien rajaaminen (lokitapahtumat lähetetään työkalulle rajapinnan kautta) aiheuttaa monitoroinnissa haasteita, sillä se estää samalla myös metriikan seuraamisen. Jokainen testattu työkalu nojautui siihen, että sille annetaan lukuoikeus metriikkaan ja tapahtumiin. Kaiken datan lähettäminen työkalun rajapinnan kautta vaikuttaisi tuhoon tuomitulta ratkaisulta, joka aiheuttaisi monitoroinnin instrumentoijalle suuren määrän työtä. Ilman oikeuksia työkalujen käyttö jäisi siis hyvin laihaksi, eikä niitä olisi suositeltavaa hyödyntää.

Pohjakonfiguraation validointi aiheutti myös haasteita. Validointiympäristönä toimi sovelluksen kehitysympäristö, joka hankaloitti normaalitilan määrittämistä. Kehitysympäristössä määritetty normaalitila on korkeintaan suuntaa antava, joten konfiguraation siirryttyä tuotantoympäristöön se on todennäköisesti konfiguroitava osin uudestaan.

8.2 Onnistumiset

Työkalujen arvioinnissa käytetty vaatimusmäärittelyn suunnittelu onnistui hyvin, joka tutkimuksen aikana validoitiinkin ylläpitotiimin kesken. Läpikäynnin yhteydessä saatiin korjausehdotuksia, joiden pohjalta vaatimusmäärittelystä saatiin halutunlainen.

Tutkimuksen pohjakonfiguraation suunnittelu ja sen validoinnin toteuttaminen onnistui hyvin. Vaikka validoinnin toteuttaminen jäikin hieman puolitiehen, se vastasi kysymykseen pohjakonfiguraation toimivuudesta. Konfiguraation nojaaminen kultaisiin signaaleihin oli onnistunut päätös. Validointivaiheessa saatiin konfiguraation mukaiset perushälytykset asetettua ja samalla käsitys siitä, mitä vielä olisi suositeltavaa toteuttaa.

Kokonaisuuteena onnistumisena voitaisiin nähdä ylläpitotiimin osallistuttaminen tutkimuksen tärkeimmissä vaiheissa. Tiimiläisiltä saatiin palautetta ja tukea askarruttaviin tilanteisiin.

8.3 Soveltuvuustutkimuksen johtopäätökset

Soveltuvuustutkimuksen johtopäätöksenä voidaan todeta, että systemaattinen sovellusten monitorointi on mahdollista toteuttaa ja siihen soveltuvia työkaluja on saatavilla. Hyviä työkaluja on monipuolisesti, mutta myös pilvialustojen omat työkalut tekevät tehtävänsä. Mielestäni niinkin hyvin, että suosittelen niiden käyttämistä ensisijaisena ratkaisuna. AWS-ympäristön tapauksessa lokitapahtumien ja metriikoiden kerääminen toteutetaan aina CloudWatchin kautta, oli käytössä kolmannen osapuolen sovellus tai ei. Käyttämällä ympäristön monitorointiratkaisua vältytään vaikeilta kysymyksiltä, kuten esimerkiksi onko erilliseen työkalulle lähetettävä ja tallennettava data varmasti säilötty turvallisesti (GDPR:n mukaisesti).

Ainakin AWS:n tapauksessa monitorointikonfiguraatiot voidaan tehdä ohjelmallisesti Serverless-sovelluskehyksen tai Cloud Formation Templaten konfiguraatioihin. Tällä mahdollistetaan konfiguraatioiden uudelleenkäyttö muissa projekteissa, jotka käyttävät konfiguroituja palveluita. Tällä ratkaisulla monitoroinnin instrumentoija voi säästää aikaa ja keskittyä normaalitilan määrittämiseen ja raja-arvojen asettamiseen.

Lähteet

Basili, V. R., Caldiera, G. & Rombach, H. D. N.d. Julkaisu GQM -menettelytavasta. Viitattu 15.5.2018. http://www.cs.umd.edu/~mvz/handouts/gqm.pdf

Chang, E. 2017. Monitoring services and setting SLAs with Datadog. Blogikirjoitus SLA:n määrittelystä ja käytöstä Datadogin verkkosivulla. Viitattu 20.4.2018. https://www.datadoghq.com/blog/set-and-monitor-slas/

Configure a Hosted Collector. 2018. Ohjeet kerääjän konfiguroimiseen Sumo Logicin virallisella ohjesivulla. Viitattu 20.5.2018. https://help.sumologic.com/Send-Data/Hosted-Collectors/Configure-a-Hosted-Collector

Creating Amazon Route 53 Health Checks and Configuring DNS Failover. N.d. AWS-pilvipalvelun virallinen Route 53 -palvelun ohjesivu. Viitattu 15.5.2018. https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-failover.html

Dorsey, T. 2015. 11 Code Profiling and Performance Tools for Visual Studio. Artikkeli koodiprofiloinnin työkaluista Visual Studio Magazinen verkkosivulla. https://visualstudiomagazine.com/articles/2015/07/01/11-code-profiling-and-performance-tools.aspx

Dragich, L. 2012. The Anatomy of APM – 4 Foundational Elements to a Successful Strategy. Viitattu 14.12.2017. http://www.apmdigest.com/the-anatomy-of-apm-4-foundational-elements-to-a-successful-strategy

Ewaschuk, R. & Beyer, B. N.d. Monitoring Distributed Systems. Viitattu 11.11.2017. https://landing.google.com/sre/book/chapters/monitoring-distributed-systems.html

Forrest, C. 2017. Google Cloud Platformin historiaa ja palveluita käsittelevä artikkeli Techrebublicin sivulla. Viitattu 15.4.2018. https://www.techrepublic.com/article/google-cloud-platform-the-smart-persons-guide/

Gasparyan, A. 2017. Synthetic Transaction Monitoring Vs. Real User Monitoring: When To Use Each Approach? Viitattu 4.1.2018. https://www.monitis.com/blog/synthetic-transaction-monitoring-vs-real-user-monitoring-when-to-use-each-approach/

Holmwood, L. 2014. Applying cardiac alarm management techniques to your on-call. Viitattu 3.1.2018. http://fractio.nl/2014/08/26/cardiac-alarms-and-ops/

ITIL incident management. N.d. Ohje häiriöiden hallintaprosessista BMC:n verkkosivuilla. Viitattu 17.1.2018. http://www.bmc.com/guides/itil-incident-management.html

Jones, C., Wilkes, J., Murphy, N., Smith, C. & Beyer, B. N.d. Service Level Objectives. Googlen SRE -kirjan kappale palvelutasotavoitteista. Viitattu 20.4.2018. https://landing.google.com/sre/book/chapters/service-level-objectives.html

Jones, D. 2013. What You Should Know About Application Performance Management – Part 1. Viitattu 14.12.2017. https://web.archive.org/web/20131214140935/http://nexus.realtimepublishers.com/content/?tip=what-you-should-know-about-application-performance-management-part-1

Kautto, M. 2009. Palvelutasosopimukset (SLA). Viitattu 14.12.2017. https://www.cs.helsinki.fi/group/cinco/teaching/2009/soc-seminaari/abstracts/kautto_abstract.pdf

Key Performance Indicators. N.d. Artikkeli KPI:n määrittämisestä ja käytöstä. Viitattu 22.2.2018. http://www.applicationperformancemanagement.org/performance-testing/key-performance-indicators/

Kowall, J. 2014. Application Monitoring is not Application Performance Monitoring (APM). Viitattu 11.1.2018. https://blogs.gartner.com/jonah-kowall/2014/02/14/application-monitoring-is-not-application-performance-monitoring-apm/

Kraft, J. 2017. Artikkeli jäljityksestä ElectronicDesignin verkkosivulla. Viitattu 15.5.2018. http://www.electronicdesign.com/test-measurement/11-myths-about-software-tracing

Laaksonen, A. 2015. Pilvipalvelut pähkinänkuoressa. Viitattu 16.11.2017. http://www.pilveen.net/2015/09/pilvipalvelut-pahkinankuoressa.html

Leader in native cloud applications. 2017. Yrityksen esittely verkkosivulla. Viitattu 16.11.2017. https://sc5.io

Mathew, S. 2017. Yleiskatsaus AWS:n palveluista. Viitattu 14.4. https://docs.aws.amazon.com/aws-technical-content/latest/aws-overview/introduction.html

Matson, J. 2016. Metric graphs 101: Timeseries graphs. Blogikirjoitus monitorointidatan visualisoinista Datadogin verkkosivulla. Viitattu 23.1.2018. https://www.datadoghq.com/blog/timeseries-metric-graphs-101/

Matson, J. & Young, K. N.d. Monitoring modern Infrastructure. Datadogin julkaisema PDF muodossa oleva e-kirja.

Mushero, S. 2017. How to Monitor the SRE Golden Signals. Julkaisu Mediumin verkkosivustolla. Viitattu 17.4.2018. https://medium.com/devopslinks/how-to-monitor-the-sre-golden-signals-1391cadc7524

Nordcloudista ja SC5:stä Euroopan johtava pilvinatiivien IT-palvelujen tarjoaja. 2017. Julkaisu SC5:n sivulla. Viitattu 16.11.2017. https://sc5.io/posts/nordcloudista-ja-sc5sta-euroopan-johtava-pilvinatiivien-it-palvelujen-tarjoaja/#gref

Part III. Practices. N.d. E-kirja Googlen sivulla. Viitattu 11.11.2017. https://landing.google.com/sre/book/chapters/part3.html

Partners. N.d. Pilvialusta- ja teknologiakumppanit Nordcloudin verkkosivulla. Viitattu 20.5.2018. https://www.nordcloud.com/partners

Pilvipalvelijat yhteen - tähtäävät Euroopan ykköseksi. 2017. Uutinen Kauppalehden sivulla. Viitattu 16.11.2017. https://www.kauppalehti.fi/uutiset/pilvipalvelijat-yhteen---tahtaavat-euroopan-ykkoseksi/yD7Ye7WD

Rouse, M. N.d. Application performance monitoring (APM). Viitattu 6.1.2018. http://searchenterprisedesktop.techtarget.com/definition/Application-monitoring-app-monitoring

Rubin, A. N.d. Trendianalyysi tulevaisuudentutkimuksen menetelmänä. Viitattu 4.1.2018. https://tulevaisuus.fi/menetelmat/toimintaympariston-muutosten-tarkastelu/trendianalyysi-tulevaisuudentutkimuksen-menetelmana/

Sloss, B. & Beyer, B. N.d. Introduction. Viitattu 11.11.2017. https://landing.google.com/sre/book/chapters/introduction.html

SOASTA Real User Monitoring Best Practices. N.d. Dokumentti loppukäyttäjän monitoroinnista Soastan verkkosivulla. Viitattu 5.1.2017. https://www.soasta.com/wp-content/uploads/2014/09/mPulse-Whitepaper1.pdf

Soveltavat tutkimusmenetelmät. N.d. Ohje Jyväskylän ammattikorkeakoulun verkkosivulla. Viitattu 16.11.2017. https://oppimateriaalit.jamk.fi/yamk-kasikirja/soveltavat-tutkimusmenetelmat/

Turnbull, J. 2016. The art of monitoring. Kindle-lukulaitteella luettava e-kirja.

Values That You Specify When You Create or Update Health Checks. N.d. AWS-pilvipalvelun virallinen Route 53 -palvelun ohjesivu. Viitattu 15.5.2018. https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/health-checks-creating-values.html

Walz, A. 2016. App Marketing Metrics for Pirates: Commandeering the Purchase Funnel. Blogikirjoitus sovellusmarkkinoinnin metriikoista. Viitattu 19.4.2018. https://www.apptentive.com/blog/2016/03/29/app-marketing-metrics-for-pirates/

Watson, M. 2017. What Is Application Performance Monitoring and Why It Is Not Application Performance Management. Viitattu 11.1.2018. https://stackify.com/what-is-application-performance-monitoring/

What is AWS X-Ray. N.d. Kehittäjille suunnattu X-Ray -palvelun virallinen ohjesivu. Viitattu 19.4. https://docs.aws.amazon.com/xray/latest/devguide/aws-xray.html

What is Azure? N.d. Microsoft Azuren virallinen verkkosivusto. Viitattu 14.4.2018. https://azure.microsoft.com/en-gb/overview/what-is-azure/

What is a KPI. N.d. Artikkeli Klipfolion sivulla. Viitattu 11.1.2018. https://www.klipfolio.com/resources/articles/what-is-a-key-performance-indicator

What is Application Performance Management? Overview, Common Terms, and 10 Critical APM Features. 2015. Artikkeli Stackifyn sivulla. Viitattu 4.1.2018. https://stackify.com/what-is-apm/

What is code profiling? Learn the 3 Types of Code Profilers. 2016. Artikkeli Stackifyn sivulla. Viitattu 7.3.2018. https://stackify.com/what-is-code-profiling/

What is ELK Stack. N.d. Elasticin esittelysivu ELK-stackille. Viitattu 31.3.2018. https://www.elastic.co/elk-stack

What is Real User Monitoring? How It Works, Examples, Best Practices, and More. 2017. Artikkeli Stackifyn sivulla. Viitattu 4.1.2018. https://stackify.com/what-is-real-user-monitoring/

Wright, N. N.d. Nailing the incident management process. Artikkeli Atlassianin verkkosivulla. Viitattu 18.1.2018. https://www.atlassian.com/it-unplugged/itsm/incident-management-process

Young, K. 2015. The Docker monitoring problem. Blogikirjoitus Datadogin verkkosivulla. Viitattu 12.4.2018. https://www.datadoghq.com/blog/the-docker-monitoring-problem/

Liitteet

Liite 1. Pohjakonfiguraation määrittelytiedosto

Application monitoring baseline

Tips for determining a baseline and monitoring

Establish a baseline for normal performance in your environment, by measuring performance at various times and under different load conditions.

Store historical monitoring data. It will give you a baseline to compare against with current performance data, identify normal performance patterns and performance anomalies, and devise methods to address issues. Monitoring is a living thing, it shapes continuously. Don't be afraid to tweak things.

For monitoring golden signals, we use a superset of SRE Golden signals, USE and RED:

Rate — Request rate, in requests/sec
Errors — Error rate, in errors/sec
Latency — Response time, including queue/wait time, in milliseconds.
Saturation — How overloaded something is, which is related to utilization but more directly measured by things like queue depth (or sometimes concurrency). As a queue measurement, this becomes non-zero when you are saturated, often not much before. Usually a counter.
Utilization — How busy the resource or system is. Usually expressed 0–100% and most useful for predictions (as Saturation is probably more useful). Note we are not using the Utilization Law to get this (~Rate x Service Time / Workers), but instead looking for more familiar direct measurements.

Typical settings for specific environment are listed in the following sections.

Backend monitoring

AWS

Lambda

Metric	Trigger	Severity	Action	Type
Rate	Invocations / min	-	-	Informative
Errors	Errors* / Invocations ratio exceeds x for one min	Critical	Check for memory, timeout and permission errors. Runtime errors shouldn't be here (see Handled runtime errors section)	Alert
Latency	Duration exceeds x milliseconds	Critical	Check if the downstream service is making the latency. If lambda is the source, check if memory etc needs tweaking.	Alert
Latency	Duration exceeds x milliseconds	Warning	Check if the downstream service is making the latency. If lambda is the source, check if memory etc needs tweaking.	Alert
Saturation	Concurrent-Executions / min	-	-	Informative
Utilization	Concurrent-Executions / limit** x 100 exceeds 90	Critical	Check the source, what is creating concurrent events? Possible bug? If limit is set really tight, scale it.	Alert
Utilization	Concurrent-Executions / limit** x 100 exceeds 80	Warning	Check the source, what is creating concurrent events? A possible bug? If limit is set to be really tight, scale it.	Alert

* Errors includes handled exceptions, unhandled exceptions, out of memory exceptions, timeouts, permission errors

** Function Level Concurrent Execution Limit.

Good to know

- AWS Lambda sends metrics to CloudWatch in one minute intervals.
- Throttling behavior: On reaching the concurrency limit associated with a function, any further invocation requests to that function are throttled, i.e. the invocation doesn't exe-

cute your function. Each throttled invocation increases the Amazon CloudWatch Throttles metric for the function. AWS Lambda handles throttled invocation requests differently, depending on their source:

  - Event sources that aren't stream-based:
    - synchronous invocation: Lambda returns 429 error and the invoking service is responsible for retries. Each service may have its own retry policy. For example, CloudWatch Logs retries the failed batch up to five times with delays between retries.
    - asynchronous invocation: AWS Lambda automatically retries the throttled event for up to six hours, with delays between retries. Remember, asynchronous events are queued before they are used to invoke the Lambda function.
  - Stream-based event sources:
    - For stream-based event sources (Kinesis and DynamoDB streams), AWS Lambda polls your stream and invokes your Lambda function. When your Lambda function is throttled, Lambda attempts to process the throttled batch of records until the time the data expires. This time period can be up to seven days for Kinesis.

- More here https://docs.aws.amazon.com/lambda/latest/dg/concurrent-executions.html
- If you haven't set up any function-level concurrency limit, then the unreserved concurrency limit is the same as the account level concurrency limit. For more, see https://docs.aws.amazon.com/lambda/latest/dg/concurrent-executions.html

For full metrics list, see https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/lam-metricscollected.html

Handled runtime errors

For handling runtime errors, we use metric filters. See https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/FilterAndPatternSyntax.html

RDS

Metric	Trigger	Severity	Action	Type
Rate	ReadIOPS (count / second)	-	-	Informative
Rate	WriteIOPS (count / second)	-	-	Informative
Latency	ReadLatency exceeds x seconds	Warning	Check history, scale if needed	Alert
Latency	WriteLatency exceeds x seconds	Warning	Check history, scale if needed	Alert
Saturation	DiskQueueDepth exceeds x	Warning	Check history, scale if needed	Alert
Saturation	DiskQueueDepth exceeds x	Critical	Check history, scale if needed	Alert
Utilization	CPUUtilization (in percents)	-	-	Informative
Utilization	CPUUtilization exceeds 80 percents for 5 minutes	Warning	Check history, scale if needed	Alert
Utilization	CPUUtilization exceeds 90 percents for 5 minutes	Critical	Check history, scale if needed	Alert
Utilization	FreeableMemory (Bytes)	-	-	Informative
Utilization	FreeableMemory / max memory x 100 exceeds 80 for 5 minutes	Warning	Check history, scale if needed	Alert
Utilization	FreeableMemory / max memory x 100 exceeds 90 for 5 minutes	Critical	Check history, scale if needed	Alert
Utilization	DatabaseConnections	-	-	Informative
Utilization	DatabaseConnections / max connections x 100 exceeds 80 for 5 minutes	Warning	Check history, scale if needed. Check the connection sources, e.g. a bug might leave open connections.	Alert

Utilization	DatabaseConnections / max connections x 100 exceeds 90 for 5 minutes	Critical	Check history, scale if needed. Check the connection sources, e.g. a bug might leave open connections.	Alert
Utilization	FreeStorageSpace	-	-	Informative
Utilization	FreeStorageSpace / max space x 100 exceeds 85	Warning	Investigate disk space consumption. See if it is possible to delete data from the instance or archive data to a different system to free up space. Is Logrotate possible?	Alert
Utilization	FreeStorageSpace / max space x 100 exceeds 95	Critical	Investigate disk space consumption. See if it is possible to delete data from the instance or archive data to a different system to free up space. Is Logrotate possible?	Alert

Good to know

- Amazon Relational Database Service sends metrics to CloudWatch for each active database instance every minute.
- Note that the Amazon RDS Service Level Agreement requires that you follow their guidelines, see the list of guidelines and other recommendations here https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_BestPractices.html
- For full metrics list, see https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/rds-metricscollected.html
- Check also enhanced monitoring https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_Monitoring.OS.html

DynamoDB

Metric	Trigger	Severity	Action	Type
Rate	SuccessfulRequestLatency (use SampleCount to get number of requests) / min	-	-	Informative
Latency	SuccessfulRequestLatency exceeds X ms	Warning	Is the provisioned read / write throughput set properly? Scale if needed.	Alert
Errors	ConditionalCheckFailedRequests*, ThrottledRequests** and/or UserErrors (status code 400) count exceeds X in one min***	Critical	Investigate the source.	Alert
Errors	SystemErrors (status code 500) count exceeds X in one min***	Critical	Check the service status	Alert
Saturation	ThrottledRequests	-	-	Informative
Utilization	ConsumedReadCapacityUnits (sum of one minute) / 60 (results read capacity per second)	-	-	Informative
Utilization	ConsumedReadCapacityUnits (sum of one minute) / 60 (results read capacity per second) / provisioned read throughput * 100 exceeds 80 for 5 min	Warning	Recheck the throughput bounds. Investigate the source.	Alert
Utilization	ConsumedReadCapacityUnits (sum of one minute) / 60 (results read capacity per second) / provisioned read throughput * 100 exceeds 80 for 10 min	Critical	Recheck the throughput bounds. Investigate the source.	Alert
Utilization	ConsumedWriteCapacityUnits (sum of one minute) / 60 (results write capacity per second)	-	-	Informative
Utilization	ConsumedWriteCapacityUnits (sum of one minute) / 60 (results write capacity per second) / provisioned write throughput * 100 exceeds 80 for 5 min	Warning	Recheck the throughput bounds. Investigate the source.	Alert

Utilization	ConsumedWriteCapacityUnits (sum of one minute) / 60 (results write capacity per second) / provisioned read throughput * 100 exceeds 80 for 10 min	Critical	Recheck the throughput bounds. Investigate the source.	Alert

* A failed conditional write will result in an HTTP 400 error (Bad Request). These events are reflected in the ConditionalCheckFailedRequests metric, but not in the UserErrors metric.

** A throttled request will result in an HTTP 400 status code. All such events are reflected in the ThrottledRequests metric, but not in the UserErrors metric.

*** Amazon CloudWatch aggregates these DynamoDB metrics at one-minute intervals

Good to know

- If your read or write requests exceed the throughput settings for a table, DynamoDB can throttle that request. DynamoDB can also throttle read requests exceeds for an index. Throttling prevents your application from consuming too many capacity units. When a request is throttled, it fails with an HTTP 400 code (Bad Request) and a ProvisionedThroughputExceededException. The AWS SDKs have built-in support for retrying throttled requests, so you do not need to write this logic yourself.

- For more about throughputs, see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ProvisionedThroughput.html

- For full metrics list, see https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/dynamo-metricscollected.html

Availability monitoring

AWS

Route 53

Metric	Trigger	Severity	Action	Type
Latency	SSLHandshake-Time (in ms)	-	-	Informative
Latency	TimeToFirstByte (in ms)	-	-	Informative
Availability	Alert when application is not available over 1 minute	Warning	Check the service status. Escalate if needed. Is there some event correlating, e.g. unsuccessful deployment. Start investigating.	Alert
Availability	Alert when application is not available over 5 minutes	Critical	Check the service status. Escalate if needed. Is there some event correlating, e.g. unsuccessful deployment. Start investigating.	Alert
Latency	Alert when latency of frontend exceeds X ms over 5 minutes.	Warning	Check the service status. Start investigating.	Alert
Latency	Alert when latency of frontend exceeds X ms over 5 minutes.	Critical	Check the service status. Start investigating.	Alert

Good to know

- For full metrics list, see https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/r53-metricscollected_shared.html

Liite 2. Monitorointipalveluiden vertailudokumentti

Monitoring tool comparison

Sumo Logic

Date: 16.5.2018

Reviewer: Perttu Savolainen

Security: MFA support, push based transaction possible through HTTP API.

Cloud readiness: Own maintained SDK (js, python, ruby etc). Support for Lambda, RDS, DynamoDB, Azure Functions, Azure SQL and Azure Table Storage.

Monitoring: No direct exception tracking support, but has queries based alerts. It's possible to query e.g. "error" from logs.

Usability: Relatively easy integration through HTTP API. Multi-application monitoring support. Seems relatively easy to use.

Alerts: Email + Slack alerts available. Support for Pagerduty.

Load: Minimal if push based data transfer is used. Polling metrics can create some sort of load.

Pricing: Starting at 275$ / month

Overall feeling: Looks good. Maybe a little pricy, but has a lot of features. Worth checking out.

Thoughts after testing: Sumo Logic has a good mindset for security. It has a support for MFA without any excessive configuring. It also supports a push-based data transferring through HTTP API. This allows data handling without the AWS level credentials. Logs are sent to Sumo Logic using that specific HTTP API. The user interface gets quickly a little bit heavy, when the number of tabs grows. Amount of handled and parsed data grows fast, which results in long periods of waiting. There's also Sumo Logic's own query language, which you need to learn. Still, very potential tool (because of Log reduce and other modern features).

Datadog

Date: 16.5.2018

Reviewer: Perttu Savolainen

Security: MFA support, push based transaction possible through HTTP API.

Cloud readiness: Open source library for implementation (Node.js). Support for Lambda, RDS, DynamoDB, Azure Functions, Azure SQL and Azure Table Storage.

Monitoring: No straight exception tracking support (available with e.g. Rollbar), but availability support found. No JavaScript APM support yet. Infrastructure and component monitoring, reporting and analytics found.

Usability: Relatively easy integration through HTTP API. Multi-application monitoring support. Seems relatively easy to use (drag n' drop dashboards etc).

Alerts: Email + Slack alerts available. Support for Pagerduty.

Load: -

Pricing: 15$ / host / month (with 15 month metric retention)

Overall feeling: Has a lot of features, worth checking out.

Thoughts after testing: Testing skipped, since it's already in use.

New Relic

Date: 16.5.2018

Reviewer: Perttu Savolainen

Security: No direct MFA support, but available e.g. through Auth0 login. HTTP API support.

Cloud readiness: Has support for AWS and Azure, and their serverless solutions. Doesn't have SDK, since New Relic uses installed agents and its APIs.

Monitoring: Exception tracking accessible through Error Analytics service.

Usability: Relatively easy integration via HTTP API. Multi-application monitoring support. Seems relatively easy to use.

Alerts: Email + Slack alerts available. Support for Pagerduty.

Load: -

Pricing: 14,40$ / host / month

Overall feeling: Has a lot of features, worth checking out.

Thoughts after testing: Before starting a trial period, it was mandatory to install an agent for "to be maintained server", which seemed a little odd, since all we care is cloud environments (and don't have old school servers). Logs were transferred via Lambda function, which was easily installed through Serverless Application Repository. Showing metrics and events in the same view was really a good solution. Sadly,

this is only accessible in Infrastructure service (Cloud resource data and dashboards are accessible in Insights). In New Relic, there's also a query language to learn. Slack messages were configured nicely by default. Overall it's a nice tool but can't compare to Sumo Logic in our case.

AWS Elasticsearch Service

Date: 16.5.2018

Reviewer: Perttu Savolainen

Security: MFA support through AWS console, push based transaction possible through API.

Cloud readiness: Has several APIs to deal with data, CloudWatch Logs etc can be delivered e.g. with AWS Kinesis Firehose delivery stream.

Monitoring: No straight monitoring, since it's used for search and analytics (for big data, e.g. logs etc). Use CloudWatch to handle monitoring with this. Monitoring for itself (Elasticsearch cluster) available.

Usability: Easy integration with Kinesis Firehose. Seems relatively easy to use.

Alerts: No alerting, but possible to create custom Lambda to make queries to Elasticsearch and alert based on them.

Load: -

Pricing: T2.medium machine costs around 60$/month. In smaller machines like this EBS is mandatory, so volume (SSD) costs 0,161$ per GB / month. This means for example with t2.medium and 100GB volume, the price would be 80$/month. This doesn't include possible data transfer costs.

Overall feeling: It has some configuring (custom Lambdas, Firehose etc) to do to get it up and running. I don't think this is good for us at the moment.

Thoughts after testing: -

Elastic Cloud

Date: 16.5.2018

Reviewer: Perttu Savolainen

Security: No out-of-the-box MFA support. MFA Support through SAML solutions.

Cloud readiness: Has several APIs to deal with data.

Monitoring: Query based monitoring, since it's used for search and analytics (for big data, e.g. logs etc). Monitoring for itself (Elasticsearch cluster) available.

Usability: Seems relatively easy to configure and use.

Alerts: Alerts possible via Alerting module (former Watcher). Support for email and Slack/HipChat/Pagerduty.

Load: -

Pricing: Starting at 121$ / month (https://www.elastic.co/cloud/as-a-service/pricing). Depends highly on usage (e.g. t2.medium.elasticsearch ~50$ / month)

Overall feeling: Good integration support and alerting out-of-the-box. Still needs quite a lot to configure.

Thoughts after testing: -

Dashbird

Date: 16.5.2018

Reviewer: Perttu Savolainen

Security: No out-of-the-box MFA support.

Cloud readiness: Only support for AWS Lambda.

Monitoring: Tracing support (seamlessly integrated with AWS X-Ray).

Usability: Good overview across all functions and function specific views. Seems relatively easy to use.

Alerts: Email + Slack alerts available.

Load: -

Pricing: 299-350$/month with unlimited functions.

Overall feeling: Supporting only AWS Lambda, it is quite pricy. Overall feeling is it's CloudWatch and X-Ray wrapped in one SaaS service. Not for us.

Thoughts after testing: -

Riemann + Logstash + Grafana

Date: 16.5.2018

Reviewer: Perttu Savolainen

Security: No out-of-the-box MFA support.

Cloud readiness: Opensource SDKs for major languages, which can be used for pushing data to Riemann.

Monitoring: Monitoring and stream query support.

Usability: Needs a lot of manual configuring.

Alerts: Email and Slack alerts available. Support for Pagerduty.

Load: -

Pricing: Free, except hosting it somewhere and costs related to that.

Overall feeling: Would need whole lot of configuring and knowledge of Clojure. Not for us at the moment.

Thoughts after testing: -

Prometheus

Date: 16.5.2018

Reviewer: Perttu Savolainen

Security: No out-of-the-box MFA support. Push via HTTP API supported.

Cloud readiness: Official and unofficial (Node.js is unofficial) SDKs and tools e.g. for getting CloudWatch data to Prometheus. Uses pull type of data collection. Also push supported.

Monitoring: Flexible query language and alerting support.

Usability: Needs manual configuring. UI seems relatively easy to use.

Alerts: Supports alert grouping. Email + Slack alerts available.

Load: Pulling type generates more load to monitored endpoints.

Pricing: Free, except hosting it somewhere and costs related to that.

Overall feeling: Even with pull type of data collecting, Prometheus seems intriguing. Needs more setting up than SaaS services.

Thoughts after testing: -

GrafanaCloud

Date: 16.5.2018

Reviewer: Perttu Savolainen

Security: No out-of-the-box MFA support. MFA support through LDAP credentials.

Cloud readiness: Supports Azure (Azure Monitor and Application Insights) and AWS CloudWatch.

Monitoring: Dashboards and alerting.

Usability: Seems relatively easy to configure and use. A lot of plugins and shared dashboards to start with.

Alerts: Email, Slack and Pagerduty support.

Load: -

Pricing: Starting at 39$/month.

Overall feeling: In all its simplicity, GrafanaCloud seems worth checking out.

Thoughts after testing: Defining an alert is not so dynamic as it should be. Alerts are based on dashboard queries, and alerts are not supporting variables inside the query. That means creating alerts should be done one by one which would take too much time. Cloudwatch logs are not available on Grafana, only Service related metrics are. Not for us.