LAB University of Applied Sciences

Teollisuuden IoT-laitteiden kunnonvalvonnan kehittäminen

Elastic-pinon käyttöönotto

LAB-ammattikorkeakoulu
Insinööri (AMK), Tieto- ja viestintätekniikka
2021
Aapo Kärki

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Kärki, Aapo	Opinnäytetyö, AMK	2021
	Sivumäärä	
	40	

Työn nimi

Teollisuuden Iot-laitteiden kunnonvalvonnan kehittäminen

Elastic-pinon käyttöönotto

Tutkinto

Insinööri (AMK)

Toimeksiantajan nimi, titteli ja organisaatio

Markus Järvinen, development team leader, Lahti Precision Oy

Tiivistelmä

Opinnäytetyössä tavoitteena oli kehittää teolliseen kunnossapitoon järjestelmä, joka kerää internetiin liitettyjen laitteiden lokidataa sekä luo hälytyksiä vikatilanteissa. Järjestelmä toimi integroituna pilvipalveluun. Teknologiaksi oli valittu Elastic-pino. Työn tilaaja on punnitus- ja annostuslaitteiden sekä vaakojen toimittaja, punnituksen digitalisaatiota kehittävä Lahti Precision Oy.

Lokidatan automatisoitu valvonta on kunnonvalvontaa etänä, mikä on osa teollista kunnossapitoa. Lyhyellä viiveellä tietoon tullut vikailmoitus voi lyhentää laitteen toimimattomuusaikaa. Kun teollisilta verkkoon liitetyiltä laitteilta kerätään dataa, voi dataa määrällisesti syntyä paljon ja siten relevanttia teoriaa löytyy Big Data -kirjallisuudesta. Elasticsearch on etenkin nopeita tekstihakuja mahdollistava hakumoottori, jossa hyödynnetään Big Data -aihepiirin tekniikoita. Elastic-pinoon kuuluvat ohjelmistot, jotka toimivat Elasticsearchin kanssa. Elastic-pinossa on Elasticsearchin lisäksi esimerkiksi datan lataajia ja Elasticsearchin käyttöliittymä, Kibana.

Toteutuksessa ohjelmistojen paikalliseen kehitysympäristöön asennettiin Elastic-pinosta Elasticsearch, Kibana, Filebeat sekä Elastalert. Datan kerääminen laitteilta, datan muokkaaminen ja tallettaminen kehitettiin sekä testattiin paikallisessa testausympäristössä, jossa oli myös testilaitteita. Lokidatalle kehitettiin tarkoituksenmukaista struktuuria, joka mahdollistaa analyysin. Hälytyksiä testattiin testilaitteiden datalla.

Paikallisen kehityksen jälkeen toiminnallisuus otettiin käyttöön tuotantoympäristössä. Tuotantoympäristöön viennissä ilmeni yllätyksiä. Siten kehitystyö jatkuu. Tuloksena on uuden ohjelmistoteknologian onnistunut käyttöönotto ja sen prosessikuvaus, jossa tutkitaan, kehitetään ja testataan, jatkuvasti laajentaen ja parantaen toiminnallisuuksia.

Asiasanat

Kunnonvalvonta, esineiden internet, Elasticsearch

Abstract

Author(s)	Type of Publication	Published
Kärki, Aapo	Thesis, UAS	2021
	Number of Pages	
	40	

Title of Publication

Enhancing health management of industrial IoT-devices

Deployment of Elastic-stack

Name of Degree

Engineer (UAS)

Name, title and organization of the client

Markus Järvinen, development team leader, Lahti Precision

Abstract

Subject of this thesis is development of a system that collects logging data from industrial devices. The system should alert in case of defect in a device. Main objective was to develop and deploy Elastic-stack. Developed software system is an enhancement to remote health management of industrial devices. The system developed is part of cloud-based weighing service. The study was commissioned by Lahti Precision Oy. Lahti Precision manufactures weighing equipment and the study is part of weighing digitalization.

Automated log monitoring is remote management of devices and therefore creates the value of this system. Industrial devices can create a lot of data and for this matter, relevant theory can be found from Big Data -literature. Elasticsearch as a distributed indexing engine also has some mechanics, like memory-based features and distributed indexing, that are based on themes discussed under Big Data -thematic. Elastic-stack consists of supporting software around Elasticsearch. Main parts are Kibana, the user interface for Elasticsearch and data shippers Logstash and Beats-family.

Practical part of study includes developing and testing of Elastic-stack and devices in a local environment. Installed applications were Elasticsearch, Kibana, Filebeat and Elastalert. Logging data was enriched during local development to correspond production environment.

Finally, the product was taken to production environment. The result of this study is a successful deployment of technology stack and description of still ongoing process of software development.

Keywords

Elasticsearch, Internet of Things, Device health management

Sisällys

Liitteet

Liite 1. Esimerkkitapaus lokituksesta osana uuden teknologian testausta

Liite 2. Elasticsearch-kontin käynnistysasetukset

Liite 3. Kibana-kontin käynnistysasetukset

Liite 4. Filebeat-kontin käynnistysasetukset

1 Johdanto

Hajautettujen järjestelmien jatkuva, reaaliaikainen valvonta on ihmiselle käytännössä hyvin työlästä ja vaikeutuu entisestään järjestelmän kasvaessa. Riippumatta järjestelmän koosta, kannattaa monitorointia järjestää keskitetysti ja automatisoidusti. Tässä opinnäytetyössä luodaan lokitieto- ja hälytysjärjestelmän malli teollisuuden IoT-laitteiden kunnonvalvonnan tueksi. Tallennusteknologiaksi on valittu Elastic-pino.

Työn tilaaja on Lahti Precision. Lahti Precision on punnitusalalla toimiva vaakojen ja annostusjärjestelmien toimittaja. Yrityksen historia ylettyy yli sadan vuoden päähän (Lahti Precision). Lahti Precisionin liikevaihto oli 14,7 miljoonaa euroa ja työntekijöiden määrä 102 vuonna 2019. Lahti Precisionin myymiin palveluihin kuuluu myös digitaalinen punnituspalvelu. Opinnäytetyön tuotos toimii integroituna pilvipalveluun ja on osa toimialan digitalisaation kehitystä.

Tämän opinnäytetyön tavoitteiksi on määritelty käyttäjätarinat. Kehitystyön tulosta verrataan näihin tarinoihin:

- Pilvipalvelu kerää laitekohtaisia status- ja lokitietoja
- Ylläpitäjä voi priorisoida ja luokitella viestejä ja poikkeustilanteita
- Ylläpitäjä voi saada ilmoituksen laitteiston vikatilasta
- Ylläpitäjä voi nähdä status- ja lokitietoja ongelmaan ja tapahtumien kulkuun liittyen

Asetelmassa haaste on kehittää paras mahdollinen ratkaisu käytössä olevilla teknologioilla. Asiakkaalle arvon tuottamisen kannalta kyse on teollisten laitteiden kunnonvalvonnasta. Käytännön haasteena on sekä datan määrän hallinta että dataputken luonti valituilla teknologioilla. Näitä osioita tutkitaan teoriaosuuden kolmessa kokonaisuudessa. Ensimmäisessä käsitellään teollisen kunnossapidon ja kunnonvalvonnan perusteita sekä hälytysten luontia osana näitä. Toisessa osa-alueessa tutkitaan datan keräämistä hajautetuista järjestelmistä. Kolmannessa kokonaisuudessa käsitellään Elastic-pinoa, sen osien ominaisuuksia ja pinon käyttökohteita.

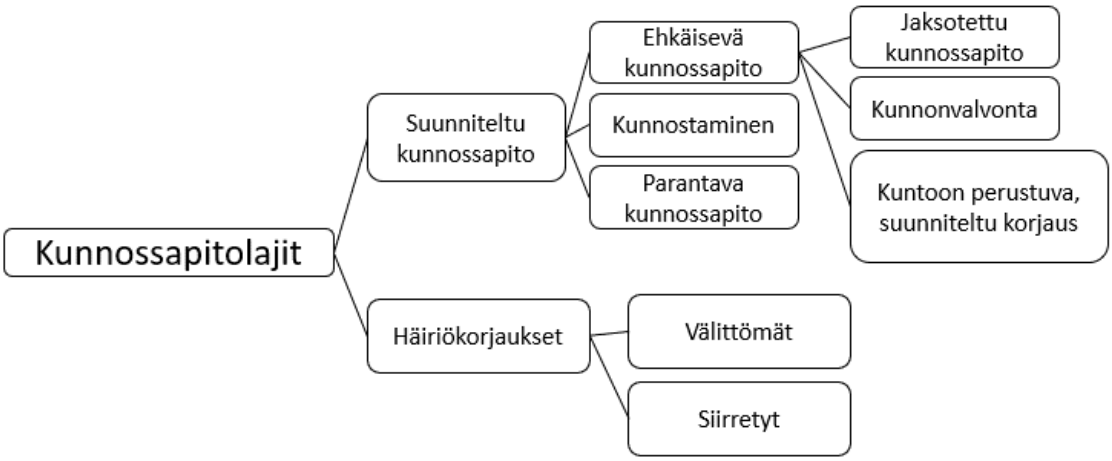
Käytännön osuudessa kehitetään järjestelmää paikallisessa kehitysympäristössä, jossa datan keräämistä ja hälytysten toimintaa voidaan testata. Kehitysympäristö koostuu Docker-konteista, joissa käytettävät ohjelmistot toimivat. Paikallisen kehityksen jälkeen toiminnallisuus viedään tuotantoon.

2 Teollinen kunnossapito

2.1 Kunnossapidon lajit

Kunnossapito on suuri toimiala. Kunnossapitoon liittyviä tehtäväkenttiä ovat muun muassa maantie- ja rautatieverkot, viestintä- ja sähköverkot, satamat ja niin edelleen. Kunnossapito työllistää noin 200 000 henkilöä ja jos se laskettaisiin omaksi toimialakseen, olisi se Suomessa kolmanneksi suurin. (Järviö & Lehtiö 2012, 31.)

Standardissa SFS-EN 13306:2010 kunnossapidon kaksi lajia ovat proaktiivinen ja reagoiva. Erottelu tapahtuu vian havaitsemisessa. Proaktiivisia eli ennakoivia toimia ovat kaikki kunnossapidon toimet ennen vian tai häiriön havaitsemista. Häiriökorjaukset eli reagoivat toimet tehdään toiminnon estävän tilan jo synnyttyä. PSK-standardi 7501:2011 muuttaa lähestymistapaa suunniteltuihin ja suunnittelemattomiin toimenpiteisiin eli häiriöiden korjauksiin. PSK 6201:2010 (kuvio 1) on käytännössä sama, erona vain kunnonvalvonta omana sektorinaan. (Järviö & Lehtiö 2012, 46-47.)



Kuvio 1. Kunnossapidon lajit PSK 6201:2010 (mukailtu Järviö & Lehtiö 2012, 47)

Kunnossapito ei tarkoita ainoastaan korjaamista, vaan siihen voidaan laskea myös vikojen ja vikaantumisen hallinta. Tehokas käyttö ja luotettavuus eivät riipu vain kunnossapitäjistä, vaan käyttäjillä on myös merkitystä. Laitoksen tehokkuus ja luotettavuus ovat osa suurempaa kokonaisuutta, omaisuuden tai laitteiden hallintaa. (Järviö & Lehtiö 2012, 14.) Ehkäisevän kunnossapidon osa-alueita ovat esimerkiksi kunnonvalvonta, toimintakunnon toteaminen, käynninvalvonta ja vikaantumistietojen analysointi (Järviö & Lehtiö 2012, 50).

2.2 Kunnonvalvonta

Koneet, koneiden käyttö ja niiden kunnossapito on monimutkaistunut reilusti viimeisen sadan vuoden aikana. Vanha olettamus on, että vikaantuminen on yhteydessä työn raskauteen ja käytön määrään. Yksinkertaisille, vanhoille laitteille tämä on mahdollista, mutta useiden teknologioiden käyttö ja prosessien kehittyminen on tuonut mukanaan uusia vikaantumismalleja, jotka eivät ole riippuvaisia ajasta tai käytön määrästä. Modernit teknologiat kuten automaatio, pneumatiikka ja tietotekniikka tuovat uusia osaamisvaatimuksia kunnossapitäjille. Osaamisvaatimusta nostaa myös esimerkiksi sensorivalvontaan liitetty sumea logiikka, mutta toisaalta se tuo tuoreita näkökulmia kunnonvalvontaan. Kunnossapitoon voidaan tuoda älykkyyttä ja samalla kunnossapidon kohde ei ole enää mekaaninen laite, vaan myös laitetta ohjaavat ohjelmistot. Datan keruu, monitorointi ja etävalvonta ovat moderneja ilmiöitä. (Järviö & Lehtiö, 22-24.)

Monitorointia on kahdenlaista. Ulkoisessa monitoroinnissa, niin sanotussa black-box monitoroinnissa, järjestelmää havainnoidaan ulkoisesti, kuten käyttäjät järjestelmän havaitsevat. White-box monitoroinnissa data tulee suoraan järjestelmän sisäisistä kanavista. (Ewaschuk 2017.) Yksinkertaista ulkoista monitorointia on havainnoida, toimiiko järjestelmä niin kuin sen tulisi toimia. Järjestelmä tai laite voi ulkoisesti vaikuttaa toimivan kuten pitää, mutta sisäisesti voi virhetila olla päällä. Vastaavasti sisäisesti kaikki voi näyttää olevan kunnossa, mutta käyttäjä ei pysty käyttämään järjestelmää. Kyseessä olisi siis vika esimerkiksi rajapinnassa tai laitteen ja sen fyysisen käyttöliittymän välissä.

Vuonna 2000 on arvioitu, että 60 % vioista voidaan ehkäistä tai poistaa vaikuttamalla koneen toimintaympäristöön, olosuhteisiin ja päivittäiseen havainnointiin sekä käyttöön. Koneen rakenteiden ja komponenttien korjaaminen voi poistaa 15 % vioista. Näin ollen kaikista vioista vain 25 % jäisi näissä havainnoissa ehkäistäviksi kunnonvalvonnalla ja ennakkohuolto-ohjelmalla. (Järviö & Lehtiö 2012, 92.) Tuon arvion jälkeen teknologiat ovat kehittyneet huomattavasti. Päivittäisen käytön ja havainnoinnin merkitystä ei kuitenkaan tule vähätellä, mikäli tavoitteena on vikaantumaton toiminta. Jos vikaantumisen välttämisen lähtökohta on, että vikoja pidetään seurauksina muutoksista koneessa käytön tai ikääntymisen seurauksena (Järviö & Lehtiö 2012, 88), tulisi näistä saada myös dataa monitoroitavaksi.

2.3 Hälytykset osana kunnonvalvontaa

Selkeä tavoite systeemien ja automaation suunnittelussa on ihmisen ja automaation välisen tehtävänjaon luominen. Automaatiossa on omat rajoitteensa siinä mitä automaatiolla voidaan tehdä. Tehokkaissa järjestelmissä automaatio voi toimia ihmisen ”silminä” eli

havainnoijana ja ”lihaksina”, mikäli suoria toimia tarvitaan. Abstraktiotason päättely ja järjestelmän optimointi jäävät ihmiselle. (Heinonkoski, Asp & Hyppönen 2008, 42-43.)

Monet järjestelmät on suunniteltu ihmis- tai käyttäjäkeskeisiksi. Visualisoinnit, käyttöliittymät ja raportoinnit ovat vain ihmisten käyttöön. Automaation suunnittelussa on tärkeää luoda selkeä työkokonaisuus systeemiä käyttävälle ihmiselle. Järjestelmien monimutkaistuessa käyttöliittymän avulla välitetään ymmärrettävää tilannekuvaa sekä kokonaisuuksista että tarkoista ohjattavan systeemin yksityiskohdista. (Heinonkoski ym. 2008, 41-43.)

Googlella on periaatteena, että hälytys tehdään vain, mikäli ongelman käsittelyä ei voi automatisoida ja ongelman ratkaisu vaatii älykkyyttä. Tarkkailua siis automatisoidaan ja datamäärien kasvaessa se on välttämätöntä. Valvontapaneelien graafit ja niihin pääsy on hyödyllistä, mutta vain tarpeen tullen. Tilannetta, jossa joku tuijottaa valvontapaneeleita jatkuvasti ongelmien varalta on vältettävä. (Ewaschuk 2017.)

Googlella kokemus on osoittanut, että hälytysjärjestelmä muodostuu helposti monimutkaiseksi. Hälytyksiä lähtee eri raja-arvoilla monenlaisista metriikoista. Ylimääräistä koodia syntyy kehittämään ilmoitustyyppejä, joiden aiheuttajia voi olla lukuisia. Jokaista aiheuttajaa varten luodaan erilaisia hallintapaneeleja. Monimutkainen järjestelmä on vaikea ylläpitää ja pahimmillaan muodostuu taakaksi. Hyvän hälytyksen tulee olla mahdollisimman yksinkertainen, ennustettava ja luotettava. Tehokas valvontajärjestelmä antaa selkeitä merkkejä ja aiheuttaa vain vähän kohinaa. (Ewaschuk 2017.)

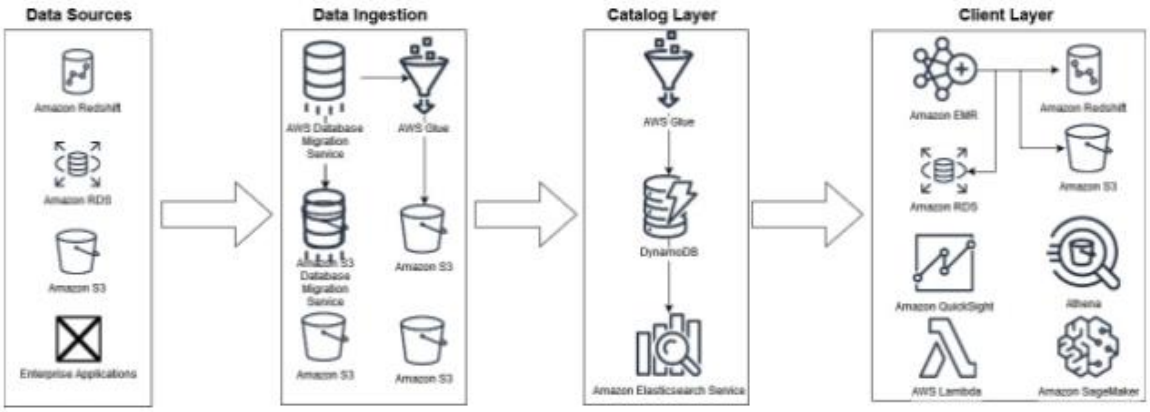
Huollolle on tärkeää ensinnäkin saada tieto vikatilasta silloin kun se syntyy. Yksinkertainen hälytys saattaa kertoa, että on vika havaittu. Se ei kuitenkaan vastaa kysymykseen, kuinka kauan vika on ollut päällä tai onko se vielä päällä. Mikäli yksittäiselle hälytykselle luodaan tila päällä/pois, saadaan lisää informaatiota huollon käyttöön. Ilmoituksen mukana tulevan viestin sisältöä hyödyntämällä pyritään palauttamaan laite mahdollisimman pian toimintakelpoisuustilaan.

3 Datan keräys

3.1 Big Data

Kun dataa on paljon, on käsittelyssä oltava siihen sopivia teknologioita. Paljon dataa on suhteellinen käsite. Dataa syntyy jatkuvasti ”virtaamalla” ja talletettuna se on pysyvämpää. Karkeasti datan määrä on säilytysaika kertaa virtausnopeus, jolla uutta dataa tulee sisään. McLarenin formulan tuottama 1,5 teratavua viikonlopussa (Hodge 2020), 500 miljoonaa Twitter-viestiä päivässä (Salo 2014, 13) tai LinkedInin 75 teratavua pelkkää datacenterin tilaa tallettavaa lokia per datacenter (Kreps 2013) ovat määriä, joiden hallinnointiin vaaditaan erityisiä työkaluja.

Big Data -termiä ei ole yksiselitteisesti määritelty eikä se ole tarpeenkaan. Big Data -käsitteeseen liitetään yleensä Hadoop-tyyppiset ratkaisut. Hadoop on ensimmäisen kerran vuonna 2007 julkaistu Apache-projekti, jonka innoittajana toimivat Googlen julkistamat työpaperit sisäisesti käyttämistään tekniikoista hajautettuun tallentamiseen ja hajautettuun analyysiin (Salo 2014, 72). Google File Systemin (GFS) tyyppisiä teknologioita ovat esimerkiksi Hadoop Distributed File System ja Amazonin S3. Hadoopissa ja Googlella nimellä MapReduce kulkeva työkalu on hajautettu analytiikkakerros. Hajautetun tiedostojärjestelmän päälle rakennettuja analytiikkateknologioita on useita, kuten Apache Spark, Apache Hive, Apache Drill ja käytännössä jokaisella suurella pilvipalvelujen tarjoajalla on hajautetuista analytiikkateknologioista omat versionsa sekä toteutuksensa. Kuvassa 1 on havainnollistus Amazonin oman sisäisen järjestelmän rakenteesta Data Lakeen tallennetun datan päälle. Tallennetun tiedon analysointiin on tehty katalogiksi kutsuttu kerros, jossa on datan lataaja (AWS Glue), tallennusteknologiana DynamoDB tietokanta ja Amazonin itse myymää jakelua Amazon ElasticSearchia hyödynnetään hakujen nopeuttamiseksi.

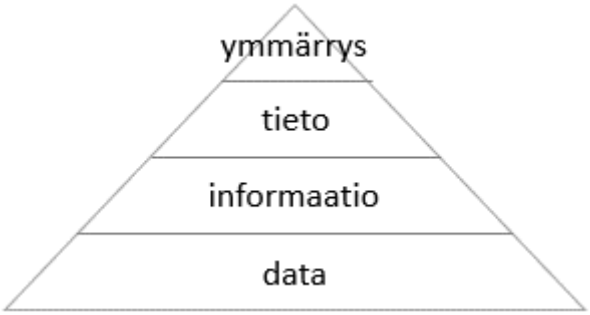


Kuva 1. Amazonin Data Lake -ratkaisu Big Datan haasteisiin (Vogels 2020)

Yksi suuren datamäärän käsittelyyn kehitetty tekniikka on muistinvarainen datan analysointi. Automaattinen, liikennettä analysoiva tallennushierarkia integroi perinteistä tekniikkaa uuteen seuraamalla datan käyttöä. Tiheästi käytetty data on kuumaa ja pidetään tehokkaasti saatavilla. Harvoin käytetty on kylmää ja siirretään alas hierarkiassa, edulliseen ja hitaampaan paikkaan. Data voi muuttua kuumasta kylmäksi ja toisinpäin. Tallennukseen tulee kuitenkin yhdistää myös perinteisiä menetelmiä, ettei dataa katoa. (Salo 2014, 40.)

3.2 Datan transformaatio ja talletus

Datan kerääminen, muokkaus, tallettaminen (extract, transform, load eli ETL) ja analysointi on kokonaisuus. Pelkkä datan tallettaminen sinänsä ei ole itseisarvo. Salon (2014, 32) mukaan data on raaka-aine, jota louhimalla saadaan informaatiota. Informaatiosta voidaan jalostaa tietoa. Tieto lisää ymmärrystä ja tietämystä (kuvio 2). Teknologioilla voidaan muokata datasta informaatiota, mutta tieto ja ymmärrys jäävät ihmisille, kone ei voi niitä tuottaa. (Salo 2014, 32.)



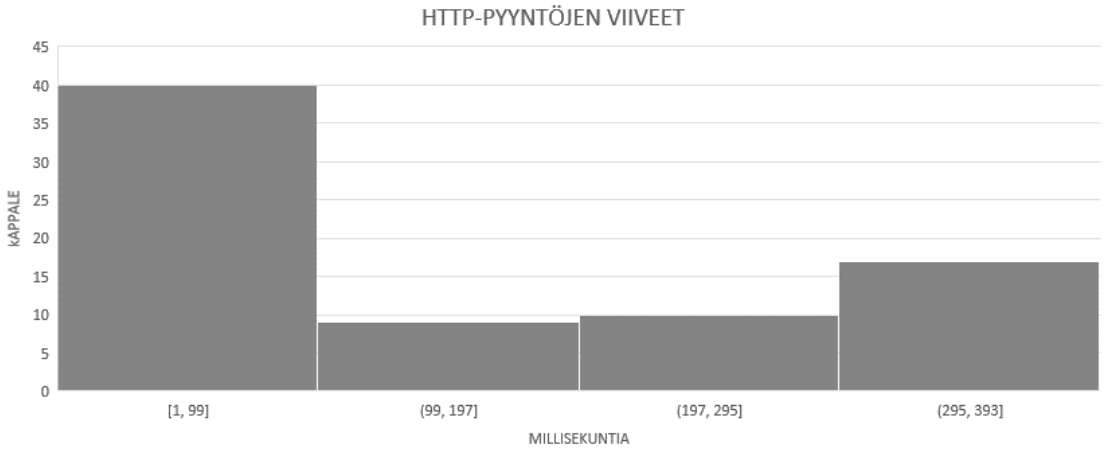
Kuvio 2. Datan ja tiedon suhde (mukailtu Salo 2014, 33)

Datan lähteenä voivat toimia monenlaiset eri järjestelmät ja siten raakadatan muoto voi vaihdella. Ennen tallentamista data muotoillaan tallennukseen ja analyysiin sopivaksi, mikäli analyysitapa on tiedossa.

Tallennusmuoto määrää osaltaan latausvaihetta. Perinteinen tallennusmuoto on esimerkiksi tekstitiedosto, jossa alkiot on eroteltu pilkulla eli csv-formaatti. Tiedostoja on helppo siirrellä paikasta toiseen, mutta esimerkiksi hakutoiminnallisuudet ja koosteet ovat hankalia toteuttaa. Csv-tiedostot perustuvat rivi-sarake-malliin. Rivi-sarake- sekä perinteisemmille relaatiomalleille vaihtoehtona ovat modernit NoSQL-tietokannat. Levylle talletettu tekstitiedosto on yleensä vain paikallisesti tai sisäverkossa saatavissa. Internetin kautta saavutettavissa oleva tietokanta on moderni vaihtoehto.

Jokaisen yksittäisen pyynnön pitkäaikainen tallettaminen ei välttämättä ole mielekästä. Datamäärän kasvaessa käytännöllinen tapa tallettaa tietoja on yksittäisten tietueiden sijaan

tehdä jakaumista tietueita. Ewaschukin (2017) mallissa on esimerkkidatana käytetty http-pyyntöjen viiveitä (kuvio 3). Koosteet vievät huomattavasti vähemmän tilaa ja ovat silti informatiivisia. Laitteetkin saattavat lähettää vikatilassa useita samanlaisia viestejä minuutissa. Siten lokidatan kohdallakin on aiheellista miettiä, onko useilla samanlaisilla viesteillä lisäarvoa ja kannattaako niitä tallettaa. Joissakin aikasarjatietokannoissa on sisäänrakennettuja tapoja arkistoida dataa koosteiksi (Elastic a, Influxdb a).



Kuvio 3. Kooste http-pyyntöjen viiveistä

Talletusmuodon lisäksi toinen ratkaistava seikka on ajallinen resoluutio. Järjestelmän eri aspektit kannattaa tallettaa eri tarkkuuksilla (Ewaschuk 2017). Mikroprosessorin kuormitusasteen tarkastelu minuutin välein ei välttämättä kerro pitkäaikaisistakaan kuormitusongelmista, mutta esimerkiksi lämpötila-anturin tilatiedon kysely minuutin välein voi olla tarpeettoman tiheää. Liian tiheä talletusväli vie tarpeettomasti levytilaa. Huomioon on otettava myös pitkän aikavälin tarpeet. Jos talletusajaksi on suunniteltu pitkiä aikoja, on datalle suunniteltava elinkaari.

3.3 Lokiviestit datana

Yksi tapa erotella datatyyppejä on jako strukturoituun ja strukturoimattomaan dataan. Tähän väliin sopii käytännössä kaikki mahdollinen data. Strukturoidulla datalla on rakenne ja strukturoimattomalla sitä ei ole. Video ja kuvat ovat esimerkkejä strukturoimattomasta datasta. Mikäli niihin liitetään metatietoja, on kyse välimuodosta eli semistrukturoidusta datasta. Semistrukturoidun datan kohdalla metatietojen liittämisessä ongelmaksi voi pitkällä aikavälillä muodostua seikka, jota ei aikanaan pidetty tärkeänä eikä siksi liitetty dataan. (Salo 2013, 25.) Jälkikäteen tärkeää, mutta puuttuvaa metatietoa on vaikea lisätä.

Pelkkä tekstimuotoinen data, kuten esimerkiksi lokidata, twitterviestit tai muut vastaavat, ovat itsessään myös strukturoimatonta dataa. Kun niihin liitetään metatietoja, kuten

lähettäjä ja aikaleima, muodostuu niistä semistrukturoitua dataa. Analyysin kannalta metatiedot voivat olla informatiivisempia kuin tekstiä sisältävä viesti itsessään, tilanteesta riippuen. Tekstien analysointi ja indeksointi muodostaa myös rakennetta.

Lokiviestit kertovat järjestelmän tilasta. Lokiviesteistä pitää saada selville, mitä tapahtui ja milloin. Siten tärkeä (meta)tieto lokiviestiä talletettaessa on aikaleima. Aikaleimojen läsnäolo mahdollistaa aikasarjamaisen datan käsittelyn. Lokiviestit datana on käytännössä muuttumatonta, koska sitä yleensä ei muuteta jälkikäteen. Lokiviestit ovat arvokkaimmillaan tuoreena ja vanhetessaan lokit menettävät arvoaan.

Analyysia helpottavaa struktuuria voidaan luoda luokittelemalla lokitapahtuman tyyppi. Lokiviesti voi kertoa perustoiminnosta, poikkeustilasta tai tilan muutoksesta kuten päivityksestä. Siten analysoinnin kannalta lokidataa voidaan muuttaa informaatioksi muodostamalla koosteita. Esimerkiksi koosteena voidaan laskea, kuinka monta tietynlaista ilmoitusta tulee tunnissa tai vuorokaudessa. Laitteen kannalta kooste voi olla, kuinka monta kertaa tai kuinka pitkään laite on suorittanut tehtäväänsä rajatulla ajanjaksolla.

Esimerkkitapaus suuresta lokidatamassasta on liitteessä 1. Mikäli yksittäisessä dokumentissa olisi vain lokiviesti ja aikaleima, tällaisen datamäärän manuaalinen selaaminen olisi hyvin epäkäytännöllistä. 10,7 miljoonan vuorokaudessa syntyneen dokumentin informatiivisuutta on lisätty strukturoimalla dataa, esimerkiksi lisäämällä kenttiä koosteita varten.

3.4 Katsaus lokitus- ja hälytysteknologioihin

Puhtaasti lokittamiseen erikoistuneita palveluita ovat muun muassa Loggly ja Logz.io. Googlen Cloud Logging, Amazonin CloudWatch ja Microsoftin Azure Monitor ovat isojen pilvipalvelutoimijoiden omat IoT-alustojen lokitusjärjestelmät. Jokaisessa on myös omat työkalut hälytysten luontiin (Microsoft 2020, Google a, Amazon Web Services).

Splunk on pörssiinkin listautunut data-alusta-palvelun tarjoaja. Splunkin referenssilistalla on useita tunnettuja brändejä, kuten Honda, Bosch, Airbus, McLaren, Porsche, Deutsche Bahn, Lenovo, Intel (Splunk). Splunk on data-alusta siihen liittyvine ominaisuuksineen ja lokit ovat yksi mahdollinen datan muoto. Splunkin käyttötavoista muun muassa Porschen kerrotaan hyödyntävän Splunkia data-alustana sähköauton toiminnan kehittämiseen (Davies 2019).

Elasticsearchia hallinnoivalla Elasticilla on asiakkuuksissa myös tunnettuja yhtiöitä, kuten eBay, Facebook, Docker, Shopify, Uber. Elasticin asiakkaiden käyttötavoissa korostuu Elasticsearchin hakutoiminnallisuuden ja skaalautumisen vahvuudet kuten eBayn (Elastic b) tapauksessa.

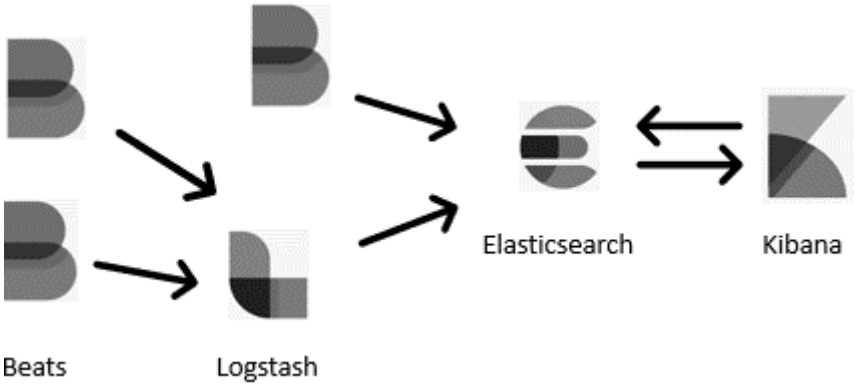
Itse ylläpidettävissä ison volyymin lokitusteknologioissakin on useita vaihtoehtoja. Prometheus markkinoi itseään johtavana avoimen lähdekoodin monitorointiratkaisuna. Prometheukseen voi tallettaa metriikoita ja sieltä voi luoda hälytyksiä. Prometheukseen voi visualisointityökaluksi liittää esimerkiksi Grafanan.

Potentiaalinen avoimen lähdekoodin vaihtoehto itse ylläpidetylle Elastic-pinolle on Tick-pino. Tick-pinon ydin on suuren volyymin aikasarjoihin erikoistunut tietokanta InfluxDB. Samaan tapaan kuin Elastic-pinossa on käyttöliittymä ja tiedon lataamisesta vastaavat agentit, on Tick-pinossa käyttöliittymäkerros Chronograf, tiedon kerääjä Telegraf ja valvontatyökalu Kapacitor. InfluxDB:tä hallinnoivan yrityksen Influxdatan testeissä Influxdb:n on raportoitu olevan nopeampi, suorituskykyisempi ja vievän vähemmän levytilaa kuin kilpailevat aikasarjatietokannat, Elasticsearch mukaan lukien (Influxdata b). Influxdata myy sekä Tick-pinoa pilvipalveluna että lisensoitua versiota, joka sisältää skaalaamisen useampaan klusteriin. Ilmaisversio sisältää itse ylläpidettäväksi vain yhden klusterin. Influxdatan myymä tuote on nimetty Metrics as a Serviceksi (Influxdata c).

4 Elastic-pino

4.1 Elasticsearch

Elasticsearch on hajautettu haku- ja analytiikkamoottori sekä Elastic-pinon keskeisin osa. Nimensä mukaisesti hakutulosten sekä koosteiden muodostamisen nopeus on Elasticsearchin vahvuus. Elasticsearch vastaa useimpiin kyselyihin sekunnissa eli lähes reaaliajassa (Elastic c). Kuvassa 2 on esitettynä yleisimmät Elastic-pinon komponentit eli Beatsit ja Logstash datan lataajina, Elasticsearch keskiössä ja Kibana käyttöliittymänä Elasticsearchiin. Näitä komponentteja ylläpitää ja kehittää hollantilainen yritys Elastic NV.



Kuva 2. Elastic-pinon komponentteja

Elasticsearch käyttää perustana Apache Lucenea ja hyödyntää sen ominaisuuksia. Siksi Elasticsearch esimerkiksi tukee käytännössä kaikkia tietotyyppejä: tekstiä, numeerista dataa, aikaleimoja sekä erikoisuutena maantieteellisiä tietueita. Elasticsearch täydentää Lucenen hakutyyppejä faceteiksi nimetyillä ominaisuuksilla, jotka tuottavat ryhmittelyjä ja koosteita hakutuloksista (Elastic d).

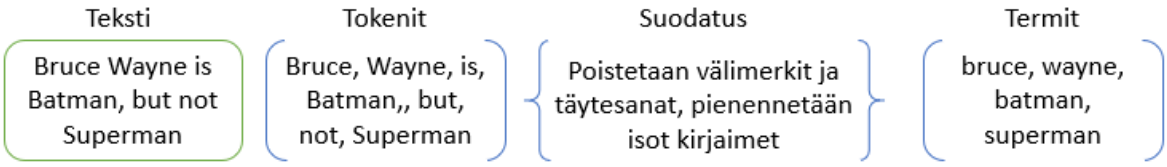
Elasticsearchin ekosysteemin ympärille on kasvanut yhteisöä, josta on seurannut muun muassa monimuotoisia Elastic-pinon ohjelmistoja sekä erilaisia käyttökohteita. Ainakin seuraavia ominaisuuksia markkinoidaan Elastic-pinon käyttötarkoituksina:

- Hakutoiminnallisuuden lisääminen sovellukseen tai nettisivulle
- Lokien, järjestelmän metriikoiden ja tietoturvaan liittyvien tapahtumien valvonta
- Reaaliaikaisen koneoppimisen implementointi
- Liiketoiminnan työnkulkujen automatisoinnin tallettaminen
- Avaruudellisen tiedon hallinta, analysointi ja integrointi
- Geneettisen datan tallennus ja prosessointi

4.1.1 Apache Lucene Elasticsearchin perustana

Elasticsearchin ytimessä toimii hakumoottori Apache Lucene. Lucene, kuten Elasticsearch ja Kibanakin, ovat Javalla koodattuja. Lähes kaikkien yleisten tietotyyppien tallettaminen Luceneen on mahdollista. Erikoisuutena Lucenessa on myös spatiaalisten muotojen, kuten piste, neliö ja ympyrä, indeksointi ja haut. Lucenea voi mieltää dokumenttitietokantana, jonka kaikki kentät indeksoidaan. Tekstikentät, joihin on asetettu indeksointitavaksi täysi tekstihaku, indeksoidaan käännettyyn indeksiin.

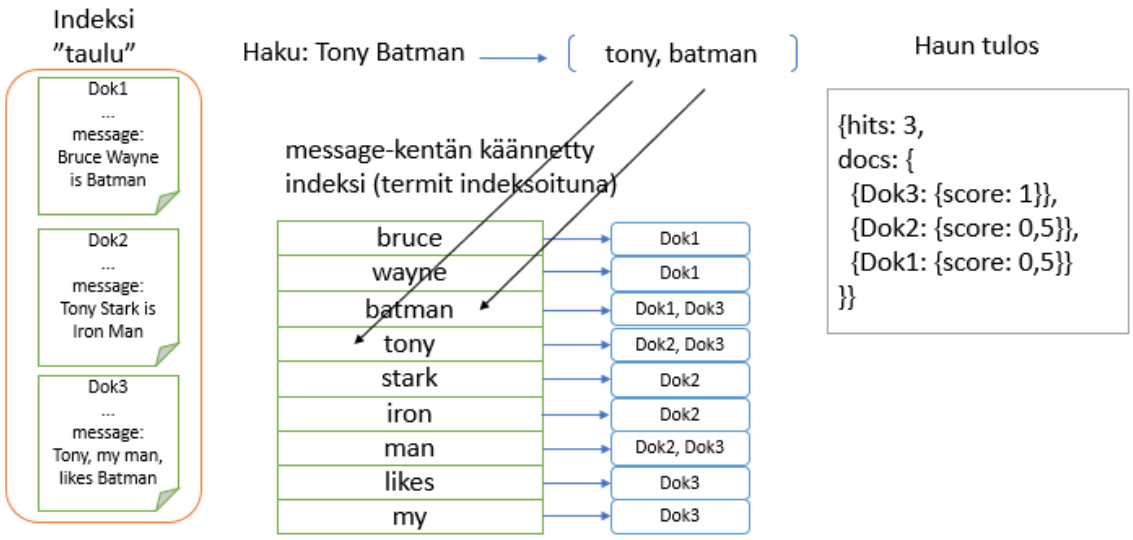
Tekstikenttiin Lucene ottaa tekstiä vain käsittelemättömänä vastaan. Käyttöön valittu analysoija käsittelee tekstin. Analysoija koostuu kahdesta osasta, esi- ja jälkierittelijästä. Syötteenä tulevasta termistä voidaan esierittelyssä esimerkiksi poistaa kirjaimia, HTML-merkintöjä, vakiomerkkijonoja tai merkkijonokuvioita. Jälkierittelyssä syötettä voidaan muotoilla, kuten useimmiten on käytäntönä, muuttamalla termin kaikki kirjaimet pieniksi ja ennen eteenpäin syöttämistä vielä suodattaa termejä. Käytännössä tämä voi olla täytesanojen poistamista kuten englannissa sanat a, any ja niin edelleen, tai lisätä polveutuvia sanoja ”pyörä”, "pyörät", "pyörän". Näiden vaiheiden tuotoksena muodostuu termien virta Lucenelle indeksoitavaksi. Kuviossa 4 on esimerkki syötteenä tulevasta tekstistä ja sen analysoidusta tuloksesta eli indeksoitavista termeistä. Vakioanalysoijien lisäksi on mahdollista luoda omia analysoijia. (Apache Software Foundation a.)



Kuvio 4. Havainnollistus Lucenen tekstin analysointiprosessista

Tekstihaun mahdollistamiseksi Lucene luo käännetyn indeksin. Kuviossa 5 on havainnollistus täyden tekstihaun käännetystä indeksistä. Käännetyssä indeksissä termit osoittavat

dokumentteihin. Käännetyn indeksin arvosta ei voi päätellä dokumentin kentän todellista arvoa kokonaisuudessaan.



Kuvio 5. Lucenen käännetty indeksi yksinkertaistettuna

Tyypillisessä tietokantahaussa rajataan hakutulosta. Tavallisessa skeemassa tietue joko kuuluu haettuun joukkoon tai ei, mutta Lucenessa tulokset voivat kuulua ”enemmän” tai ”vähemmän” haettuun joukkoon. Lucene siis pisteyttää tekstihakutulokset. Kuvion 5 havainnollistavassa yksinkertaistetussa esimerkissä molempien haettujen termien (tony, batman) esiintymisellä sai pisteeksi 1. Jos vain toinen haetuista termeistä esiintyi, oli pistemäärä 0,5. Pistetytykseen on erilaisia malleja kuten avaruusvektorimalli ja kielimallit (Apache Software Foundation a). Tiedon hakemiseen Lucenessa on useita hakutyyppejä, kuten sumea haku, termihaku, termien yhdistelmähaku, piste-etäisyys, boolean-tyyppiset sekä regexp- ja villikorttihaut. Lucenessa on mahdollista hakea esimerkiksi etäisyyttä, lähintä naapuria ja osumia tietyltä väliltä. (Apache Software Foundation b.)

4.1.2 Elasticsearchin perusosat

Elasticsearchin tietueet ovat dokumentteja. Dokumentti on kokoelma kenttiä, joissa on arvoja. Arvo voi olla jokin yleisistä datatyypeistä, JSON-objekti tai viite toiseen dokumenttiin. Dokumenttia vastaava käsite relaatiokannoissa olisi rivi. Elasticsearchissa on rajapinta, joka mahdollistaa dokumenttien tallettamisen ja niiden kyselyn. Siinä mielessä Elasticsearchia voikin luonnehtia dokumenttitietokannaksi tai NoSQL-tietokannaksi. Elasticsearch ei nojaa riveihin ja relaatioihin, vaan kenttiin ja indekseihin. Elasticsearch tallettaa perinteisen rivi- ja sarakemuodon sijaan tarvittaessa sisäkkäisiä ja monimutkaisiakin dokumentteja. Kentän arvon tyyppi riippuu indeksin kenttätyypityksestä (mapping). Jo luoduissa indekseissä ei tyypitettyjen kenttien tyyppejä voi muuttaa, mutta uusia kenttiä voi lisätä. Mikäli

kenttää ei ennestään ole tyypitetty, voi Elasticsearchissa tyypittää uusia kenttiä dynaamisesti sisään tulevan datan tyypin perusteella.

Automaattisilla tyypityksillä pääsee nopeasti liikkeelle. Huomiota vaativat kuitenkin erityisesti tekstikentät. Tekstikenttiä voi tyypittää useille eri tyypeille. Elasticsearchin oletustyyppi ei aina ole paras mahdollinen. Oletuksena tekstikentät tyypitetään täyteen tekstihakuun sekä luodaan alikenttä nimeltään keyword, jossa on sama kentän arvo tyypitettynä keyword-tyypiksi. Kaikkia tekstikenttiä ei välttämättä suorituskyvyn optimoinnin takia ole tarpeen tyypittää täyteen tekstihakuun. Keyword on kenttätyyppi, joka mahdollistaa koosteiden tekemisen ja suodattamisen nopeasti. Esimerkiksi tiedon lähteen yksilöivä kenttä, kuten laitteen tunniste, kannattaa huolehtia tyypitetyksi keywordina koosteiden ja suodatuksen takia.

Tyypityksien hallintaan on luotu mallineet (template), joilla tyypitys voidaan luoda jo ennen ensimmäisen tietueen tallettamista. Elasticsearch hylkää datan, jos kenttään yritetään tallettaa vääräntyyppistä tietoa.

Elasticsearchissa jokainen dokumentti talletetaan indeksiin (myöhemmin indeksitaulu). Elasticsearchin termistössä indeksi tarkoittaa dokumenttien kokoelmaa. Siten tässä yhteydessä on kiinnitettävä huomiota indeksi-termin käyttöön. Indeksi on tavallisesti järjestysnumeroa tarkoittava luku, joka yksin tai yhdessä muiden indeksien kanssa yksilöi yksityisen tiedon jonossa tai taulukossa (ATK-sanakirja 22, 1971). Dokumenteista ja niiden kenttien arvoista luotuja järjestettyjä listoja, joihin haut kohdistuvat, kutsutaan myös indekseiksi, joten kontekstista on pääteltävä, kumpaa tarkoitetaan. Englanniksi termit ovat erilliset index (indeksitaulu) ja indice.

Relaatiotietokannoissa indeksitaulua vastaava käsite on taulu. Käytännössä käyttötavoissa on kuitenkin eroja. Elasticsearchin indeksitauluille on tyypillistä, että samankaltaisia indeksitauluja luodaan useita, esimerkiksi päiväkohtaisesti. Elasticsearchissa yksi virtaavan datan hallintaa helpottava ominaisuus onkin automaattinen indeksitaulujen elinkaaren hallinta. Asetuksilla on mahdollista tehdä automaattiset vanhojen indeksien poistot sekä vuorokauden vaihtuessa uuden indeksitaulun luonti rotatoimalla. Tällaiselle elinkaaren hallinnalle ei ole vastinetta perinteisissä relaatiokannoissa. Käytännössä nimetty indeksi siis jakaantuu useampaan osaan, kuten päiväkohtaisesti, jolloin indeksiin kohdistettu hakee kaikista sen osista. Silloin ei ole merkityksellistä, missä indeksitaulussa tietue on, jos se on indeksoitu yhtenäiseen indeksiin.

Yksittäinen indeksi voi kasvaa käytännössä rajattomasti. Levytila on kuitenkin aina rajallista. Skaalautuvuutta Elasticsearchiin luodaan sirpaleilla (shard). Luotaessa indeksitaulua Elasticsearchiin määritellään, moneenko sirpaleeseen se hajautetaan. Yksi sirpale on yksi

Lucenen indeksi. Pienelle datamäärälle on tarpeetonta määrittää useita sirpaleita. Suuri datamäärä hidastuu yhdellä sirpaleella. Optimaalinen sirpaleiden määrä on ennen datan keruuta hyvin vaikeaa määrittää.

Sirpaleita voi olla myös eri noodeissa. Noodi on yksi Elasticsearch-prosessi. Kun noodi käynnistetään, se joko luo oman klusterin tai liittyy toiseen klusteriin. Noodit ovat yleensä eri koneilla.

Noodeja toimii eri rooleissa. Perusrooli on datanoodi, johon indeksejä hajautetaan ja joissa hakuja prosessoidaan. Master noodi on johtava noodi klusterissa ja masteriin tehdään kirjoitukset, hallinnointi ja asetusten muutokset. Lisäksi on erikseen koneoppimisen noodeja sekä tiettyjä klusterin toimintaa tukevia noodeja. Elasticsearch tallettaa automaattisesti indeksien replikoita eri noodeille.

4.1.3 Elasticsearchin muistinkäyttö

Lucene hyödyntää kyselyihin vastaamisen nopeuttamiseksi käyttöjärjestelmän tiedostojärjestelmän välimuistia. Täysi tekstihaku toimii osittain muistinvaraisesti, muistissa pidetään erityisesti käänteisiä indeksejä (McCandless 2010). Ennen raskasta levylle kirjoitusta, eli Lucenen commitia, uusien muutosten lokia myös säilytetään muistissa (Elastic e). Elasticsearch vaatii paljon muistia, noin puolet käytössä olevasta. Elasticsearchille muistikeko on nopeuden takia tärkeä osa, mutta Lucenen välimuistin käyttö ei ole osa Elasticsearchille varattua muistikekoa (Elastic f), eikä myöskään Java-ajoympäristön virtuaalikoneen toiminta. Lisäksi Elasticsearch käyttää sille erikseen varatun muistikeon ulkopuolisia puskureita verkkoyhteyden tehostamiseksi. Java-ajoympäristön virtuaalikone kompressoi olioosoittimia, mikäli muistin koko ylittää noin 32 gigatavua, joten suositus on rajata muistia nimenomaan Elasticsearchin muistikekoon noin 26-30 gigatavua, järjestelmästä riippuen.

Kuten todettu, Elasticsearchin indeksit koostuvat sirpaleista, joista jokainen on yksittäinen Lucenen indeksi. Lucenen indeksi koostuu segmenteistä, joita ei voi muuttaa. Muuttumattomuutensa vuoksi indeksit sopivat hyvin pidettäväksi tiedostojärjestelmän välimuistissa. Muuttumattomuus vaatii kuitenkin ratkaisun tilanteeseen, jossa dataa halutaan muuttaa. Tämä toteutetaan luomalla uusi segmentti, jossa data on päivitetty. Vanha segmentti, jossa on vanhentunut data, jää tällöin tarpeettomaksi.

Itse ylläpidetyssä järjestelmässä ohjelmiston optimointi on osa kehitystyötä. Elasticsearchin muistinkäytön optimoinnissa on omat haasteensa. Lucenen muistinkäyttö on yksinkertaista. Lucene pitää kuumia segmenttejä muistissa. Elasticsearchin muistissa pitämät rakenteet eivät löydy koostetusti ja yksiselitteisesti. Yksi rakenne, jota Elasticsearch pitää muistikeossa, on kenttädatan välimuisti (field data cache). Kenttädatan välimuisti on monimutkainen

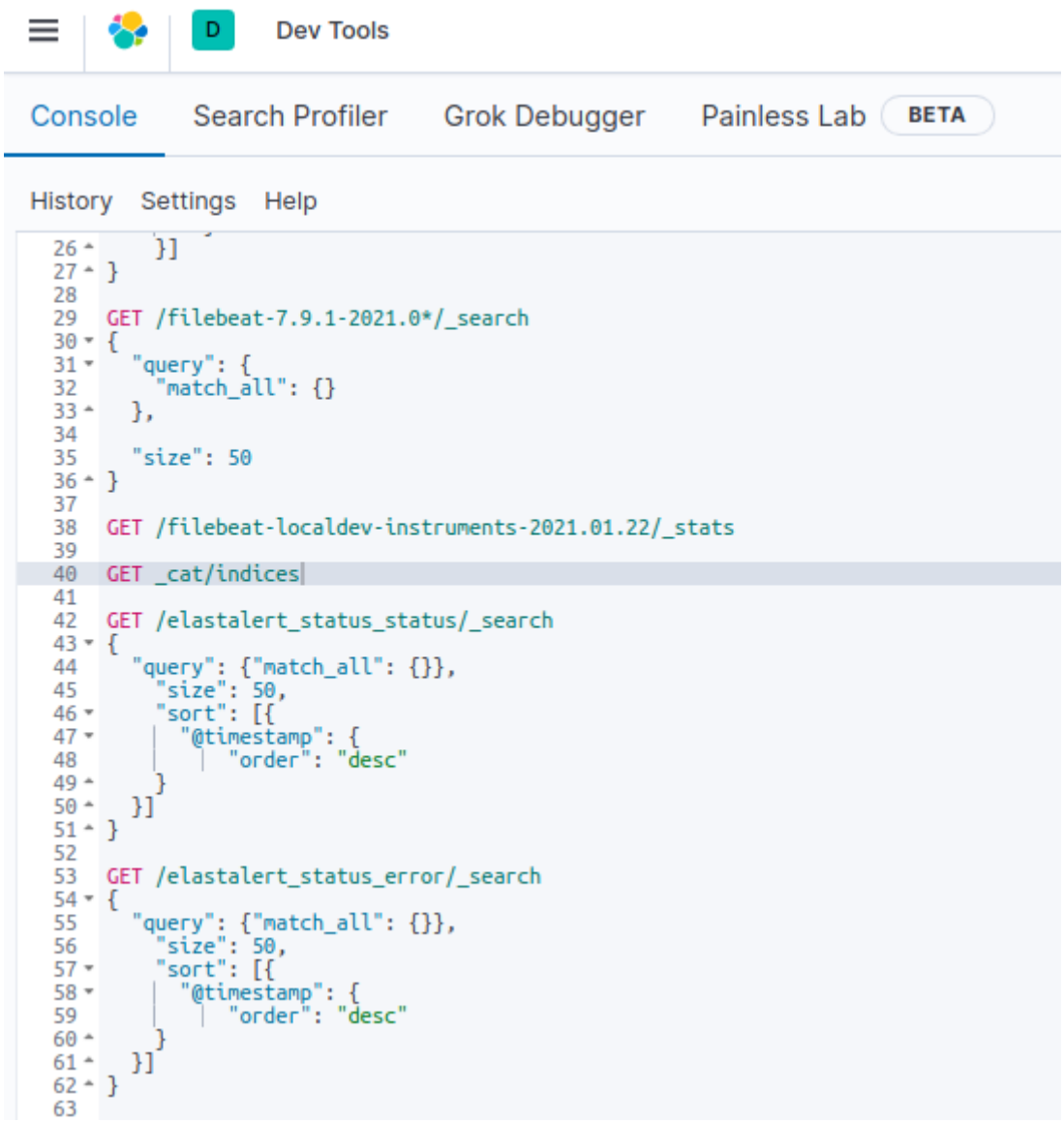
rakenne, jota hyödynnetään muun muassa koosteiden ja ryhmittelyjen luontiin. Kun käännetyssä indeksissä indeksoidaan kentässä olevat termit, joita voi olla kentässä useita, on yksi kenttä indeksoitu monella arvolla. Kenttädatan välimuistin indeksointitavassa indeksoidaan dokumentin kentässä olevan arvon perusteella, eli yhdellä alkiolla. Elasticsearch tallettaa jokaisesta dokumentista kentän arvoa vastaavan tietueen, jotka voidaan keskenään asettaa järjestykseen. Näitä tietueita kutsutaan ordinaaleiksi. Elasticsearch yhdistää ordinaalit kaikista sirpaleista yhteen paikkaan globaaleiksi ordinaaleiksi. Globaalien ordinaalien avulla muodostetaan kenttädatan välimuisti, josta koosteet, järjestämiset ja skriptihaut ovat mahdollisia. Haku tehdään ordinaalien perusteella, ei kentässä olevasta arvosta riippuen. Ordinaalista ei voi myöskään päätellä kentän todellista arvoa. Kyseessä on siis Lucenesta riippumaton indeksointitapa, joka raskautensa vuoksi ei oletuksena ole aktivoituna tekstikenttiin. Tekstikentän jokainen termi olisi oma ordinaalinsa, kun käännetyssä indeksissä termit ovat uniikkeja. (Elastic g, Elastic h, Elastic i.)

4.2 Kibana

Kibana on selainpohjainen analytiikka- ja visualisointityökalu, käytännössä Elasticsearchin käyttöliittymä. Valmiita visualisointipohjia on monia, esimerkiksi eri Beats-perheen oletustietorakenteille ja lokidatalle. Visualisoinnin lisäksi Kibanassa on mahdollista konfiguroida koneoppimisen algoritmeja eri käyttötarkoituksiin ja erilaisia hälytyksiä, mutta monet tämän tyyppiset edistyneet analytiikkatoiminnot ovat käytettävissä vain maksavilla tilaajilla.

Kibanasta voi tehdä kyselyjä Elasticsearchiin ja siten muodostaa raportointinäkymiä. Tyypillinen tapa on tehdä koosteita esimerkiksi lukumääristä tai keskiarvoista ja näyttää niitä ajallisesti järjestettynä.

Kibana ei rajoitu pelkästään näihin. Kibanan konsoli on hyvin käyttökelpoinen ohjelmistokehitystyön aikana. Dev tools -osiosta löytyvään konsoliin tallettuvat kyselyt Elasticsearchin rajapintoihin. Kibana tallettaa osan tilastaan selaimeen ja osan Elasticsearchiin, mikä selittää tietojen säilymisen. Kuvassa 3 on esimerkkejä käytännöllisistä kehitysaikaisista kyselyistä, joissa hyödynnetään Elasticsearchin omaa kyselykieltä. Kyselykieli tukee muun muassa tähteen loppuvia villikorttihakuja indeksien nimissä.



Kuva 3. Kibanan konsoliin tallentuvat käytetyt kyselyt

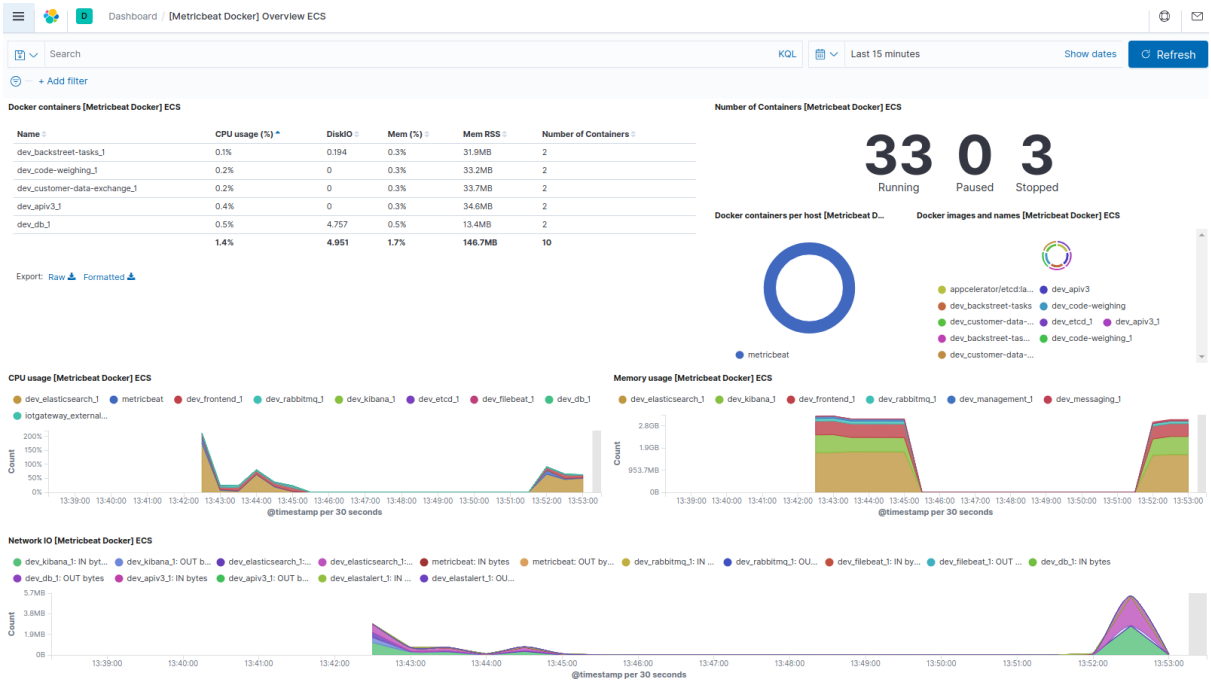
4.3 Beats-perhe

Vanhin Elastic-pinon datan lataaja on yleiskäyttöinen Logstash. Beats-perhe on luotu tukemaan Logstashia. Beats-perhe on ohjelmoitu Golangilla, joka tekee Beatseista huomattavasti kevyempiä datan lataajia Logstashiin verrattuna. Logstash vaatii Java-ajoympäristön, kun Golangissa pohjana on c-kielen kirjasto. Beatsit ja Logstash eivät ole toisensa poissulkevia vaihtoehtoja. Beatseista voi lähettää dataa suoraan Elasticsearchiin tai Logstashin kautta. Yksi syy Logstashin kautta lähettämiseen voisi olla esimerkiksi SSL-yhteyteen tarvittavien sertifikaattien keskitetty hallinta yhdessä paikassa, Logstashissa, jos käytössä on useita lataajia. Muita syitä voisivat olla esimerkiksi palomuurin rajaukset tai muutosten keskittäminen Logstashiin.

Beatseja on yhteensä seitsemän, joista jokainen on erikoistunut omaan käyttökohteeseensa. Winlogbeat on luotu Windowsin tapahtumalokien talletukseen, Auditbeat Linuxin audit-lokeja varten. Packetbeatilla voidaan kerätä dataa verkkoliikenteestä ja Heartbeat on black-box-monitorointia palvelun saatavuuden testaamiseen. Functionbeat on serverless-ratkaisu integroitavaksi pilvipalveluihin. Metricbeatilla voidaan kysellä metriikoita useista eri järjestelmistä, kuten muistinkäyttöä, levytilaa ja prosessorin kuormitusta. Filebeatilla voi lukea tiedostoja.

Filebeat kiinnittää tiedostoihin kerääjiä. Yhdellä Filebeat-instanssilla voi olla useita kerääjiä. Kerääjä tarkkailee tiedostoa ja tunnistaa tiedoston muokkauksen. Kun tiedostoon ilmestyy uusi rivi, lähettää kerääjä uuden rivin Filebeatille, joka käsittelee uuden rivin yksittäisenä prosessoitavana tapahtumana. Filebeat muokkaa, suodattaa ja rikastaa tarvittaessa dataa sekä lähettää sen eteenpäin. Filebeatin puskurointi perustuu rekisteriin, johon Filebeat kirjoittaa jokaisesta tiedostosta viimeisimmän onnistuneesti lähetetyn rivin, johon Elasticsearch on myös vastannut. Poikkeuksena juuri avattu tiedosto, jossa viimeksi on lähetetty rivi numero 0. Mikäli yhteys Elasticsearchiin tai Logstashiin on poikki, säilyy tieto viimeisimmästä perille menneestä rivistä. Siitä eteenpäin tiedot ovat lähettämättä. Tällä varmistetaan, että dataa ei kadoteta. Tätä kutsutaan vähintään kerran -toimitukseksi. (Elastic j.)

Monet Beatsit tarjoavat valmiita visualisointi-hallintapaneeleja Kibanaan. Kuvassa 4 on Metricbeatin mukana tuleva oletusvisualisointipaneeli Docker-konttien muistinkäytöstä, prosessorin kuormituksesta sekä verkkoliikenteestä.



Kuva 4. Metricbeatin mukana tuleva Dockerin metriikoiden hallintapaneeli

4.4 Elastalert-ohjelmisto

Elastalert on yksinkertainen ohjelmistokehys luomaan hälytyksiä poikkeamista, datapiikeistä tai muista tunnistettavista malleista datassa. Elastalertin kehitystyö on aloitettu ennen kuin Kibanassa on ollut hälytystoiminnallisuutta ja Elastalert on yhä ilmainen avoimen lähdekoodin ohjelmisto. Koodia on ylläpitänyt Yelp, mutta aktiivinen kehitys on lopetettu (Yelp a). Elastalert repositoriossa on 1187 raportoitua ongelmaa ja 156 muutospyyntöä (pull request). Versionhallinnassa on uusin versio 0.2.4 julkaistu huhtikussa 2020, minkä jälkeen joitakin bugikorjauksia on vielä yhdistetty koodikantaan. (Yelp b.) Elastalert on kirjoitettu Pythonilla.

Elastalertin kolme osaa ovat säännöt, prosessointi ja hälytykset. Elastalert lähettää kyselyjä määrätyin väliajoin Elasticsearchille. Kyselyn tuloksia verrataan sääntöön. Kun sääntö täsmää kyselyyn, syntyy osuma. Hälytykseen tarvittavien osumien määrän voi asettaa. Osumia voidaan prosessoida esimerkiksi rikastamalla tapahtumia tai suodattamalla niitä pois. Prosessoidut osumat siirretään hälytyksille, joita voi olla useita erilaisia per sääntö.

4.4.1 Sääntötyypit

Sääntötyyppejä on monenlaisille yksinkertaisille algoritmeille. Eri sääntötyypit on listattu taulukossa 1.

Blacklist	Mustalle listalle asetetut arvot tuottavat osuman.
Whitelist	Listan ulkopuoliset arvot tuottavat osuman.
Change	Kentän arvon muuttuessa tuottaa osuman.
Frequency	Tietyn lukumäärän ylittyessä määrätyllä aikavälillä tuottaa osuman.
Spike	Tuottaa osuman, mikäli nykyisellä aikaikkunalla tapahtumien määrä on raja-arvon verran suurempi kuin verrokki-ikkunassa.
Flatline	Tuottaa osuman, mikäli tapahtumia on määrätyn ajan vähemmän kuin raja-arvo.
New Term	Ennen näkemätön arvo kentässä tuottaa osuman.
Cardinality	Yksilöllisten arvojen yläarvon ylitys tai alaraja alitus tietyllä aikavälillä.
Metric Aggregation	Kooste, kuten keskiarvo, summa tai lukumäärä, on rajojen ulkopuolella määrätyssä aikaikkunassa.
Spike Aggregation	Kooste on raja-arvon verran korkeampi nykyisessä aikaikkunassa kuin verrokki-ikkunassa.

Percentage Match	Kyselyyn sopivia dokumentteja on prosentuaalisesti raja-arvojen välillä.
Any	Jokainen osuma tuottaa hälytyksen.

Taulukko 1. Elastalertin hälytystyypit

Hälytystyypeistä useampikin sopisi laitteiden hälytyksiä valvomaan. Sopivia hälytystyyppejä käyttötarinoina olisivat:

- Osuma, jos X määrä tapahtumia ajassa Y (frequency)
- Osuma, jos kentän arvo täsmää listaan (blacklist/whitelist)
- Osuma, mikäli yksikin tapahtuma täsmää suodatettuun kyselyyn (any)

4.4.2 Hälytykset eli toiminnot

Yksi hälytys on käytännössä sama kuin toiminto. Valmiita toimintoja ovat esimerkiksi sähköpostin lähettäminen, HTTP POST -pyynnön lähettäminen, konsolissa komennon ajaminen tai webhookit Slackiin, Telegramiin, Teamsiin, Jiraan ja moneen muuhun vastaavaan palveluun.

Hälytyksille syötetään parametri, joka on tietotyypiltään Pythonin sanakirja tai lista sanakirjoja. Parametrin perusteella hälytykselle luodaan sisältö. Oletuksena sisällön runko muodostuu hälytyksen nimestä, hälytystekstistä, hälytyksen tyypistä, eniten osumia tuottaneen kentän nimestä ja osumien määrästä. Lisäksi oletusrungon viestiosa sisältää kaikki parametrin kentät arvoineen.

4.5 Elastic-pinon sovellutuksia

Käytännön toteutuksia Elastic-pinon käytöstä on lukuisia. Pino soveltuu moneen, mutta toteutus on suunniteltava vaatimusten ja tarpeiden mukaan. Lokidatan ollessa kyseessä yksi vaihtoehto on käyttää Elasticsearchia ainoana datan talletuspaikkana. Pitkäaikaisessa talletuksessa tällöin on haasteensa, mutta lokidatan arvo laskee joka tapauksessa ajan kanssa. Elasticsearchin etukäteen määritelty tyypitys ja sirpaleiden määrä sekä indeksointi on vaikea määritellä vuosien päähän.

Kappaleessa 3.1 on esitetty Amazonin tapa käyttää AWS Elasticsearchia hakujen nopeuttajana katalogikerroksella. Tiirikainen (2018) on käyttänyt Elastic-pinoa langattoman viestintäjärjestelmän kehitysvaiheessa radioarvojen vertailuun. Kaurto (2018) on hyödyntänyt Elasticsearchin lokien keruuta verkonvalvonnassa, tarkemmin tapahtumapohjaisessa vikojen valvonnassa. Lokien tueksi ja verkon suorituskyvyn valvonnan takia Kaurto keräsi

Elasticsearchiin myös verkkoliikenteen virtaustallenteita. Pfizerilla on sovellus, jossa data on Hadoopiin perustuvassa datapilvi-ratkaisussa ja Elasticsearch on tuotu siihen päälle indeksointikerrokseksi (Elastic k).

Fiktiivinen käyttötapa voisi olla analyysi, jossa analysoidaan sosiaalisen median postauksia kahdesta eri lähteestä, kuten Twitteristä ja Instagramista. Raakadata syötetään Elasticsearchille, josta voidaan tehdä hakuja analyysia varten. Tässä mallissa Elasticsearch on siistityn ja indeksoidun datan väliaikainen varasto (data warehouse) analyyseja varten. Hälytyksiä voisi luoda mustalle listalle asetetuista sanoista. Hälytyksistä tai koosteista voitaisiin tutkia esimerkiksi yhtiön avainsanan esiintymistä tviiteissä ja markkinointikampanjan vaikuttavuutta.

5 Elastic-pinon käyttöönotto ja konfigurointi

5.1 Lähtötilanne

Tämän opinnäytetyön käytännön osuus alkoi kehitysympäristön luonnilla. Elasticsearchista, Kibanasta ja Logstashista oli jo aikaisempi käyttöönotto, joten niistä oli saatavilla tietoja. Tässä kehitystyön vaiheessa päivitettiin Elasticsearch ja Kibana uusimpiin versioihin sekä vaihdettiin datan lataajaksi Filebeat Logstashin sijaan.

Kaikki ohjelmistot otettiin käyttöön Docker-konttiteknologialla. Konttien kerroksille katsottiin mallia internetin monista vaihtoehdoista. Elasticsearch-, Kibana- ja Filebeat-konttien käynnistysasetukset ovat liitteissä 2, 3 ja 4. Kehitysaikaiset ohjelmistojen asetukset ovat huomattavasti yksinkertaisempia kuin tuotantokäytössä.

5.2 Elasticsearch ja Kibana

Elasticsearchille varattua muistia on paikallisessa kehityksessä rajattu yhteen gigatavuun. Gigatavu on myös Elasticsearchin oletusasetus, tosin tuotantokäytössä harvoin riittävä. Tähän ei sisälly Lucenen käyttämä muisti, joten mikäli kontille ei erikseen rajata muistia, voi datamäärän kasvaessa kehitys käydä hyvin raskaaksi koneelle.

Kibanan käyttöönotto on myös suoraviivaista, pitkälti Elasticin luoman valmiin levykuvan takia (liite 3). Paikallisessa kehitystyössä vaadittavia asetuksia on vain vähän. Huomattavaa on, että avoimen lähdekoodin projektina Elastic-pino on alun perin rakennettu toimimaan luotettavan järjestelmän sisällä ja siksi tietoturvaominaisuudet eivät ole oletuksena toiminnassa. Tietoturvatoimintoja olisi mahdollisia esimerkiksi SSL-liikenne ja tunnistautuminen salasanalla tai rajapinta-avaimella. Käyttäjäoikeuksien, käyttäjätunnuksien tai SSL-liikenteen avainten konfigurointi eivät paikallisen kehitystyön aloitukseen kuuluneet.

Kibanassa on oma työkalu hälytysten luontiin. Se on osittain maksullinen, ilmaiseksi voi kokeilla joitakin sen toimintoja. Hälytysten luontia varten Kibana vaatii asetuksiinsa salausavaimen (encryptionKey). Kibanan peruskäyttö ei vaadi salausavainta.

5.3 Filebeat

Filebeatin käyttöönotossa kontti ei Elasticin valmiiseen levykuvaan (liite 4) vaadi tässä vaiheessa konfiguraatiotiedoston kopioimisen lisäksi muita toimenpiteitä. Konttiin liitetään isäntäkoneesta lokit sisältävä kansio sekä Docker daemonin portti konttien metadatatietoja varten.

Filebeatin asetuksissa määritellään syöte, lähtö sekä niiden väliin mahdollinen prosessointi. Prosessoinnissa yksittäistä tapahtumaa voidaan rikastaa, sen sisältämää dataa voidaan suodattaa tai tarvittaessa poistaa. Filebeatin asetuksissa määritellään myös indeksien nimeämiskäytäntö Elasticsearchiin, jos sellainen on, sekä indeksin elämänkaaren hallinta.

Filebeatissa on valmiita input-tyyppejä, joista valittiin container. Log-tyyppi olisi toinen tähän tarkoitukseen mahdollinen tyyppi ja sitä kehitysvaihessa kokeiltiinkin. Container-tyypissä etu on Docker daemonilta oletuksena saatavat metadatat.

Useammalle riville jakaantuvien tietojen tunnistaminen yhtenäiseksi viestiksi on sääntö, jonka toimimattomuudella on seurauksensa. Esimerkiksi dokumentti, jonka viestikentässä on pelkkä sulkumerkki ”}”, on turhaa kohinaa datassa. Vastaavasti ennen virhetilannetta suoritetut viimeisimmät funktiokutsut eli niin sanottu pinojälki tulostuu aina eri riveille. Tässä tapauksessa ne aina alkavat välilyönnillä ja välilyöntiä käytetäänkin monirivisen viestin tunnistamiseen.

Prosessoinnissa container-tyyppinen syöte luo automaattisesti message-kentän ja sijoittaa lokitekstin siihen. Mikäli viesti on merkkijonomuotoon muunnettu JSON-objekti, tämä parsitaan takaisin JSON-muotoon data-nimiseen kenttään. Mikäli parsinta ei onnistu, ei Filebeat aiheuta prosessointia keskeyttävää virhettä. Tapahtumaa rikastetaan lisäksi kontin metadatalla. Prosessoidun viestin perusteella valitaan indeksi, johon dokumentti lähetetään. Mikäli JSON-kenttien dekoodaus on onnistunut, tapahtumassa on kenttä data.assetId ja dokumentti ladataan päivämäärällä varustettuun laitteet-indeksiin.

Kuvassa 5 on kehitysaikaisia, erilaisia indeksien nimeämiskäytäntöjä. Malli voi olla agentti-versio-päivämäärä-järjestysluku tai agentti-ympäristö-järjestelmä-päivämäärä(-järjestysluku). Järjestysluku tarkoittaa tapausta, jossa indeksi on täyttänyt elämänkaaren hallinnassa määritellyn rotaation (rollover). Ehto voi olla esimerkiksi koon kasvaminen yli 10Gb tai vuorokauden vaihtuminen. Kuvassa 5 Kibanan konsoliin syötetyn kyselyn ”GET _cat/indices” vastauksessa näkyvät järjestyksessä indeksin terveys, tila, nimi, tunniste, ensisijaisten sirpaleiden määrä, replikoiden määrä, dokumenttien määrä, poistettujen dokumenttien määrä, ensisijaisen indeksin viemä levytila sekä koko indeksin viemä tila. Valitettavasti kuvaan ei saa sarakkeiden nimiä. Tämä on myös normaalia Kibanan käyttäjäkokemusta.

Indeksien ollessa kunnossa niiden tila olisi vihreä. Indekseille on asetettu replikat. Elasticsearch yrittää asettaa replikat toiseen noodiin talletukseen, mutta koska paikallisesti klusterissa on vain yksi noodi, ovat replikat unassigned-tilassa ja siten indeksin terveyttä indikoidaan keltaisella. Lisäksi, kun replikat ovat unassigned-tilassa, on indeksin viemä levytila sama kuin ensisijaisen sirpaleen.

```
yellow open filebeat-localdev-instruments-2021.01.14 BNBJJIgeTI-duSs-SZqZMQ 1 1    1731   0   336kb   336kb
yellow open filebeat-localdev-instruments-2021.01.15 SD0CszrMQfSjKIvqeuLbLA 1 1   24210   0   4.1mb   4.1mb
yellow open elastalert_status_silence                PuNJpRPxR06jCosDlO2giw 1 1     303   0  72.6kb  72.6kb
yellow open filebeat-localdev-testindex-2021.01.27   vsZS3Oi8Qui6Fz_oilr8zQ 1 1     473   0 348.5kb 348.5kb
yellow open filebeat-7.9.1-2021.01.25-000010         q0lwODUcR9CeS5TKrzQhHQ 1 1       0   0    208b    208b
yellow open filebeat-localdev-testindex-2021.01.26   4yuF2C8RRd2QWc_NwJHtDA 1 1    6428   0 507.6kb 507.6kb
yellow open filebeat-localdev-instruments-2021.01.18 YzmxcgomQb2Qwpn-dCx9fw 1 1   20106   0   3.2mb   3.2mb
yellow open filebeat-localdev-instruments-2021.01.19 -2DjcA3ITbS_P1kykNDhJA 1 1   30432   0     4mb     4mb
```

Kuva 5. Kibanan konsolissa Elasticsearchin indeksejä tietoineen

5.4 Laiteviestit prosessoiva järjestelmä pilvipalvelussa

Laitteiden virheviestejä käsittelevässä järjestelmässä Filebeatin saataville lokit saadaan yksinkertaisesti tulostamalla konsoliin. Filebeat lukee konttien lokeja, jonne konsolitulostukset oletuksena tallentuvat. Filebeatin prosessointi oli valmiiksi asetettu ohjaamaan dokumentit oikeaan indeksiin kentän assetId perusteella. Kuvassa 6 on laitteen virheviesti JSON-muodossa tarpeellisine kenttineen. Viestien luokittelua voi käsittelevässä järjestelmässä ohjelmallisesti säätää level-kentässä. Tähän tasoon on myös helppo kiinnittää sääntö Elastalertissa.

assetAlert.json > ...

```json
{
    "assetId": "testAsset",
    "assetType": "instrument",
    "assetLocation": "Tallinn",
    "assetOwner": "Customer x",
    "level": "fatal",
    "message": "Fatal error in instrument"
}
```

Kuva 6. Esimerkki laitteen virheviestistä JSON-muodossa

Tässä vaiheessa kaikki data talletetaan. Satojen virheviestien tallettaminen tunnissa ei pitkällä aikavälillä ole mielekästä. Ongelma ei ole Elasticsearchin kapasiteetti. Elasticsearch pystyy helposti käsittelemään satoja viestejä minuutissa. Virhetilanteessa voi tulla paljon samanlaisia viestejä, eikä samanlaisten viestien tallettaminen välttämättä lisää informaatiota. Kaikki viestit voi tallettaa muutamiksi viikoiksi, mutta sitä pitempään ei kaikkea ole hyödyllistä säilyttää. Järjestelmää käytettäessä pitemmällä aikavälillä onkin pohdittava, riittääkö pelkkä hälytysten tallettaminen vai talletetaanko virheviestit koosteina. Tuotannossa syntyvän datamäärän ennakointi ei tässä vaiheessa ole mahdollista, joten tuotannossa datan hallinnan suunnitelma on tehtävä tietotaidon kerryttyä.

5.5 Elastic-pinon hälytysjärjestelmät

Elastic-pinossa on sisäänrakennettuna kaksi hälytystyökalua. Molemmat on integroitu Kibanaan ja molemmat kuuluvat Elastic-pinon maksulliseen tilaukseen. Elastic-pinon

maksullinen tilaus ei kuulu projektin suunnitelmiin. X-Packiksi kutsuttu lisäosa oli aiemmin kokonaan maksullinen, mutta nykyään jotkut siihen kuuluvista ominaisuuksista ovat ilmaiseksi käytettävissä.

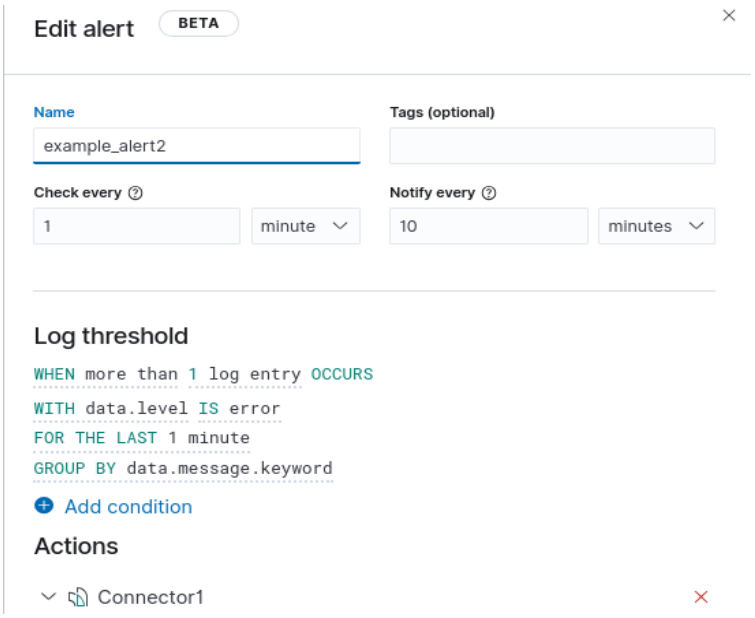
Watcher on ensimmäinen Elastic-pinoon luotu hälytysjärjestelmä. Myöhemmin on lisätty nimellä Kibana Alerts and Actions toimiva työkalu. Elasticsearchin dokumentaatiossa nämä kuvataan käyttötarkoituksiltaan samantyyppisiksi, mutta toisistaan riippumattomiksi järjestelmiksi. Perusosat muistuttavat Elastalertia ja ovat molemmissa syöte, aikataulu, ehto ja toiminnot. Syöte voi olla yksittäinen tai useita eri indeksejä Elasticsearchista. Aikataulu tarkoittaa sykliä, kuinka tiheästi kysely suoritetaan. Ehto on tila, jota tarkkaillaan. Ehdon täyttyessä laukaistaan toiminto. Toiminto voi olla esimerkiksi webhookin tai sähköpostin lähettäminen.

Watcher on kokonaan maksullinen ominaisuus, eikä sitä testattu. Kibanan hälytystoimintoa voi käyttää rajatusti ilmaiseksi. Mikäli ilmaiset toiminnot olisivat riittävät, olisi hälytysjärjestelmän käyttöönotto huomattavan helppoa.

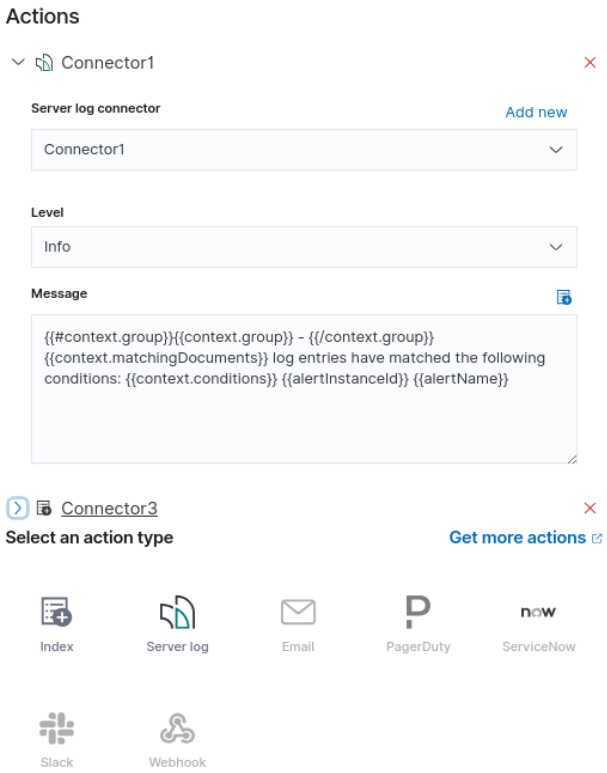
Kibanan hälytysten testaus

Hälytysten luonti onnistuu tekemällä valinnat lomakkeelle (kuva 7). Syöte eli skannattava indeksi on valittu jo edellisessä vaiheessa. Hälytyksen tyyppi on valittu lokidatalle sopivaksi. Kentän data.level arvon ollessa error tai korkeampi ja kun minuutissa rekisteröidään yksi kirjaus, tehdään hälytys. Valinta ”For the last minute” ei ehkä ole lokidatan kohdalla itsensä selittävä optio. Ehto sopii paremmin mitattavaan arvoon, kuten prosessorin kuormitukseen. Mikäli kuorma olisi esimerkiksi minuutin yli raja-arvojen, aiheutuisi hälytys. Prosessorikuormahan voi tippua raja-arvon alle aikajakson aikana, jolloin hälytystä ei syntyisi.

Kuvan 8 alaosassa on konnektorit, joita käytetään toimintoihin. Konnektorille määritellään kriteeri, jolloin sitä käytetään, tässä tapauksessa taso. Konnektorille luodaan viesti, jossa voidaan käyttää hälytyksen dataa. Viimeiseksi konnektori yhdistetään toimintoon, kuten sähköpostiin tai webhookiin. Maksullisia ovat kaikki muut paitsi hälytysten kirjaaminen indeksiin tai kirjoittaminen lokiin.



Kuva 7. Kibanassa hälytyksen luonti



Kuva 8. Toiminnot linkitetään konnektoreihin

Kuvassa 9 on käynnissä oleva hälytysinstanssi. Hälytystä ei ole vielä tehty, vaan tarkkailtava ehto on ollut käynnissä 8 sekuntia. Kun määritelty kesto tulee täyteen, suoritetaan hälytykseen liitetty toiminto.

Kuva 9. Kibanassa käynnistynyt hälytysinstanssi

Ongelmallista oli, ettei ilmaiseksi voi lähettää webhookeja tai sähköpostia, mitkä ovat ensisijainen hälytystapa tässä projektissa. Siksi Kibanan hälytykset hylättiin. Kibanan hälytyksiä hallinnoi Elastic ja niitä kehitetään jatkuvasti. Siten on hyvin mahdollista, että jatkossa kätevin tapa hälytysten luontiin ovat Kibanan hälytykset, varsinkin jos Elastalertin kehitystyöhön ei tule uutta virallista jatkajaa.

6 Elastalertin käyttöönotto

6.1 Elastalertin asentaminen

Elastalertin käyttöönotossa oli kaksi selkeää haastetta. Elastalertista ei ollut aiempaa pohjatietoa tiimissä tai käyttöönottoa. Toiseksi Elastalertin aktiivisen kehityksen loputtua luotettavan ja ajantasaisen asennuskonfiguraation löytäminen internetistä oli yllättävän hankalaa. Yelpin (a) Elastalert-repositoriossa viitataan konttitoteutukseen, jota on viimeksi päivitetty toukokuussa 2019 (Bitsensor). Ansible-pelikirjoista monet ovat vanhentuneita, useimpia niistä on päivitetty vuosia sitten, tuoreinta heinäkuussa 2020 (Freedomofpress). Kehitystyön aikana kokeiltiin monia eri asennusvaihtoehtoja ja useimmat osoittautuivat haastaviksi saada toimimaan. Ongelmia ilmeni pääasiassa vanhentuneiden ohjelmistojen versioiden takia. Muutamassa vuodessa Python, Elastalertin riippuvuksien riippuvuudet ja Linux-levykuvat ovat kehittyneet ja tulleet keskenään epäsopiviksi. Useimpia netistä löytyviä Elastalertin Dockerfilejä voi käyttää vain pohjana, koska sellaisenaan ne eivät toimi.

Kehitystyötä tehtiin kuvassa 11 esitetyllä konttimäärittelyllä. Kyseistä repositoriota on päivitetty viimeksi maaliskuussa 2021. Tekijä ilmoittaakin version haarautetuksi alkuperäisestä Yelpin repositoriosta ja ylläpitävänsä haaraa siihen asti, kunnes uusi ylläpitäjä on määrätty alkuperäiselle repositoriolle. (Jertel.)

Elastalertin kontti luodaan Alpine-Linux levykuvasta. Hyvä puoli on tarkka versio 3.6, latest-merkityt imaget olivat päivittyneet Elastalertin suhteen yhteensopimattomiksi. Konttiin asennetaan useita Elastalertin asentamiseen vaadittavia apk-paketteja. Tässä määrittelyssä huomionarvoista on yksittäinen komento, jossa Elastalert asennetaan suoraan Pythonin pakettienhallinnasta (kuva 10). Yksinkertaisimmillaan käyttöönotto on näinkin helppoa, kunhan asennukseen tarvittavat edellytykset ovat kunnossa.

```
12  RUN pip install elastalert==${ELASTALERT_VERSION} && \
```

Kuva 10. Elastalertin asennus pip-paketinhallintaohjelmalla

Konttiin kytketään emokoneen levyltä peruskonfiguraatiotiedosto sekä kansio (kuva 11), joka sisältää hälytysten luomisen säännöt. Tämä on riittävä alustus Elastalertin sääntöjen ja hälytysten paikalliseen kehitykseen. Käytäntö osoitti, että aina kyselyn jälkeen ja sääntöä testatessaan Elastalert lukee säännön tiedostosta, eli sääntöä voi muuttaa Elastalertin ollessa käynnissä. Elastalertin virallisessa dokumentaatiossa oli ilmoitettu, että aina sääntöä muutettaessa ohjelma pitäisi käynnistää uudestaan.

docker-compose.yml

```yaml
295   elastalert:
296     build: elastalert
297     environment:
298       ELK_VERSION: "7.9.1"
299     volumes:
300       - ./elastalert/elastalert.yaml:/opt/config/elastalert_config.yaml
301       - ./elastalert/rules:/opt/elastalert/rules
302     depends_on:
303       - elasticsearch
304       - kibana
```

elastalert > Dockerfile > ...

```
1   FROM python:3.6-alpine
2
3   LABEL description="ElastAlert suitable for Kubernetes and Helm"
4   LABEL maintainer="Jason Ertel (jertel at codesim.com)"
5
6   ARG ELASTALERT_VERSION=0.2.4
7
8   RUN apk --update upgrade && \
9       apk add gcc libffi-dev musl-dev python3-dev openssl-dev tzdata libmagic && \
10      rm -rf /var/cache/apk/*
11
12  RUN pip install elastalert==${ELASTALERT_VERSION} && \
13      apk del gcc libffi-dev musl-dev python3-dev openssl-dev
14
15  RUN mkdir -p /opt/elastalert && \
16      echo "#!/bin/sh" >> /opt/elastalert/run.sh && \
17      echo "set -e" >> /opt/elastalert/run.sh && \
18      echo "elastalert-create-index --config /opt/config/elastalert_config.yaml" >> /opt/elastalert/run.sh && \
19      echo "exec elastalert --config /opt/config/elastalert_config.yaml \"\$@\"" >> /opt/elastalert/run.sh && \
20      chmod +x /opt/elastalert/run.sh
21
22  ENV TZ=Europe/Moscow
23
24  WORKDIR /opt/elastalert
25  ENTRYPOINT ["/opt/elastalert/run.sh"]
```

Kuva 11. Elastalert-kontin käynnistysasetukset

Elastalertin perusasetuksissa (kuva 12) vaaditaan vain muutamia pakollisia tietoja. Elastalert tallettaa takaisinkirjoitusindeksiin tilansa häiriöiden, kuten Elastalertin kaatumisen tai Elasticsearch-yhteyden katkeamisen takia. Indeksin luomiskomento on Elastalertin Dockerfilessä rivillä 18 (kuva 11).

Takaisinkirjoitusindeksi elastalert_status, johon Elastalert tallettaa metatietoja, ei ole ainut indeksi, jonka Elastalert luo. Elastalert luo yhteensä neljä indeksiä. Nimettyyn elastalert_statukseen talletetaan luodut hälytykset. Toisen indeksin Elastalert luo liittämällä edellisen perään ”_status” eli tuloksena on elastalert_status_status. Tähän indeksiin talletetaan suoritetut kyselyt. Siten esimerkiksi yhteyden katkettua ja palauduttua Elastalert sisällyttää kyselyyn kaiken datan viimeisimpään kyselyyn asti. Kahteen muuhun indeksiin, elastalert_error ja elastalert_silence talletetaan virheet sekä hälytysten hiljennykset.

```
elastalert.yaml > ...
  # The elasticsearch hostname for metadata writeback
  # Note that every rule can have its own elasticsearch host
  es_host: elasticsearch
  es_port: 9200
  # This is the folder that contains the rule yaml files
  # Any .yaml file will be loaded as a rule
  rules_folder: rules
  # How often ElastAlert will query elasticsearch
  # The unit can be anything from weeks to seconds
  run_every:
    minutes: 1
  # ElastAlert will buffer results from the most recent
  # period of time, in case some log sources are not in real time
  buffer_time:
    minutes: 1
  # The index on es_host which is used for metadata storage
  # This can be a unmapped index, but it is recommended that you run
  # elastalert-create-index to set a mapping
  writeback_index: elastalert_status
  # If an alert fails for some reason, ElastAlert will retry
  # sending the alert until this time period has elapsed
  alert_time_limit:
    days: 2
```

Kuva 12. Elastalertin perusasetuksia

6.2 Hälytysten testaaminen

Hälytysten testaamista varten paikalliseen kehitysympäristöön kytkettiin testilaitteita ja saatettiin ne mekaanisesti vikatilaan. Siten sääntöjen asetuksia ja hälytysten toimivuutta voitiin testata vastaavalla sisään tulevalla datalla kuin tuotannossa, vaikka volyymi pysyikin pienempänä.

Hälytyksiä kaiutettiin konsoliin sekä lähetettiin testikanavalle Teamsiin (kuva 13, rivit 15–16) kehitystyön aikana. Testilaitteista huomattiin, että virheen tyypistä riippuen laitteet on asetettu lähettämään virhe aina vähintään kaksi kertaa, joten sääntö asetettiin huomioimaan toistuvat viestit. Sääntönä käytettiin kahta error- tai fatal-tason tapahtumaa kahdessa minuutissa. Avainsanalla query_key tulokset muodostetaan per uniikki alkio. Koska laitteen tunniste on yksilöllinen, jokaisen laitteen tuloksia tarkastellaan ja koostetaan muiden laitteiden virheitä huomioimatta. Kenttä, johon query_key kohdistuu, kannattaa olla tyypitetty keywordiksi.

Hälytykset ovat hiljennettyinä kuvassa 13 asetettuun realert-arvoon asti. Näin pystytään luomaan tilatieto. Virhe voi pysyä pitkäänkin päällä. Uudelleen tulevasta hälytyksestä

nähdään, onko hälytys vielä päällä. Kun hälytyksiä aktivoidaan vain 50 minuutin välein, vältytään ylimääräiseltä kohinalta.

```
name: Development frequency alarm

# the frequency rule type alerts when num_events events occur with timeframe time
type: frequency
index: filebeat-laitteet-*
filter:
- query:
    query_string:
      query: "data.level:error OR data.level:fatal"

num_events: 2
timeframe:
  minutes: 2
alert:
  - ms_teams
  - command

command: ["echo",  "{@timestamp} {num_hits} hits {data[message]} message {data[assetName]} assetName"]

ms_teams_webhook_url: "https://webhook.url"
alert_text_type: alert_text_only
ms_teams_alert_summary: "test"
realert:
  minutes: 50

query_key: data.assetId
```

Kuva 13. Testihälytyksen asetukset

Hälytyksen runkona käytettiin alert_text_only-tyyppiä, jossa hälytyksen sisällöksi tulee vain hälytyksen nimi sekä alert_text-kenttä (kuva 14). Kenttään alert_text voi argumentteina asettaa sääntöön täsmänneestä dokumentista kenttiä, koska osuma tulee syötteenä hälytykselle.

```
alert_text: |
  Alert Data<br>
  Aika: {0}<br>
  Dokumentteja: {1}<br>
  Osumia: {2}<br>
  Laitteen id: {3}<br>
  Indeksi: {4}<br>
  Tyyppi: {5}<br>
  Viesti: {6}

alert_text_args:
- "@timestamp"
- num_hits
- num_matches
- data.assetId
- _index
- data.type
- data.message
```

Kuva 14. Hälytyksen tekstirunko

Testivaiheessa selvisi, että kaikkea ei kannata yrittääkään tehdä yhdellä säännöllä. Kanava, joka on Teams, sähköposti tai joku muu, johon lähetetään hälytys, saattaa helposti

saada useita ilmoituksia päivässä, kun laitteita on paljon. Siten hyvä käytäntö olisi esimerkiksi luoda sääntö muuten samoilla asetuksilla, mutta hälyttää vuorokauden välein ja eri kanavalle. Tällöin tieto välittyy heti, mutta ylimääräisiltä ilmoituksilta vältytään. Vuorokauden välein ilmoittaminen sopisi hyvin ensisijaiselle kanavalle kuten sähköposti.

Tässä vaiheessa on paikallisesti tehty tarvittava. Toistuvat laiteviestit olivat suurin mietinnän aihe. Vikatilanteissa viestejä voi minuutissa tulla paljon, joten hälytyksissä osumien määrä ei sinänsä ole kuin suuntaa antava. Useita kymmeniä olisi paljon, vain muutama vähän. Sen päättely, mitä hälytysten määrä indikoi, jää ihmiselle. Itsestään poistuvan, satunnaisesti tai säännöllisesti ilmenevän vikahälytyksen analysointi ja ongelman selvittäminen on hankalaa. Silloin historiatietoa, eli lokiviestejä ennen hälytystä, voi hyödyntää päättelyssä. Olisi mielenkiintoista testata vikatilaa edeltävää dataa koneoppimisen algoritmeilla.

7 Tuotantoon vienti

Tuotantoon viennissä oli monia vaiheita. Tuotannon asennuskonfiguraatiot ovat erilaiset kuin paikallisesti käytetyt ja tuotannossa on eri tavalla kapasiteettia laitteistossa kuin paikallisessa kehityksessä. Sovellukseen tulevia pyyntöjä ja muuta käyttäjien liikennettä on tuotantoympäristössä enemmän, joka näkyy muun muassa moninkertaisena Filebeatin resurssien käyttönä. Rauhallisessa järjestelmässä Filebeat voi varata muistia esimerkiksi 50 megatavua, mutta 400–1200 megatavua on kiireisemmässä järjestelmässä tavallista.

Kehitysympäristössä ennen tuotantoa Filebeatista löytyi ominaisuus, joka tulee huomioida Filebeatin käynnistyksessä. Filebeat aloittaa käynnistyessään kaikkien sille annettujen tiedostojen lukemisen. Mikäli järjestelmä on ollut pitkään käynnissä, voivat vanhimmat lokitukset olla vuosia vanhoja. Filebeat oli konfiguroitu lähettämään dataa nimettyihin, päiväkohtaisiin indekseihin. Näin ollen jokaiselta päivältä, vanhimmasta lähtien, luodaan oma päiväkohtainen indeksitaulu, jossa on sirpaleita vähintään yksi.

Ongelmaksi muodostui, kun Elasticsearchin sirpaleiden maksimimäärä täyttyi ja Filebeat alkoi lokittamaan virhettä. Filebeat luki omaa lokiaan ja yritti lähettää sitä Elasticsearchiin, mikä ei onnistunut. Virheen lähetyksen epäonnistuminen aiheutti uuden virheen. Näin jokainen Filebeatin lokiin kirjattu tapahtuma alkoi kumuloitumaan. Filebeat-kontin loki täytti koko kehityspalvelimen kovalevyn yön aikana.

Kaksi ratkaisua otettiin käyttöön. Filebeatin oma lokitus siirrettiin Filebeatin ulottumattomiin. Toiseksi nimettyjen päiväkohtaisten indeksien sijaan Elasticsearchiin luotiin lokidatalle huomattavasti sopivampi indeksointimalli, niin kutsuttu datastriimi. Lyhyt kuvaus datastriimistä on, että se sisältää Elasticsearchin automaattisesti hallinnoimat sirpaleet, joista vain uusimpaan kirjoitetaan. Kyselyt kohdistuvat indeksirekisteriin, eli kaikkiin kyseisen datastriimin sirpaleisiin. Elasticsearchin datastriimi on kehitetty metriikoille ja lokidatalle. Datastriimin etuna on, ettei esimerkiksi pieniä sirpaleita tule paljon. Itse asiassa Elasticsearchin sirpaleiden maksimimäärän täyttyminen oli heikkoa optimointia ja siten myös ongelma. Ylisirpaloituminen (oversharding) on Elasticsearchin ideaalisen toiminnan kannalta vältettävä tilanne. Optimimäärä ensisijaisille sirpaleille joidenkin lähteiden mukaan per indeksi on 1–2 ja koko 10–50 gigatavua. Päiväkohtaisesti nimettyihin indekseihin lataamalla sirpaleet jäävät helposti pienemmiksi. Sirpalemäärälle on suositus 20 tai vähemmän per gigatava Elasticsearchille varattua muistikekoa. (esim. Elastic l.)

Filebeat-kontin päivitys kehityspalvelimella aiheutti kaikkien lokien, eli myös vuosia vanhojen, lokien uudelleenlähetyksen. Tämä ratkaistiin tallettamalla Filebeatin datakansio

pysyvään paikkaan, eli isäntäkoneelle, ulos kontista. Data-kansio sisältää rekisterin, johon Filebeat tallettaa kaikista tiedostoista vähintäänkin viimeisen onnistuneesti lähetetyn rivin.

Datastriimiin siirtymisessä tavoiteltiin myös optimointia, joka saattaa olla ennenaikaista. Parissa viikossa Elasticsearch varasi muistia yhdeltä palvelimelta 64:sta gigatavusta kaiken. Muistin varaaminen vaikuttaa olevan ominaisuus, käyttämätön muisti on hukattua muistia. 99 prosenttinen muistinkäyttö viikkotolkulla ei tunnu intuitiivisesti normaalilta, mutta todellisuudessa on hyvin vaikea sanoa, milloin koneelta oikeasti loppuu kapasiteetti. Koska esimerkiksi tyypityksien muutokset jälkikäteen voivat olla työläitä, etukäteen optimointi on tässä tapauksessa perusteltua. Todellisuudessa välimuistissa (cache) on n. 50 % käytetystä muistista.

Kuvassa 15 on otos Elasticsearchin resurssien käytön ilmoituksesta. Luvut ovat suuria: 16 gigatavua muistia (keossa) ja yli 2800 avointa tiedostoa (open_file_descriptors).

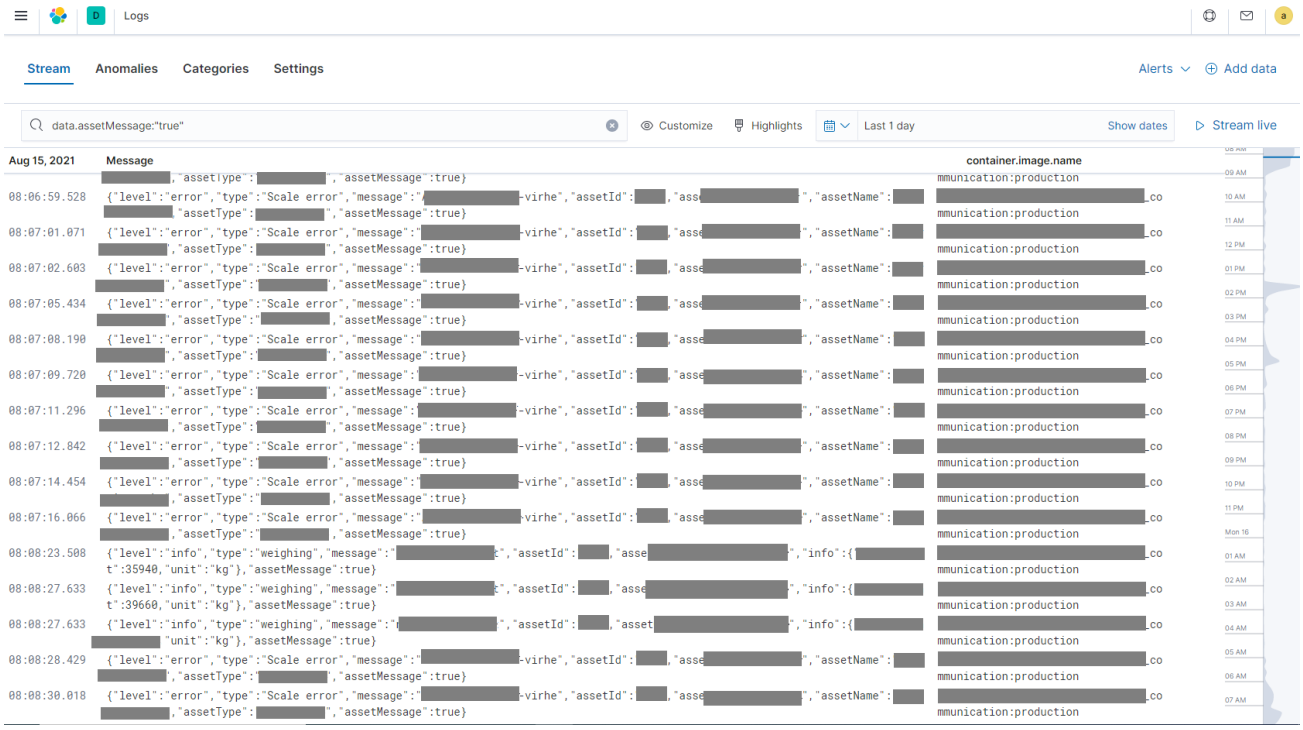

```
286    "process" : {
287      "cpu" : {
288        "percent" : 1
289      },
290      "open_file_descriptors" : {
291        "min" : 2828,
292        "max" : 2828,
293        "avg" : 2828
294      }
295    },
296    "jvm" : {
297      "max_uptime_in_millis" : 1899835219,
298      "versions" : [
299        {
300          "version" : "14.0.1",
301          "vm_name" : "OpenJDK 64-Bit Server VM",
302          "vm_version" : "14.0.1+7",
303          "vm_vendor" : "AdoptOpenJDK",
304          "bundled_jdk" : true,
305          "using_bundled_jdk" : true,
306          "count" : 1
307        }
308      ],
309      "mem" : {
310        "heap_used_in_bytes" : 15983788072,
311        "heap_max_in_bytes" : 27917287424
312      },
313      "threads" : 116
```


Kuva 15. Elasticsearchin dataa resurssienkäytöstä

Elasticsearchin tietoturva-asetukset ovat monipuoliset. Käyttöoikeuksien hallinnassa lukijoiden ja kirjoittajien oikeudet roolitetaan erikseen. Tietoturva-asiat vaativat myös muuta

ohjelmistoinfraa tueksi ja ne piti lisäksi järjestää. Elastalertille oikeuksien luonnissa yllätyksenä tuli, että Elastalert ei anna minkäänlaista virheilmoitusta, mikäli kysely ei onnistukaan esimerkiksi oikeuksien puuttuessa.

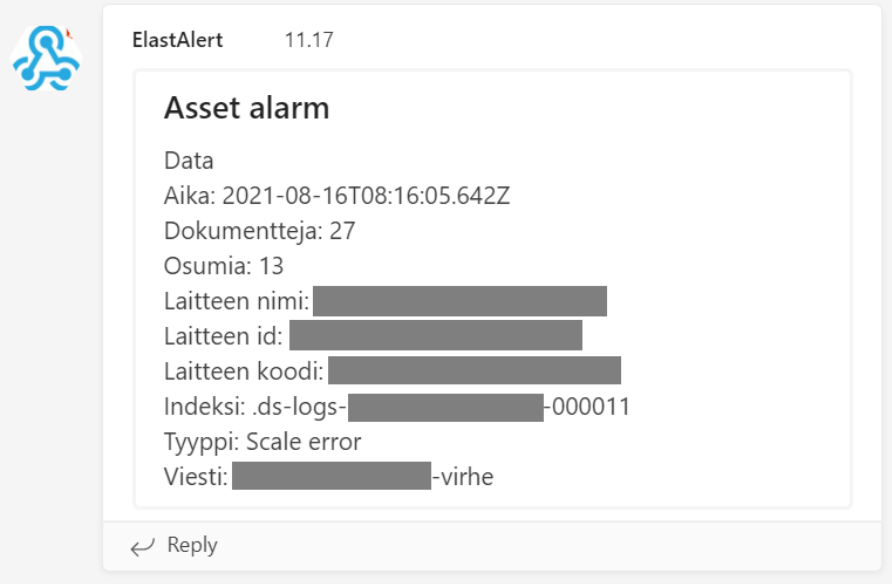
Yllättävintä tuotannossa oli Filebeatin resurssien käyttö kuormitetummassa ympäristössä. Tämä ei kuitenkaan ole toistaiseksi ongelma, eikä siten vaadi optimointia. Lisäksi Filebeat ilmoittaa vaikeasti ymmärrettävistä virheistä. Virheiden käsittelyä vaikeuttaa Go-kielen joskus olemattomat tai lyhyet virhetilanteiden pinojäljet. Esimerkiksi jotkut palvelut konteissa voivat lokittaa arvaamattomia asioita, kuten 5 megatavua null-merkkiä. Iso määrä nullia aiheuttaa myös Filebeatille ongelmia. Jos ja kun tällaiset sattumukset on hoidettu, nähdään että Elasticsearch sujuvasti tallettaa miljoonia dokumentteja päivässä. Kuvassa 16 tuotantoympäristöstä Kibanan oletusnäkymä lokien tarkasteluun. Lokeja on rajattu laiteviesteihin kentän data.assetMessage arvolla true.



Kuva 16. Kibanan lokinäkymä tuotantoympäristössä

Kuvassa 17 on Teamsiin tullut hälytys. Hälytyksiä tulee useita ja viestissä on vielä kehitettävää. Osumat ovat sääntöjen täsmäämisiä. Tuotantolaitteiden virheiden syyt ja tuotannossa syntyvät määrät oli vaikea ennakoida. Nämä ovat peräisin laitteiden lähdekoodista,

joka voi olla vanhaakin. Tuotantoympäristön laitteiden hälytyksistä kuitenkin huomataan, että data pitää tuntea ennen mahdollista asiakkaille julkaisua.



Kuva 17. Elastalertin hälytys Teamsissa

Kun Elastic-pinoa tukeva ohjelmistoinfra on kehitetty, Elastic-pinon asennus automatisoitu, tietoturva-asiat ratkottu ja vakavat virheet korjattu, prosessi jatkuu. Käyttöönoton jälkeen tarkennetaan asetuksia, kuten datan strukturointia, ohjelmistojen optimointia ja hälytysten raja-arvoja. Datan elinkaari kannattaa suunnitella vasta kun tuntee dataa ja elinkaaren mukaan suunnitella myös arkistointi. Lokitusta ja sitä myöten hälytyksiä on mahdollista laajentaa muihinkin pilvipalvelun toimintoihin. Muiden datan lataajien, kuten Metricbeatin käyttöönotolla, voidaan laajentaa jo olemassa olevia toiminnallisuuksia.

8 Yhteenveto

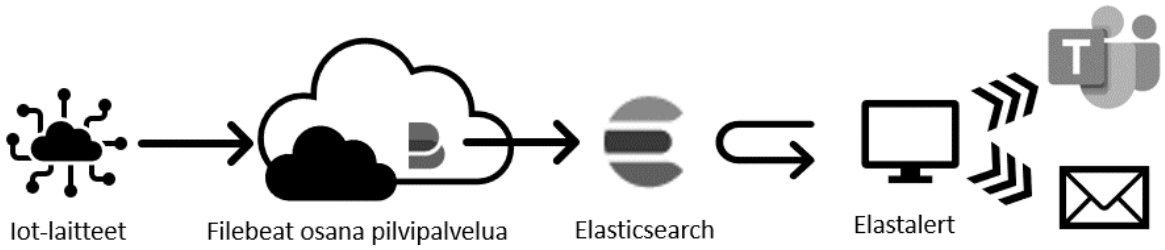
Työssä tavoitteena oli kehittää teollisten laitteiden kunnossapitoon järjestelmä, jossa laitteiden lokeja talletetaan ja vikatiloista saadaan hälytys. Teoriaosuudessa tutkittiin teollista kunnossapitoa ja hälytyksiä osana sitä, datan keräystä Big Data -kehyksessä sekä Elastic-pinoa data-alustana.

Käytännön osuudessa luotiin paikallinen Elastic-pinon kehitysympäristö, johon liitettiin myös testilaitteita. Oikeiden laitteiden viestien perusteella kehitettiin sekä dataputkea että Elastalertin hälytyksiä ja sääntöjä, joiden toimivuus testattiin paikallisesti. Kibanan hälytyksien toiminnallisuutta testattiin osana kehitystyötä. Kibanan hälytykset kuuluvat maksaville Elastic-pinon tilaajille ja ilmaiset toiminnot eivät olleet riittäviä tämän työn käyttötarkoituksiin.

Käytännön osuuden tavoitteena oli luoda malli, joka toteuttaa seuraavat käyttäjätarinat:

- Pilvipalvelu kerää laitekohtaisia status- ja lokitietoja
- Ylläpitäjä voi priorisoida ja luokitella viestejä ja poikkeustilanteita
- Ylläpitäjä voi saada ilmoituksen laitteiston vikatilasta
- Ylläpitäjä voivat nähdä status- ja lokitietoja ongelmaan ja tapahtumien kulkuun liittyen

Luodulla mallilla pystytään toteuttamaan kaikki yllä mainitut käyttäjätarinat. Lokitietoja vastaanotetaan, muunnetaan ja lähetetään talletettavaksi pilvipalvelussa. Muunnettaessa pystytään lisäämään struktuuria kiinnittämällä metatietoja, kuten luokittelua ja priorisointia. Elastalert asetetaan lähettämään vikatilan alkamisesta sähköposti ja vikatilasta voidaan tilatietoa päivittää eri kanavaan kuten Teamsiin. Dataan lisättyjen metatietojen avulla datamassasta voidaan etsiä yksittäisten laitteiden tietoja ja tapahtumia eri aikaväleillä. Kuviossa 6 on esitetty järjestelmän rakenne ja datan kulku.



Kuvio 6. Lokidatan kulku laitteilta pilvipalvelun kautta Elasticsearchiin Elastalertin saataville

Elastic-pinosta voidaan todeta sen olevan data-alusta, jossa on Big Datan termistöstä ominaisuuksia kuten muistinvarainen ja useille noodeille hajautettu analytiikka. Elasticsearchin vahvuudet ovat erityisesti ilmainen skaalattavuus ja nopeat tekstihaut. Heikkoutena on

useiden kehittyneiden toimintojen rajaus maksaville käyttäjille. Koneoppimisen algoritmit ja hälytykset kuuluvat muun muassa maksulliseen tilaukseen.

Ympäristön pystyttäminen ja dataputken rakentaminen tietoturvallisesti on pohjatyötä ennen varsinaista datan louhintaa, tässä tapauksessa hälytysten luontia. Kun laitteiden lähettämä data on tunnetussa muodossa, on luokittelu, haut ja suodatus ohjelmallisesti helppoa.

Vaikeinta datan keräämisessä on ennakoida, mitä ominaisuuksia datalta halutaan tulevaisuudessa. On vaikea sanoa, mitä kannattaisi tallettaa pitkäaikaisesti: ilmoituksia, hälytyksiä, vai koosteita vain toisesta. Kun dataa oppii tuntemaan, on sen elinkaaren ja käyttötarkoituksien suunnitteluun enemmän tietotaitoa.

Järjestelmän pystyttämisen jälkeen on hyvin strukturoidusta datasta helppoa luoda sääntöjä ja hälytyksiä. Testilaitteilla nähtiin, että viestejä voi tulla vaihtelevia määriä. Vikailmoituksia ja hälytystä edeltäviä tilatietoja olisi mielenkiintoista käyttää koneoppimisen algoritmeissa vikaantumisen mallintamiseen.

Tuotantokäyttöön viedyssä toiminnallisuudessa on vielä paljon kehitettävää. Monenlaisia hälytyksiä tulee ehkä liiankin kevyesti ja viestikanavat tukkeutuvat helposti. Hälytyksille pitää mahdollisesti luoda oma käyttöliittymä, varsinkin hälytysten tyyppien, lähteiden ja volyymin kasvaessa. Googlella on käytössä hälytysten analysointiin oma työkalu Outalator (Krabbe 2017) joka antaa osviittaa ison järjestelmän hälytysten hallinnasta. Hälytysten hallinnan lisäksi pitempiaikainen datan säilöntä ja arkistointi vaatii suunnittelua, kuin myös Filebeatin tietyt kuormittavat ominaisuudet, jotka ilmenivät vasta suuren mittakaavan järjestelmässä.

Tässä opinnäytetyössä toteutettiin onnistuneesti prosessi uuden teknologiapinon käyttöönotosta tuotantoympäristöön. Tämän prosessin alussa tutustuttiin teoriaan ja vähitellen kehitys, tutkiminen ja testaaminen limittyvät. Uusiin ympäristöihin vienti aiheutti uusia ongelmia ja haasteita ratkottaviksi. Kun ohjelmistopino toimii tuotannossa niin luotettavasti, ettei se tarvitse jatkuvaa valvontaa, alkaa laajentaminen, jo olemassa olevien toimintojen tarkentaminen ja optimointi. Elastic-pinossa on mahdollisuuksia kerätä muutakin dataa pilvipalvelun osista ja käyttöliittymätyyppinen datan havainnollistaminen auttaa datan hyödyntämisessä. Ohjelmiston kehitysprosessi ei lopu käyttöönottoon.

Lähteet

ATK-sanakirja. Tietokoneyhdistyksen julkaisu 12. 1971.

Amazon Web Services. Monitor AWS IoT alarms and metrics using Amazon CloudWatch. Viitattu 27.2.2021. Saatavissa https://docs.aws.amazon.com/iot/latest/developerguide/monitoring-cloudwatch.html

Apache Software Foundation. a. Package org.apache.lucene.analysis. Viitattu 27.2.2021. Saatavissa https://lucene.apache.org/core/8_7_0/core/org/apache/lucene/analysis/package-summary.html#package.description

Apache Software Foundation. b. Class Pointvalues. Viitattu 28.3.2021. Saatavissa https://lucene.apache.org/core/7_3_0/core/org/apache/lucene/index/PointValues.html

Bitsensor. Elastalert Server repository. Viitattu 14.1.2021. Saatavissa https://github.com/bitsensor/elastalert

Elastic. a. Rollup Jobs. Viitattu 1.9.2021. Saatavissa https://www.elastic.co/guide/en/kibana/current/data-rollups.html

Elastic. b. eBay and Elasticsearch: This is not small data. Viitattu 27.2.2021. Saatavissa https://www.elastic.co/videos/ebay-and-elasticsearch-this-is-not-small-data

Elastic. c. Near real-time search. Viitattu 27.2.2021. Saatavissa https://www.elastic.co/guide/en/elasticsearch/reference/master/near-real-time.html

Elastic. d. Facets guide. Viitattu 13.3.2021. Saatavissa https://www.elastic.co/guide/en/app-search/current/facets-guide.html

Elastic. e. Translog. Viitattu 27.2.2021. Saatavissa https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-translog.html

Elastic. f. Heap sizing. Viitattu 28.2.2021. Saatavissa https://www.elastic.co/guide/en/elasticsearch/guide/current/heap-sizing.html

Elastic. g. Field data cache settings. Viitattu 13.3.2021. Saatavissa https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-fielddata.html

Elastic. h. Text field type. Viitattu 14.3.2021. Saatavissa https://www.elastic.co/guide/en/elasticsearch/reference/current/text.html#fielddata-mapping-param viitattu

Elastic. i. What are global ordinals? Viitattu 14.3.2021. Saatavissa https://www.elastic.co/guide/en/elasticsearch/reference/current/eager-global-ordinals.html

Elastic. j. How Filebeat works. Viitattu 27.2.2021. Saatavissa https://www.elastic.co/guide/en/beats/filebeat/current/how-filebeat-works.html

Elastic. k. Elastic as a Fundamental Core to Pfizer´s Scientific Data Cloud. Viitattu 27.2.2021. Saatavissa https://www.elastic.co/elasticon/tour/2019/boston/elastic-as-a-fundamental-core-to-pfizers-scientific-data-cloud

Elastic. l. Size your shards. Viitattu 14.3.2021. Saatavissa https://www.elastic.co/guide/en/elasticsearch/reference/current/size-your-shards.html

Davies, M. 2019. Data Electrified – Porsche is bringing Data to Everything with Splunk and the Taycan. Viitattu 27.2.2021. Saatavissa https://www.splunk.com/en_us/blog/conf-splunklive/data-electrified-porsche-is-bringing-data-to-everything-with-splunk-and-the-taycan.html

Ewaschuk, R. 2017. Monitoring distributed systems. Google. Viitattu 7.3.2021. Saatavissa https://sre.google/sre-book/monitoring-distributed-systems/

Freedomofpress. Elastalert Ansible role repositorio. Viitattu 2.4.2021. Saatavissa https://github.com/freedomofpress/ansible-role-elastalert

Google. a. Creating charts and alerts. Viitattu 27.2.2021. Saatavissa https://cloud.google.com/logging/docs/logs-based-metrics/charts-and-alerts

Heinonkoski, R., Asp, R. & Hyppönen, H. 2008. Automaatio – helppoa elämää?. Opetushallitus

Hodge, J. 2020. Splunk and McLaren Racing: Driven by Data. Viitattu 27.2.2021. Saatavissa https://www.splunk.com/en_us/blog/partners/splunk-and-mclaren-racing-driven-by-data.html

Influxdata. a. Downsampling and data retention. Viitattu 1.9.2021. Saatavissa https://docs.influxdata.com/influxdb/v1.7/guides/downsampling_and_retention/

Influxdata. b. Compare. Viitattu 13.3.2021. Saatavissa https://www.influxdata.com/products/compare

Influxdata. c. Metrics as a service model with influxdb. Viitattu 13.3.2021. Saatavissa https://www.influxdata.com/resources/metrics-as-a-service-model-with-influxdb/

Jertel, J. Elastalert-Docker repositorio. Viitattu 2.4.2021. Saatavissa https://github.com/jertel/elastalert-docker

Kaurto, A. 2018. Ethernet-pohjaisten automaatioverkkojen reaaliaikainen kunnonvalvonta. Tampereen teknillinen yliopisto. Diplomityö. Viitattu 28.3.2021. Saatavissa https://docplayer.fi/199576431-Aleksi-kaurto-ethernet-pohjaisten-automaatioverkkojen-reaali-aikainen-kunnonvalvonta-diplomityo.html

Krabbe, G. 2017. Tracking Outages. Google. Viitattu 2.4.2021. Saatavissa https://sre.google/sre-book/tracking-outages/

Kreps, J. 2013. The Log: What every software engineer should know about real-time data's unifying abstraction. LinkedIn. Viitattu 13.3.2021. Saatavissa https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

Järviö, J. & Lehtiö, T. 2012. Kunnossapito – tuotanto-omaisuuden hoitaminen. Kunnossapidon julkaisusarja n:o 10. Helsinki: KP-Media Oy.

Lahti Precision. Viitattu 3.3.2021. Saatavissa https://lahtiprecision.com/yritys/

McCandless, M. 2010. Lucene's RAM usage for searching. Viitattu 27.2.2021. Saatavissa http://blog.mikemccandless.com/2010/07/lucenes-ram-usage-for-searching.html

Microsoft 2020. Azure Monitor Logs overview. Viitattu 27.2.2021. Saatavissa https://docs.microsoft.com/en-us/azure/azure-monitor/logs/data-platform-logs

Salo, I. 2013. Big data - tiedon vallankumous. Jyväskylä: Docendo Oy.

Salo, I. 2014. Big data & pilvipalvelut. Jyväskylä: Docendo Oy.

Splunk. Customers. Viitattu 27.2.2021. Saatavissa https://www.splunk.com/en_us/customers.html

Tiirikainen, P. 2018. Lokitiedon visualisointi. Lahden ammattikorkeakoulu. Opinnäytetyö (ylempi amk). Viitattu 14.2.2021. Saatavissa https://www.theseus.fi/bitstream/handle/10024/142764/Tiirikainen_Pekka.pdf?sequence=1&isAllowed=y

Vogels, W. 2020. How Amazon is solving big-data challenges with data lakes. Viitattu 13.3.2021. Saatavissa https://siliconangle.com/2020/01/30/amazon-solving-big-data-challenges-data-lakes/

Yelp. a. Status of the Elastalert project. Viitattu 3.3.2021. Saatavissa https://github.com/Yelp/elastalert/issues/2911

Yelp. b. Elastalert repository. Viitattu 14.1.2021. Saatavissa https://github.com/Yelp/elastalert

Liite 1. Esimerkkitapaus lokituksesta osana uuden teknologian testausta

Uuden teknologian testaustyössä järjestelmän tilaa valvottiin muun muassa tallettamalla lokeja Elasticsearchiin. Dataa kertyi tasaisesti noin 444 000 dokumenttia tunnissa, eli yhteensä yli 10,7 miljoonaa vuorokaudessa. Tilaa nämä veivät levyltä 3,6gb. Huomattavaa tilankäytössä on, että jokaisessa dokumentissa oli vähintään 15 indeksoitavaa kenttää.

Liite 2. Elasticsearch-kontin käynnistysasetukset

elasticsearch > Dockerfile > ...

```
1  FROM docker.elastic.co/elasticsearch/elasticsearch:7.9.1
2
3  COPY elasticsearch.yml /usr/share/elasticsearch/config/
4  RUN chown elasticsearch:elasticsearch /usr/share/elasticsearch/config/elasticsearch.yml
5
6  USER elasticsearch
```

docker-compose.yml

```
286    elasticsearch:
287      build: elasticsearch
288      environment:
289        ES_JAVA_OPTS: "-Xmx1g -Xms1g"
290      ports:
291        - 9200:9200
292      volumes:
293        - elasticsearch:/usr/share/elasticsearch/data
```

Liite 3. Kibana-kontin käynnistysasetukset

kibana > Dockerfile > ...

```
1  FROM docker.elastic.co/kibana/kibana:7.9.1
2
3  COPY kibana.yml /usr/share/kibana/config/
```

docker-compose.yml

```
339    kibana:
340      build: kibana
341      ports:
342        - 5601:5601
343      depends_on:
344        - elasticsearch
```

kibana > ! kibana.yml

```
1  server:
2    name: kibana
3    host: 0.0.0.0
4    basePath: "/kibana"
5    rewriteBasePath: true
6  elasticsearch:
7    hosts: http://elasticsearch:9200
8  logging:
9    quiet: true
10
11 xpack.encryptedSavedObjects.encryptionKey: "1111111111222
```

Liite 4. Filebeat-kontin käynnistysasetukset

filebeat > Dockerfile > ...

```
FROM docker.elastic.co/beats/filebeat:7.9.1

COPY filebeat.yml /usr/share/filebeat/filebeat.yml
USER root
RUN chown root:filebeat /usr/share/filebeat/filebeat.yml
USER filebeat
```

docker-compose.yml

```
  filebeat:
    build: filebeat
    user: root
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "10"
    volumes:
      - /var/lib/docker:/var/lib/docker:ro
      - /var/run/docker.sock:/var/run/docker.sock
    environment:
      - ENVIRONMENT_NAME=localdev
```

filebeat.yml > ...

```
filebeat.inputs:
- type: container
  paths:
    - '/var/lib/docker/containers/*/*.log'
  multiline.pattern: '^[[:space:]]'
  multiline.negate: false
  multiline.match: after
  processors:
    - decode_json_fields:
        fields: ["message"]
        target: "data"
        overwrite_keys: true
    - add_docker_metadata:
        host: "unix:///var/run/docker.sock"

output.elasticsearch:
  hosts: ["elasticsearch:9200"]
  protocol: http
  ssl.verification_mode: none
  indices:
    - index: "filebeat-laitteet-%{+yyyy.MM.dd}"
      when.has_fields: ["data.assetId"]
    - index: # Others to other indexes

setup.ilm.enabled: auto
setup.ilm.policy_file: /usr/share/filebeat/ilm_policy.json
setup.ilm.overwrite: true
```