Haaga-Helia
ammattikorkeakoulu Oy

Skaalautuvan Mobiiliapplikaation Lokitus

Santeri Friman

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2016

Haaga-Helia ammattikorkeakoulu Oy

Tiivistelmä

7.5.2016

Tekijä(t)
Santeri Friman.

Koulutusohjelma
Tietojenkäsittelyn Koulutusohjelma

Opinnäytetyön otsikko
Skaalautuvan Mobiiliapplikaation Lokitus

Sivu- ja liitesivumäärä
43

Opinnäytetyön otsikko englanniksi
Logging for Scalable Mobile application

Opinnäytetyössä toteutettiin keskitetty lokitusjärjestelmä skaalautuvalle mobiilisovellukselle. Opinnäytetyön toimeksiantajana toimi Kvantia Oy ja tuotteen nimi on Quanter. Lokitusjärjestelmän toteuttamiseen käytettiin ELK-ohjelmistonpinoa, joka koostuu kolmesta ohjelmasta: Elasticsearch, Logstash ja Kibana. Pilvipalveluntarjoana toimi Amazon Web Services ja käytettyjen ohjelmien konfigurointien hallintaan käytettiin SaltStack-ohjelmistopinoa.

Tämän opinnäytetyön tavoite on helpottaa mobiilisovelluksessa syntyneiden odottamattomien virheiden selvittämistä ja korjaamista. Koska sovellus toimii pilvessä ja järjestelmät skaalautuvat automaattisesti, kuuluu tarvittavien konfigurointien luonti tavoitteisiin.

Lokitusjärjestelmä toteutettiin kehitysympäristöä varten. Tuotantotason lokitus ei kuulu tämän opinnäytetyön rajoituksiin. Jotain yksityiskohtia, esimerkiksi käytetyistä järjestelmistä, on myös jätetty kertomatta niiden arkaluontoisuuden vuoksi.

Tässä opinnäytetyössä kerrotaan, mitä on lokitus, mitä sen avulla voidaan saavuttaa ja mitä tekijöitä hyvä lokituksen luomisessa on. Lokituksen lisäksi käsitellään myös hieman järjestelmien arkkitehtuuria sekä hajautetun ja keskitetyn järjestelmän hyviä ja huonoja puolia. Käytössä olleiden ohjelmien käyttöönotto ja konfigurointi kuvataan tässä työssä yksityiskohtaisesti. Lopussa pohditaan, kuinka tämä projekti onnistui ja kuinka lokitusta lähdetään viemään eteenpäin tuotantoympäristöön.

Asiasanat
lokitiedostot, pilvipalvelut, mobiilisovellukset, tietotekniikka-arkkitehtuuri

Haaga-Helia
University of Applied Sciences

Abstract

7.5.2016

Author(s)
Santeri Friman.

Degree programme
Degree programme in Business Information Technology

Report/thesis title
Logging for Scalable Mobile application

Number of pages and appendix pages
43

This thesis presents an implementation of a centralized logging system for a scalable mobile application. The client for this project was Kvantia LTD and the name of the mobile application is Quanter. The software stack used to implement this system was called ELK-stack and it consists of three programs: Elasticsearch, Logstash and Kibana. Cloud service provider to host the application was Amazon Web Services and SaltStack was used to manage configuration of the software used in this thesis.

The goal of this thesis is to make investigating and fixing of unexpected problems in the application easier. Other goal in this thesis was to also create proper configurations for the used software because this application runs in the cloud and the system scales automatically.

The logging system was implemented for development environment. Production level system is not within the scope of this project. Some details will not be told because of the sensitivity of those details.

This thesis goes through following topics: what is logging, what can you achieve with it and what factors are important in creating a good logging environment. In addition to logging this thesis goes through the differences of a distributed and centralized systems in system architectures. Installation and configuration of software used in this thesis will be explaine. The end of this thesis will reflect on the succession of this project and how it is possible to continue the development to production stage.

Keywords
Log, Log files, Mobile application, Cloud services, IT-architectures

Sisällys

1 Johdanto

Kaikille järjestelmäylläpitäjille ja ohjelmistokehittäjille tulee jossain vaiheessa eteen tilanne, kun sovelluksessa tapahtuu jotain mitä ei pitäisi tapahtua. Kun virhettä aletaan paikallistamaan, on tärkeää saada selville muutamia perustietoja mikä helpottaa ja nopeuttaa ongelman ratkaisua huomattavasti.

Jotta tarvittavat tiedot olisi jälkikäteen mahdollista löytää, on ohjelman tallennettava ne tietokoneen kiintolevylle. Tätä tallennettua tietoa tapahtumien kulusta kutsutaan lokitekstiksi tai lyhyesti lokiksi.

Tämä opinnäytetyö toteutetaan työnantajani, Kvantia Oy:n Quanter nimiseen sovellukseen. Quanter on joukkueurheiluun suunnattu joukkueenhallintatyökalu. Sovelluksen avulla valmentajat ja pelaajat saavat automatisoidusti analysoitua dataa helpottamaan joukkueen rutiinien ja tehtävänhallinnan organisointia. Opinnäytetyössä käsitellään hyvään lokitukseen tarvittavia ominaisuuksia sekä toteutetaan lokitusjärjestelmä skaalautuvaan mobiilisovelluksen kehitysympäristöön.

Quanterin kriittisimmät osat sijaitsevat skaalautuvaksi suunnitellussa klusterissa, pilvessä. Sovelluksessa tai sen back end puolella tapahtuva virhe saattaa johtua hyvinkin monesta seikasta, jotka saattavat olla ympäristöön, koodiin tai vaikka käyttäjän fyysiseen sijaintiin liittyviä. Sovelluksen tallentamat lokitekstit täytyy saada yhdestä paikkaa etsittäviksi jotta ongelmia pystytään ratkaista vaivattomasti.

Ennen projektia lokitekstit sijaitsivat hajautettuina paikallisiin tiedostoihin ympäri klusteria. Opinnäytetyö kerää, analysoi ja indeksoi nämä lokitekstit eri lähteistä ja tallentaa ne yhteen hakukoneeseen josta ne ovat erillisen rajapinnan kautta helposti haettavissa.

Opinnäytetyön teoriaosuudessa pohditaan mistä hyvä lokitus koostuu analysoimalla lähteitä ja tutkimalla eri vaihtoehtoja. Teoriaosuudessa käydään läpi myös hajautetun ja keskitetyn arkkitehtuurin eroja.

Hyvän lokituksen suunnittelua varten on rajattava monta tekijää. Tämä opinnäytetyö on rajattu nopeuttamaan ja helpottamaan ohjelmointivirheiden tutkintaa. Muita tekijöitä, mitä kehitetään myöhemmin opinnäytetyön valmistuttua, on esimerkiksi suorituskyvyn mittaaminen, tuntemattoman koodin ymmärtäminen ja lisätietojen saaminen virheistä (Garry Shutler, 30.12.2012).

Tämä opinnäytetyöprojekti toteutettiin niin kutsun ELK-ohjelmistopinon avulla. ELK-ohjelmistopino koostuu kolmesta ohjelmasta: Elasticsearch, Logstash ja Kibana. Nämä kaikki kolme ohjelmaa käydään läpi opinnäytetyössä. Näiden lisäksi järjestelmien nopeaan käynnistämiseen ja konfigurointien hallintaan tarkoitettu ohjelma SaltStack käydään pääpiirteittäin lävitse. Opinnäytetyö rajautuu lokituksen toteutukseen. Joitain yksityiskohtia esimerkiksi koodista tai järjestelmistä jätetään kertomatta niiden arkaluontoisuuden takia.

Käsitteet

Automaatio = Itsetoimiva (Suomen Automaatioseura ry., 2010). Automaatiossa toiminnot tapahtuvat ilman ihmisen toimintaa tai hyvin vähäisellä ihmisen väliintulolla.

Sovellus = ”tiettyä tehtävää t. tiettyjä tehtäviä toteuttamaan tarkoitettu ohjelma, joka ei liity kiinteästi järjestelmän ylläpitoon t. Hallintaan.” (Kielitoimiston sanakirja)

Data = ”asian säännönmukainen esitys viestitettävässä t. käsittelykelpoisessa muodossa; tieto, tiedot. Binaarinumeroin esitetty data. Datan syöttö, siirtäminen tietokoneeseen.” (Kielitoimiston sanakirja)

Loki = Tietokoneohjelman tallentama teksti, joka kertoo mitä ohjelmassa on tapahtunut

Järjestelmäylläpitäjä = Henkilö, joka on erikoistunut tietokonejärjestelmien ylläpitämiseen.

Klusteri = Klusteri on monta tietokonetta, jotka ovat yhteydessä toisiinsa verkon yli. Käytännössä verkko tietokoneita.

Pilvi = Kun tietokone- ja tietotekniikkaresurssit toimivat verkon ylitse jonkin palveluntarjoajan ylläpitämänä, ne toimivat pilvessä.

Ohjelmistokehittäjä = Henkilö, joka on erikoistunut suunnittelemaan ja toteuttamaan ohjelmistoja. Ts. Ohjelmoija.

Ohjelmointivirhe = Bugi. Virhe tietokoneohjelmassa.
Filtteröinti = Tekstin läpikäymistä ja muokkaamista ennalta määritetyillä ehdoilla. Voidaan esimerkiksi poimia sanoja tai muuta tietoa erilleen tekstistä.

Rajapinta = standardin mukainen käytäntö t. yhtymäkohta, joka mahdollistaa tietojen siirron laitteiden, ohjelmien t. käyttäjän välillä. Käyttöliittymä ohjelman ja käyttäjän välisenä rajapintana. (Kielitoimiston Sanakirja)

Asiakaspää = Asiakas-palvelin sovellusarkkitehtuurissa käytettävä termi, jossa asiakaspäällä tarkoitetaan asiakkaan käyttämää päätelaitetta, jossa itse sovellus pyörii.

Palvelinpää = Asiakas-palvelin sovellusarkkitehtuurissa käytettävä termi, jossa palvelinpäällä tarkoitetaan palvelinta tai palvelimia, joissa ajetaan sovelluksen toimenpiteitä joita ei voi tai kannata asiakaspäässä ajaa.

Tietovarasto = Säilytyspaikka datalle

Tietokanta = Organisoitu säilytyspaikka, joka on yhdistelmä kokoelmia, pyyntöjä, näkymiä ja muita rakenteita.

Hajautettu Järjestelmä = Järjestelmä, jossa resurssit ovat jaettuna monen eri tietokoneen välillä

Keskitetty Järjestelmä = Järjestelmä, jossa resurssit ovat keskitettynä yhdelle koneelle.

Publish-Subsrcibe = Viestinvälitysmalli jossa lähettäjät eivät lähetä viestiä suoraan vastaanottajalle vaan viestit kulkevat välittäjän kautta. Vastaanottaja saa viestit välittäjältä vain niistä aiheista, joihin vastaanottaja on ilmoittanut kiinnostuksensa.

JSON = JavaScript Object Notation. Standardisoitu muoto, jossa data voi esittää ihmiselle helposti luettavassa muodossa.

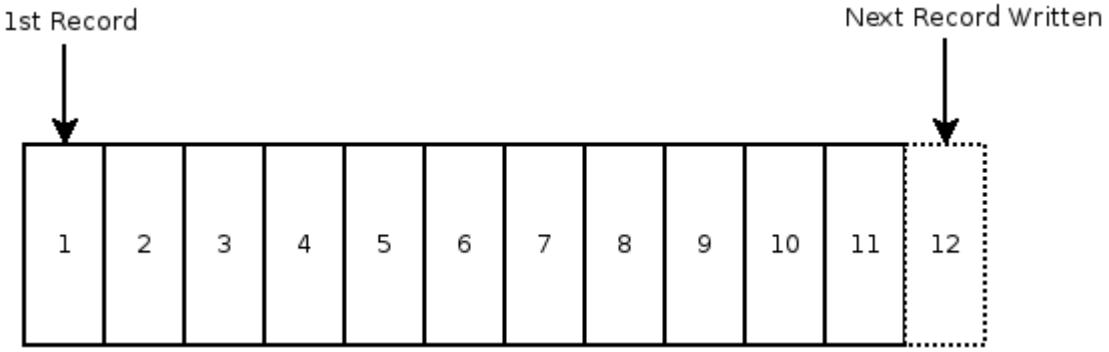
REST-rajapinta = Rajapinta, jonka kautta internetin yli toimiva sovellus lähettää ja vastaanottaa pyyntöjä ja kyselyitä.

2 Tietoperusta

Tässä luvussa käsitellään lokitusta yleisesti sovelluksissa. Luku keskittyy siihen, mitä on hyvä lokitus, kuinka siitä voi olla yritykselle hyötyä ja miten sellainen toteutetaan. Luku käy myös läpi arkkitehtuureja, yleisimpiä malleja lokituksen toteutukseen, muutamia esimerkkejä missä lokituksesta voi olla hyötyä ja yleisimpiä konsepteja.

2.1 Mitä on loki?

Pähkinänkuoressa loki on tallennettu luettelo kaikesta ohjelmassa tapahtuneista asioista. Lokiteksti on yksinkertaisimmillaan ja yleisimmin vain tavallista tekstiä. Kaikki ohjelmat tallentavat tätä lokia niin kutsuttuihin lokitiedostoihin. Loki on ehkä yksinkertaisin mahdollinen varastoinnin abstraktio (Jay Kreps, 16.12.2013). Yksinkertaisesti ohjelma kirjoittaa uuden rivin aina edellisen perään. Ohjelmat eivät poista lokeista mitään, joten kaikki tiedot pysyvät aikajärjestyksessä.



Kuva 1 Kuinka lokitekstit tallennetaan tiedostoon (Jay Kreps, 16.12.2013)

Loki kertoo mitä ohjelmassa on tapahtunut ja milloin. Aikajärjestys on tässä hyödyllistä, koska silloin voi olla täysin varma että aikaisempana oleva rivi on myös tapahtunut ajassa aikaisemmin. Skaalautuvissa hajautetuissa järjestelmissä, kuten tämän opinnäytetyön kohteena oleva applikaatio, tämä edellä mainittu lokin määritelmä on myös itse ongelman ydin (Jay Kreps, 16.12.2013).

Toinen erittäin hyvä ominaisuus aikajärjestyksessä on myös se, että virherivin löydyttyä tarvitsee vain katsoa tämän kyseisen rivin ympäriltä löytääkseen tapahtumat ohjelmassa, mitkä ovat mahdollisesti johtaneet tämän virheen syntyyn.

Jay Kreps kutsuu kirjoituksessaan tällaista tekstimuodossa olevaa, rakenteetonta ja ihmisten luettavaksi tarkoitettua lokia sovelluslokiksi (Jay Kreps, 16.12.2013). Hänen

mielestä konsepti, jossa ihmiset lukevat lokia yksittäisiltä koneilta on vanhentunut. Muihin vaihtoehtoihin kuitenkin palataan hieman myöhemmin.

2.2 Lokin elinkaari

Lokin elinkaari alkaa, kun ohjelma tulostaa lokin tiedostoon. Tiedostosta jokin ohjelma lukee uuden lokirivin ja lähettää sen eteenpäin lokien prosessointipalvelimelle. Lokin saavuttua tälle palvelimelle, se prosessoidaan ja tarvittaessa pilkotaan tietovarastolle sopivaan muotoon. Prosessoinnin jälkeen viesti joko lähetetään viestijonoon tai suoraan tietovarastoon. Tietovarasto säilöö lokit haluamallaan tavalla, mahdollisesti luo varmuuskopion siitä ja palauttaa hyväksytyn vastauksen viestijonolle tai ohjelmalle joka alun perin syötti lokin tietovarastolle.
Lokiviestit merkataan tunnisteella, jotta ne voidaan poistaa ajan kuluessa. Tärkeimmät lokit säilytetään pitemmän aikaa palvelimella, mutta tässä projektissa maksimissaan kolmen kuukauden ajan jolloin nekin poistetaan.

2.3 Arkkitehtuuri: keskitetty vai hajautettu järjestelmä?

Kuten ylempänä on mainittu, hyvä lokitus riippuu monesta tekijästä. Hyvällä lokituksella on mahdollista saavuttaa monenlaista lisäarvoa. Lisäarvoa tuovia parannuksia ovat muun muassa suorituskyvyn mittaus, ongelmien ratkonta ja hyödyllisen datan kerääminen.

Hyvän lokituksen suunnittelussa tärkeitä aspekteja ovat lokituksen arkkitehtuurin ja lokin elinkaaren suunnittelu sekä haluttu laajuus.

Vaikka yksinkertaisimmillaan loki on vain rivejä lokitiedostossa peräkkäin jossain ennalta määritetyssä sijainnissa tietokoneella, tätä tekstiä tulee joka ikinen sekunti huomattavia määriä. Vielä enemmän, kun sovelluksen koko ja komponenttien määrä kasvaa.

Lokituksen arkkitehtuuria suunnitellessa on otettava huomioon kohdejärjestelmän rakenne ja se, mitä lokituksen avulla halutaan saavuttaa. Järjestelmä voi olla keskitetty, hajautettu tai vaikkapa lokaalissa päätelaitteessa toimiva.

2.3.1 Keskitetty järjestelmä

Keskitetyssä järjestelmässä kaikki järjestelmän osat (web-palvelin, käyttäjänhallinta, tietokannat, lokitus) ovat keskittyneet yhteen palvelimeen. Keskitetty järjestelmä on yleensä nopea ja helppo pystyttää sekä ylläpitää (Saurabh Goyal, 2015). Lokituksessa tämä tarkoittaa sitä, että lokituksen arkkitehtuuri voi olla hyvinkin yksinkertainen.

Yksinkertaisimmillaan lokitus voi tapahtua vain Syslogiin, joka on yleisesti käytössä oleva standardi, jossa yksi osa (ohjelma) tulostaa viestit ja toinen (palvelin) säilöö ne ja kolmas prosessoi ne halutulla tavalla.

Tällaista keskitettyä järjestelmää on helppo ylläpitää, mutta ongelmaksi muodostuu sen skaalautuvuus sekä niin kutsuttu Single Point of Failure. Single Point of Failure tässä tapauksessa viittaa siihen, kuinka tämän yhden palvelimen sammuessa koko järjestelmä, lokitus mukaan lukien, lakkaa toimimasta. Minkä tahansa palvelun tai sovelluksen maineen ylläpitäminen edellyttää sitä, että sovelluksen ja sen komponenttien käynnissä oloaika maksimoidaan. Myös jos käyttäjämäärät kasvavat nopeasti, voi palvelin hyvinkin nopeasti täyttyä suuresta määrästä lokia jolloin pahimmassa tapauksessa voi olla edessä palvelimen tallennuskapasiteetin täyttyminen. Suuri käyttäjämääränkasvu voi myös käyttää kaikki palvelimen resurssit, jolloin osat lakkaavat toimimasta.

2.3.2 Hajautettu järjestelmä

Vastakohta keskitetylle järjestelmälle on hajautettu järjestelmä. Hajautetussa järjestelmässä järjestelmän eri osat on hajautettu moneen eri tietokoneeseen ja ne kommunikoivat toistensa kanssa internetin avulla (Wolfgang Emmerich, 1997).

Koska hajautetut järjestelmät skaalautuvat horisontaalisesti, lisäämällä uusia koneita verkkoon, niiden teoreettinen potentiaalinen skaalautuvuus on ääretön (Royans Tharakan, 2007).

Hyvässä ja toimivassa lokituksessa on otettava huomioon hajautetun järjestelmän luomat ongelmat. Kuten aikaisemmin on mainittu, lokituksessa on tärkeää, että lokit tallentuvat lopulliseen tietovarastoon siinä järjestyksessä, missä ne on alun perin kirjoitettu. Lokitiedostoissa myöhemmän tapahtuman on oltava lokitettuna aikaisemman tapahtuman jälkeen. Tämä ei kuitenkaan monista asioista johtuen ole aina mahdollista hajautetuissa järjestelmissä. Esimerkiksi jos kaksi eri palvelinta sijaitsee eri maassa, voi samaan aikaa syntynyt lokirivi saavuttaa määränpään eri aikoihin.

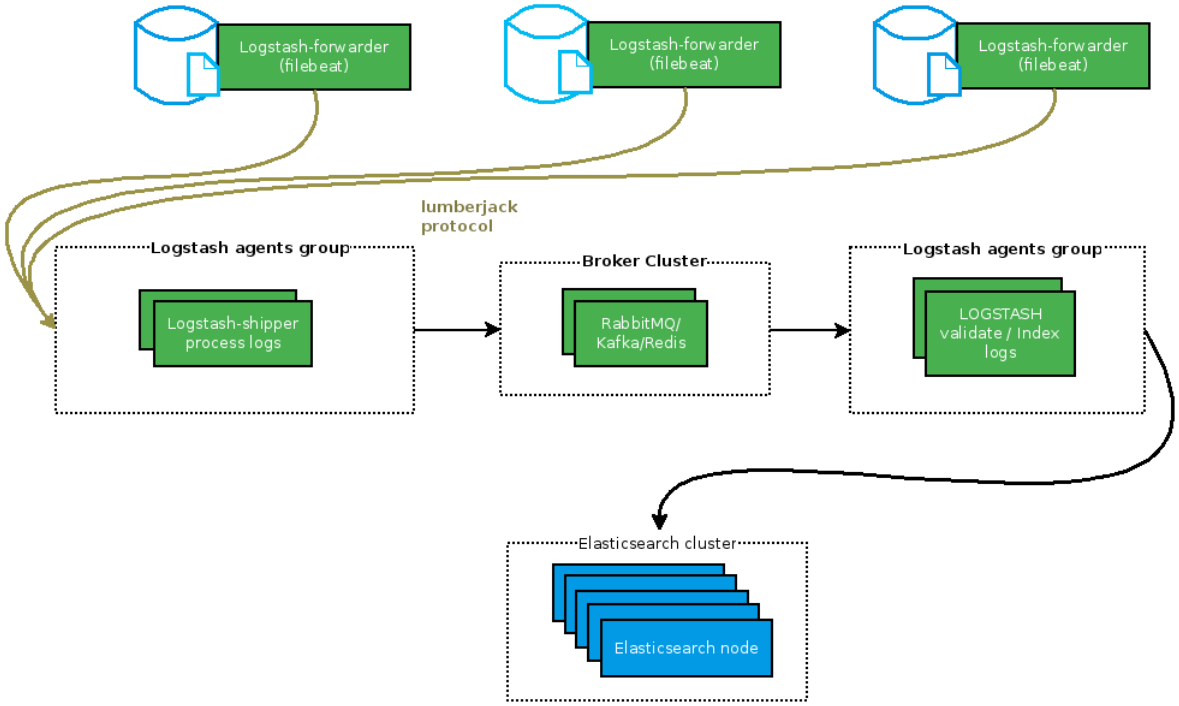
Kun järjestelmässä on monia tietokoneita, jotka ovat yhdistetty toisiinsa verkon avulla ja kaikki tulostavat lokia, on tärkeää tietää milloin ja missä jokainen lokikirjaus on tapahtunut. Lokeissa olevat aikaleimat ovat korvaamattomassa asemassa hajautetuissa järjestelmissä (Jay Kreps, 16.12.2013).

Lokeja selatessa, minkä tahansa rajapinnan lävitse, on äärimmäisen tärkeää, että lokit ovat aikajärjestyksessä. Muuten on erittäin vaikea hahmottaa ohjelmassa tapahtuneita asioita ja niiden syy-seuraussuhteita.

Jay Kreps (16.12.2013) mainitsee artikkelissaan, että jos on olemassa loki muutoksista, voi näitä muutoksia käyttää taulun luomiseen, joka pitää sisällään nykyisen tilan. Optimaalisessa tilanteessa tämä mahdollistaisi teoriassa järjestelmän uudelleenluonnin johonkin aikaisempaan tilaan.

Olennaisin ominaisuus lokissa on aikaleima. Aikaleiman avulla verkon yli kerätyt lokit on mahdollista pitää järjestyksessä myös lopullisessa säilytyspaikassa.

Alla olevassa kuvassa (Kuva 2) on esiteltynä esimerkkiarkkitehtuuri lokitukselle. Tämä arkkitehtuuri on suunniteltu hajautetulle järjestelmälle. Kuviossa on hyvin esitelty tarvittavat osat hyvän lokituksen toteuttamiseksi.



Kuva 2 Esimerkkiarkkitehtuuri lokitukselle (Azarmi, 2016. 5.)

Ensimmäisenä lokit pitää kerätä kaikista koneista joita klusterissa on. Lokien keräämiseen tarkoitettu ohjelma on hyvä olla kevyt ja yksinkertainen, koska on parasta säästää koneen resursseja itse pääsovellukselle. Tässä esimerkissä käytetty Logstash-forwarder on juuri sellainen ohjelma. Käytännössä se lukee ja kuuntelee jokaisen sille määritellyn lokitiedoston loppupäätä ja lähettää ne verkkoa pitkin eteenpäin. Logstash-forwarderin

kehittäminen on nykyisin lopetettu ja sen on korvannut ohjelma nimeltä Filebeat. Toimintaperiaate on kuitenkin samanlainen.

Toisessa päässä vastaanottava kone ottaa kaikista lokia tulostavista koneista vastaan niiden lokit. Protokollia lokien siirtoon on monia ja useimmiten ne käyttävät TCP-tietoliikenneprotokollaa. Yllä olevassa esimerkissä on käytetty Lumberjack-protokollaa lokien siirtoon. Lumberjack-protokolla eroaa tavallisesta TCP-protokollasta salaamalla viestit käyttäen TLS-salausta. Siirrettävät lokit on hyvä salata, koska niissä saatetaan siirtää hyvinkin arkaluontoista tietoa.

Kun lokit on lähetetty ja vastaanotettu halutussa paikassa, täytyy niitä usein hieman prosessoida. Eri ohjelmat tulostavat lokia erilaisessa formaateissa, joten loppusäilytyspaikkaa varten, joka on tässä esimerkissä Elasticsearch, on hyvä muotoilla lokiteksti yhteneväiseen muotoon.

Skaalautuvissa applikaatioissa, varsinkin käyttäjämäärien kasvaessa, lokia voi tulla niin paljon, että vaarana on suorituskyvyn lasku. Käytetyillä ohjelmilla voi olla esimerkiksi rajallinen maksimi tiedonsiirto/vastaanotto. Myös TCP-protokollalla on olemassa raja läpisyötön määrään, tämä kuitenkin määrittyy esimerkiksi käytössä olevan kaistan ja muiden tekijöiden avulla, jotka eivät liity tämän projektin aihepiiriin. Nämä seikat on syytä ottaa huomioon lokituksen arkkitehtuurissa.

Jotta vältyttäisiin turhilta kopioilta ja pakettien katoamisilta, täytyy lokien prosessointityökalun (Logstash) ja tietovaraston (Elasticsearch) välissä oltava jokin puskuri. Yksi hyvin suosittu puskurinakin toimiva ohjelma on Apachen Kafka.

Kuten huomataan, hajautetussa järjestelmässä on monia seikkoja joita täytyy ottaa huomioon arkkitehtuuria suunnitellessa. Hajautetun järjestelmän suoma etu on sen skaalautuvuudessa. Tällainen järjestelmä, joka on yllä käyty läpi, kestää isonkin kuormituksen. Yhden klusterin koneen sammuessakin tämä järjestelmää jatkaa toimintaansa.

Kolikolla on tietysti kääntöpuolensa, ja tässä tapauksessa se on järjestelmän pystyttämisen ja ylläpidon vaativuus. Monet tehtävät toimenpiteet vaativat tarkkaa suunnittelua ja jokaisen järjestelmän osan hahmottamista. Nämä kaikki vievät aikaa ja vaivaa.

Se, että valitseeko hajautetun järjestelmän vai keskitetyn järjestelmän riippuu tietysti aina käyttötapauksesta eikä sitä voi yksiselitteisesti ratkaista, onko toinen lähestymistapa parempi.

2.4 ELK-ohjelmistopino

ELK-ohjelmistopino on kokoelma Elastic Oy:n kehittämiä ohjelmia. Nämä ohjelmat ovat Elasticsearch, Logstash ja Kibana. Kaikki kolme ohjelmaa ovat hyvin suosittuja ohjelmia ja käytössä laajalti ympäri maailmaa. Yhdistämällä nämä kolme ohjelmaa yhteen, hoitaa tämä ohjelmistopinon lokien keräämisen päästä päähän eli aina pilvestä keskitettyyn palvelimeen ja sieltä internet-selaimeen. Tätä ohjelmistopinoa voidaan käyttää äärettömään määrään käyttötarkoituksia, esimerkiksi lokitukseen, pikaviestintään tai vaikka yrityksen talouden analysointiin.

2.4.1 Elasticsearch

Elasticsearch on kaiken kattava ohjelma hakuorientoituneille sovelluksille. Elasticsearch on rakennettu Apachen Lucene-hakumoottorin päälle. Lucene on kokonaan Java-ohjelmointikielellä kirjoitettu tekstihakukone. Todennäköisemmin Lucene on valittu Elasticsearchin perustaksi koska se on kypsä, skaalautuva, kevyt, mutta hyvin tehokas ja todella suorituskykyinen (Kuć & Rogoziński. 2013. 8).

Elasticsearch julkaistiin helmikuussa 2010 Shay Baronin aloittamana projektina. Julkaisun jälkeen Elasticsearchin suosio on kasvanut räjähdysmäistä vauhtia vain muutamassa vuodessa. Nykyisin Elasticsearch on muun muassa github.com:n eniten tähtimerkattu Java-ohjelma 16048 tähdellään.

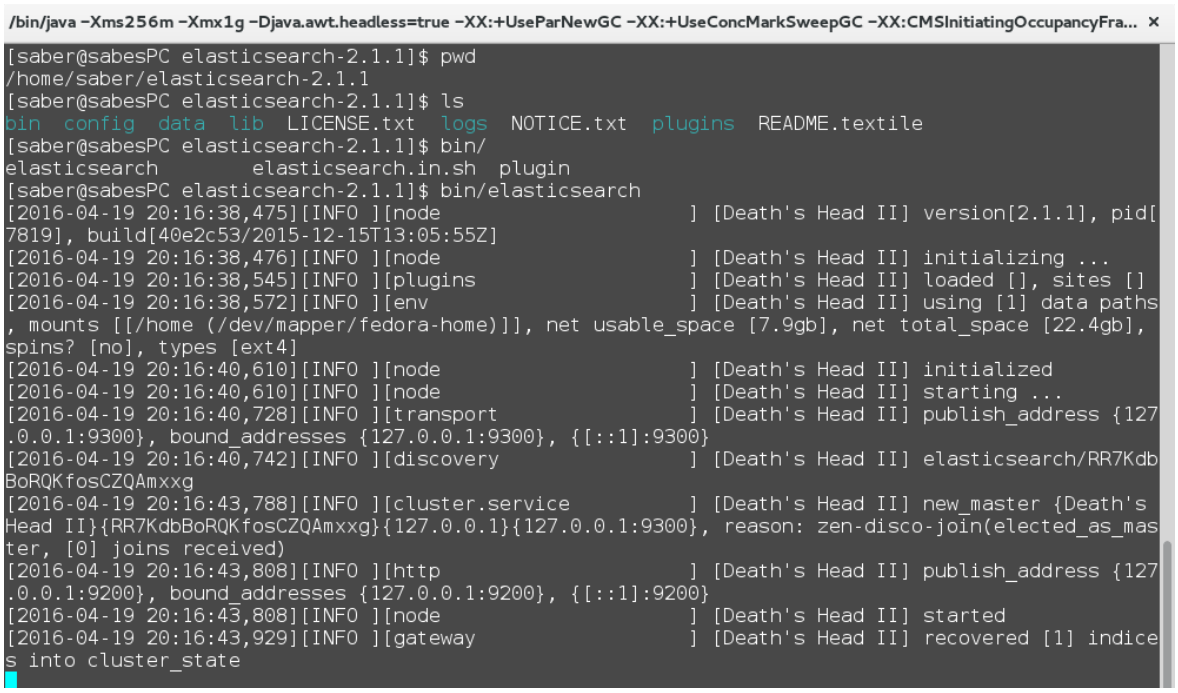
Githubin tähtimerkinnät kuvastavat yleisesti ottaen sitä, kuinka paljon kiinnostusta projektilla on (Baishakhi, Daryl, Filkov & Devanbu. 2014. 2.). Tämä merkintä ei kuitenkaan yksinomaan riitä kertomaan ohjelman hyödyllisyyttä, vaan monesti esimerkiksi suosittujen yritysten ohjelmat keräävät julkaisun jälkeen nopeasti paljon tähtiä. Elasticsearchin suosio saattaa perustua hyvinkin moneen asiaan. Muutamia näistä on sen pieni aloituskynnys, automaattinen skaalautuvuus ja korkea suorituskyky.

Elastisearch-klusteri muodostuu yhdestä tai useammasta koneesta, jotka ovat yhteydessä toisiinsa verkon yli. Klusteriin luodaan indeksejä, jotka koostuvat dokumenteista, joilla on

jotakuinkin samankaltaisia ominaisuuksia. Lokituksessa voidaan esimerkiksi luoda indeksit päivittäin, jolloin tieto tulee automaattisesti jaoteltua päivien mukaan.

Dokumentit ovat Elasticsearchissa JSON-muodossa olevia tiedostoyksiköitä. Lokien kanssa käytännössä yksi lokirivi on yksi dokumentti. Kun lokirivi indeksoidaan Elasticsearchiin, sille annetaan tyyppi ja sen eri tiedot jaotellaan omiksi nimi-arvo-pareiksi dokumenttiin.

Sen käytön aloittaminen ei voisi olla yhtään sen helpompaa. Elasticsearch ladataan koneelle heidän sivuiltaan (https://www.elastic.co/downloads/elasticsearch). Oletusasetuksilla sen voi suoraan komentoriviltä ilman mitään erikoisasetuksia:



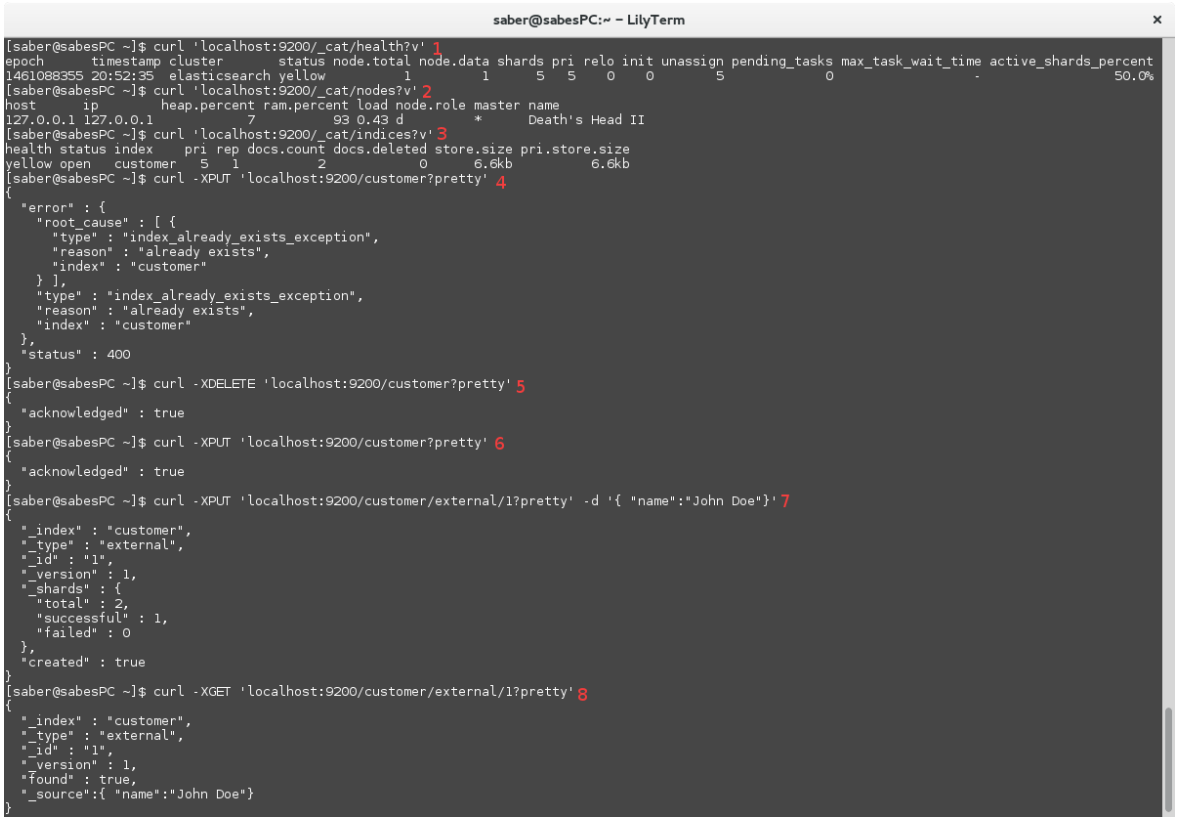
Kuva 3 Elasticsearchin käynnistys komentoriviltä

Yllä olevassa kuvassa ensimmäisenä tulostaan nykyinen hakemisto, joka on käyttäjän *saber* kotihakemiston alla, tässä tapauksessa *elasticsearch-2.1.11*. Tämän jälkeen tulostetaan hakemiston sisältö komennolla *ls*, ja ajetaan bin hakemistosta Elasticsearchin käynnistyskomento komennolla *bin/elasticsearch*. Tällä yhdellä komennolla Elasticsearch on käynnistetty ensimmäisen kerran ja sen käytön voi aloittaa.

Elasticsearchin käyttöliittymä on myös suunniteltu helposti lähestyttäväksi. Elasticsearchia ohjataan HTTP:llä heidän REST-rajapinnan kautta. Tämä mahdollistaa monipuolisen määrän mahdollisuuksia klusterin hallintaan. Koska Elasticsearch ja sen

ohjelmointirajapinta ovat avointa lähdekoodia, mahdollistaa tämä myös kolmansien osapuolien kehittämät rajapinnat.

Alla olevassa kuvassa esitellään muutama perustoiminto, miten Elasticsearchia voidaan hallita komentoriviltä käyttäen *curl*-ohjelmaa. Curl on vakio-ohjelma unix-pohjaisissa käyttöjärjestelmissä jolla voidaan tehdä http-kutsuja tai -pyyntöjä palvelimelle.



Kuva 4 Elasticsearchin perustoimintoja

Elasticsearch käyttää oletuksena porttia 9200, josta sen REST-ohjelmointirajapintaan vastaa pyyntöihin. Yllä olevassa kuvassa on esitelty numeroituna muutamia pyyntöjä, mitä Elasticsearchille voi antaa. Alla on Komennot selitettyinä:

1. Tulostaa Elasticsearch klusterin ”terveydentilan” eli erinäisiä tietoja, jotka kertovat klusterin tilasta. Vastauksesta selviää muun muassa aikaleima, klusterin nimi, klusterin tila (joka on vihreä (green), keltainen (yellow) tai punainen (red)), noodien eli koneiden määrä, shardien eli sirpaleiden määrä, määrittämättömien sirpaleiden määrä ja aktiivisten sirpaleiden määrä
2. Tulostaa koneiden tilan. Vastauksesta selviää muun muassa nimi, resurssien käyttö jne
3. Indeksien tila. Muun muassa koko, lukumäärä ja niin edelleen

4. Tällä komennolla käytetään HTTP-protokollan PUT-metodia, jolla luodaan uusi indeksi nimeltä *customer*, mutta kuten edellisestä komennosta ja tämän komennon vastauksesta voidaan päätellä, indeksin luonti epäonnistuu koska se on jo olemassa
5. Poistetaan olemassa oleva indeksi käyttäen HTTP-protokollan DELETE-metodia
6. Sama, kuin komento 4
7. Syötetään, indeksin *customer* kohtaan 1*,* uusi asiakas nimeltä *John Doe* käyttäen JSON-formaattia
8. Haetaan, indeksin *customer,* ensimmäinen dokumentti.

Yllä on käyty läpi vain jäävuoren huippu Elasticsearchin ominaisuuksista. Se on kuitenkin hyvin monipuolinen ja sopeutuvainen ohjelma erilaisiin käyttötarkoituksiin. Sen helpoksi suunniteltu REST-ohjelmointirajapinta mahdollistaa monien ohjelmien helpon integroinnin Elasticsearchiin. Yksi näistä ohjelmista on Logstash.

2.4.2 Logstash

Logstash tarjoaa integroidun rajapinnan lokien hallintaan. Ominaisuuksiin sisältyy lokien keräys, keskittäminen, parsinta, säilytys ja haku (Turnbull. 2014. 18). Logstash on ilmainen ja avointa lähdekoodia ja sen on alun perin kehittänyt amerikkalainen ohjelmistokehittäjä Jordan Sissel.

Logstashin vahvuuksia on sen valmiudet sopeutua lähes mihin tahansa ympäristöön. Logstash osaa lukea lokeja muun muassa TCP/UDP-väylää pitkin, suoraan tiedostoista niin UNIX- kuin Windows ympäristöissä tai suoraan ohjelmien ulostulosta. Logstashin tarjoamat lukumahdollisuudet jättävät hyvin vähän tilanteita, missä lokin tallentaminen ei onnistuisi.

Kun lokit keskitetään palvelimelle, joissa Logstash on käynnissä, Logstashista löytyy myös kattava kokoelma filttereitä. Filttereiden avulla lokiviestit voidaan prosessoida haluamaansa muotoon ja niistä voidaan eristää tietoja omiin kenttiinsä analysointia varten.

Turnbull kertoo kirjassaan, The LogStash Book, Logstashin arkkitehtuurin olevan viestipohjainen ja erittäin yksinkertainen. Yleensä palvelimella pyörii yksi niin kutsuttu Logstash-agentti (Logstash agent), jonka tehtävänä on suorittaa eri toimenpiteitä yhdessä muiden avoimen lähdekoodin komponenttien kanssa. Yksi tällainen komponentti olisi esimerkiksi Elasticsearch.

2.4.3 Kibana

Logstash kerää, analysoi ja prosessoi lokit ja lopulta lähettää ne säilöön Elasticsearchiin. Elasticsearch-luvussa esitelty tekstikäyttöliittymä ei kuitenkaan ole paras mahdollinen käyttöliittymä suuren ja monimuotoisen datan visualisointiin. Elasticsearchin REST-rajapinta antaa monipuoliset mahdollisuudet web-käyttöliittymien tekemiselle ja yksi suosituimpia web-käyttöliittymistä Elasticsearchille on Kibana.

Kibana mahdollistaa kustomoitujen hallintapaneelien tekemisen. Hallintapaneeleihin voi liittää tarkoitukseen sopivia kaavioita, joiden parametrit tulevat suoraan Elasticsearchista. Hallintapaneelien lisäksi on mahdollista selata Elasticsearchissa olevia lokeja ja visualisoida niitä mielen mukaan.

Kuten muutkin ELK-ohjelmistopinon ohjelmat, myös Kibana on avointa lähdekoodia. Kibanaan on rakennettu monia laajennuksia, joita on helppo asentaa ja ottaa käyttöön. Laajennuksien avulla voidaan esimerkiksi monitoroida systeemien tilaa tai vaikka lähettää hälytyksiä

2.5 SaltStack

SaltStack (lyhyesti Salt) on Thomas Hatchin vuonna 2011 aloittama avoimen lähdekoodin projekti. Alun perin Saltin ideana oli toimia nopeana tapana käynnistää järjestelmiä etänä (Myers. 2015. 1). Tämän projektin edetessä ja kasvaessa Saltiin on kuitenkin lisätty monia kerroksia ja nykyään siitä muodostuu hyvin monipuolinen ohjelma palvelimien konfigurointien hallintaan.

Saltin pääkomponentit ovat Salt Master ja Salt Minion. Salt Master on tavallaan Saltin ohjausjärjestelmä. Salt Masterin alapuolelle käynnistetään Salt Minioneita. Kaikki Salt minionit ottavat käskyjä vastaan masterilta. Masterista käsin hallitaan kaikki Salt minioneja ja minionit pitävät Salt Masterin käskyjä totena (Myers. 2015. 8).

Selkeämmin sanottuna Salt Master on päätietokone ja Salt Minionit ovat myös tietokoneita, mutta ottavat käskyt vastaan masterilta. Minionit voivat toki myös toimia ilman Salt Masterin olemassa oloa. Salt Masteriin tehdään niin kutsuttuja reseptejä joita ajamalla on mahdollista suorittaa erilaisia toimenpiteitä minioneille. Masterista käsin voi muun muassa käynnistää uusia minioneita tai asentaa niille päivityksiä tai uusia ohjelmia.

Saltin konfigurointihallintasysteemiä kutsutaan nimellä States. SLS-tiedostot (Salt States File) määrittävät aina jonkin tilan missä minionin kuuluu olla kun tämä tiedosto on ajettu.

Näitä tiloja yhdistellään toisiinsa luomalla niin kutsuttuja reseptejä. Reseptit ovat periaatteessa yhdistelmiä eri Stateista. Nämä tilat määrittyvät asetusten mukaan, jotka välitetään minioneille pilareitten (Pillars) mukana. Näihin resepteihin kirjoitetaan operaatiot mitä tehdään kun resepti ajetaan. Minioneille määritetään rooleja ja kun reseptejä ajetaan masterilta, ajetaan reseptit asetusten mukaisesti minionilla olevien roolien mukaan. Esimerkiksi jos yhdellä minionilla on rooli elasticsearch, mutta ei ole roolia logstash, tällöin voidaan ajaa esimerkiksi asennusrespeti ja tämä asentaisi kyseiselle minionille elasticsearchin, mutta jättäisi logstashin asentamatta. Tämä asennusresepti voi siis sisältää molempien (Elasticsearch ja Logstash) Statet, mutta asentaa vain sen kumman rooli minionilta löytyy.

Pilarit (Pillars) ovat puurakenteisia tietorakenteita, jotka ovat määriteltynä Salt Masterilla. Pilareiden avulla voidaan lähettää luottamuksellista tietoa salattuna minioneille. Esimerkiksi tässä projektissa pilareiden avulla lähetetään minionille tiedot, minkä mukaan luodaan asennettavien ohjelmien konfiguraatiot.

Salt on hyvin monipuolinen ohjelma joka taipuu mitä erilaisempiin käyttötarkoituksiin. Saltin perusteellinen läpikäynti ei kuitenkaan kuulu tämän opinnäytetyön aiheeseen.

3 Sovelluksen lokituksen suunnittelu

Tässä luvussa käydään läpi mobiiliapplikaation arkkitehtuuria ja sen lokituksen suunnittelua. Luku keskittyy tämän halutun lopputuloksen määrittelyyn, siihen luotuihin tavoitteisiin, aiheen rajaamiseen sekä teknologiavalintojen perustelemiseen.

Määrittelyssä kerrotaan, mitä lisäarvoa lokituksen toteuttaminen halutaan tuovan sovelluksen. Tämän lisäksi siinä määritellään lokituksen tavoiteltu kattavuus, laajuus sekä rajataan osa-alueet mihin osa-alueisiin lokitusta sovelletaan.

Suunnitteluosiossa käydään läpi keskeisimmät työkalut ja teknologiavalinnat sekä tehdään konkreettinen suunnitelma lokituksen toteuttamiseksi. Tässä kappaleessa käydään läpi järjestelmän vaativat osa-alueet aina arkkitehtuurista yksittäisten lokirivien parsimiseen ja analysointiin.

3.1 Määrittely ja rajaus

Projektin tavoitteena on saada toimiva lokitusjärjestelmä skaalautuvaan mobiiliapplikaatioon. Koska sovellus sijaitsee pääosin pilvessä, hajautetussa järjestelmässä, on lokien tutkiminen ihmisvoimin vaivalloista ja hidasta. Toimivalla lokitusjärjestelmällä tarkoitetaan sellaista, että jokaisesta järjestelmän koneesta ja palvelusta kerätään lokit ja lähetetään ne yhteen koneeseen. Vastaanottava kone analysoi nämä lokitekstit, parsii niistä halutut kentät ja tallentaa ne tietovarastoon.

Opinnäytetyön tavoitteena on myös toteuttaa pilvipalvelun konfigurointi, jolla skaalautuvuus voidaan automaattisesti hoitaa uusia klustereita pystyttäessä tai vanhaa klusteria päivittäessä.

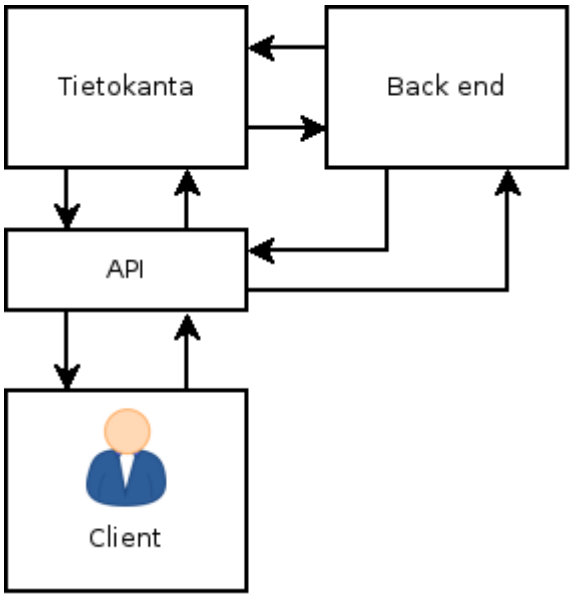
Oletetaan, että lokituksen avulla on mahdollista kehittää seuraavia osa-alueita: virheiden tutkinta, suorituskyvyn profilointi, koodin ymmärtäminen ja lisätiedon liittäminen virheilmoituksiin (Garry Shutler, 30.12.2012). Tämä opinnäytetyö rajoittuu virheiden tutkinnan sekä virheilmoitusten informaation kehittämiseen. Suorituskyvyn profilointi ja muut kehityskohteet ovat jatkokehityksen tuotteita.

Tietovaraston vaatimus muodolle, missä lokitekstit säilytetään, on otettava huomioon kun lokitekstejä prosessoidaan. Esimerkiksi Elasticsearch säilöö tietoja JSON-muodossa, joten prosessointivaiheessa on syytä ottaa huomioon mahdollisuus säilöä tietovarastoon avain-arvopareja.

3.2 Suunnittelu

Kuten yllä on mainittu jo useaan kertaan, järjestelmän arkkitehtuuri on olennainen kun lähdetään suunnittelemaan lokitusta ja valitsemaan oikeita ohjelmia. Tässä luvussa käydään ensiksi läpi sovelluksen yleinen arkkitehtuuri, jonka jälkeen tarkastellaan hieman mahdollisia teknologiavalintoja kyseiseen arkkitehtuuriin sopivaan lokitukseen.

3.2.1 Arkkitehtuuri



Kuva 5 Sovelluksen arkkitehtuuri

Yllä olevassa kuvassa (Kuva 5) on kuvattuna sovelluksen arkkitehtuuri. Kuva on hyvinkin yksinkertaistettu ja yleinen katsaus arkkitehtuuriin. Todellisessa maailmassa, pilvessä, nämä osat ovat suunniteltu skaalautuviksi ympäri klusteria. API (Application Programming Interface, suomeksi ohjelmointirajapinta), tietokanta ja back end ovat sovelluksen palvelinpuolella, kun taas kuvassa (Kuva 5 Sovelluksen arkkitehtuuriKuva 5) näkyvä client-osa kuuluu taas asiakaspuolelle.

Lokitusta varten olennaista tässä arkkitehtuurissa on back end sekä API, nämä osat hoitavat suurimman osan ohjelman toiminnasta ja tulostavat halutut lokitekstit. Tietokannan tai asiakaspään (Client) lokeja ei varsinaisesti tässä vaiheessa kerätä.

Kun käyttäjä tekee jotain, joka vaatii hakuja tietokannasta tai muuta operointia back end puolella, viestinvälittäjänä toimii ohjelmointirajapinta. Ohjelmointirajapinta vastaanottaa

viestin, välittää sen palvelinpuolelle ja jää odottamaan vastausta. Vastauksen saapuessa ohjelmointirajapinta palauttaa vastauksen sisällön asiakaspuolelle.

Tietokanta toimii informaation säilytyspaikkana. Tietokantaan tallennetaan käyttäjä- ja tapahtumatietojen lisäksi esimerkiksi tehtäviä, joita back endin on suoritettava kun aika koittaa.

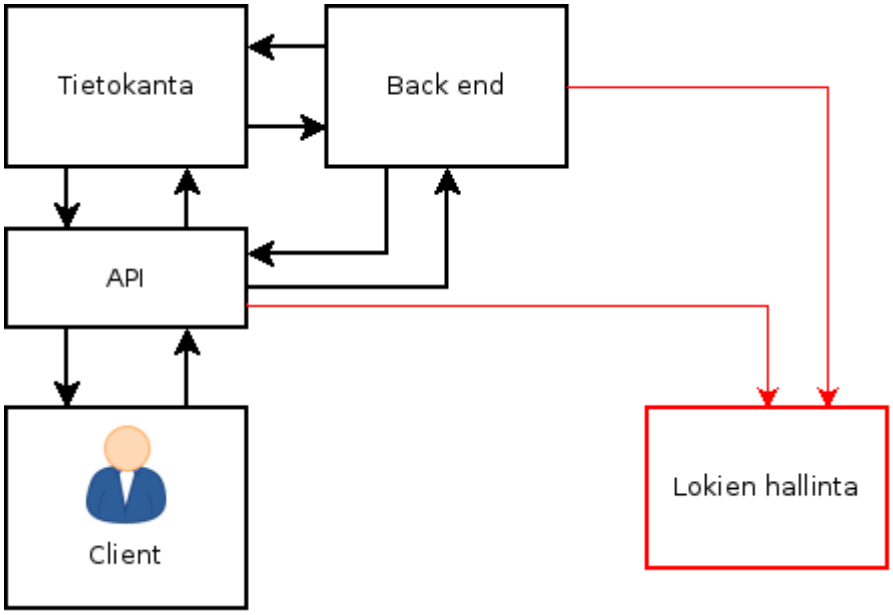
Ohjelmointirajapinta keskustelee tietokannan lisäksi suoraan back endiin. Esimerkiksi, jos käyttäjän toimenpiteet vaativat suoraan jonkin tehtävän laukaisemista tai laskutoimituksen tekemistä, voidaan viesti välittää suoraan ohjelmointirajapinnan kautta back endiin. Tällöin ohjelmointirajapinta jää odottamaan hyväksyttyä vastausta, jonka saavuttua sovellus voi taas jatkaa toimintaansa.

Back end hoitaa ohjelman eniten resursseja vaativat operaatiot, esimerkiksi, kun täytyy tehdä raskaita laskutoimituksia, hallita tai hakea suuria määriä dataa. Back end pitää huolen, että kaikki viestit saapuvat käyttäjille eikä mitään tärkeää dataa häviä kovassakaan kuormituksessa.

Kuten määrittelyssä mainittiin, lokituksen tavoitteena on helpottaa virheiden löytämistä ja niiden korjaamista. Virheiden löytämiseen eniten vaikuttavat lokit syntyvät ohjelmointirajapinnassa sekä back end-puolella. Näistä kahdesta osasta on siis saatava kerättyä lokitiedot. Tiedot täytyy myös jollakin tavalla muovata jossain määrin samanlaiseen muotoon, jotta niitä voidaan helpommin analysoida.

Koska kaikki palvelut pyörivät skaalautuvassa klusterissa, ohjelmointirajapinnassa olevien palveluiden määrä voi kasvaa hyvinkin suureksi. Myös back end puoli skaalautuu, joten seurattavia tekijöitä syntyy sielläkin hyvin monta. Tämän skaalautuvuuden myötä lokia syntyy siis hyvin moneen sijaintiin ja lopullisena tehtävänä on saada nämä kaikki lokit keskitettyä yhteen paikkaan.

Lokien kerääminen, analysointi ja hakeminen on myös tietokoneen resursseja vievää homma. Erityisesti muistia tarvitaan tiedostojen nopeaan käsittelyyn. Toimiva lokitusjärjestelmä ei myöskään missään nimessä saa vaikuttaa itse sovelluksen toimintaan. Hajautetun järjestelmän yksi tärkeimmistä tehtävistä on olla käytettävissä kokoajan, joten myöskään lokitus ei saa vaikuttaa järjestelmän 100 prosenttiseen saatavuuteen. Tästä syystä lokitekstien analysointi on hyvä eristää omaksi komponentikseen.

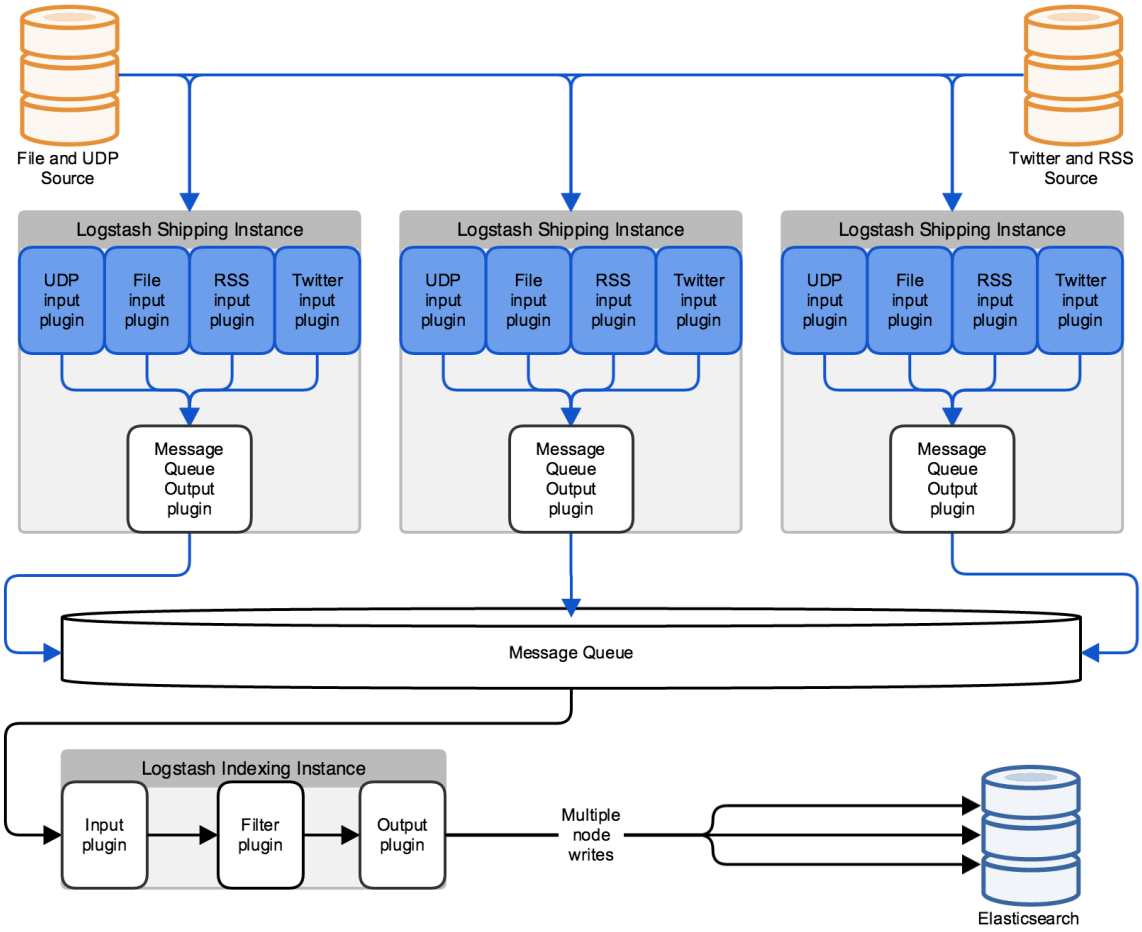


Kuva 6 Lokien hallinta lisättynä arkkitehtuuriin

Yllä olevassa kuvassa (Kuva 6) näkyvä lokien hallinta -komponentti koostuu kolmesta osasta: Vastaanottaminen ja analysointi, säilyttäminen ja indeksointi sekä niiden esittäminen.

Tässä klusterissa lokien hallinta toimii käytännössä omassa koneessaan. Lokiviestit täytyy kuitenkin saada keskitettyä omaan koneeseen jossa varsinainen prosessointi ja säilytys toteutetaan. Tämän toteuttamiseen on monia vaihtoehtoja, yksi hyvin suosittu työkalu on Logstash. Logstash tarjoaa ”putken” jonka avulla lokit saadaan kerättyä ja keskitettyä omaan koneeseen.

Logstashin (Elastic, 2016) dokumentaatiossa käydään läpi esimerkkiarkkitehtuuri, miltä Logstashin käyttöönottoputkisto (pipeline) voisi näyttää:



Kuva 7 Esimerkkiarkkitehtuuri Logstashille ja Elasticsearchille (Logstash dokumentaatio)

Ylläolevassa kuvassa (Kuva 7) on Logstashilla käytännössä neljä eri tasoa. Ensimmäisessä tasossa (input-tier) syötetään Logstashille halutut lokit halutuista lähteistä. Kuten kuvasta voi päätellä, lähteet voivat olla mitä tahansa aina tiedostoista Twitter-viesteihin. Tässä tasossa ei vielä tehdä minkäänlaista prosessointia viesteille vaan lähetetään ne suoraan eteenpäin seuraavalle tasolle.

Seuraava taso toimii käytännössä viestijonona (message queue), tarkemmin sanottuna puskurina. Puskurin ideana on ehkäistä koko lokitusjärjestelmän virhealttiutta siinä tapauksessa jos Logstashissa tapahtuu jokin virhe ja se lakkaa toimimasta tai jos uusien viestien määrä kasvaa niin isoksi, ettei Logstash tai Elasticsearch pysty käsittelemään niitä.

Kolmannen tason (filtering-tier) tehtävä on ajaa viestit suodattimen läpi. Tässä tasossa lokiviestit vastaanotetaan viestijonosta ja käsitellään ne. Viesteistä esimerkiksi erotellaan halutut tiedot omiin kenttiin. Esimerkkejä tiedoista on vaikka aikaleima, alkuperäinen sijainti tai lokiviestin taso.

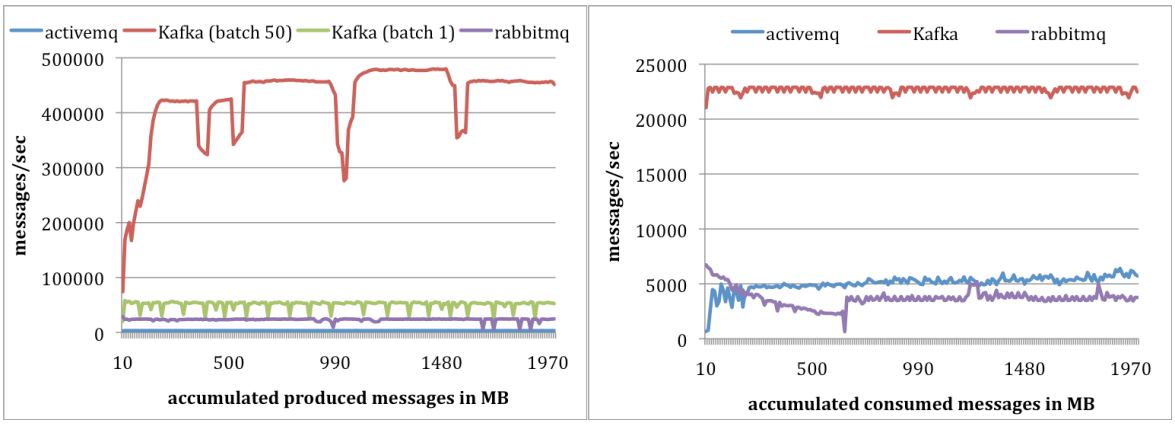
Viimeiseissä tasossa (indexing tier) lokiviestit lähetetään Elasticsearchille indeksoitavaksi ja säilytettäväksi. Elasticsearch on skaalautuvaksi suunniteltu hakumoottori, joka toimii kuvassa olevassa järjestelmässä myös tietovarastona lokeille. Elasticsearchista kerrotaan lisää myöhemmissä luvuissa.

Logstashia pystyy skaalaamaan niin horisontaalisesti kuin vertikaalisestikin. Jokaiseen näistä tasoista voidaan suoraan lisätä resursseja jolloin varsinainen suorituskyky kasvaa. Logstash ei ole ainoa työkalu lokien keskittämistä varten. Muita suhteellisen suosittuja työkaluja lokien keskittämiseen on muun muassa Graylog2, Fluentd ja Splunk.

Luotettavuuden kannalta
tärkeä osa tämän kaltaisessa arkkitehtuurissa on viestijono (message queue), joka toimii myös puskurina. Kun ohjelmassa tulee äkillisiä kasvuja käyttäjämäärissä tai tapahtuu jotakin odottamatonta, lokia voi tulla hyvinkin paljon lyhyen ajan sisään. Tässä tapauksessa on tärkeää, ettei lokitietoja menetetä, eikä lokitusjärjestelmät kaadu niiden vastaanottokyvyn johdosta.

Yksi ohjelma, joka toimii tällaisena puskurina, on Kafka. Kafka on uudenlainen viestittämisjärjestelmä joka yhdistää perinteisten lokisyötteenlukijat ja viestijärjestelmien hyödyt (Kreps, Narkhede & Rao 2015, 1.).

Sen lisäksi, että Kafka toimii viestijonona, se myös säilöö viestejä ja muistaa mitä on luettu viimeksi. Kafka antaa jokaiselle viestille oman identifioivan numeron, offsetin, jonka avulla Kafka muistaa missä kohtaa edellisen kerran ollaan oltu. Kun, esimerkiksi uudelleenkäynnistyksen yhteydessä, otetaan uusiksi yhteys Kafkaan, Kafka jatkaa edellisestä käsitellystä offsetista eteenpäin. Kafkan suorituskyky on myös parempi kuin monien samankaltaisien ohjelmien.



Kuva 8 Kafkan suorituskykykuvaajat (Kreps, Narkhede & Rao. 2015, 6.)

Yllä olevissa kuvaajissa (Kuva 8) on kuvattu kaksi testitapausta, mittaamaan Kafkan suorituskykyä. Kafka toimii niin sanotulla Publish-Subscribe-menetelmällä. Publish-subsrcibe -menetelmässä viestin julkaisija (Publisher) ei lähetä viestiä suoraan vastaanottajalle (Consumer/Subscriber) vaan viestit menevät niin sanotun välittäjän (Broker) kautta. Vastaanottaja ei täten saa ilmoitusta siitä, milloin julkaisija on viestin julkaissut vain vasta kun välittäjä kertoo sen vastaanottajalle. Vastaanottaja eli tilaaja saa viestit vain niiltä kanavilta, mitä tilaaja on tilannut.

Kuvaaja kertoo Kafkan julkaisukapasiteetiksi 50.000 ja 400.000 viestiä sekunnissa, riippuen erän koosta. Vastaanottokapasiteetti on testissä keskiarvoltaan 22.000 viestiä sekunnissa, yli neljä kertaa ActiveMQ:n ja RabbitMQ:n kapasiteetti. ActiveMQ ja RabbitMQ ovat myös viestijono-ohjelmia.

Viimeisenä osana tämän kaltaista arkkitehtuuria on tietokanta tai tietovarasto. Kuvassa 5 tietovarastona toimii Elasticsearch. Elasticsearch tarjoaa tiedostojen nopean hakemisen, sekä turvallisen tavan säilöä lokiviestejä.

Elasticsearch ei ole ainoa mahdollinen loppusijoituspaikka lokeille. Tähän tarkoitukseen on olemassa paljon muitakin vaihtoehtoja. Lokeja voidaan säilyttää myös perinteisissä tietokannoissa, kuten relaatiotietokannoissa (esimerkiksi MySQL), tai uudemmissa NoSQL tietokannoissa (esimerkiksi Cassandra). Nykyisin, pilvipalveluiden yleistyessä, on muitakin vaihtoehtoja säilyttää tietoa tehokkaasti. Esimerkiksi Amazon Web Service (AWS) tarjoaa tällaista palvelua nimellä Amazon S3. Amazon S3 on automattisesti skaalautuva ja kestävä verkkotallennus palvelu.

3.2.2 Teknologiavalinnat

Kuten arvata saattaakin monet edellisissä luvuissa mainitut teknologiat valikoituivat myös tässä projektissa käytetyiksi teknologioiksi. Tässä luvussa perustellaan hieman valintoja.

Elasticsearch, Logstash ja Kibana ovat hyvin suosittuja ohjelmia. Ne ovat saman tahon ylläpitämiä ja kehittämiä, joten ne ovat suunniteltu toimimaan hyvin toistensa kanssa. Avoin lähdekoodi myös takaa sen, että minkä tahansa näistä kolmesta vaihtoehdosta voi aina korvata vaihtoehtoisella.

Elasticsearch valikoitui lokien säilöntätavaksi pitkälti suuren suosionsa, korkean skaalautuvuuden ja kattavan dokumentaation johdosta. Yksi vaihtoehdoista olisi ollut

säilyttää lokit esimerkiksi Cassandra-tietokannassa, joka olisi myös ollut skaalautuva. Elasticsearch-hakukoneen tarjoamat ominaisuudet kuitenkin vakuuttivat. Se on suunniteltu hyvin nopeisiin tekstihakuihin. Nopeus ei ole tärkein ominaisuus lokeja luettaessa, koska niitä luetaan vain ylläpidossa, eikä se juuri vaikuta käyttäjän käyttökokemukseen. Sen avulla on myös helppo tehdä esimerkiksi analyysiä lokeista, sillä on hyvin korkea skaalautuvuus ja sitä on mahdollista käyttää tulevaisuudessa muihinkin käyttötarkoituksiin. Elasticsearchin korkea suosio sen sijaan on osaltaan vaikuttanut hyvän ja auttavan yhteisön syntyyn. Tästä yhteisöstä saa apua kiperiinkin kysymyksiin nopeasti ja vaivattomasti.

Seuraavaksi valikoitui työväline lokien keräämiseen ja parsimiseen. Lokien keräämiseen ja parsimiseen harkittiin muutamia eri vaihtoehtoja. Näitä olivat esimerkiksi: PaperTrail, Loggly, SumoLogic, AWS CloudWatch ja Graylog2. Lopulta kuitenkin valittiin Logstash.

Elasticsearchin käyttö hakukoneena ja tietovarastona kasvatti Logstashin asemaa vertailussa muita ohjelmissa vastaan. Edellisessä kappaleessa mainituista ohjelmista vain Logstash ja Graylog2 ovat ilmaisia. SumoLogic ja Loggly ovat enemmän kokonaisvaltaisia tuotteita, tarjoten ominaisuudet aina lokien keräämisestä niiden analysointiin. Korkeat hinnat ja se, ettei kumpikaan ohjelmista (SumoLogic ja Loggly) ole avointa lähdekoodia, poistivat ne laskuista. PaperTrail on kevyt ohjelma, joka käytännössä lukee lokitiedostot melkein mistä tahansa lähteestä ja syöttää eteenpäin haluttuun säilytysosoitteeseen. PaperTrail toimii kuitenkin heidän omilla palvelimillaan ja on myös maksullinen, joten sekään ei sopinut vaihtoehdoksi. Graylog2 vaikutti vaihtoehtona hyvältä ja kattavalta, joten se oli hyvinkin varteenotettava vaihtoehto. Graylog2:n dokumentaatio ei kuitenkaan ollut aivan samalla tasolla kuin Logstashin dokumentaatio, ja se yhdistettynä Logstashin erinomaiseen yhteensopivuuteen Elasticsearchin kanssa teki Logstashista selvän valinnan. Kun tietovarastoksi oli valittu Elasticsearch ja lokien kerääjäksi Logstash, oli Kibanan valitseminen visualisointi- ja selaustyökaluksi ilmiselvä.

Muut projektissa käytetyt teknologiat, kuten AWS, SaltStack ja Kafka, olivat jo käytössä toimeksiantajalla. Näiden teknologioiden valintaan ei ollut vaikutusmahdollisuutta.

4 Sovelluksen lokituksen toteutus

Tässä kappaleessa käydään lävitse lokituksen käytännön toteutus. Tämä projekti toteutettiin käytännössä kahdessa osassa. Ensimmäiseksi lokitusta testattiin vain lokaalissa ympäristössä asentamalla käytetyt ohjelmat omalle tietokoneelle. Tässä vaiheessa testattiin ohjelmien käyttöönotto ja niiden konfiguroinnin luomiset. Testaukseen käytetyn tietokoneen spesifikaatiot olivat seuraavan laiset:

- Merkki & Malli: Lenovo ThinkPad S440
- Käyttöjärjestelmä: Windows 8.1 Pro 64-bit
- Prosessori: Intel Core i7-4510U CPU @ 2.00GHz (4 CPUs), ~2.6GHZ
- Muisti: 8192MB RAM
- Näytönohjain: Intel® HD Graphics 4400

Yhden kehitysympäristön pyörittämiseen käytettiin virtuaalikonetta. Virtuaalikoneen spesifikaatiot olivat seuraavan laiset:

- Virtualisointiohjelma: VirtualBox
- Käyttöjärjestelmä: Fedora 21 LTS 64-bit
- Prosessori: Intel Core i7-4510U CPU @ 2.00Ghz
- Muisti: 4426MB RAM

Virtuaalikoneissa kaikki komponentit ovat virtualisoituja, eivätkä oikeita fyysisiä komponentteja. Tästä johtuen suorituskyky ei ole samalla tasolla kuin aidossa ympäristössä. Pelkästään kehitysympäristön pyörittäminen tässä projektissa on jo resursseja vievää. Kaikki prosessit vaativat rutkasti muistia käyttöönsä. Kun tavallisen kehitysympäristön päälle asensi vielä lokitusohjelmat, tulivat suorituskykyrajoitteet nopeasti vastaan. Töitä ei voinut tehdä hyvinkään montaa minuuttia putkeen, ennen kuin tietokone meni kokonaan jumiin.

Toisessa vaiheessa kehitysympäristö siirrettiin pilveen. Pilvessä on käytössä huomattavasti suuremmat resurssit, ja kun kehitysympäristö pyörii palvelimella, ei tietokoneen tarvitse jakaa laskentatehoja kehitysympäristön pyörittämiseen ja koneen muiden toimintojen pyörittämiseen. Muita toimintoja ovat esimerkiksi internet-selain ja tekstieditori.

Kehitysympäristöä pilvessä ei voi kuitenkaan vielä kutsua vielä skaalautuvaksi, koska käytännössä se toimii kokonaan yhdellä koneella. Tämän luvun loppupuolella käsitellään kuitenkin lisää tarvittavia konfiguraatiota, mitä täytyy ottaa huomioon, kun siirrytään

kehitysympäristöstä tuotantoympäristöön. Tämä opinnäytetyöprojekti ei käsittele tuotantoympäristöön siirtymistä.

Kappaletta voi käyttää myös opetustarkoituksessa kyseessä olevien teknologioiden käyttöönottoon. Ensimmäisenä käydään lävitse lyhyesti Amazon Web Servicen (AWS) roolia tässä arkkitehtuurissa. Tämän opinnäytetyön toteuttamiseen, ei tarvinnut AWS:n toiminnasta tietää paljoa, sillä tarvittavat palvelut olivat jo pystytetty etukäteen. AWS:n läpikäymisen jälkeen luvussa 4.2 käsitellään ELK-ohjelmistopinon konfiguraatioita ja kuinka ne rakennetaan.

Työläin osuus toteutuksesta oli SaltStackin opetteleminen ja sitä varten tarkoitettujen reseptien luominen. Reseptit ja muut SaltStackin ominaisuudet käydään tarkemmin lävitse luvussa 4.3.

4.1 Amazon Web Services (AWS)

Tämän opinnäytetyön otsikossakin mainittu pilvi voi kuulostaa abstraktilta käsitteeltä. Käytännössä se on Amazon Web Services eli AWS. AWS on maailman suurin pilvialusta, joka tarjoaa lukuisia eri palveluita kaikenlaisiin käyttötarkoituksiin. Amazon Web Servicen avulla on mahdollista siirtää käytännössä koko yrityksen toiminta pilveen. Kenties suosituin AWS:n tarjoamista palveluista on Amazon Elastic Compute Cloud (Amazon EC2). Amazon EC2 on palvelu, joka tarjoaa skaalautuvaa laskentatehoa Verkon yli (Amazon Web Services. 2011). Sen web-käyttöliittymä on suunniteltu helpoksi käyttää ja sen saa käyttöönsä minuuteissa. Käytännössä se tapahtuu siten, että AWS:n avataan uusi Amazon EC2 instanssi, mihin on mahdollista asentaa haluamansa käyttöjärjestelmä. Amazon EC2:n hinnoittelu on vaikuttanut myös osaltaan sen suosioon, siitä maksetaan toteutuneen käytön mukaan. Amazon EC2-instanssi on mahdollista esimerkiksi konfiguroida siten että, jos hinta nousee määritetyn rajan ylitse, se sammutetaan automaattisesti.

AWS:ssä EC2-instanssien kanssa säilytystila abstraktoidaan Amazon Elastic Block Storagen (Amazon EBS) avulla. EBS maksaa myös käytön mukaan. Tässä opinnäytetyössä kehitysympäristö pystytettiin Amazon EC2 instanssin päälle. Kuinka tämmöinen instanssi pystytettiin, ei kuitenkaan kuulu projektin laajuuteen, joten se jätetään käymättä lävitse.

Instanssi oli m4.large -niminen instanssi. M4.large-instanssiin kuuluu 2.4GHZ Intel Xeon® E5-2676 v3 (Haswell) prosessori, 8GB RAM muistia ja 100GB Amazonin Elastic Block Storage säilytyskapasiteettia, jonka nopeus on 450Mbps. Käyttöjärjestelmänä käytettiin Centos 7-käyttöjärjestelmää

4.2 ELK-ohjelmistopinon asentaminen

Elasticsearchin, Logstashin ja Kibanan asentaminen on helppoa. Asennustapoja on käytännössä kaksi. Joko ohjelmat ladataan Elasticin sivuilta tai RPM- ja Debian-paketinhallintaa käyttävissä GNU/Linux-käyttöjärjestelmissä voidaan ottaa käyttöön jokaiselle ohjelmalle tarkoitettu oma pakettivarasto, jolloin ohjelmat voidaan asentaa paketinhallinnan kautta. Paketinhallinnan kautta asentaminen on suositeltavaa, koska tämän avulla voidaan esimerkiksi päivittää ohjelmat paketinhallinnan kautta sen sijaan, että ladattaisiin aina uusi versio Elasticin sivulta.

Koska tässä projektissa kehitysympäristö toimi AWS:ssä, on paketinhallinnan kautta asentaminen huomattavasti vaivattomampaa kuin sivuilta lataaminen. Kun asentaa ohjelmat paketinhallinnan kautta, asentuu käyttöjärjestelmälle myös tarvittavat käynnistyskomennot oikeisiin paikkoihin, jolloin ohjelmat voidaan käynnistää taustalle palveluina. Kuvasta 1 nähdään esimerkiksi, kun ohjelma on käynnistetty niin kutsutusti etualalle, eikä palveluna. Tämä estää esimerkiksi muiden komentojen ajamisen tästä istunnosta, koska ohjelma itse on käynnissä kyseisessä istunnossa. Palveluna pyörivä ohjelma pyörii taustalla piilossa, jolloin se ei häiritse muuta käyttöä. Samoin jos etu alalla olevan istunnon lopettaa, loppuu myös tämän ohjelman ajaminen. Palveluna ajettaessa ohjelma on käynnissä taustalla niin kauan kuin sen halutaan olevan.

4.2.1 Elasticsearchin asentaminen

Alla olevassa kuvassa (Kuva 9) näkyy Elasticsearchin asentamisen ensimmäinen vaihe. Rpm-komennolla tuodaan Elasticin tarjoama julkinen avain rpm-paketinhallinnan asetuksiin. Julkisen avaimen avulla voidaan Elasticsearch-asentaa Elasticin-pakettivarastosta salatusti ja varmistetaan, että paketti on varmasti aito Elasticsearch, eikä esimerkiksi kolmannen osapuolen huijauspaketti. Tämän jälkeen lisätään elasticsearchin-pakettivarasto käyttöjärjestelmän pakettivarastoja hallitseviin asetuksiin.

```
rpm --import https://packages.elastic.co/GPG-KEY-elasticsearch
vim /etc/yum.repos.d/elasticsearch.repo
```

Kuva 9 Ensimmäiset komennot Elasticsearchin asentamiseksi

Vim on tekstieditori, jolla luodaan elasticsearch.repo niminen tiedosto. Tiedostoon tulee seuraava teksti:

```
[elasticsearch_repo]
gpgcheck=1
gpgkey=http://packages.elastic.co/GPG-KEY-elasticsearch
name=Elasticsearch repository for 2.x packages
humanname=Elasticsearch repository for 2.x packages
baseurl=http://packages.elastic.co/elasticsearch/2.x/centos
```

Kuva 10 Pakettivaraston käyttöönottaminen

Kun pakettivaraston tiedot on asetettu haluttuun tiedostoon, tunnistaa Centoksen paketinhallintaohjelma *yum* pakettivaraston olemassaolon. Tämän jälkeen Elasticsearch on hyvin yksinkertaista asentaa seuraavalla komennolla:

```
yum install elasticsearch
```

Tai käyttäen uudempaa paketinhallintaohjelmaa, *dnf:ää:*

```
dnf install elasticsearch
```

Nämä kolme vaihetta ovat kaikki, mitä tarvitaan Elasticsearchin asentamiseen. Paketinhallintaohjelma pitää huolen siitä, että kaikki tarvittavat käynnistyskomennot ja konfiguraatiot löytyvät oikeasta paikasta. Nyt Elasticsearch on mahdollista käynnistää palveluna. Centos käyttöjärjestelmässä palveluita hallitaan *Systemd*-nimisellä ohjelmalla. Systemd on järjestelmän ja palveluiden hallitsemiseen tarkoitettu ohjelma (Red Hat). Elasticsearch käynnistetään ajamalla seuraava komento:

```
service elasticsearch start
```

Tämän jälkeen elasticsearch on käynnissä oletusasetuksilla huomaamattomasti taustalla. Oletusasetukset kelpaavat elasticsearchille testaamiseen, mutta kehitysympäristön koosta ja palvelimen rajatuista resursseista johtuen, tässä projektissa oletusasetuksia jouduttiin muutamaan.

Palvelimella, jossa kehitysympäristö on käynnissä, on 8GB RAM-muistia. Elasticsearchille suositellaan dokumentaatiossa tällaisessa tapauksessa jakaa puolet palvelimen muistista. Koska kehitysympäristössä pyörii koko skaalautuvaksi suunniteltu ohjelma kaikkine komponentteineen yhdellä palvelimella, jotka kaikki vievät osansa muistia, täytyy Elasticsearchin käyttämän muistin määrää pienentää. Tämä ei kuitenkaan haittaa siitä

syystä, että tällä palvelimella ei ole lähelläkään tuotantoympäristön laajuutta lokien määrässä, joten Elasticsearch kyllä pystyy toimimaan pienemmälläkin muistimäärällä.

Elasticsearchin konfiguraatiota muokataan yml-päätteisestä tiedostosta. Kun on asentanut ohjelman paketinhallinnan kautta, tämä konfigurointitiedosto löytyy /etc/elasticsearch-hakemistosta. Testikäyttöön oletusasetukset toimivat hyvin, mutta tässä projektissa joutui Elasticsearchin konfiguraatiota hieman muokkaamaan. Elasticsearch-klusterissa koneet liittyvät toisiinsa klusterin nimen perusteella. Jotta vältyttäisiin väärien Elasticsearch-instanssie liittymistä toisiinsa, klusterin nimi täytyi vaihtaa seuraavalla tavalla:

```
27  ################################### Cluster ###################################
28
29  # Cluster name identifies your cluster for auto-discovery. If you're running
30  # multiple clusters on the same network, make sure you're using unique names.
31  #
32  #cluster.name: elasticsearch
33
```

Kuva 11 Elasticsearch-klusterin nimen vaihtaminen

Kuvassa 5 rivillä 32 näkyy Elasticsearchin klusterin nimi asetus. Risuaita tällä rivillä tarkoittaa, että se kohta on kommentoitu ulos ja Elasticsearch käyttää oletus nimeään. Ottamalla pois risuaidan ja laittamalla halutun nimen kohdan ”*elasticsearch*” tilalle on mahdollista vaihtaa klusterin nimi haluamakseen. Tässä vaiheessa muita muutoksia ei Elasticsearchille tarvitse tehdä

4.2.2 Logstashin ja Kibanan asentaminen

Logstashin ja kibanan asentamine sisältää käytännössä samat vaiheet Elasticsearchin kanssa. Tästä johtuen käydään kohdat listana lävitse:

1. Pakettivarastojen käyttöönotto:
    1. Luodaan halutulla tekstieditorilla tiedostot logstash.repo ja kibana.repo ja laitetaan alla olevissa kuvissa näkyvät tekstit:

        ```
        [logstash-2.2]
        name=Logstash repository for 2.2.x packages
        baseurl=http://packages.elastic.co/logstash/2.2/centos
        gpgcheck=1
        gpgkey=http://packages.elastic.co/GPG-KEY-elasticsearch
        enabled=1
        ```

        Kuva 12 Logstash-pakettivaraston käyttöönotto

```
[kibana-4.4]
name=Kibana  repository for 4.4.x packages
baseurl=http://packages.elastic.co/kibana/4.4/centos
gpgcheck=1
gpgkey=http://packages.elastic.co/GPG-KEY-elasticsearch
enabled=1
```

Kuva 13 Kibana-pakettivaraston käyttöönotto

2. Asennetaan ohjelmat komennolla: *yum install logstash kibana*

Kibanaa varten ei tarvitse muuttaa konfiguraatioita, vaan se toimii suoraan portissa 5601. AWS:ssä täytyy tehdä EC2 instanssille oma niin kutsuttu security rule, joka ohjaa julkisen IP-osoitteen porttiin 5601 tulevat pyynnöt palvelimella osoitteeseen 127.0.01:5601. Tämän jälkeen Kibanaan pääsee käsiksi kätevästi internet-selaimella

4.2.3 Logstash konfigurointi

Johtuen Systemd:n rakenteesta, käynnistyskomennot eivät näe ympäristömuuttujia suoraan, vaan tarvittavat ympäristömuuttujat täytyy tuoda erikseen koodiin. Logstashin käynnistyskomennot eivät ainakaan versiossa 2.2 ottaneet tätä asiaa huomioon, joten se ei käynnistynyt palveluna suoraan asennuksen jälkeen. Logstash käyttää Java Virtual Machinea (JVM) ja täten ohjelman tarvitsee tietää Javan sijainti käyttöjärjestelmässä. Tämä täytyi muokata mukaan Logstashin käynnistyskomentoihin.

Ensimmäisena Javan sijainti piti tuoda alustavaan käynnistyskomentoon, joka sijaitsi /etc/init.d/ -hakemistossa nimellä logstash. Kyseiseen tiedostoon lisättiin seuraavan lainen rivi:

```
20 export JAVA_HOME=/usr/lib/java
```

Kuva 14 Javan kotihakemiston lisäys käynnistyskomentoon

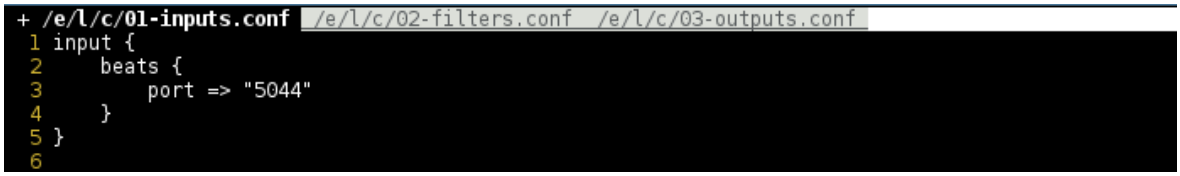
Tämä käynnistyskomentosarja ajaa lävitse monia komentoja lopulta käynnistäen itse logstashin. Logstashin käynnistetään hakemistosta /opt/logstash. Tämän hakemiston alta bin/ -hakemistosta löytyy oma komentosarja *logstash.lib.sh* joka käynnistää Logstashin ja koska Logstash on Java-ohjelma, se myös tarvitsee Java käynnistyskomennon sijainnin. Alla oleva kuvassa esitetty rivi lisättiin *logstash.lib.sh*-tiedostoon:

```
28 export JAVACMD=/usr/lib/java/bin/java
```

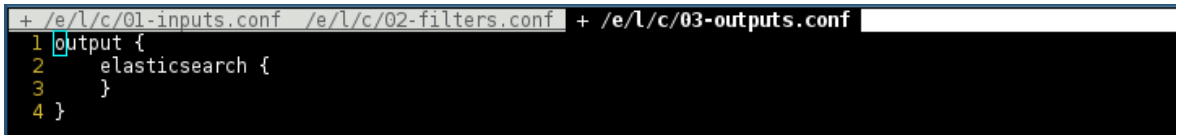
Kuva 15 Java-komennon sijainnin lisäys logstash.lib.sh-tiedostoon

Logstashin putkisto muodostuu siis käytännössä kolmesta osasta: sisääntulo, filtteri ja ulostulo. Näiden haluttu toimintalogiikka tehdään /etc/logstash/conf.d/ hakemistoon. Logstash tunnistaa kaikki hakemistoon laitetut tiedostot ja käynnistäessä kokoaa näistä tiedostoista yhden kokonaisen konfiguraation Logstashin kolmelle osalle.

Tässä projektissa tiedostot nimettiin 01-inputs.conf 02-filters.conf 03-outputs.conf. Tiedostojen nimistä voi päätellä mistä löytyy sisääntuloasetukset, mistä filtteri- ja mistä ulostuloasetukset.



Kuva 16 Logstash sisääntuloasetukset



Kuva 17 Logstash ulostuloasetukset

Kuva 16 ja Kuva 17 kertoo tässä projektissa käytetyt sisääntulo ja ulostulo asetukset. Sisääntulossa käytetään Logstashin Beats-laajennusta ja sille annetaan ohje käyttää porttia 5044. Beats-laajenuksesta kerrotaan lisää seuraavassa luvussa. Ulostulossa puolestaan kerrotaan Logstashille käytettäväksi sisäänrakennettua Elasticsearch laajennusta, jolla Logstash osaa itsestään syöttää lokit käsittelyn jälkeen Elasticsearchiin.

Väliin jäävää filtteröintiasetuksia ei voida valitettavasti tässä projektissa näyttää. Käytännössä filtterissä parsitaan Logstashin läpikulkevat lokit haluttuun muotoon. Lokitekstit jaotellaan alkuperäisen lähteen mukaan oman nimisiin lokeroihin, joiden avulla niitä on helppo etsiä Elasticsearchista. Filtterissä lokiteksteistä myös erotellaan halutut tiedot omiin kenttiin. Näitä tietoja ovat muun muassa aikaleimat, lokituksen tasot, erilaiset tunnistimet ja tietysti itse lokiviestit.

Logstashissa lokien parsimiseen käytetään Grok-työkalua. Grok toimii yhdistelemällä tekstikaavoja, jotka poimivat lokiteksteistä kaavoja vastaavat kohdat. Grok käyttää avukseen säännöllisiä lauseita (Regular Expression). Säännölliset lauseet ovat lausekkeita, jotka tietojenkäsittelytieteessä määrittelevät säännöllisen kielen.

4.2.4 Beatsin asentamine ja käyttöönotto

Beats on Logstash-lisäosa, jonka avulla on mahdollista siirtää dataa erityyppisistä lähteistä Logstashiin ja halutessaan Logstashista eteenpäin. Koska tässä projektissa lokit luetaan suoraan ohjelmien lokitiedostoista, tarvitsemme käyttöön vain Beatsin lisäosan joka pystyy lukemaan rivit tiedostoista. Tämän osan nimi on Filebeat.

Beats lisäosa asennetaan Logstashiin sen omalla lisäosien hallinta-komennolla. Komento löytyy hakemistosta /opt/logstash/bin/ nimella plugin. Ajamalla /opt/logstash/ hakemistossa komennon *bin/plugin install logstash-input-beats* saa asennettua logstashin pakettivarastosta beats plugin.

4.3 ELK-ohjelmistopinon käyttö SaltStackin kanssa.

Tässä opinnäytetyössä ei käsitellä koko SaltStackin (lyhyesti Salt) asentamista ja käyttöönottoa. Sen sijaan tässä luvussa käsitellään kuinka ELK-ohjelmistopinon ohjelmat otettiin käyttöön Saltin kanssa.

Salt on yksi suosituimpia avoimen lähdekoodin projekteja ja sillä on todella iso ja auttava yhteisö. Monien ohjelmien käyttöönottoon on jo valmiiksi kirjoitettu paljon reseptejä. Salt on tehnyt yhteiset pelisäännöt näille resepteille, jotta ne olisivat yhdenmukaisia. Kokoelma Saltin resepteistä löytyvät osoitteesta https://github.com/saltstack-formulas.

Tässä projektissa hyväksikäytettiin näitä valmiiksi tehtyjä reseptejä Logstashin, ja filebeatin osalta. Koska Kibanaan ei tarvitse sen kummempia konfiguraatioita, sen asentamiseen ei tarvinnut käyttää erillistä reseptiä. Elasticsearchin valmiit reseptit olivat hieman vajavaisia, joten sitä varten tehtiin kokonaan oma resepti.

Reseptit löytyvät omasta formulas-hakemistosta, mikä on määritelty Salt Masterin pääkonfiguraatiotiedostossa. Tämä löytyy masterin hakemistosta /etc/salt/master. Formuloissa määritellään yksitellen operaatiot mitä minionilla ajetaan, jotta saadaan haluttu ohjelma asennettua.

SaltStackin määrittelemien ohjeiden mukaan määritellyissä resepteissä on aina yksi init.sls-tiedosto joka ajetaan kun kyseistä reseptiä kutsutaan. Esimerkiksi jos minionia käynnistäessä on määritelty elasticsearch rooli ja on asetettu elasticsearch roolin löytyessä asentamaan elasticsearch niminen tila, Salt etsii elasticsearchin respetihakemistosta init.sls tiedoston ja ajaa tämän.

Käytännössä tässä projektissa käytetyt reseptit ajavat edellisissä luvuissa kuvatut asennusvaiheet lävitse ja käynnistävät palvelut.

4.3.1 Esimerkkirespeti Saltilla

Alla käydään läpi Elasticsearchin salt-resepti vaihe vaiheelta:

Ensimmäisessä vaiheessa (Kuva 18) tuodaan halutut asetukset, joka sisältää esimerkiksi muuttujien arvoja, erillisestä settings.sls tiedostosta. Tämän jälkeen käytetään Saltin sisäänrakennettun Staten *pkgrepo* metodia managed, joka ottaa käyttöön Elasticsearchin pakettivaraston asennusvaiheessa.

```
1  {%- from 'elasticsearch/settings.sls' import elasticsearch with context %}
2
3  elasticsearch_repo:
4    pkgrepo.managed:
5      - humanname: Elasticsearch repository for 2.x packages
6      - baseurl: http://packages.elasticsearch.org/elasticsearch/2.x/centos
7      - gpgcheck: 1
8      - gpgkey: http://packages.elastic.co/GPG-KEY-elasticsearch
9
10
```

Kuva 18 Elasticsearchin salt reseptin ensimäinen kohta

Toisessa vaiheessa (Kuva 19) määritetään Ryhmä ja käyttäjät, jonka oikeuksilla kyseistä reseptiä ajetaan. Tässä asetetaan ryhmäksi ja käyttäjäksi oletuksena *root.*

```
11 # user/groups
12 elasticsearch_group:
13   group:
14     - present
15     - name: {{ elasticsearch.group|default('root') }}
16
17 elasticsearch_user:
18   user:
19     - present
20     - name: {{ elasticsearch.user|default('root') }}
21     - groups:
22       - {{ elasticsearch.group|default('root') }}
23
```

Kuva 19 Elasticsearch reseptin toinen kohta

Kolmannessa vaiheessa (Kuva 20) luodaan Elasticsearchin konfigurointi-tiedostot. Tässä käytetään Saltin *file* Statea ja sen metodia *managed*, jonka avulla on mahdollista luoda käyttöjärjestelmälle uusia tiedostoja ja hallita olemassa olevia tiedostoja. Alla olevassa kuvassa ensiksi luodaan hakemisto /etc/elasticsearch ja tämän jälkeen sinne luodaan elasticsearch.yml ja logging.yml tiedostos. ”source”-kohdassa määritetään, että tämä

elasticearch.yml ja logging.yml tiedostot löytyvät salt masterilta hakemistosta /elasticsearch/conf eli nämä tiedostot ovat jo valmiiksi olemassa ja muokattuna haluttuun muotoon. Niille annetaan myös oikeudet 644.

```
24  /etc/elasticsearch:
25    file.directory:
26      - user: {{ elasticsearch.user|default('root') }}
27      - group: {{ elasticsearch.group|default('root') }}
28      - mode: 755
29
30  elasticsearch_config_main:
31    file.managed:
32      - name: /etc/elasticsearch/elasticsearch.yml
33      - source: salt://elasticsearch/conf/elasticsearch.yml
34      - template: jinja
35      - mode: 644
36      - local_dir: {{ elasticsearch.local_dir }}
37      - user: {{ elasticsearch.user|default('root') }}
38      - group: {{ elasticsearch.group|default('root') }}
39
40  elasticsearch_config_logging:
41    file.managed:
42      - name: /etc/elasticsearch/logging.yml
43      - source: salt://elasticsearch/conf/logging.yml
44      - template: jinja
45      - mode: 644
46      - local_dir: {{ elasticsearch.local_dir }}
47      - user: {{ elasticsearch.user|default('root') }}
48      - group: {{ elasticsearch.group|default('root') }}
49
```

Kuva 20 Elasticsearch salt reseptin kolmas kohta

Neljännessä vaiheessa (Kuva 21) luodaan samaa managed-metodia käyttäen elasticsearchin systeemi konfiguraatio hakemistoon /etc/sysconfig/.

```
50  /etc/sysconfig/elasticsearch:
51    file.managed:
52      - source: salt://elasticsearch/conf/default_conf
53      - template: jinja
54      - mode: 644
55      - require:
56        - pkg: elasticsearch
57
```

Kuva 21 Elasticsearch salt reseptin neljäs kohta

Viidennessä (Kuva 22) ja viimeisessä kohdassa tarkistetaan pkg Statella, että elasticsearch paketti on varmasti asennettuna ja tämän jälkeen ajetaan service Staten running-metodi, joka tarkistaa onko tällainen elasticsearch palvelu jo käynnissä. Jos palvelu ei ole käynnissä tällöin se käynnistetään palvelimella.

```
58  elasticsearch:
59    pkg:
60      - installed
61      - require:
62        - pkgrepo: elasticsearch_repo
63    service.running:
64      - name: elasticsearch
65      - enable: True
66      - require:
67        - pkg: elasticsearch
68        - file: /etc/elasticsearch/elasticsearch.yml
69
```

Kuva 22 Elasticsearch salt reseptin viides kohta

5 Pohdinta

Tämä luku käsittelee opinnäytetyön projektin lopputuloksia. Luvussa pohditaan onko halutut tavoitteet saavutettu, mitä olisi voinut tehdä paremmin sekä jatkokehitystä. Tämä opinnäytetyön tekeminen on myös opettanut paljon uusia asioita ja tekniikoita, joten tässä luvussa pohditaan myös työn opetusarvoa.

5.1 Lopputulos

Opinnäytetyön tavoitteena oli saada toteutettua keskitetty lokitusjärjestelmä, joka on mahdollista ottaa käyttöön hajautetussa, pilvessä toimivassa, järjestelmässä. Tässä projektissa kuitenkin tuotantoympäristö toimii huomattavasti erilaisessa infrastruktuurissa kuin kehitysympäristö. Käytännössä erona kehitysympäristön ja tuotantoympäristön välillä on käytössä olevien koneiden määrä. Kehitysympäristössä on vain yksi kone jossa pyörii kaikki palvelun osat, kun taas tuotantoympäristössä koneita on useampia.

Elasticsearch, Logstash ja Kibana valikoituivat käytetyiksi työvälineiksi pitkälti niiden skaalautuvuuden ja suosion takia. Vaikeuksia tässä projektissa aiheutti tuotantoympäristön simulointi käyttäjämäärien osalta. Elasticsearchin hakumoottori toki toimii yhdellä koneella, mutta esimerkiksi sen sirpaleiden replikointi, joka takaa sen ettei tiedot häviä, ei toimi yhdellä koneella. Tämän opinnäytetyön aikana ei selvinnyt näiden kolmen ohjelman kestämä rasitus. Kehitysympäristössä simuloitiin käytännössä vain muutamien käyttäjien käyttöä. Tämä ei ainakaan aiheuttanut suurempia ongelmia minkään ohjelman kanssa.

Päätavoite tehtävässä kuitenkin saavutettiin. Opinnäytetyön tuloksena on mahdollista käynnistää AWS:n uusi instanssi, jossa käynnistyy Saltin avulla automaattisesti kaikki tarvittavat ohjelmat, Elasticsearch, Logstash, Filebeat ja Kibana, oikeilla konfiguraatioilla. Filebeat kerää lokit kaikista tiedostoista, mistä lokit halutaan saada talteen. Logstash prosessoi lokit haluttuun muotoon ja Elasticsearch indeksoi ja säilöö lokit. Kibanan avulla on helppo selata tallennettuja dokumentteja Elasticsearchista, analysoida niitä, tehdä niistä erilaisia kuvaajia ja mittareita tai luoda koelautoja, joissa yhdistyy kaikki nämä edellä mainitut toiminnot.

Asia missä ei aivan saavutettu haluttua tasoa oli itse lokituksen suunnittelu. Tehtävä osoittautui haasteellisemmaksi kuin alun perin oli odotettu. Tämän hetkiseen ohjelmassa tulostuviin lokeihin ei paljoa koskettu. Monet tiedot, mitä alun perin oli suunniteltu kerättävän, osoittautuikin odotettua vaikeammaksi. Muut henkilöt yrityksessä eivät vielä

ole käyttäneet opinnäytetyössä asennettuja systeemejä, joten on vaikea sanoa mihin suuntaan esimerkiksi filttereitä ja muita on kehitettävä. Ajan kanssa paljastuvat eri henkilöiden mieltymykset ja kun tulee eteen oikeita ongelmia oikeasta ympäristöstä, tiedetään paremmin mitä halutaan etsiä.

Teoriaosuuden tekemisessä lähteiden löytäminen osoittautui paljon haastavammaksi kuin etukäteen odotin. Itse lokituksesta ei hirveästi ole lähdekirjallisuutta saatavilla, vaan yleensä lokituksesta kertovat asiat oli piilotettuina esimerkiksi järjestelmäarkkitehtuureista kertoviin teoksiin. Hyvän lokituksen suunnittelua ja tekemistä olisi voinut avata enemmän.

5.2 Oppiminen

Tämän opinnäytetyön kanssa työskentely opetti asioita monista elämän osa-alueista. Oppiminen ei siis rajoittunut pelkästään tekniseen osaamiseen, kuten uudet teknologiat tai järjestelmien kanssa työskentely, mutta myös henkilökohtaiset intressit sekä työskentelytavat saivat paljon uusia näkökulmia.

Sitä on vaikea sanoa, kuinka tärkeitä kaikki nämä teknologiat ovat työelämässä. Elasticsearch, Logstash, Kibana, AWS ja Salt ovat kuitenkin omissa luokissaan suosituimpia ohjelmia tai alustoja. Kaikkien näiden teknologioiden osaaminen vahvistaa asemaa työmarkkinoilla. Tietotaidot sain sellaiselle tasolle, että tästä on hyvä lähteä kehittämään kyseessä olevia taitoja sille tasolle, että niillä kehtaisi jopa hakea työpaikkaa, mutta en usko että vielä pelkästään tämän projektin myötä ole sillä tasolla.

Opinnäytetyön tekeminen pakotti selvittämään asioita paljon perinpohjaisemmin, mitä yleensä kun alkaa opettelemaan uutta teknologiaa. Teoriaosuus pakotti etsimään laajalta alalta lähteitä ja käyttämään esimerkiksi Google Scholaria. Kaikki dokumentaatioiden ja eri lähdekirjallisuuden ja blogien lukeminen paransi myös minun taitoja niin järjestelmien ylläpidosta, ohjelmoinnista kuin arkkitehtuureista.

Lokituksen arkkitehtuurin suunnittelu opetti todella paljon myös projektissa käytössä olevasta järjestelmästä, josta olen jo nyt huomannut olleen todella paljon hyötyä muissa kehitysprojekteissa.

5.3 Jatkokehitys

Kuten aikaisemmin on mainittu, tämä opinnäytetyö toteutti lokituksen kehitysympäristöön. Seuraava vaihe on tuoda tämä lokitus tuotantoon. Sitä varten joutuu tehdä vielä lisää

suunnittelua, sillä lokitusjärjestelmä ei missään nimessä saa esimerkiksi estää muuta järjestelmää toimimasta.

Muutokset joita täytyy tehdä, koskevat lähinnä Saltia. Hajautetussa järjestelmässä ohjelman eri osat saattavat olla satunnaisesti hajautunut eri koneisiin. Filebeatin ja Logstashin konfiguraatioiden täytyy mukautua sen mukaan, mitä prosesseja kohdetietokoneessa on käynnissä.

Aidossa ympäristössä myös käyttäjämäärät ja sitä kautta viestien määrä moninkertaistuu. Filebeatilla ja Logstashilla saattavat tulla rajat vastaan kun viestien määrä äkillisesti nousee ruuhka-aikoina. Tähän ongelmaan on vastaus jo projektissa käytössä oleva tuote, Kafka. Kafka voi toimia viestijonona Filebeatin ja Logstashin välillä, puskuroiden viestejä ja pitämässä kirjaa niiden järjestyksestä. Kafkaa ei kuitenkaan otettu tässä opinnäytetyössä arkkitehtuuriin mukaan, koska Kafka-lisäosaa ei vielä ollut saatavilla Filebeatiin. Tämä lisäosa on kuitenkin tulossa ehkä jopa lähitulevaisuudessa, joten jos sitä ei vielä tarvita se tullaan implementoimaan tulevaisuudessa. Mikäli tuotantoon siirtymisessä ilmestyy ongelmia viestien määrän kanssa, täytyy Kafka lisäosa toteuttaa itse.

Kehitystä tullaan tekemään myös lokien analysoinnin suhteen. Lokien avulla voidaan analysoida esimerkiksi käyttäjien seurantaan ohjelman sisällä ja erilaisten tapahtumien laskemiseen ja niin edelleen. Tämänkin projektin pitää saada tuottamaan riittävästi lisäarvoa tuotteelle ja sen kehitykselle ja toivon mukaan tulevaisuudessa voidaan saada hyviä tuloksia aikaiseksi.

Lähteet

Amazon Web Services. Luettavissa: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html.

Automaatioseura Ry. Luettavissa: http://www.automaatioseura.com/component/content/article/5-uusimmat-tiedotteet/186-automaation-maeaeritelmae-keskustelua

Bahaaldine Azarmi. 2016. Scalable Big Data Architecture - A practitioner's guide to choosing relevant big data architecture. Apress. Luettu 15.04.2016

Baishakhi, Daryl, Filkov & Devanbu. 2014. A Large Scale Study of Programming Languages and Code Quality in Github. Luettavissa: http://macbeth.cs.ucdavis.edu/lang_study.pdf. Luettu: 18.04.2016.

CLOUD, Amazon Elastic Compute. Amazon web services. 2011. Luettu: 28.04.2016. Luettavissa: http://dclug.tux.org/200611/AmazonEC2.pdf.

Colton Myers. 2015. Learning SaltStack. Packt Publishing. Birmingham. Luettu 05.05.2016.

Elasticsearch. Basic Concepts. Luettavissa: https://www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html#_index. Luettu:13.04.2016

Elasticsearch. Luettavissa: https://www.elastic.co/guide/en/elasticsearch/guide/current/important-configuration-changes.html

Garry Shutler 2012. Logging. Luettavissa: http://gshutler.com/2012/12/logging/. Luettu: 5.2.2016

James Turnbull. 2014. The LogStash Book. Omakustanne. Luettu: 23.04.2016

Jay Kreps 2013. The Log: What every software engineer should know about real-time data's unifying abstraction. Luettavissa: https://engineering.linkedin.com/distributed-

systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying. Luettu: 5.2.2016

Kreps, J., Narkhede, N. & Rao, J. 2015. Kafka: a Distributed Messaging System for Log Processing. Luettavissa: http://people.csail.mit.edu/matei/courses/2015/6.S897/readings/kafka.pdf. Luettu: 9.4.2016

Logstash. Deploying and Scaling Logstash. Luettavissa: https://www.elastic.co/guide/en/logstash/current/deploying-and-scaling.html. Luettu: 5.4.2016

Rafael Kuć & Marek Rogoziński. 2013. Mastering Elasticsearch. Packt Publishing. Birmingham. Luettu: 17.04.2016

Red Hat. Luettavissa: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/System_Administrators_Guide/chap-Managing_Services_with_systemd.html

SaltStack. Luettavissa: https://docs.saltstack.com/en/latest/

Saurabh Goyal. 2015. Centralized vs Decentralized vs Distributed. Luettu: 15.02.2016. Luettavissa: https://medium.com/@bbc4468/centralized-vs-decentralized-vs-distributed-41d92d463868#.wxq92xp2h

Wolfgang Emmerich. 1997. Distributed System Principles. Luettavissa: http://www0.cs.ucl.ac.uk/staff/ucacwxe/lectures/ds98-99/dsee3.pdf (kalvoja). Luettu 23.02.1016