TAMPEREEN
AMMATTIKORKEAKOULU

JÄRJESTELMÄN LOKIEN JA METRIIKOIDEN AUTOMAATTINEN VALVONTA

Elastalert-työkalun testaus ja käyttöönotto

Lauri Pudas

Opinnäytetyö
Toukokuu 2018
Tietotekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Ohjelmistotekniikka

PUDAS, LAURI:
Järjestelmän lokien ja metriikoiden automaattinen valvonta
Elastalert-työkalun testaus ja käyttöönotto

Opinnäytetyö 52 sivua, joista liitteitä 8 sivua
Toukokuu 2018

Tässä opinnäytetyössä testattiin ohjelmistojen ja järjestelmien valvontatyökalu Elastalert. Työn tavoitteena oli selvittää Elastalertin kykyä toimia osana hajautetun järjestelmän monitorointia. Samalla Elastalertia verrattiin vanhempaan, jo käytössä olevan valvontajärjestelmään. Työssä selvitettiin myös uudelleenkäytettävän ja konfiguroitavan asennusautomaation käytettävyyttä Elastalertin kanssa. Työssä esitellään lisäksi ohjelmistojen monitoroinnin ja valvonnan teoriaa.

Elastalert todettiin toimivaksi valvontatyökaluksi toimimaan osana Elastic-pinoa. Elastalertin helppokäyttöisyys ja yksinkertaisuus olivat työkalun vahvoja puolia. Elastalertista kuitenkin puuttui ominaisuuksia, joiden puute nähtiin esteeksi projektissa olleen haastavan hajautetun järjestelmän kanssa. Elastalert-työkalua ei sen vuoksi otettu tuotantokäyttöön heti testijakson jälkeen.

Elastalertin testauksella saatiin aikaan hyvää keskustelua projektin sisällä. Lisäksi huomattiin kohteena olleen järjestelmän monitoroinnin ja valvonnan tärkeys sekä sen monimutkaisuus. Testijakson jälkeen todettiin tarve kokeilla useita muitakin valvontajärjestelmiä, jotta juuri oikea järjestelmä löydettäisiin valvonnan tehostamiseksi. Työn tuloksena todettiin myös tekniikoiden ja työkalujen testauksen tärkeys yleisesti tärkeäksi asiaksi nopeasti kehittyvällä ohjelmistoalalla.

Luottamuksellinen aineisto on poistettu liitteinä olevista lähdekooditiedostoista sekä tekstin mukana esitetyistä lähdekoodileikkeistä.

Asiasanat: elastalert, monitorointi, hälyttäminen

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software engineering

PUDAS, LAURI:
Automatic monitoring of system logs and metrics
Testing and deploying Elastalert tool

Bachelor's thesis 52 pages, appendices 8 pages
May 2018

In this thesis, software and system monitoring tool Elastalert was tested. The aim on the thesis was to find out Elastalert's ability to work as part of distributed system monitoring. At the same time, Elastalert was compared with older existing monitoring and alerting system. The use of installation automation and configuration with Elastalert was also studied. The thesis also introduces the theory of software monitoring and alerting basics.

Elastalert was found to be a functional monitoring tool to work as part of the Elastic-Stack. Elastalert's ease-of-use and simplicity were the strengths of the tool. However, some features lacked from Elastalert which were obstacles with challenging distributed system in the project. The Elastalert tool was therefore not used for production immediately after the test period.

Testing Elastalert started a good discussion within the project. In addition, the importance and complexity of the monitoring and alert handling of the target system were noted. After the test period, it was found that there is a need to test several other control systems to find the right system for more effective monitoring and alerting. Because of this work, the importance of testing techniques and tools were also recognized as a matter of major importance in the rapidly evolving software industry.

Confidential data has been removed from the enclosed source code files and from the source code snippets that are included with the text.

Key words: elastalert, monitoring, alerting

SISÄLLYS

1 JOHDANTO ... 6
2 HAJAUTETUN JÄRJESTELMÄN MONITOROINTI ... 7
3 JÄRJESTELMÄN AUTOMAATTINEN VALVONTA ... 11
3.1 Hälyttäminen ongelmatilanteessa ... 11
3.2 Tilaperusteinen vs. tilaton hälytys ... 12
4 ELASTIC-TUOTEPERHE ... 15
5 ELASTALERT-TYÖKALUN TESTAUS ... 17
5.1 Asennusautomaatio ... 17
5.1.1 Tavoite ... 17
5.1.2 Docker ... 18
5.1.3 Ansible ... 22
5.1.4 Jenkins ... 26
5.2 Elastalert ... 29
5.2.1 Elastalert työkaluna ... 29
5.2.2 Toiminta ... 31
5.2.3 Säännöt ... 35
5.2.4 Hälyttäminen ... 38
6 POHDINTA ... 42
LÄHTEET ... 44
LIITTEET ... 45
Liite 1. Elastalert Ansible-roolin lähdetiedostot ... 45
Liite 2. Elastalert Docker lähdetiedostot ... 50
Liite 3. Elastalert Jenkins lähdetiedosto ... 52

LYHENTEET JA TERMIT

PoC	Proof of Concept. Demovaihe, jossa täyden implementaation sijaan tehdään kevyempi testaus kyseiselle tekniikalle tai tuotteelle. PoC:in tarkoitus on antaa tärkeää informaatiota testatun asian toimivuudesta sekä käytettävyydestä.
repository, repo	Ohjelmistojen, tiedostojen tai pakettien tallennuspaikka, josta tiedostot tai paketit voidaan hakea käyttöä ja asennusta varten. Versionhallintajärjestelmistä käytetään yleensä myös nimitystä repo.
Artifactory	Artifactory on JFrogin kehittämä erityisesti pakettien hallintaan keskittynyt tallennusjärjestelmä eli repo.
CI/CD	Continuous Integration, jatkuva integraatio. Ohjelmistojen ominaisuuksien tai osien tuominen osaksi tuotetta nopealla syklillä. Continuous Delivery, jatkuva toimitus. Ohjelmistojen nopea ja automatisoitu toimittaminen käyttöön.
NOC	Network Operations Center. Tilannehuone, jossa valvotaan järjestelmien ja verkkoyhteyksien tiloja ja hallitaan niistä tulevia hälytyksiä.
API	Application Programming Interface. Rajapinta, jonka kautta ohjelmat, laitteet ja järjestelmät voivat keskustella toisilleen.

1 JOHDANTO

Korkean saatavuuden ohjelmistojen tai järjestelmien tilan monitorointi on erittäin tärkeää. Monitoroinnilla halutaan varmistaa ohjelmiston vakaa toiminta kaikissa tilanteissa. Monitorointia on hyvä tehdä manuaalisesti ja tutkia ohjelmiston käyttäytymistä erityisissä tilanteissa, esimerkiksi korkean kuorman alla. Täydellistä jatkuvaa monitorointia on kuitenkin mahdoton tehdä manuaalisesti. Automaattiseen monitorointiin perustuvat hälytykset tarjoavat ohjelmiston kehittäjille ja käyttäjille tärkeää informaatiota ohjelmiston tilasta silloin, kun jokin menee pieleen. Monitorointi ja hälyttäminen tulisi implementoida kaikkiin sitä vaativiin ohjelmistoihin erityistä tarkkuutta käyttäen.

Tämän opinnäytetyön aiheena oli kokeilla ja tehdä Proof of Concept Elastalert-työkalulle. Elastalert on kevyt avoimen lähdekoodin hälytystyökalu, jonka on kehittänyt alun perin omiin tarpeisiinsa yhdysvaltalainen yritys Yelp. Motivaation Elastalertin testaukseen muodosti erityisesti kiinnostus ja tarve siirtyä käyttämään Elastic-tuoteperheen tuotteita osana tuotteen monitorointia ja valvontaa. Elastic-tuotteilla saatiin muodostettua tehokkaasti kattavaa datan visualisointia sekä joustavaa datan keräystä erilaisista lähteistä, kuten ohjelmiston metriikasta, lokeista ja tiladatasta ajoympäristössä.

Opinnäytetyön alkupuolella esitellään hajautetun järjestelmän monitoroinnin sekä automaattisen valvonnan perusteita. Lisäksi pohditaan erilaisia keinoja tuottaa hälytyksiä tehokkaasti. Työn loppuosassa esitellään Elastalertille luotu asennus- ja käyttöönottoautomaatio ja siihen liittyvien tekniikoiden osakokonaisuudet. Asennusautomaatiossa käytetyt tekniikat ovat Ansible, Docker ja Jenkins. Lopuksi käydään läpi Elastalertin toimintaperiaatteet, sääntötiedostojen perusteet sekä hälyttämiseen liittyvät asiat. Elastalertin toiminnallisuuksia esitellään kappaleissa esimerkkien kautta.

Opinnäytetyön tavoitteena oli tutkia ja testata Elastalert-työkalun sopivuutta osana hajautetun järjestelmän monitorointia ja valvontaa. Lisäksi haluttiin tutkia Elastalertin helppokäyttöisyyttä ja asennusautomaation toteutusta erilaisten ympäristöjen kanssa. Työkalun toiminnallisuuksia ja ominaisuuksia haluttiin myös verrata nykyisen käytössä olevan hieman vanhemman monitorointi- ja hälytysinfrastruktuurin kanssa. Tulevaisuuden tavoite projektissa on löytää sopivin järjestelmä tuotteen kokonaisvaltaiseen valvontaan.

2 HAJAUTETUN JÄRJESTELMÄN MONITOROINTI

Luotettavan hajautetun järjestelmän ylläpitämiseksi on järjestelmään integroitava tai rakennettava riittävän kattava monitorointi-infrastruktuuri. Järjestelmän tilasta on hyvä saada yleinen tilannekuva ja tarvittaessa yksityiskohtaista dataa vian sattuessa. Ilman kattavaa monitorointia järjestelmän tilasta ilmenneisiin ongelmiin ei voida reagoida tehokkaasti. Ongelmien tunnistaminen ennen käyttäjille näkyvää oiretta on myös huomattavan hankalaa ilman oikeanlaista metriikkaa ja lokiviesteistä saatua dataa.

Monitorointiin ja siihen liittyviin termeihin ei ole täysin yksiselitteistä yhteistä linjausta mitä kukin tarkoittaa. Jopa alan pioneerina tunnetun Googlen sisällä samat termit voivat tarkoittaa hieman eri asioita. Yleisimmät tulkinnat ovat kuitenkin seuraavia:

- Monitorointi
    - Järjestelmän datan keräystä, prosessointia, yhdistelyä ja esittämistä reaaliajassa. Tämä voi olla esimerkiksi kutsumääriä, virhemääriä, prosessointiaikoja tai palvelinten tilatietoa.
- Hälytys
    - Ilmoitus, joka tulee hälytysjärjestelmän kautta tietoon. Hälytys on tarkoitettu ihmisen luettavaksi ja käsiteltäväksi.
- Näkymä (Dashboard)
    - Applikaatio tai verkkopalvelu, joka tarjoaa kokonaiskuvan järjestelmän tärkeimmistä metriikoista. Näkymä voi metriikoiden lisäksi tarjota kehitystiimille informaatiota bugi- tai kehitystilanteesta.
- Juurisyy
    - Näkyvät oireet aiheuttanut ongelma ohjelmakoodissa tai järjestelmään liittyvän ihmisen toiminnassa. Korjaamalla ongelma voidaan sanoa suurella varmuudella, ettei ongelma toistu enää uudelleen samanlaisena. Ongelmalla voi olla useita juurisyitä ja nämä on hyvä erottaa toisistaan. (Site reliability engineering 2016, 55-56.)

Erilaisilla järjestelmillä on erilaisia tarpeita ja tavoitteita tuottamalleen monitoroinnille. Järjestelmästä olisi hyvä tunnistaa esimerkiksi sellaisia tilanteita, joissa jokin on hajonnut tai hajoamassa pian. Järjestelmäkohtaisia monitorointivaatimuksia löydetään kattavan

testauksen kautta sekä tutkimalla järjestelmälle asetettuja vaatimuksia. Monitorointi- ja valvontajärjestelmän tulee myös vastata järjestelmälle asetettuihin vaatimuksiin ja arkkitehtuuriratkaisuihin.

Yleisiä syitä laadukkaan monitoroinnin tekoon on monia. O'Reillyn kirjassa Site reliability engineering (2016) on kuvattu useita syitä tehdä monitorointia järjestelmän koosta ja tarkoituksesta riippumatta:

- Pitkä-aikaisen kehityksen seurannalla voidaan varautua kasvaviin käyttäjämääriin tai nousseisiin palvelinvaatimuksiin.
- Hälyttämällä ongelmatilanteissa saadaan ongelma tietoon nopeasti ja korjaustoimet aloitettua mahdollisimman pian.
- Oikeanlaisten näkymien avulla saadaan järjestelmän tilannekuva haltuun kootusti.
- Vertailevalla monitoroinnilla voidaan huomata, jos jokin palvelu on esimerkiksi hidastunut viimeviikosta huomattavasti.
- Ongelmatilanteissa kattava monitorointi tarjoaa myös mahdollisuuden tutkia, mitä muuta järjestelmässä tapahtui, kun jokin tietty komponentti aiheutti ongelman. (Site reliability engineering 2016, 55-56.)

Bugien ja ongelmien ratkominen tehokkaasti vaikeutuu myös kehitysvaiheessa huomattavasti ilman kattavaa monitorointia. Mahdollisuus palata tutkimaan vasteaikoja ja palvelun kutsujonoja bugitiketin yhteydessä tuo tarvittavaa lisädataa, jotta juuri oikea korjaus osataan tehdä ohjelmakoodiin. Kehitysvaiheen aikana voi olla hyödyllistä kerätä suuria määriä dataa järjestelmästä, jotta voidaan tutkia ja tunnistaa yksityiskohtaisesti kaikki halutut piirteet järjestelmästä testijakson ajalta.

Tuotantovaiheessa järjestelmän eri osioita pitäisi monitoroida eri tarkkuuksilla järjestelmälle vaaditun käyttökohteen mukaan. Esimerkiksi kovalevytilan tarkistaminen minuutin välein järjestelmässä, joka on suunniteltu pitkäaikaiseen käyttöön ja korkeaan saatavuuteen, on todennäköisesti turhan usein. Toisaalta minuutin välein tarkistettu prosessorikuorma jättää varmasti useita pitkiäkin piikkikuormia täysin huomaamatta. Oikean tarkkuuden valitseminen mittareille tulisi tehdä huolella ja perustellen. Lyhyen aikavälin monitorointi on hyvin raskasta ja vie paljon tilaa sekä resursseja. Pitkän aikavälin monitorointi taas jättää näyttämättä tärkeitä datapisteitä ja kiinnostavin informaatio voi jäädä täysin näkemättä. (Site reliability engineering 2016, 62.)

Eräs hyvä tapa kerätä riittävän tarkkaa metriikkaa on toteuttaa järjestelmän omaan ohjelmakoodiin datan aggregointia. Prosessorikuormaa voidaan tutkia sekunnin välein ja muodostaa näistä minuutin välein keskiarvo. Tämä keskiarvo haetaan monitorointijärjestelmään minuutin välein. Näin ollen saadaan riittävän kattavaa dataa, joka sisältää informaatiota myös pitkistä piikkikuormista kuluneen minuutin aikajaksolta. (Site reliability engineering 2016, 62.)

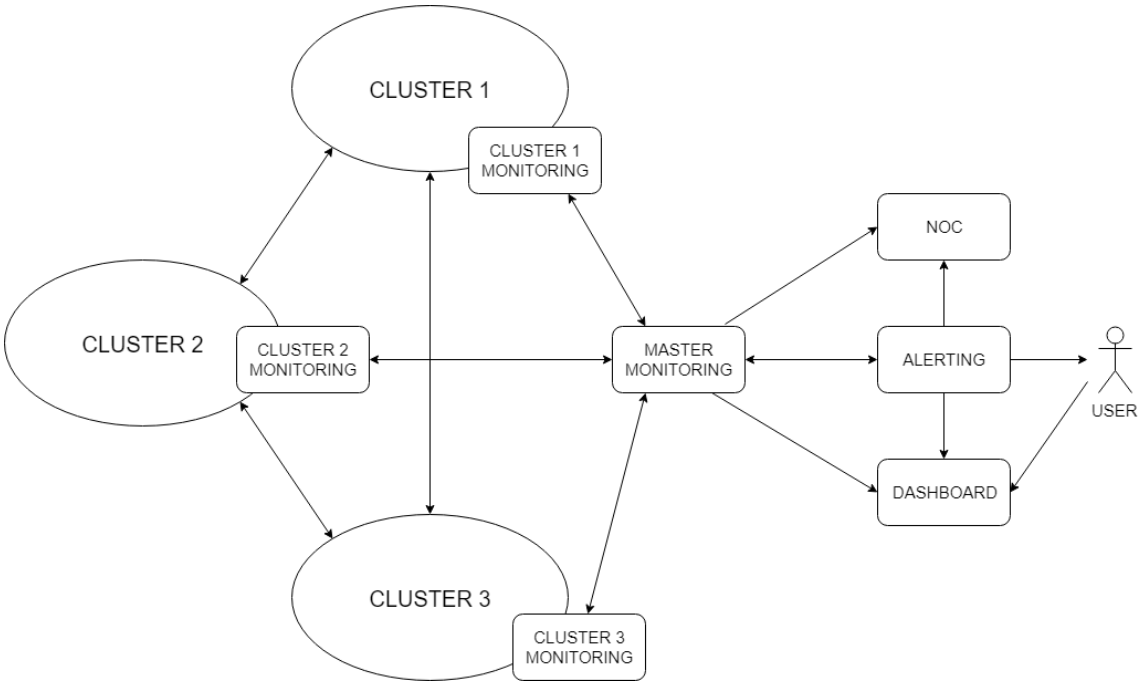
Monitoroinnille on määritetty neljän kultaisen mittarin lista. Listalla esitetyt asiat ovat yleisesti ottaen kaikista tärkeimmät asiat järjestelmän monitoroinnissa. Jos monitoroinnissa ilmenee esimerkiksi resurssipulaa tai muu este monitoroinnin kehittämiseksi halutulle tasolle, kannattaa keskittyä aluksi näiden neljän parametrin mittaamiseen:

- Latenssi
    - Aika, jonka tietty palvelu tai kutsu kestää. Epäonnistuneet kutsut täytyy ymmärtää tarvittaessa suodattaa pois, koska nämä tapahtuvat yleensä hyvin nopeasti. Kääntöpuolena taas kauan kestävä epäonnistunut kutsu on vielä pahempi kuin nopea epäonnistunut kutsu.
- Ruuhka
    - Mittari, joka kertoo, kuinka paljon kysyntää järjestelmälle on korkealla tasolla asetettu. Järjestelmästä riippuen tämä voi olla esimerkiksi http-kutsua sekunnissa.
- Virhetilanteet
    - Virhetilanteiden, poikkeuksien ja epäonnistuneiden kutsujen määrä. Virhetilanteiden kasvava tai korkea määrä indikoi useasti järjestelmässä esiintyvästä ongelmasta.
- Kyllästyneisyys (Saturation)
    - Mittari joka kuvaa kuinka ”täynnä” järjestelmä on korostaen järjestelmän rajoittavia resursseja. Monimutkaisissa järjestelmissä kannattaa pyrkiä tarjoamaan korkean tason kuormitustilannetta, esimerkiksi kestääkö järjestelmä tämän hetkisen kuorman kaksinkertaisena. (Site reliability engineering 2016, 60-61.)

Hajautetun järjestelmän monitorointivaatimukset kasvavat järjestelmän koon ja vaatimustason mukana nopeasti. Pienen järjestelmän monitorointiin riittää yksi keskitetty mo-

nitorointi-instanssi. Jos monitoroinnille on asetettu tiettyjä korkean saatavuuden vaatimuksia, voidaan yksittäinen instanssi kahdentaa, mutta edelleen suorittaa monitorointia keskitetysti.

Suuremmissa järjestelmissä tulee monitorointia jakaa luotettavuuden, saatavuuden ja suorituskyvyn takia useampiin instansseihin. Useat modernit monitorointijärjestelmät kykenevät tuottamaan yksinkertaisten konfiguraatiomuutosten jälkeen koko järjestelmän kattavaa monitorointia, jossa on useita monitorointi-instansseja. Kuviossa 1 on esitetty kuvitteellisen järjestelmän eräs mahdollinen monitorointi-infrastruktuurirakenne. Monitorointidata on saatavilla laajassakin järjestelmässä keskitetyltä palvelulta, joka kommunikoi muiden monitorointipalvelujen kanssa.



KUVIO 1. Hajautetun järjestelmän monitorointi.

Kuviossa 1 esiintyvä NOC tarkoittaa Network Operation Centeriä eli tilannehuonetta. NOCissa valvotaan järjestelmiä ja verkkoyhteyksiä korkean tason näkyvyydellä. Tämä tarkoittaa, että tilannehuone harvoin tarkastelee tietyn järjestelmän spesifisiä metriikoita. Sen sijaan NOCissa keskitytään hälytysten ja kokonaiskuvan hallintaan. Jos esimerkiksi NOC saa hälytyksen erään järjestelmän verkkoyhteyden toimimattomuudesta, voi sama ongelma pian olla toisessakin järjestelmässä. NOCia operoi yleensä palveluntarjoaja, jonka verkossa tai konesaleissa järjestelmät toimivat. Järjestelmistä voidaan lähettää monitorointidataa, tilannekuvaa ja hälytyksiä NOCiin.

3 JÄRJESTELMÄN AUTOMAATTINEN VALVONTA

3.1 Hälyttäminen ongelmatilanteessa

Monitoroinnin ja hälyttämisen avulla saamme tietää, kun järjestelmässä on jokin vialla. Hyvin rakennettu automaattinen valvonta voi myös kertoa etukäteen, jos jokin on hajoamassa pian. Jos järjestelmä ei voi toipua itse ongelmatilanteesta, tarvitaan ihminen tutkimaan saatua hälytystä. Hälytyksen sisältö tarvitsee analysoida ja mahdollinen juurisyy tunnistaa. Joissain tapauksissa hälytys voi olla myös aiheeton. Tällöin on syytä evaluoida ja tarkastaa hälytyssääntöjen oikeellisuus. (Site reliability engineering 2016, 57.)

Hälytysten rakentaminen on monille monitorointi-infrastruktuurin käyttöönottaneille yrityksille tai projekteille itsestäänselvyys. On kuitenkin mahdollista, että hälytyssääntöjä tehdään huolimattomasti. Hälytyssääntöjä voi olla esimerkiksi kaikille yksittäisille mittareille monitoroinnissa, jolloin valvonta voi hukkua hälytyksiin. Tällöin myös tärkeät hälytykset voivat kadota massaan ja sääntöjen ylläpito vaikeutuu huomattavasti. Toisaalta taas hälytyssääntöjä voi olla myös liian vähän, jolloin järjestelmän tilasta ei monitoroida riittävää määrää dataa. Hälytyssääntöjen laatimiseen ja testaamiseen tulisi käyttää riittävä määrä aikaa ja resursseja. Nämä asiat korostuvat entisestään sitä mukaan, mitä korkeammat saatavuusvaatimukset järjestelmällä on.

Kuten ohjelmistoissa ja järjestelmissä, voi valvontainfrastruktuuristakin tulla liian monimutkainen sekä vaikea ylläpitää ja päivittää. Eräs hyvä motto valvontaa suunnitellessa voisi olla ”As simple as possible, no simpler” (Site reliability engineering 2016, 62). Tällä tarkoitetaan ajatusmallia, joka ohjaa kehittäjiä luomaan valvontajärjestelmää yksinkertaiseen ja luotettavaan suuntaan. Hälytyssääntöjen, joilla tullaan saamaan tai oletetaan saatavan kiinni kaikista vakavimmat ongelmat, tulisi olla mahdollisimman yksinkertaisia ja luotettavia. Monitoroitua dataa, jota ei käytetä hälytysten muodostamiseen tai muuhunkaan valvontaan, tulisi silloin poistaa monitoroinnista. Jos poistoa ei suoriteta, tulisi vähintäänkin arvioida onko data tällöin varmasti hyödyksi valvonnan toteuttamiselle. (Site reliability engineering 2016, 63.)

Monet modernit valvontajärjestelmät tukevat useiden eri tärkeysasteiden hälytyksiä ja varoituksia. Valvontajärjestelmään voidaan luoda korkean tason kriittisiä hälytyksiä, vähemmän kriittisiä hälytyksiä ja näiden lisäksi matalan prioriteetin varoituksia. Tämä antaa hyvin vaihtoehtoja toteuttaa valvontajärjestelmä ajatellen kaikkia ylläpitoon osallistuvia osapuolia. Samalla monikerroksinen hälytysinfrastruktuuri voi muuttaa järjestelmän liian monimutkaiseksi. Lisäksi liian herkästi aiheutuvat hälytykset muuttuvat pian harmaaksi massaksi valvojien silmissä ja oikeasti kriittisiä hälytyksiä voi olla vaikea tunnistaa matalamman prioriteetin hälytyksistä ja varoituksista.

Pienemmissä järjestelmissä saatetaan hälytykset lähettää suoraan työntekijöille eikä esimerkiksi keskitettyyn NOCiin. Näissä järjestelmissä tulee hälytysten lähetyksen tärkeyttä pohtia erityisen tarkasti. Hälytyksen lähettäminen ihmisen luettavaksi on melko kallista työaikaa. Saapuva hälytys keskeyttää aina työntekijän työnkulun. Työntekijän ollessa kotona, hälytys kuluttaa työntekijän vapaa-aikaa ja pahimmassa tapauksessa keskeyttää työntekijän yöunen. Jos hälytyksiä saapuu usein, alkaa työntekijä turtumaan hälytyksiin ja huonoimmassa tapauksessa jopa jättää ne kokonaan huomioimatta. Tällöin oikeasti tärkeät hälytykset jäävät hyvin todennäköisesti vähäiselle huomiolle. Tehokkaat hälytysjärjestelmät omaavat aina hyvän tavan antaa vain ja ainoastaan tärkeitä hälytyksiä. Turhat sekä liialliseksi massaksi muodostuvat hälytykset ja varoitukset eivät ole osa tehokasta hälytysjärjestelmää. (Site reliability engineering 2016, 57.)

3.2 Tilaperusteinen vs. tilaton hälytys

Hälytyksen ollessa yksittäinen ilmoitus järjestelmässä olevasta ongelmasta, voi hälytyksellä ajatella olevan myös tilatieto. Hälytyksellä on yksinkertaisimmillaan kaksi tilaa; päällä ja pois päältä. Kokonaisvaltaiset valvontajärjestelmät tarjoavat mahdollisuuden nähdä hälytyksen lähettämisajankohdan lisäksi, kuinka kauan hälytys oli ollut päällä sekä milloin hälytys hävisi eli muuttui tilaan pois päältä. Yksinkertaiset hälytysjärjestelmät ja hälytyksien muodostamiseen tarkoitetut lisäosat tarjoavat monesti vain itse hälytyksen, muttei helppoa tapaa seurata hälytyksen aktiivista päällä oloaikaa.

Englannin kielen sanat *alert* ja *alarm* kuvaavat hyvin tilaperusteisen ja tilattoman hälytyksen eroja. Suomen kielessä molemmat tarkoittavat karkeasti samaa eli hälytystä. Alert

ymmärretään kuitenkin laajalti kerran tapahtuvaksi hälytykseksi. Tämä voisi olla esimerkiksi toisen henkilön varoittamista lähestyvästä vaarasta. Alarm sanan merkitys voidaan taas mieltää enemmän pidempiaikaiseksi hälytykseksi. Pitkäaikainen hälytys voisi olla esimerkiksi hälyttävä palohälytin, joka lopettaa vasta, kun palo on sammutettu.

Mittareiden ja arvojen, joita on helppo mitata ja monitoroida, valvonnasta kannattaa muodostaa tilaperusteinen hälytyslogiikka. Valvontajärjestelmä lähettää tällöin hälytyksen, kun raja-arvot tai muut monitoroidut arvot ylittyvät. Hälytys kuitenkin pysyy päällä ja indikoi erilaisilla näkymillä virhetilanteen päällä oloa, kunnes ongelma korjataan. Vaikeammin jatkuvasti monitoroitavana datan, kuten tietyn yksittäisen lokiviestin perusteella on vaikeaa muodostaa järkevää tilaperusteista hälytyslogiikkaa. Tällöin haitallisen lokirivin ilmestyessä lähetetään tilaton kertaluontoinen hälytys. (Wilson 2011.)

Tämän opinnäytetyön aiheena olleen Elastalert-työkalun kanssa huomattiin tarve tarjota tilatietoa hälytyksestä. Elastalert ei tue itsenäisesti hälytyksen tilan päivittämistä. Huomattiinkin, että käyttämällä vain Elastalertia on hälytykselle hyvin hankala tuottaa tila. Jos hälytyksiä lähetettäisiin minuutin välein, voisi valvoja tästä huomata, kun hälytykset loppuvat. Tämä tapa on kuitenkin hyvin huono, koska hälytysten määrä oli todella suuri ja niiden käsittely käytännössä mahdotonta. Jos taas hälytyksiä lähetettäisiin harvoin, vaikkapa 6 tunnin välein, ei valvoja hukkuisi hälytysvirtaan. Tällöin taas hälytyksen tilasta ei saisi riittävän tarkasti tietoa. Tilatieto pitäisi päivittyä vähintään muutaman minuutin välein, jotta hälytyksien reaaliaikaista tilaa voidaan tarkastella luotettavasti.

Toinen vaihtoehto olisi käyttää jotain korkeamman tason järjestelmää, joka keräisi minuutin välein Elastalertilta tulevia hälytyksiä. Järjestelmä päättelisi hälytysten loppumisen jälkeen hälytyksen tilan palanneen ennalleen. Tämäkin huomattiin huonoksi vaihtoehdoksi järjestelmän monimutkaisuuden ja ylimääräisten työkalujen integroinnin takia.

Kolmas vaihtoehto olisi tuottaa hälytystilamonitorointia Kibana-työkalulla. Kibanaan oli mahdollista luoda erilaisia graafeja Elastalertin indeksiä hyväksi käyttäen. Yleiskuvaa hälytysten tilasta voitiin analysoida graafisesti luomalla mittareita ja diagrammeja, jotka kuvasivat esimerkiksi hälytysten määrää. Myös tietyksi ajaksi vaimennetun hälytyksen tilatieto oli saatavilla indeksissä vaimennetuksi merkittynä kenttänä. Isoin ongelma Ki-

banan kanssa tuli vastaan, kun hälytysviestiin haluttiin tuoda mukaan suora linkki hälytyksen aiheuttaneeseen mittariin. Elastalertin luoma indeksi ei myöskään sisällä aina sopivia data-kenttiä helppoon hälytysdatan käsittelyyn.

Elastalert-työkalun kanssa päädyimme pitämään hälytykset tilattomina ja lähettämään hälytyksen ongelmatilanteen sattuessa. Hälytyksille määritettiin kohtalaisen pitkä aika jonka hälytykset odottavat, kunnes hälytys lähetetään uudestaan. Uudelleenlähetys tehdään, vaikka hälytys olisi ollut koko ajan päällä tai ilmestynyt useita kertoja edellisen hälytyksen lähettämisen jälkeen.

Tehokas valvontajärjestelmä pystyy tuottamaan yksinkertaisen hälytyksen, joka voidaan ajatella tilattomana hälytyksenä. Hälytysviestiin voidaan kuitenkin integroida esimerkiksi linkki suoraan sellaiseen näkymään, jossa hälytyksen aiheuttanut parametri on esitettynä. Näin valvoja saa helposti ja nopeasti paremman kokonaiskuvan sattuneesta tilanteesta. Useat modernit hälytys- ja monitorointijärjestelmät tarjoavat erilaisia ominaisuuksia, joilla voidaan yhdistää tilattoman ja tilaperusteisen hälytyksen hyviä puolia tehokkaasti.

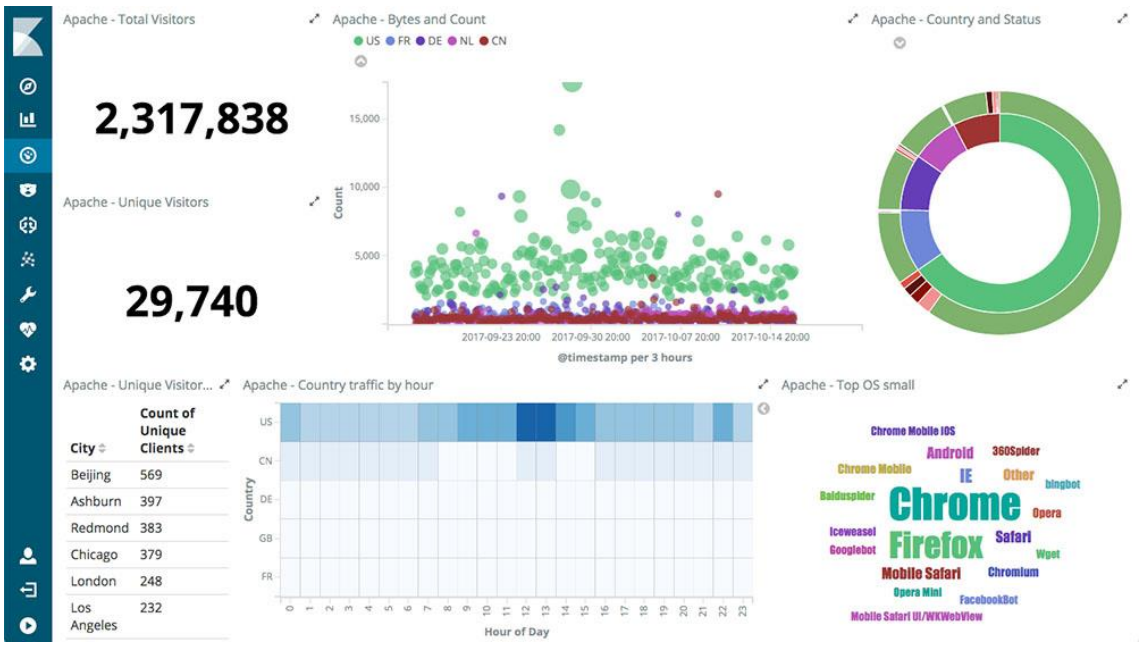
4 ELASTIC-TUOTEPERHE

Elastic-tuoteperheeseen kuuluu Elastic-yrityksen ylläpitämiä datan käsittelyyn, keräykseen ja analysointiin erikoistuneita tuotteita ja tuotteiden lisäosia. Tunnetuimmat tuotteet tuoteperheessä ovat Elasticsearch, Logstash ja Kibana. Nämä kolme tuotetta muodostavat nimen suositulle ELK-pinolle (Elastic Stack 2018). Elastic-tuoteperheen tuotteiden suosio on ollut kasvavaa ja tuotteet ovat saatavilla useissa pilvipalveluissa valmiiksi asennettuina. Datan säilöntään ja analysointiin tarkoitettu Elasticsearch on DB-Engines sivuston suosituimpien hakukoneiden listalla ylivoimaisesti ensimmäisenä (DB-Engines 2018).

Elastic on tuonut uusia avoimen lähdekoodin tuotteita saataville käyttäjien toiveiden mukaisesti viime vuosina perinteisen ELK-pinon lisäksi. Beats-tuotteet tulivat tärkeäksi osaksi Elasticin tuotevalikoimaa vuonna 2015. Uusien tuotteiden myötä ELK-pinon nimi ei enää vastannut täysin tarjontaa ja nimi päätettiin muuttaa Elastic-pinoksi. (Elastic Stack 2018.)

Elasticsearch on helposti skaalattavissa oleva avoimen lähdekoodin hakukone. Vaikka Elasticsearch oli aluksi vain vapaatekstihakukone, on se kehittynyt paljon kehityksen alkamisesta. Elasticsearchilla voidaan tehdä monimutkaisia analyysejä, hakuja ja datan aggregointia. Elasticsearch on suunniteltu skaalattavaksi useiden rinnakkaisten instanssien avulla, kun datan määrä ja muut vaatimukset alkavat kasvaa. (Andhavarapu 2017, 8.)

Elasticsearchin ollessa Elastic-pinon tärkeimpiä osia, ovat muut pinon työkalut myös keskeisessä roolissa kokonaisvaltaisen monitoroinnin tuottamisen kannalta. Logstash on palvelinpäässä toimiva työkalu, joka ottaa vastaan dataa erilaisista lähteistä yhtä aikaa ja lähettää datan Elasticsearchiin (Logstash 2018). Beats-tuotteet keräävät dataa ja lokitiedostojen sisältöä ympäristössä olevilta koneilta sekä palvelimilta ja lähettävät sitä keskitetysti Elasticsearchiin käsiteltäväksi (Beats 2018). Kibana on Elasticsearchin graafinen käyttöliittymä, jolla voidaan visualisoida mitä tahansa Elasticsearchiin kerättyä dataa tehokkaasti (Kibana 2018). Kuviossa 2 on esitetty esimerkki Kibana-työkalun käyttöliittymästä.



KUVIO 2. Kibana-työkalun käyttöliittymä (Kibana 2018)

Tämän opinnäytetyön keskeisenä osana oli myös tutustua tarkemmin Elastic-pinon toimintaan. Työssä testatun Elastalertin toiminta pohjautuu täysin Elasticsearchiin kerättyyn dataan ja datasta muodostettaviin hälytyksiin. Jotta hälytyssääntöjä voidaan muodostaa Elastalertille, täytyy kehittäjällä olla perusymmärrys siitä, minkälaista dataa Elasticsearchiin on kerätty ja minkälaisen rakenteen säilötty data omaa. Elastalertille hälytyssääntöjä muodostaessa selainpohjaisella Kibanalla voitiin helposti etsiä ja parsia Elasticsearchista löytynyttä dataa. Kibanalla voitiin myös tutkia Elasticsearchin indeksejä, joiden avulla voitiin päätellä minkälaisia kenttiä kukin Elasticsearchin json-dokumentti tulisi sisältämään. Näiden tietojen avulla hälytysten muodostaminen helpottui ja mitään ei tarvinnut jättää arvausten varaan. Lisäksi Kibanalla voitiin visualisoida Elastalertin indeksiä piirtämällä graafeja esimerkiksi sattuneiden hälytysten tilasta.

5 ELASTALERT-TYÖKALUN TESTAUS

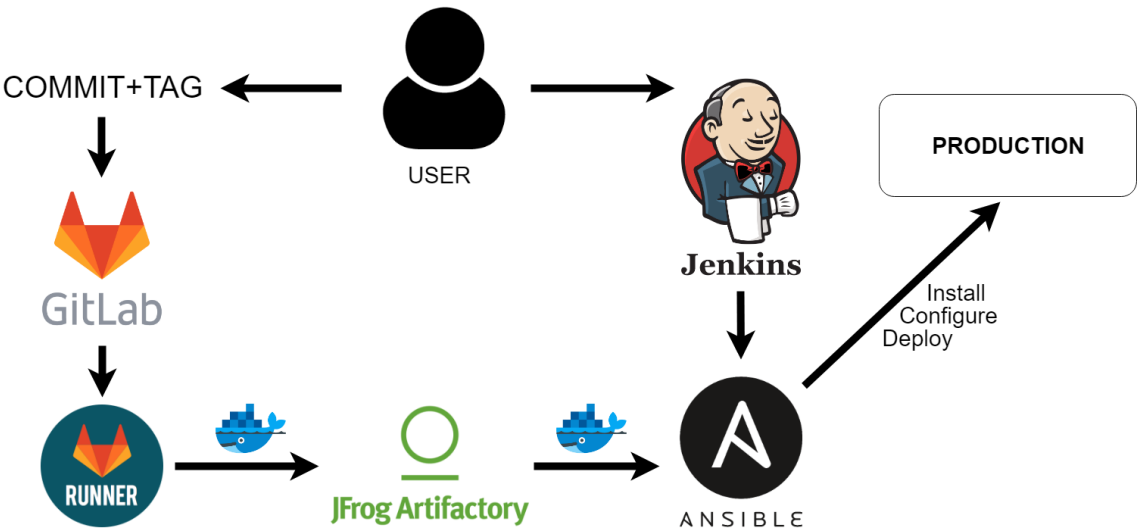
5.1 Asennusautomaatio

5.1.1 Tavoite

Elastalertille haluttiin toteuttaa helppo ja uudelleenkäytettävä asennusautomaatio. Testivaiheessa asennusautomaation vaatimukset olivat helppo integroitavuus nykyisiin asennusputkiin, riittävä konfiguroitavuus erilaisia ympäristöjä ja vaatimuksia varten sekä helppo jatkokehitettävyys. Näiden asioiden merkitys kasvaisi entisestään, jos testijakson jälkeen Elastalert haluttaisiin ottaa virallisesti käyttöön.

Käytettävät työkalut valikoituivat hyvin vahvasti sen perusteella, mitä työkaluja oli jo entuudestaan käytössä projektin sisällä. Itse Elastalert paketoitiin Docker-kontteja käyttäen. Asennus- ja konfigurointityökaluna toimi Ansible, jolla voitiin helposti täyttää monimutkaisia konfiguraatiotarpeita. Ylätason automaatioalustana toimi Jenkins, joka mahdollisti Elastalertin asennuksen, poiston ja konfiguroinnin integroitumisen olemassa oleviin automaatioputkiin.

Kuviossa 3 on esitetty toteutettu julkaisu-, asennus- ja käyttöönottoautomaatio, joka Elastalertille luotiin työn aikana. Kuvion vasemmalla puolella on julkaisuautomaatio kuvattuna korkealla tasolla. Kehittäjän julkaistessa uuden version Elastalertista versionhallintaan, rakennetaan uusi Elastalert Docker-image GitLab-runnerilla automaattisesti. Lopuksi rakennettu image siirretään Artifactoryyn käytettäväksi. GitLabissa tapahtuva automaattinen julkaisu oli luotu valmiiksi jo työtä tehdessä. Elastalertille luotu Docker-imagen rakentamisen tuki lisättiin täten jo olemassa olleeseen julkaisuputkeen. Kuvan oikean puolen osuus kattaa Jenkins-käyttöönottoautomaation sekä Ansiblella toteutetun asennusautomaation. Näistä molemmat toteutettiin työn aikana alusta loppuun Elastalertille ominaisina kokonaisuuksina. Käyttöönottoautomaatiossa Jenkins toimi käyttäjärajapintana, jonka avulla Elastalert oli helppo asentaa ja poistaa käyttäen Jenkinsin käyttöliittymää.



KUVIO 3. Elastalertin julkaisu-, asennus- ja käyttöönottoautomaatio.

5.1.2 Docker

Docker on kohtalaisen moderni, mutta huomattavan markkinaosuuden saanut virtualisointitekniikka. Docker tarjoaa käyttöjärjestelmästä riippumattoman alustan Docker-konteille, joihin voidaan paketoida haluttuja palveluja tai sovelluksia. Tämä mahdollistaa järjestelmien ajamisen ja kehittämisen riippumattomasti kohdealustasta. Yksittäistä Docker-konttia voidaan pitää kevennettynä virtuaalikoneena.

Järjestelmäasiantuntija Pitkänen (2016) tiivistää Dockerin pohjimmaisen ajatuksen hyvin vertaamalla Docker-kontteja rahtiliikenteen kuljetuskontteihin. Ne ovat helposti käsiteltäviä, pinottavia sekä siirrettäviä. Alustan (laiva) ja sovelluksen (rahti) välinen suhde on mahdollisimman riippumaton ja ennustettava. (Pitkänen 2016.)

Työssä testattu Elastalert toteutettiin paketoimalla sovellus Docker-konttiin. Näin saatiin luotua versioituva ja helposti uudelleen asennettava palvelu. Docker-kontti mahdollisti myös helpon kehitystyön sekä omalla työasemalla, että tuotannon kaltaisessa korkean datamäärän omaavissa ympäristöissä.

Docker-kontti valittiin pohjautumaan CentOS/7 käyttöjärjestelmään. Tämä mahdollistaisi tarvittaessa palvelun helpon siirtämisen Docker-kontista ajettavaksi suoraan palveluna virtuaalikoneella. Lisäksi projektin sisällä oli tehty Docker-imageja CentOSia käyttäen ja näitä varten luotu tarvittavat pohjakontit omaan Artifactoryssä olevaan Docker-repoon.

Oma repo Docker-imageille oli välttämättömyys, koska järjestelmän testausympäristöt toimivat ulkoverkosta erillään olevassa verkossa.
Docker-kontti käynnistetään ennalta määritetystä Docker-imagesta. Tässä työssä luotiin Elastalertia varten yksi Docker-image, jonka pohjalta itse Docker-kontit käynnistettiin. Docker-imagen rakentamiseen tarvitaan Dockerfile-tiedosto, joka sisältää kaikki tarvittavat ohjeet ja käskyt siitä, mitä paketteja, tiedostoja ja komentoja image tulee sisältämään. Kuviossa 4 on esitetty Elastalert Dockerfilen alkuosa, jossa määritellään tarvittavia muuttujia imagen rakentamiseen.

Elastalert Docker-imagen rakennukseen käytetty Dockerfile alkaa Docker-repon määrittelyllä, joka sijaitsee projektin Artifactoryssä. Seuraavaksi määritellään, minkä imagen pohjalle Elastalert-image halutaan rakentaa. Tässä tapauksessa pohjaksi valittiin CentOS/7 pohjautuva image. Lopuksi määritellään tiedoston ylläpitäjä. Tähän voi lisätä tarvittaessa tarkempia tietoja, kuten sähköpostiosoitteen tai yrityksen nimen.

```
ARG ARTIFACTORY_REGISTRY
FROM ${ARTIFACTORY_REGISTRY}/path/to/centos:7.4.1708
LABEL maintainer="Lauri Pudas"

# Timezone
ENV TZ=Europe/Helsinki
# Create working directory
RUN mkdir -p /opt
WORKDIR /opt
```

KUVIO 4. Elastalert Dockerfilen alkuosa.

Imagelle määritellään seuraavaksi aikavyöhyke. Aikavyöhyke määritellään kontille ympäristömuuttujaan, josta arvoa voidaan käyttää missä tahansa prosessissa kontin sisällä. Imageen luodaan /opt -kansio, jos sellaista ei ole vielä olemassa, ja siirrytään kyseiseen kansioon sisään. Tulevat komennot ajetaan tässä kansiossa ja komentoihin ei tarvitse lisätä absoluuttisia polkuja.

Kuviossa 5 on esitetty Elastalert Dockerfilen seuraava osuus, jossa ensimmäisenä Dockerfileen määritellään Yum-paketinhallintajärjestelmään liittyviä komentoja. Tarvittavat ylimääräiset pakettilähteet lisätään käyttöön ja päivitetään kaikki asennetut paketit uusimpiin versioihin. Tämän jälkeen asennetaan kaikki kontin toimintaan ja asennukseen tarvittavat paketit Yumia käyttäen. Asennuksen jälkeen on hyvä poistaa kaikki paketti-

lähteet sekä asennuksen ja päivityksen välimuistit. Näin luotavasta Docker-imagesta saadaan kevyempi. Seuraavaksi haetaan GitHubista etukäteen määritelty Elastalert zip-pakettina ja puretaan paketti. Tämän jälkeen ladattu paketti poistetaan viemästä levytilaa ja muutetaan puretun paketin pitkä nimi yksinkertaisemmin elastalertiksi. Lopuksi aikavyöhyke asetetaan aikaisemmin ympäristömuuttujaan määritetylle vyöhykkeelle.

```
# Install and update yum packages
# Fetch, unzip and rename elastalert --- v0.1.28 ---
# Set timezone
RUN yum -y install epel-release && \
  yum update -y && \
  yum -y install python-devel python2-pip wget unzip gcc && \
  yum -y remove epel-release && \
  yum clean all && \
  rm -rf /var/cache/yum && \
  wget -O elastalert-master.zip https://github.com/Yelp/elastalert/ar-
chive/4eae2e27bd95e278d50f54acfdf6ee756450a072.zip && \
  unzip elastalert*.zip && \
  rm elastalert*.zip && \
  mv elastalert* elastalert && \
  ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && \
  echo $TZ > /etc/timezone
```

KUVIO 5. Elastalert Dockerfilen keskiosa.

Kuviossa 6 on esitetty Elastalert Dockerfilen loppuosa, jossa ensimmäisenä kohdekansioksi vaihdetaan äskettäin ladattu ja purettu elastalert-kansio. Seuraavaksi Dockerfilessä määritellään itse Elastalertin asennus. Asennuksen ohjeet ja esimerkit ovat kuvattuna kattavasti Elastalertin dokumentaatiosivustolla. Pythonin paketinhallintatyökalun pip:in sekä setuptools-työkalun avulla voidaan Elastalert asentaa muutamaa komentoa käyttäen. Ladattu Elastalert-paketti sisältää myös requirements-tekstitiedoston, jonka sisältöä python osaa käyttää hyödykseen silloin, kun itse asennus suoritetaan. Tiedosto sisältää listan kaikista tarvittavista paketeista ja riippuvuuksista, joita Elastalert tarvitsee toimiakseen.

Dockerfilen lopuksi määritellään palvelulle sertifikaatit, jotta tämän testijakson puitteissa saatiin Slack/Mattermost -keskustelupalvelujen verkkoliikenne toimimaan ja palvelut luottamaan toisiinsa odotetulla tavalla. Aivan lopuksi määritetään imagelle entrypoint, joka kuvaa sitä komentoa, jonka käynnistetty Docker-kontti suorittaa aina. Elastalertin tapauksessa kutsutaan skriptiä, joka sisältää Elastalertin käynnistyksen lisäksi Elasticsearch-yhteyden testaamisen sekä indeksien luomisen Elasticsearchiin, jos sille on tarvetta.

```
# Install elastalert
WORKDIR /opt/elastalert
RUN pip install "setuptools>=11.3" && \
  python setup.py install && \
  pip --no-cache-dir install -e .
# Certificates
COPY certs/Your_ceritificate_file.crt /etc/ssl/certs/
ENV REQUESTS_CA_BUNDLE=/etc/ssl/certs/Your_ceritificate_file.crt

ENTRYPOINT ["sh", "/opt/elastalert-files/start-elastalert.sh"]
```

KUVIO 6. Elastalert Dockerfilen loppuosa.

Dockerfile-tiedostosta voidaan rakentaa ja julkaista Docker-image muutamilla eri tavoilla. Projektissa oli Docker-imagejen julkaisuun valmiina GitLab-infrastruktuuri, joten Elastalert-imagen rakentaminen ja julkaisu Artifactoryyn tehtiin tähän loogiseen paikkaan. Dockerfilen rakentamiseen käytetään Docker-compose -palvelua, joka saa syötteeksi yksinkertaisen määrittelyn rakennettavasta imagesta. Docker-compose -tiedostot ovat yaml-formaatissa. GitLabissa oleva automaatio julkaisee rakennetun imagen Docker-compose -tiedostossa määriteltyyn polkuun.

Elastalertin Docker-compose -tiedosto on esitetty kuviossa 7, jossa määritellään rakennettavat Docker-imaget. Tiedostossa context-muuttujalla on määritetty kansio tai polku, josta Elastalertin Dockerfile löytyy. Lisäksi rakennusvaiheessa annetaan muuttujana Artifactoryn osoite, jota voidaan hyödyntää Dockerfilen sisällä. Tässä tapauksessa Artifactorystä haluttiin hakea CentOS pohja-image, jonka päälle Elastalert Dockerfile pohjautuu. Lopuksi Elastalert-imagelle annetaan nimi, joka sisältää Docker-repon polun Artifactoryyn sekä Elastalertin versionumeron nimen perässä.

```
---
version: '3'

services:
  elastalert:
    build:
      context: elastalert
      args:
        ARTIFACTORY_REGISTRY: ${ARTIFACTORY_REGISTRY}
    image: ${ARTIFACTORY_REGISTRY}/PATH/TO/FOLDER/elastalert:0.1.28-2
```

KUVIO 7. Elastalert docker-compose -tiedosto.

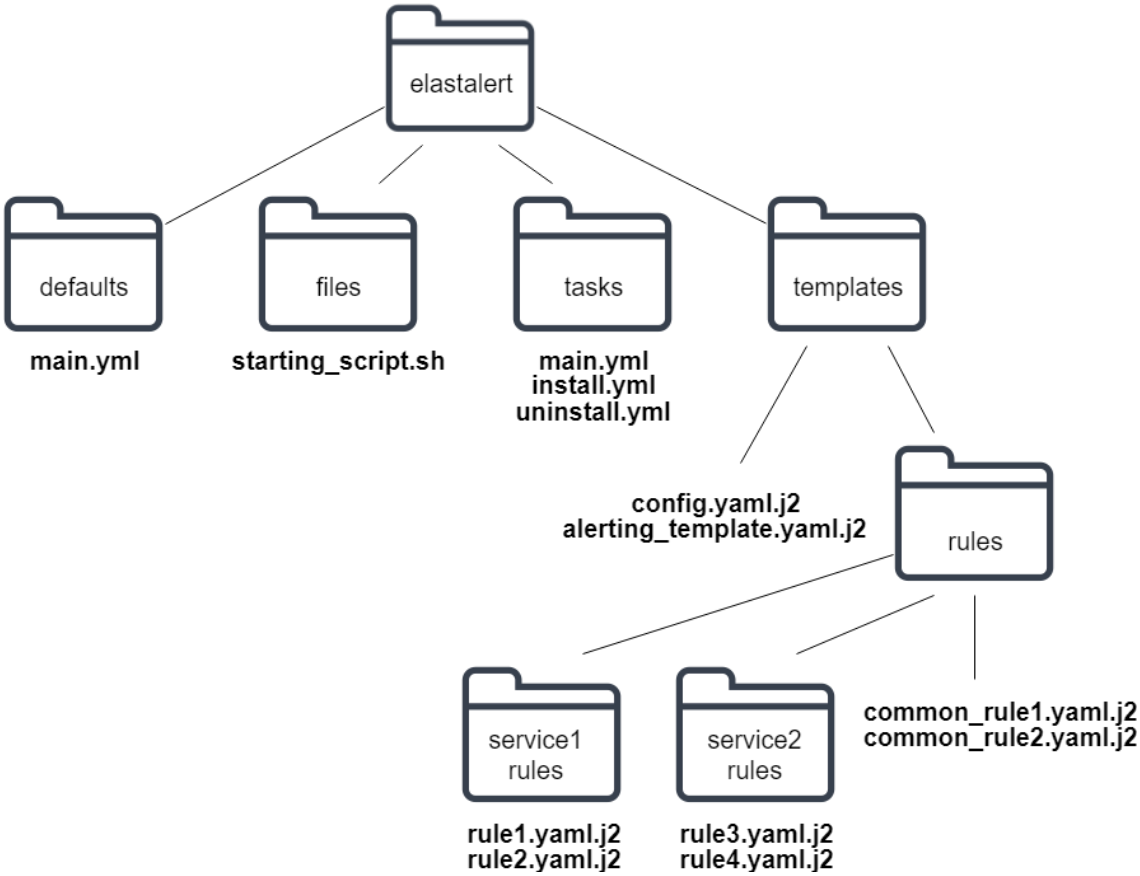
5.1.3 Ansible

Ansible on ohjelmistojen ja järjestelmien provisiointiin, konfigurointiin ja automatisointiin tarkoitettu yleiskäyttöinen työkalu. Ansiblea voidaan käyttää laajalla skaalalla pilvipalveluiden käyttöönotosta ohjelmistojen jakeluun ja konfigurointiin. Ansible on kirjoitettu pythonilla ja käyttää SSH-yhteyttä automaation tuottamiseen. Yksi Ansiblen vahvuuksista on vastaanottavan pään yksinkertaisuus. Ansible ei vaadi automatisoitavilta koneilta eli hosteilta muuta kuin SSH-yhteyden ja pythonin. (Hawkings 2017.)

Ansible valittiin automatisointityökaluksi, koska monien projektin osien automatisointi ja konfigurointi oli toteutettu käyttäen Ansiblea. Ansible tarjosi myös helposti kaikki Elastalertin automatisointiin tarvittavat osaset Docker-kontin automatisoinnista tiedostojen templatisointiin. Ansiblea käytettäessä Elastalert voitiin käynnistää minne tahansa, jossa Docker oli asennettuna, ilman erillistä kohdekoneiden konfigurointia tai muiden ohjelmistojen asennusta.

Tavoitteena Elastalertin automatisoinnissa oli luoda siitä riittävän yksinkertainen ottaen samalla huomioon testausjakson tavoitteet. Lisäksi automaation piti olla riittävän konfiguroitava, jotta erilaisten ympäristöjen ja vaatimusten ohella automaatio toimisi mahdollisimman hyvin. Elastalertille luotiin Ansible-rooli, johon rakennettiin kaikki tarvittava logiikka Elastalertin automaation tuottamiseen. Lisäksi rooliin sisällytettiin kaikki konfiguraatio- ja sääntötiedostot. Ansible-rooli mahdollisti Elastalertin asentamisen ja käytön erilaisissa konteksteissa muuttamalla vain muutamia tarvittavia ympäristökohtaisia parametrejä.

Kuviossa 8 on esitetty Elastalert Ansible-roolin kansiorakenne. Elastalert-roolin tarvitsemat tiedostot, templaatit, tehtävät sekä vakiomuuttujat ovat omissa kansioissaan. Testausvaiheessa sääntötiedostojen määrä oli vielä vähäinen ja niitä pystyi hallitsemaan roolin templaattikansiossa riittävän hyvin. Jos kuitenkin Elastalert otettaisiin täyteen tuotantokäyttöön, tulisi hälytyssääntöjä huomattavasti enemmän. Tällöin säännöt tulisi sijoittaa parempaan paikkaan versionhallinnassa ja sääntöjen kansiointi sekä ylläpito pitäisi olla riittävän yksiselitteistä. Tällaisessa tapauksessa voitaisiin sääntötemplaatit hakea jostain muusta kansiosta tai osoitteesta Ansiblella ja käyttää itse roolia edelleen samalla tavalla.



KUVIO 8. Ansible Elastalert -roolin kansiorakenne.

Elastalertin Ansible -roolin tärkeimpänä osana on tasks-kansion alta löytyvät roolin komennot. Komennot jaettiin kahteen erilliseen tiedostoon sekä oletustiedostoon. Näin ollen lisäominaisuuksia olisi voinut tarvittaessa implementoida helposti rooliin mukaan ilman, että tiedostokoot olisivat kasvaneet.

Kuviossa 9 on esitetty Elastalert-roolin oletustiedosto, joka sisältää asennus- ja poistokomentotiedostojen käytön roolia kutsuttaessa. Roolin käyttö toteutettiin käyttäen Ansiblen tageja. Elastalertin tapauksessa käyttämällä install-tagia ajetaan vain asennuksen ja konfiguroinnin tehtävät. Uninstall-tagia käyttämällä Elastalert poistetaan. Käyttämällä molempia tageja tai jättämällä tagit pois, Ansible suorittaa sekä poisto- että asennusautomaation.

```
# Uninstall -->
- include_tasks: uninstall.yml
  tags: [uninstall]

# Install -->
- include_tasks: install.yml
  tags: [install]
```

KUVIO 9. Elastalert Ansible -roolin tehtävien päätiedosto.

Kuviossa 10 on esitetty Elastalert Ansible -roolin poistoautomaatiotiedosto. Elastalertin poistaminen Ansiblella toteutettiin kahdessa osassa. Aluksi käytettiin Ansiblen docker_container-moduulia, jonka avulla poistettiin käynnissä oleva Elastalert-kontti. Kontit on nimetty käytetyn ajoympäristön mukaan, joten samalla koneella voidaan tarvittaessa ajaa useampien ympäristöjen Elastalert-instansseja. Lopuksi Elastalertin käyttämät konfiguraatio- ja sääntötiedostot poistetaan kohdekoneelta.

```
- name: Stop and remove elastalert container
  docker_container:
    name: elastalert-{{ base_environment }}
    state: absent

- name: Delete files and configs
  file:
    path: '{{ elastalert_mount_path }}'
    state: absent
```

KUVIO 10. Elastalert Ansible -roolin poistoautomaatiotiedosto.

Elastalertin asennusautomaation ensimmäinen vaihe on esitetty kuviossa 11. Asennuksessa täytyi ottaa huomioon erityisesti konfiguroinnin joustavuus sekä sääntö- ja konfiguraatiotemplaattien luominen dynaamisesti. Asennuksen ensimmäinen vaihe oli luoda kohdekoneelle tiedostorakenne, johon Elastalertin tarvitsemat tiedostot voitiin siirtää. Tiedostorakenteen luomiseen käytettiin templates-kansion rakennetta. Templaattikansiossa tiedostot ovat siinä järjestyksessä, missä ne tullaan siirtämään kohdekoneelle. Kansiorakenne voidaan näin ollen kopioida suoraan Elastalertin käyttöön.

Seuraavaksi kohdekoneelle siirretään konfiguraatiotiedosto, joka muutetaan samalla jinja2-templaattiformaatista yaml-formaattiin. Konfiguraatiotiedosto sisältää ympäristökohtaisia muuttujia, joten käyttämällä Ansiblen templatisointia voidaan samaa tiedostoa käyttää kaikkia kohdeympäristöjä vasten.

```
- name: Create folders for elastalert files if needed
  file:
    path: '{{ elastalert_mount_path }}/{{ item.path }}'
    state: directory
  with_filetree: 'templates/'
  when: item.state == 'directory'

- name: Move elastalert config to host
  template:
    src: config.yaml.j2
    dest: '{{ elastalert_mount_path }}/config.yaml'
```

KUVIO 11. Elastalert Ansible -roolin asennusautomaatiotiedoston alkuosa.

Kuviossa 12 on esitetty Elastalertin asennusautomaation keskivaihe. Elastalertin käyttämien sääntötemplaattien siirto kohdekoneelle suoritetaan lähes samalla tavalla kuin kansiorakenteen luonti aikaisemmin. Sääntötemplaatit sisältävän kansion sisältö käydään läpi ja muutetaan templatisoidut säännöt yaml-formaattiin, joita Elastalert käyttää. Tiedostopäätteen muuttaminen dynaamisesti on toteutettu Ansiblen Regular Expression -suodattimella, jolla jinja2-pääte voidaan helposti poistaa tiedostonimestä huolimatta. Elastalertin käynnistämiseen käytetty käynnistysskripti siirretään konfiguraatiotiedoston tapaan tiedostorakenteen juureen. Samalla skriptille annetaan suoritusoikeudet. Testivaiheessa suoritusoikeudet annettiin kaikille laitteen käyttäjille. Lopullisessa tuotantoversiossa suoritusoikeudet tulisi antaa vain Elastalertia käyttävälle kontille.

```
- name: Move and create elastalert rules from templates
  template:
    src: '{{ item.src }}'
    dest: '{{ elastalert_mount_path }}/rules/{{ item.path | regex_re-
place(".j2","") }}'
  with_filetree: 'templates/rules/'
  when: item.state == 'file'

- name: Move and set elastalert starting script to executable
  copy:
    src: start-elastalert.sh
    dest: '{{ elastalert_mount_path }}/'
    mode: a+x
```

KUVIO 12. Elastalert Ansible -roolin asennusautomaatiotiedoston keskiosa.

Elastalertin käynnistyksen viimeinen vaihe on käynnistää Docker-kontti Artifactoryssä olevasta imagesta. Tämä viimeinen vaihe on esitetty kuviossa 13. Käynnistyksen yhteydessä kontille annetaan ympäristöön viittaava nimi ja määritetään kontti käynnistymään.

Lisäksi kontti määrätään käynnistymään aina uudelleen, vaikka kontti olisikin tätä tehtävää ajaessa jo käynnissä. Näin varmistetaan, että Elastalert lataa aina mahdolliset konfiguraatiomuutokset. Kontille annetaan ympäristömuuttujina asetuksia, joita esimerkiksi Elastalertin konfiguraatio käyttää hyväkseen. Kohdekoneelle siirretyt tiedostot annetaan käyttöön kontille liitetyn levyosion kautta ja lopuksi määritellään kontti käynnistymään aina virhetilanteessa uudestaan. Asettamalla uudelleenkäynnistysparametri saadaan lisättyä huomattavasti kontin luotettavuutta virhetilanteissa.

```
- name: Start container from artifactory image
  docker_container:
    name: 'elastalert-{{ base_environment }}'
    image: '{{ artifactory_docker_image }}'
    state: started
    restart: yes
    env:
      CONF_FILE: '/opt/elastalert-files/config.yaml'
      ELASTIC_HOST: '{{ elasticsearch_ip }}'
      ELASTIC_PORT: '{{ elasticsearch_port }}'
      ELASTIC_INDEX: '{{ elasticsearch_write_index }}'
    volumes:
      - '{{ elastalert_mount_path }}:/opt/elastalert-files'
    restart_policy: on-failure
```

KUVIO 13. Elastalert Ansible -roolin asennusautomaatiotiedoston loppuosa.

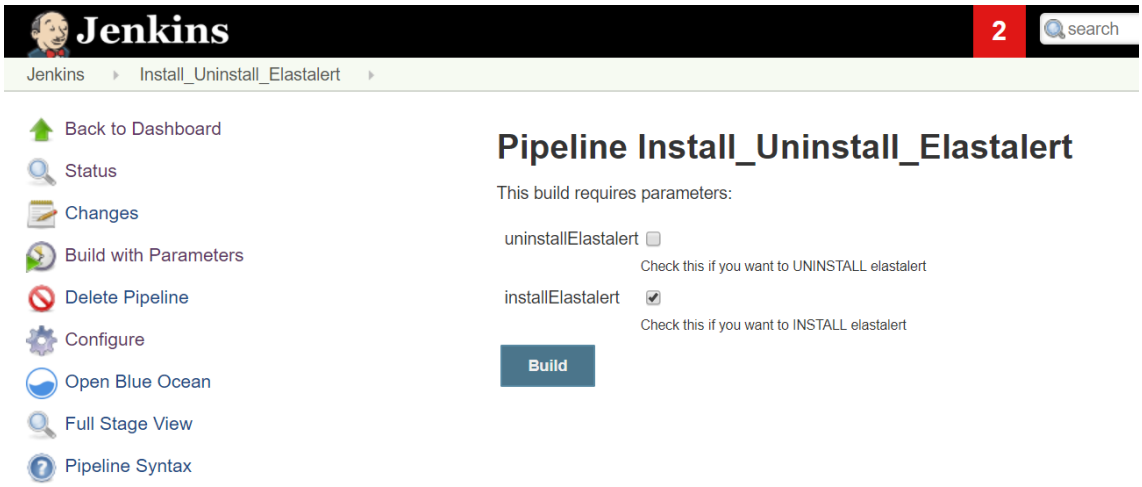
5.1.4 Jenkins

Jenkins on Javalla kirjoitettu avoimen lähdekoodin CI- ja CD-työkalu. Jenkinsiä voidaan käyttää kaikkien yleisimpien versionhallintatyökalujen ja ohjelmointikielten kanssa, mikä tekeekin Jenkinsistä yhden alan käytetyimmistä työkaluista. Yksi Jenkinsin vahvuuksista on yhteisön tarjoama tuki ja lisäosatarjonta. Jenkinsiä kehittää suuri kehittäjistä koostuva yhteisö, joka ylläpitää projektia hallitusti. (Giménez 2017.)

Elastalertin asennus-, konfigurointi- ja poistoautomaatio keskitettiin Jenkinsiin, jossa myös moni muu projektin automaatio sijaitsee. Tämä mahdollistaisi helpon integroinnin muun tuotteen kääntö-, rakennus- ja asennusvaiheisiin. Työssä suoritetussa testausvaiheessa Elastalertille luotiin yksinkertainen automaatioputki, jonka avulla käyttäjä voi poistaa sekä asentaa Elastalertin.

Jenkins-työkaluun voidaan muodostaa töitä (Job) graafisesti selaimesta määrittämällä tai täysin koodipohjaisesti. Tätä työtä tehdessä automaatio haluttiin tehdä mahdollisimman toistettavasti ja versioituvasti, joten se toteutettiin koodina. Jenkins käyttää töiden luomiseen omaa formaattiaan Jenkinsfileä. Jenkinsfile on Groovy-ohjelmointikieleen perustuva määritystiedosto. Tiedostoon määritellään, mitä parametrejä työ saa ennen suoritusta, kaikki suoritukseen liittyvät komennot ja vaiheet sekä kaikki työn tarvittavat muuttujat, tiedostot ja salatut avaimet.

Kuviossa 14 on esitetty Jenkins-työkaluun luotu asennus- ja poistoautomaatiotyö. Käyttäjä voi raksia haluamansa toiminnot Jenkinsistä ja Jenkins suorittaa kyseiset käskyt Ansiblella kohdekoneita vasten. Testausvaiheessa käytössä oli vain yksi korkean datamäärän ympäristö, joten Elastalertin Jenkins-automaatio jätettiin hyvin yksinkertaiselle tasolle. Jos Elastalert otettaisiin kokonaisvaltaiseen käyttöön, olisi Jenkinsiin hyvä lisätä kenttiä eri ympäristöjen ja konfiguraatioiden määrittämiseen.



KUVIO 14. Yksinkertainen Jenkins-automaatio Elastalertin asennukseen ja poistoon.

Jenkinsfilen alussa on määritelty työlle asetettujen parametrien tyypit, nimet ja kuvaukset. Elastalertin tapauksessa käytettiin vain kahta valintalaatikkoa testivaiheen yksinkertaisuuden takia. Jenkinsfilen alkuosa on esitetty kuviossa 15.

```
properties([
  parameters([
    booleanParam(name: 'uninstallElastalert', defaultValue: false, de-
scription: 'Check this if you want to UNINSTALL elastalert'),
    booleanParam(name: 'installElastalert', defaultValue: true, descrip-
tion: 'Check this if you want to INSTALL elastalert')
  ])
])
```

KUVIO 15. Jenkinsfilen alkuosa.

Seuraavaksi Jenkinsfileen määritellään suoritettavan työn ohjeet. Jenkinsfilen keskiosa on esitetty kuviossa 16. Node-funktio viittaa Jenkinsin käyttämään kuormanjakotekniikkaan. Jenkins valitsee itse sopivan noden, jolla annettu koodi suoritetaan. Nodeja voi olla käytössä useita ja niillä voi olla useita suorituspisteitä, joihin Jenkins-töitä jaetaan automaattisesti. Seuraavaksi tarkistetaan käyttäjän syöttämät parametrit ja muodostetaan sopivat tagit Ansiblea varten. Jos työ ajettiin ilman parametrejä, annetaan käyttäjälle virheilmoitus ja lopetetaan suoritus.

```
node() {

  // Verify that at least one option is selected before continuing
  def installTags = ""
  if (!params.installElastalert && !params.uninstallElastalert) {
    error message: """
        You must specify at least one option.
        Uninstall and install can be defined at same time.
        If both options are defined uninstall will be executed first.
        """
  } else if (params.installElastalert && !params.uninstallElastalert) {
    installTags = "install"
  } else if (!params.installElastalert && params.uninstallElastalert) {
    installTags = "uninstall"
  } else {
    installTags = "install,uninstall"
  }
```

KUVIO 16. Jenkinsfilen keskiosa.

Lopuksi Jenkinsfilessä kloonataan repositorio, jossa Elastalert-rooli sekä Ansiblen vaatimat inventaario- ja muuttujatiedostot ovat versioituna. Tiedostot siirtyvät käyttöön Jenkins-työtilan juureen, josta niitä voidaan käyttää seuraavissa vaiheissa. Viimeisenä vaiheena on itse Ansiblen suoritus Jenkinsin avulla. Ansiblelle annetaan tarvittavat parametrit ja playbook, jossa Elastalert-roolin ajo on määritetty sekä käyttäjän Jenkinsille antamat parametrit. Jenkins käyttää hyväkseen etukäteen luotuja builderStage- ja ansibleStage-

funktioita. Funktiot ovat yleiskäyttöisiä ja niitä voidaan käyttää kaikissa muissakin Jenkins-töissä. Jenkinsfilen loppuosa on esitetty kuviossa 17.

```
  builderStage('Clone YOUR_REPO_HERE') {
    hg('YOUR_REPO_HERE.SUB_REPO', 'master')
  }

  ansibleStage('Deploy or Uninstall elastalert') {
    def execPath = "YOUR_REPO_HERE.SUB_REPO/PATH/TO/ANSIBLE/FOLDER"
    sh "cd $execPath && \
        ansible-playbook -i inventories/YOUR_INVENTORY/ install-unin-
stall-elastalert.yml -t $installTags"
  }
}
```

KUVIO 17. Jenkinsfilen loppuosa.

5.2 Elastalert

5.2.1 Elastalert työkaluna

Elastalert on avoimen lähdekoodin automaattinen datan monitorointiohjelmisto, joka tarjoaa käyttäjälle hälytyksiä annettujen sääntöjen perusteella. Elastalertin on alun perin kehittänyt yhdysvaltalainen yritys Yelp. Elastalert oli Yelpille ratkaisu monitoroida automaattisesti kerättyä statistiikkaa ja lähettää hälytyksiä, jos järjestelmässä ilmenisi vikaa. Yelp on alun perin kehitetty auttamaan käyttäjiä löytämään ja arvostelemaan paikallisia palveluja tarjoavia yrityksiä. Yelpiä käyttää keskimäärin 95 miljoonaa käyttäjää kuukausittain. (Yelp 2018.)

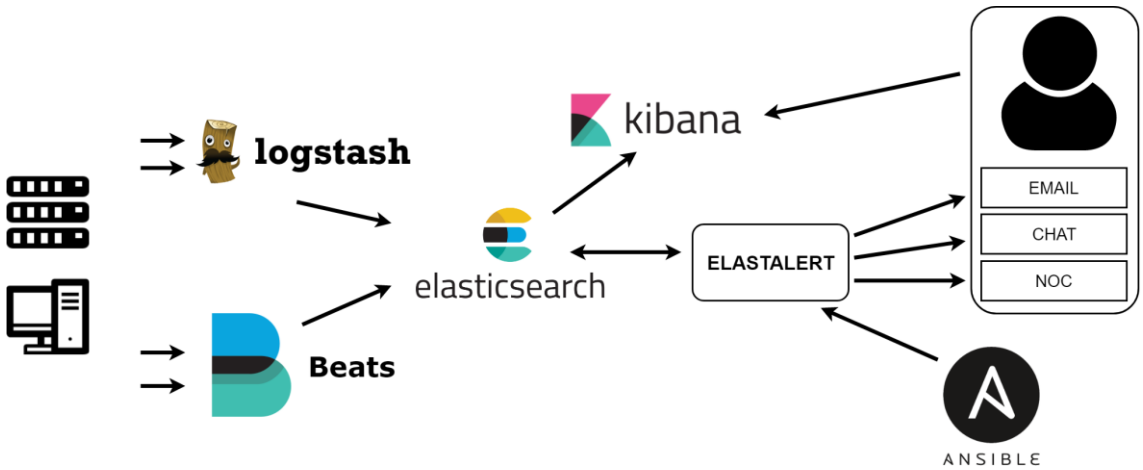
Elastalert-projekti on saatavilla GitHub-palvelussa ja projektiin on tuottanut sisältöä yli 130 kehittäjää. Elastalertille on tehty myös lisäosia, joilla käyttäjä voi laajentaa tai parantaa entisestään Elastalertin toimintaa omaan käyttökohteensa mukaan. Yksi suosituimmista lisäosista on Kibana-lisäosa, jossa käyttäjä voi esimerkiksi luoda Elastalertille sääntöjä suoraan selaimesta. Dockerhubista löytyy myös useita Elastalertin sisältäviä valmiita Docker-imageja. Suosituimmilla Docker-imageilla on yli satatuhatta latauskertaa.

Elastalertissa ei itsessään ole mitään käyttöliittymää, vaan työkalu on yksinkertainen ja kevyt komentoriviohjelma. Elastalertin monitoroimaa dataa onkin tarkoitus tutkia ja visualisoida Elasticsearchin ja Kibanan avulla. Elastalert voidaankin ottaa käyttöön suoraan niissä tapauksissa, joissa datan keräys- ja visualisointi-infrastruktuuri on kehitetty toimimaan Elastic-tuoteperheen työkaluilla jo valmiiksi.

Elastic-tuoteperhe tarjoaa virallisessa valikoimassaan maksullista hälytystyökalua Watcheria. Watcher on osa isompaa lisäosakokoelmaa X-Packia. X-Packissa on useita lisäosia, joilla voi parantaa Elasticsearchin tarjoamia toiminnallisuuksia. Watcherin saamiseksi X-Packin joutuu kuitenkin ostamaan ja hinta riippuu kohdeympäristön konfiguraatiosta. Elastalert tarjoaa huomattavan määrän samoja ominaisuuksia kuin Watcher ja on samalla ilmainen. Elastalertin edullisuus tekee työkalusta helposti kokeiltavan ja lähestyttävän, kun ostoprosessia ei tarvitse erikseen tehdä.

Elastalert voidaan asentaa lähes mille tahansa Linux-jakelulle, kunhan käyttöjärjestelmään voidaan asentaa python. Elastalert-paketit haetaan käyttäen pythonin paketinhallintajärjestelmää pip:iä tai lataamalla Elastalert pakattu tiedosto suoraan GitHubista. Elastalertin käyttämiseen voidaan myös käyttää valmiita Docker-imageja, joita on useita tarjolla Dockerhubissa. Elastalert voidaan myös paketoida itse tuotettuun Docker-konttiin, kuten tässä raportissa oli edellä esitetty.

Kuviossa 18 on esitetty yksinkertaistettu kuva Elastalertin toimintatavasta osana Elastic-pinoa. Elastic-pinon keräämä ja tuottama data on säilötty Elasticsearchiin, josta Elastalert käy kysymässä valvottavien metriikoiden tilaa. Mahdollisen hälytyksen sattuessa Elastalert lähettää hälytyksen määritettyihin kanaviin, joiden avulla hälytys annetaan käyttäjille. Kuvassa on myös esitetty Ansible-asennusautomaatio, joka tuo ja konfiguroi Elastalertin käyttöön osaksi Elastic-pinoa. Lisäksi Elastalert kirjoittaa Elasticsearchiin oman etenemänsä ja hälytysten tilan. Näin ollen Elastalert voidaan myös konfiguroida ja käynnistää tarvittaessa uudestaan ajonaikaisesti ilman datahävikkiä.



KUVIO 18. Elastalertin toiminta Elastic-pinossa.

5.2.2 Toiminta

Elastalertin toiminta perustuu pohjimmillaan yksinkertaiseen Elasticsearchin indeksin lukemiseen. Näiden tulosten pohjalta päätellään, tehdäänkö hälytys vai ei. Elastalert yhdistää luetun datan sääntöjen ja hälytysten kanssa. Jos jokin sääntö täyttyy, lähetetään yksi tai useampia hälytyksiä annetun konfiguraation mukaisesti. Elastalert on suunniteltu olemaan luotettava, erittäin joustava sekä helppo konfiguroida ja ottaa käyttöön. (Elastalert documentation 2018.)

Elastalert käyttö asentamisen jälkeen on yksinkertaista. Ohjelma käynnistetään python-komennolla *python elastalert/elastalert.py*. Jos mitään lisäparametrejä ei anneta käynnistyksen yhteydessä Elastalert käynnistyy oletusasetuksilla. Komentoriviparametreinä voidaan antaa esimerkiksi *--config <polku tiedostoon>*, jolla voidaan määrittää haluttu konfiguraatiotiedosto tai *--verbose*, jolla saadaan laajempi lokitustaso käyttöön Elastalertin suorituslokiin. Lisäksi voidaan käyttää useita muita parametrejä, jotka on dokumentoitu ja kuvattu Elastalertin virallisessa dokumentaatiossa.

Elastalertin konfiguraatiotiedoston pohjana käytettiin paketissa mukana tullutta esimerkkitiedostoa. Konfiguraatiotiedostoon voidaan määritellä laajasti Elastalertille erilaisia yleisiä sääntöjä, kansiorakenteita ja muita tarvittavia muuttujia. Iso osa konfiguraatiotiedostoon määriteltävistä parametreistä voidaan antaa Elastalertille myös ympäristömuuttujina, joka tuo tarvittaessa lisää joustavuutta käyttöön. Konfiguraation laajuudesta ja do-

kumentaation hieman epäselvästä rakenteesta johtuen kannattaa mahdollisiin konfiguraatiomahdollisuuksiin tutustua huolella kehitysvaiheen aikana. Konfiguraatiotiedoston sisältö on esitetty kuviossa 19.

```
rules_folder: ../elastalert-files/rules

run_every:
  minutes: 1

buffer_time:
  minutes: 30

es_host: {{ elasticsearch_ip }}
es_port: {{ elasticsearch_port }}

writeback_index: {{ elasticsearch_write_index }}

alert_time_limit:
  days: 2
```

KUVIO 19. Elastalertin yleinen konfiguraatiotiedosto.

Tässä työssä Elastalertille luotiin konfiguraatio vain niillä parametreillä, jotka nähtiin tarpeellisiksi testivaiheen aikana. Tiedosto on yaml-formaattia, joka on templatisoitu Ansiblea ja jinja2-templaattia käyttäen. Konfiguraatioon voidaan tuoda dynaamisesti tarvittavat muuttujat Ansiblella ympäristökohtaisesti. Ensimmäisenä konfiguraatiossa määritellään polku, johon sääntötiedostot on luotu. Seuraavaksi Elastalert asetetaan tekemään Elasticsearch-kysely minuutin välein, jotta mahdollisille hälytyksille saadaan hyvä vasteaika. Buffer_time:llä määritellään aikaikkuna, jota vasten Elastalert tekee hakuja Elasticsearchiin. Es_host- ja Es_port-muuttujilla määritellään Elasticsearchin osoite. Writeback_index-parametrillä voidaan määrittää Elastalertin käyttämälle indeksille nimi, jota Elastalert kirjoittaa Elasticsearchiin. Tästä voi olla hyötyä esimerkiksi silloin, kun samaan Elasticsearchiin tehdään hakuja usealla Elastalertilla. Alert_time_limit muuttuja määrittää kauanko Elastalert yrittää lähettää hälytystä, jos hälyttäminen epäonnistuu.

Elastalert ei sisällä itsessään Elasticsearchin tapaan isoja määriä indeksoitua dataa. Elastalert käyttää hyväkseen Elasticsearchin indeksiä ja kirjoittaa oman edistymisen sekä tilanteen Elasticsearchille luomaansa omaan indeksiin. Elastalertin luomaa dataa voi halutessaan tutkia ja visualisoida esimerkiksi Kibanalla selaimen välityksellä. Elastalertin oma indeksi sisältää dataa kaikista sääntöjen pohjalta tehdyistä hauista, muodostetuista hälytyksistä, hälytysten hiljentämisestä, virheistä Elastalertin toiminnassa sekä muusta

Elastalertin tuottamasta metadatasta. Tätä indeksiä hyödyntämällä Elastalert voi lukea ja muodostaa hälytyksiä menneeltä ajalta, jos jostain syystä Elastalert olisi ollut poissa päältä tai ei olisi omannut yhteyttä Elasticsearchiin. Samalla Elastalert siirtää raskaan datan käsittelyn siihen tarkoitetulle Elasticsearchille ja pysyy itse hyvin kevyenä palveluna, joka analysoi vain pieniä datalohkoja kerrallaan.

Elastalertin käyttämä indeksi tulee luoda Elasticsearchiin Elastalertin ensimmäisellä käynnistyskerralla. Staattisissa ympäristöissä, joissa Elasticsearchia ja muuta ympäristöä ei asenneta uudestaan tiheästi, voidaan indeksi luoda manuaalisesti. Dynaamisemmat ympäristöt vaativat kuitenkin paremman indeksin hallinnan, jotta voidaan olla varmoja Elastalertin toiminnasta aina asennuksien ja poistojen yhteydessä. Elastalertille luotiin käynnistysskripti, jolla varmistetaan indeksin olemassaolo tai luodaan uusi indeksi, jos Elastalertille määritettyä indeksiä ei löydy. Kuviossa 20 on esitetty Elastalertille luotu käynnistysskriptin alkuosa.

```
# Check that elastalert configs are available
if [ ! -f ${CONF_FILE} ]; then
  echo "ELASTALERT CONFIGS NOT FOUND!"
  echo "Config dir query path: $CONF_FILE"
  echo "Exiting..."
  exit 1
fi

# Check Elasticsearch connection before index checking
while ! wget -q -T 3 -O - "http://${ELASTIC_HOST}:${ELASTIC_PORT}"
2>/dev/null
do
  echo "Waiting for Elasticsearch..."
  echo "Connecting to:   http://${ELASTIC_HOST}:${ELASTIC_PORT}"
  sleep 5
done
echo "Connected to Elasticsearch !"
```

KUVIO 20. Elastalertin käynnistysskriptin alkuosa.

Käynnistysskripti alkaa varmistamalla, että Elastalertille muodostettu konfiguraatio on saatavilla. Konfiguraatiotiedoston polku ja nimi on asetettu etukäteen Ansiblen avulla Docker-kontin ympäristömuuttujaan, josta muuttujaa voidaan käyttää hyödyksi kontin eri prosesseissa. Käynnistysskriptin seuraavassa vaiheessa tarkistetaan Elasticsearch yhteyden toiminta. Yhteys halutaan muodostaa ennen kuin Elasticsearchiin kirjoitettavaa indeksiä luodaan tai luetaan. Yhteyttä odotetaan, kunnes Elasticsearch-yhteys saadaan

muodostettua. Tilanne, jossa Elasticsearchia joudutaan odottamaan, voi ilmentyä ohjelmistojen ja alustakoneiden asennusten sekä poistojen yhteydessä.

Kuviossa 21 on esitetty Elastalertin käynnistysskriptin loppuosa, jossa haetaan ensimmäisenä Elastalertille määriteltyä indeksiä Elasticsearchin datasta. Jos haettu indeksi löytyy, jatketaan skriptiä eteenpäin. Jos kuitenkaan indeksiä ei ole saatavilla Elasticsearchissa, se luodaan tarvittavilla parametreillä. Luominen tapahtuu käyttäen Elastalertin mukana tulevaa komentoa *elastalert-create-index*. Indeksin luonnissa käytetään Ansiblella asetettuja ympäristömuuttujia komennon parametreinä. Parametrit ovat tarpeen, jos käytössä on asetuksia ja tiedostopolkuja, jotka eroavat Elastalertin vakiokonfiguraatioista.

```
# Query elastalert index from elasticsearch
echo "Quering elastalert index from elasticsearch..."
if ! wget -q -T 3 -O - "http://${ELASTIC_HOST}:${ELASTIC_PORT}/${ELAS-
TIC_INDEX}" 2>/dev/null
then
  echo "Creating index to Elasticsearch data..."

  elastalert-create-index \
    --host "${ELASTIC_HOST}" \
    --port "${ELASTIC_PORT}" \
    --config "${CONF_FILE}" \
    --index "${ELASTIC_INDEX}"

  echo "Elastalert index created."
else
  echo "Elastalert index already exists in Elasticsearch."
fi

echo "Starting Elastalert..."
echo "Elasticsearch IP: http://${ELASTIC_HOST}:${ELASTIC_PORT}"
echo "Elastalert config file:  $CONF_FILE"
cd /opt/elastalert && python elastalert/elastalert.py --config
${CONF_FILE}
```

KUVIO 21. Elastalertin käynnistysskriptin loppuosa.

Skriptin lopuksi Elastalert käynnistetään normaalia käynnistyskomentoa käyttäen. Käynnistyksen yhteydessä annetaan parametrinä polku konfiguraatiotiedostoon. Tiedostopolun määrittely tarvitaan tämän raportin puitteissa tehtyyn toteutukseen, koska konfiguraatiotiedosto ei löydy Elastalertin juurikansiosta. Komentoon voitaisiin tarvittaessa lisätä muitakin parametrejä, joilla voidaan esimerkiksi helpottaa kehitysvaiheessa tarvittavan informaation keräämistä Elastalertista.

5.2.3 Säännöt

Elastalertin toiminta perustuu sääntötiedostoissa määriteltyihin sääntöihin, joiden perusteella Elasticsearchiin tehdään hakuja ja muodostetaan tarvittaessa hälytyksiä. Sääntötiedosto on konfiguraatiotiedoston tavoin yaml-formaatissa. Jokainen sääntö perustuu johonkin Elastalertin mukana tulevaan sääntömääritykseen tai tarvittaessa pythonilla itse luotuun sääntöön. Elastalertin mukana tulevat säännöt ovat lyhyesti kuvattuna:

- Any
    - Muodostaa hälytyksen, jos mikä tahansa määritetyistä ehdoista täyttyy.
- Blacklist
    - Muodostaa hälytyksen, jos annetuilla hakuehdoilla löytyy jokin etukäteen määritellyistä arvoista.
- Whitelist
    - Muodostaa hälytyksen, jos annetuilla hakuehdoilla ei löydy jokin etukäteen määritellyistä arvoista.
- Change
    - Muodostaa hälytyksen, jos jokin tietty arvo muuttuu.
- Frequency
    - Muodostaa hälytyksen, jos tiettyjä tapahtumia on määriteltyä useampia tietyn ajanjakson sisällä.
- Spike
    - Muodostaa hälytyksen, jos tapahtumien määrä moninkertaistuu tietyn vertailuajan aikana. Voidaan käyttää myös tapahtumien määrän laskun tarkkailuun.
- Flatline
    - Muodostaa hälytyksen, jos tiettyjä tapahtumia on määriteltyä vähemmän tietyn ajanjakson sisällä.
- New Term
    - Muodostaa hälytyksen, jos uusi arvo löytyy haetuista kentistä, joissa sitä ei ole ennen nähty.
- Cardinality
    - Muodostaa hälytyksen, jos aikaikkunan sisällä haetuista kentistä löytyy uniikkeja arvoja määriteltyä enemmän.

- Metric Aggregation
    - Muodostaa hälytyksen, jos haetuista arvoista tehdyn laskutoimituksen jälkeen saatu tulos on tosi annetun vertailuarvon kanssa. Haetuista arvoista voidaan laskea minimi, maksimi, keskiarvo, summa, kardinaali tai kokonaismäärä.
- Percentage Match
    - Muodostaa hälytyksen, jos haetun datan sisältö on poikkeava määritetyistä arvoista tietyn prosenttimäärän verran. Voidaan verrata vähimmäis- ja enimmäismäärää.

Jokaiselle hälytystyypille on määritetty pakolliset sekä vapaavalintaiset konfiguraatiokentät, joiden avulla säännöt saadaan toimimaan käytetyn datan ja halutun hälytyksen kanssa oikein. Sääntötiedostoihin voidaan määritellä myös samoja kenttiä kuin Elastalertin konfiguraatiotiedostossa. Tällöin sääntötiedostoon määritetty arvo pätee vain kyseiselle säännölle ja muut säännöt käyttävät konfiguraation määrittämää arvoa.

Kaikille säännöille yhteistä on määritellä polku ja indeksi, jota datan hakemiseen ja säännön tarkistamiseen käytetään. Määrittely tapahtuu suoraan Elasticsearchin käyttämällä Apache lucene-syntaksilla, joka kirjoitetaan tiedostoihin Elastalertille ominaisella tavalla. Elastalert tarjoaa yleisimmät Elasticsearchin hakutoiminnot, kuten query_string, terms, wildcard, range, negation, and ja or. Kuviossa 22 on esitetty esimerkki Elastalertin datahakumäärittelystä, jossa etsitään Java-kielen jmx-indeksistä käyttöjärjestelmän prosessorin kuormitustietoa. Haettu arvo pitää myös saada sellaisesta datasta, joka sisältää kentässä service halutun palvelun nimen.

```
index: jmx-v3*
filter:
- query:
    query_string:
      query: 'path: "java.lang:type=OperatingSystem.SystemCpuLoad"'
- query:
    query_string:
      query: 'service: "YOUR_SERVICE_NAME"'
```

KUVIO 22. Esimerkki Elastalert-datahakumäärittelystä.

Yksi helppo tapa kehittää Elastalertin sääntöjä on etsiä haluttuja datapisteitä ja arvoja Kibanan avulla. Kibanalla kehittäjä voi visualisoida tehokkaasti haluamaansa dataa ja luoda samanlaisten hakujen perusteella sääntöjä Elastalertiin. Kibanalla voi tutkia myös

Elasticsearchissa olevia indeksejä ja niiden sisältöä. Tämän avulla voidaan tuottaa tarkemmat hakukriteerit säännöille, jotka takaavat nopeammat hakutulokset Elasticsearchin indeksistä. Samalla minimoidaan verkkoliikennettä sekä nopeutetaan mahdollisen hälytyksien lähettämistä.

Elastalertin sääntö koostuu yleisesti seuraavista osista:

- Nimi
    - Säännölle on hyvä antaa kuvaava nimi, jotta hälytyksen sattuessa ei ole epäselvää, mistä hälytys johtuu.
- Indeksi
    - Elasticsearchin indeksi, josta monitorointidata haetaan.
- Säännön tyyppi
- Säännölle tyypille ominaiset parametrit
    - Säännöille pakolliset sekä valinnaiset parametrit ovat dokumentoituna Elastalertin dokumentaatiosivustolla ja GitHubissa.
- Hakusuodattimet
    - Lista hakusuodattimia, joiden perusteella annetusta indeksistä haetaan monitorointidataa.
- Hälytyssäännöt
    - Määritysjoukko, jossa kuvataan, mihin kanaviin hälytyksiä lähetetään ja minkälaisilla parametreillä.

Kuviossa 23 on esitetty eräs esimerkki Elastalertilla luotavasta yksinkertaisesta sääntötiedostosta. Sääntötyyppinä käytetään Any-tyyppiä, joka lähettää hälytyksen aina, kun kaikki määritellyistä hakukriteereistä täyttyy. Haku suoritetaan kaikkiin jmx-v3 -alkuisiin indekseihin Elasticsearchin datassa. Suodattimissa on käytetty kolmea eri hakukriteeriä, joiden avulla varmistetaan, että mahdollinen osuma tapahtuu vain juuri halutuilla arvoilla. Ensimmäisellä suodattimella annetaan polku, josta data löytyy Elasticsearchissa. Toinen suodatin määrittää sen palvelun nimen, jolta arvo halutaan saada. Kolmannessa suodattimessa etsitään yli tuhannen millisekunnin ylittäviä arvoja. Tuhat millisekuntia on määritetty kyseisen palvelun ylärajaksi keskimääräisen vasteajan osalta. Heti kun yli tuhannen millisekunnin arvo ilmestyy Elasticsearchin dataan, lähetetään hälytys. Hälytyksen lähettämiseen määritellyt parametrit tuodaan tiedostoon mukaan Ansiblella asennusvaiheessa.

Parametrin on määritelty erillisessä jinja2-templaattitiedostossa, jossa niitä voidaan hallita keskitetysti.

```
---
name: YOUR_SERVICE high average response time

type: any
index: jmx-v3*

filter:
- query:
    query_string:
      query: 'path: "PATH.TO.YOUR.SOFTWARE.AVERAGE.RESPONSE.TIME"'
- query:
    query_string:
      query: 'service: "YOUR_SERVICE_NAME"'
- query:
    query_string:
      query: 'value: [1000 TO *]'

{% include "rule-alerting-template.j2" %}
```

KUVIO 23. Elastalert sääntötemplaattitiedosto.

5.2.4 Hälyttäminen

Elastalertin hälyttämisperiaate on yksinkertainen. Jos annettu sääntö täyttyy, lähetetään hälytys määriteltyihin kanaviin. Hälytyksen lähetys voi myös peruuntua, jos hälytysintervalliksi määritelty aika ei ole täyttynyt edellisestä hälytyksestä. Hälyttämiseen tarvittavat konfiguraatioparametrit kirjoitetaan sääntötiedostoon mukaan. Tämä mahdollistaa hälytyssääntöjen priorisoinnin ja muokkaamisen eri hälytyssääntöjen kesken. Esimerkiksi erittäin vakavat hälytykset halutaan lähettää useisiin eri kanaviin, jotta hälytys tulee käsiteltyä varmasti mahdollisimman nopeasti. Toisaalta taas matalimman prioriteetin hälytys tai varoitus voidaan lähettää kanaviin, joista ne tulee käsiteltyä tarvittavien prosessien kautta.

Elastalert tukee vakiona pitkän listan palveluja ja työkaluja, joihin hälytys voidaan lähettää lisäämällä pieni osuus kanavalle ominaista konfiguraatiota. Näitä ovat sähköposti, Jira, Slack, Mattermost, OpsGenie, SNS, HipChat, Stride, MS Teams, Telegram, PagerDuty, Exotel, Twilio, VictorOps, Gitter, ServiceNow ja Stomp (Elastalert documentation

2018.) Kaikkien hälytyskanavien yksilökohtaiset konfiguraatiomahdollisuudet ovat listattuna Elastalertin dokumentaationsivuilla.

Valmiiksi rakennettujen palvelujen lisäksi Elastalertista voidaan lähettää hälytys täysin kustomoituun ratkaisuun kohtalaisen helposti. HTTP POST -metodi lähettää nimensä mukaisesti määritetyn http post -kutsun määritettyyn osoitteeseen annetuilla parametreillä. Suurin osa API-toteutuksista tukee sanoman vastaanottamista post-metodilla. Command-metodi antaa käyttäjälle vielä laajemmat mahdollisuudet toteuttaa oman hälytysratkaisun. Command suorittaa nimensä mukaisesti hieman kustomoidun komentorivikäskyn, jonka mukana voidaan välittää hälytykseltä tulevat tiedot komentoon mukaan. Command-metodia voisi käyttää esimerkiksi CURL-komennon käyttämiseen tai vaikkapa oman hälytysjärjestelmän kutsumiseen suoraan komentoriviltä. Command- ja HTTP POST -metodeihin voidaan liittää kaikki sama data mukaan, kuin aikaisemmin lueteltuihin valmiisiin kommunikaatio- ja hälytysjärjestelmäsanomiin.

Mahdollisten hälytyskanavien lisäksi Elastalertin dokumentaatiossa tulee mukana lyhyt ohjeistus, kuinka itse voi kehittää python-kielellä tuen sellaiseen järjestelmään, johon Elastalertissa ei tukea vielä ole. Avoimen lähdekoodin avulla kuka tahansa voi luoda Elastalertiin uusia keinoja lähettää hälytyksiä. Tästä syystä lista Elastalertin tukemista järjestelmistä on melko pitkä ja jatkuvasti kehittyvä.

Kuviossa 24 on esitetty testivaihetta varten luotu templaattitiedosto hälytyssääntöjen tuomiseen dynaamisesti sääntötiedostojen loppuun. Tämän työn puitteissa käytettiin vain yhtä templaattitiedostoa. Tällöin siis kaikki hälytyssäännöt saivat samat parametrit datan lähettämiseen, lähetyskanavien konfiguraatioon sekä uudelleenlähetyksen ajan määrittämiseen. Tiedostossa määritellään ensin lista kenttiä, jotka halutaan lähettää hälytysviestissä mukana. Jos kentät jätetään määrittämättä, lähettää Elastalert kaikki löytyneet kentät hälytysviestin mukana. Ylimääräiset kentät kannattaa karsia pois hälytysviestistä, jotta sisältö on mahdollisimman yksinkertainen ja tehokas hälytyksen analysoijalle.

```
include: ["LIST", "OF", "FIELDS"]

realert:
  {{ elastalert_realert_time }}

alert:
- debug
{% if slack_enabled %}
- slack
{% endif %}

{% if slack_enabled %}
slack:
slack_webhook_url: "{{ slack_hook_url }}"
slack_username_override: "Elastalert-{{ base_environment }}"
{% endif %}
```

KUVIO 24. Elastalertin hälytystemplaattitiedosto.

Hälytystemplaattitiedostossa määritellään seuraavaksi uudelleenlähetysaika. Tämä tarkoittaa aikaa, jonka välein kyseinen hälytys lähetetään. Ajan nostamisella pyritään välttämään hälytystulvaa vastaanottavassa päässä. Käyttökohteesta riippuen aika täytyy määritellä sopivaksi, jotta ihmisen suorittama valvonta ei heikenny liian tiheästä hälytysintervallista. Lopuksi tiedostossa määritellään, mihin kanaviin hälytys lähetetään. Tämän työn aikana hälytyksiä lähetettiin vain debug-moodilla Elastalertin lokiviesteihin sekä Mattermost-pikaviestinpalveluun. Mattermost on avoimen lähdekoodin pikaviestinpalvelu, joka toteuttaa Slack-palvelun rajapinnat. Tästä syystä tiedostossa käytetään Slack-avainsanaa.

Templaattitiedosto on jinja2-templaatti. Tiedostoon tuodaan Slack-asetukset vain, jos Elastalertin asetuksissa on määritetty slack_enabled-muuttuja todeksi. Templatisoinnin avulla voidaan asennuksessa jättää helposti ylimääräiset hälytyskanavat pois tai vastaavasti lisätä hälytyskanavien määrää, jos asennusympäristö sitä vaatisi.

Hälytysten tuottamaa tekstisisältöä voi myös muokata sopivammaksi. Viestin otsikon, sisällön sekä viestiin sisälletyt kentät voi kustomoida. Elastalertin dokumentaatio tarjoaa muutamia perusesimerkkejä siitä, kuinka hälytyksen sisältöä muokataan. Testijakson aikana huomattiin kuitenkin, että dokumentaatio oli heikohkoa myös näiden ominaisuuksien kohdalla. Parhaat käytänteet löydettiin kokeilemalla ja soveltamalla saatuja esimerkkejä omiin hälytystemplaatteihin. Testijakson yksinkertaistamiseksi kustomoidut hälytysviestit jätettiin vielä tekemättä. Elastalertin laajemmassa käyttöönotossa hälytysviestit

muokattaisiin vastaamaan tarkasti haluttua sisältöä, jotta hälytysviesti itsessään helpottaisi valvojan tehtävää mahdollisimman paljon. Elastalert tarjoaa useampia valmiita vaihtoehtoja, joista valita hälytysviestin sisältö hälytyssäännölle sopivaksi.

6 POHDINTA

Elastalert-työkalun testaus ja kokeilu osoittautuivat hyvin mielenkiintoiseksi aiheeksi tehdä opinnäytetyö. Kokeilujakso toi esiin kehityskohteita sekä monitorointi-infrastruktuurista että itse tuotteen valvontarajapinnoista. Monimutkainen hajautettu järjestelmä sekä sen tuomat asennus- ja konfiguraatiohaasteet tuli ottaa jo työn testausvaiheessa huomioon. Työn tulokset herättivät myös keskustelua projektin sisällä siitä, miten tuotteen kokonaisvaltaista valvontaa kannattaisi tehdä tehokkaasti.

Elastalert osoittautui testijakson aikana saatujen kokemusten perusteella oivalliseksi työkaluksi automaattisen valvonnan tekoon. Avoin lähdekoodi tekee työkalusta myös helposti lähestyttävän, koska mitään lisensiointeja tai maksuja ei tarvitse erikseen tehdä. Toisaalta myös Elastic-tuoteperheen työkalut ovat saaneet jatkuvasti kasvavan käyttäjäkunnan, joka kasvattaa samalla Elastalertin potentiaalisia käyttäjämääriä. Elastalertin selkeä vahvuus on työkalun yksinkertaisuus. Sekä itse työkalun yksinkertainen toiminta, että siihen liittyvien sääntöjen konfiguraatioiden tuottaminen yaml-syntaksilla ovat hyviä asioita.

Joitain Elastalertin huonompia puolia löytyi testijakson aikana silloin, kun Elastalertin avulla haluttiin toteuttaa haastavampia toiminnallisuuksia. Elastalertin ollessa yksinkertainen ja kevyt työkalu, hälytysten helppo tilan seuranta suoraan työkalusta jäi puuttumaan. Myös näkymien integrointi suoraan hälytyksiin tai hälytysviesteihin osoittautui turhan hankalaksi. Testijakson aikana huomattiin myös, että Elastalertin vahvuus on lokiviestien ja niiden kaltaisten datan valvonta. Vaikka jatkuvasti saatavien metriikoiden valvonta onnistuu Elastalertilla, ovat toiminnallisuudet selvästi enemmän epäsäännöllisemmän datan valvontaan suunnattuja.

Testijakson päätteeksi todettiin, että Elastalert ei sopinut käyttötarkoitukseemme tarpeeksi hyvin. Koska tuote on monimutkainen hajautettu järjestelmä, pitäisi valvontajärjestelmän pystyä tarjoamaan vielä enemmän toimintoja ja ominaisuuksia, joiden avulla voitaisiin helpottaa valvojan työtä. Elastalert voisi sopia oivallisesti kuitenkin pienemmän tai yksinkertaisemman järjestelmän valvontaan varsinkin silloin, jos järjestelmässä on jo käytössä Elastic-tuoteperheen tuotteita. Testijakson päätteeksi huomattiin tarve kokeilla

muitakin monitorointi- ja hälytysjärjestelmiä, jotta tuotteen kokonaisvaltaista valvontaa voitaisiin parantaa ja kehittää.

Ohjelmistoalalla erilaisten työkalujen ja tekniikoiden testaaminen on parhaimmillaan erittäin tehokas tapa lisätä tietoa ja herättää keskustelua projektin sisällä. Testijakso tuo esiin uusien työkalujen tuomia hyötyjä ja haittoja. Lisäksi opitaan paljon nykyisten tekniikoiden toiminnallisuuksista, kun verrataan useampaa työkalua tai järjestelmää keskenään. Testijakson tuloksista riippumatta saadaan aina uutta tietoa tai vähintään vahvistettua jo olemassa olleet odotukset.

Haluankin kannustaa yrityksiä tekemään pieniä testijaksoja uusille tuotteille ja teknologioille. Näiden testijaksojen avulla voidaan parantaa tuoteosaamista ja ymmärrystä oman tuotteen mahdollisuuksista. Aina testaus- ja validointityö ei tuota valmiita ratkaisuja tai muuta isoja osia kokonaisuudesta, mutta testijaksojen seurauksena saadaan kuitenkin aina avattua keskustelua teknologioiden vaikutuksesta omaan tuotekehitysprosessiin ja tuotteen kokonaiskuvaan.

LÄHTEET

Andhavarapu, A. 2017. Learning Elasticsearch. Distributed real-time search and analytics with Elasticsearch 5.x. Birmingham, UK: Packt Publishing Ltd.

O'Reilly. 2016. Site reliability engineering. How Google runs production Systems. CA, USA: O'Reilly Media, Inc.

Beats. 2018. Lightweight Data Shippers. Verkkosivusto. Luettu 28.3.2018. https://www.elastic.co/products/beats

DB-Engines. 2018. DB-Engines Ranking of Search Engines. Verkkosivusto. Luettu 28.3.2018. https://db-engines.com/en/ranking/search+engine

Elastalert Documentation. 2018. ElastAlert - Easy & Flexible Alerting With Elasticsearch. Ohjelmiston dokumentaatio. Luettu 15.3.2018. https://elastalert.readthedocs.io/en/latest/elastalert.html

Elastic Stack. 2018. What is the ELK Stack? Why, it's the Elastic Stack. Verkkosivusto. Luettu 28.3.2018. https://www.elastic.co/elk-stack

Giménez, J. SRE tiimin vetäjä. 2017. What Is Jenkins and Why Should You Be Using It. Blogi. Luettu 8.3.2018. https://bugfender.com/blog/what-is-jenkins-and-why-should-you-be-using-it/

Hawkins, A. 2017. Introduction to Ansible. Blogi. Luettu 11.3.2018. https://semaphoreci.com/community/tutorials/introduction-to-ansible

Kibana. 2018. Your Window into the Elastic Stack. Verkkosivusto. Luettu 28.3.2018. https://www.elastic.co/products/kibana

Logstash. 2018. Centralize, Transform & Stash Your Data. Verkkosivusto. Luettu 28.3.2018. https://www.elastic.co/products/logstash

Pitkänen, L. järjestelmäasiantuntija. 2016. Docker mullistaa sovelluskehityksen kuin merikontit rahdin. Blogi. Luettu 7.3.2018. https://blog.planeetta.net/docker-kontit-tuotantoon

Wilson, S. ohjelmistopäällikkö. 2011. State Based Vs. Stateless monitoring. Blogi. Luettu 22.3.2018. https://blogs.technet.microsoft.com/authormps/2011/03/07/state-based-vs-stateless-monitoring/

Yelp. 2018. About Us. Verkkosivusto. Luettu 12.3.2018. https://www.yelp.com/about

LIITTEET

Liite 1. Elastalert Ansible-roolin lähdetiedostot

elastalert/defaults/main.yml

```
---

artifactory_docker_image_version: 0.1.28-2
artifactory_docker_image_name: 'PATH/TO/ARTIFACTORY/image/elastalert'
artifactory_docker_image: '{{ artifactory_docker_image_name }}:{{ arti-
factory_docker_image_version }}'

base_environment: 'YOUR_BASE_ENVIRONMENT'
elastalert_mount_path: '/opt/elastalert/{{ base_environment }}'

elasticsearch_ip: '{{ groups["elasticserver"] }}'
elasticsearch_port: '9200'
elasticsearch_ip_port: '{{ elasticsearch_ip }}:{{ elasticsearch_port }}'
elasticsearch_write_index: 'elastalert_status'

elastalert_realert_time: 'hours: 12'

# Slack settings
slack_enabled: true
slack_hook_url: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          YOUR_CRYPTED_VAULT_SECRET_HERE
          YOUR_CRYPTED_VAULT_SECRET_HERE
          YOUR_CRYPTED_VAULT_SECRET_HERE
          YOUR_CRYPTED_VAULT_SECRET_HERE
```

elastalert/tasks/main.yml

```
---

# Uninstall -->
- include_tasks: uninstall.yml
  tags: [uninstall]

# Install -->
- include_tasks: install.yml
  tags: [install]
```

elastalert/tasks/install.yml

```
---

- name: Create folders for elastalert files if needed
  file:
    path: '{{ elastalert_mount_path }}/{{ item.path }}'
    state: directory
  with_filetree: 'templates/'
  when: item.state == 'directory'

- name: Move elastalert config to host
  template:
    src: config.yaml.j2
    dest: '{{ elastalert_mount_path }}/config.yaml'

- name: Move and set elastalert starting script to executable
  copy:
    src: start-elastalert.sh
    dest: '{{ elastalert_mount_path }}/'
    mode: a+x

- name: Move and create elastalert rules from templates
  template:
    src: '{{ item.src }}'
    dest: '{{ elastalert_mount_path }}/rules/{{ item.path | regex_re-
place(".j2","") }}'
  with_filetree: 'templates/rules/'
  when: item.state == 'file'

- name: Start container from artifactory image
  docker_container:
    name: 'elastalert-{{ base_environment }}'
    image: '{{ artifactory_docker_image }}'
    state: started
    restart: yes
    env:
      CONF_FILE: '/opt/elastalert-files/config.yaml'
      ELASTIC_HOST: '{{ elasticsearch_ip }}'
      ELASTIC_PORT: '{{ elasticsearch_port }}'
      ELASTIC_INDEX: '{{ elasticsearch_write_index }}'
    volumes:
      - '{{ elastalert_mount_path }}:/opt/elastalert-files'
    restart_policy: on-failure
```

elastalert/tasks/uninstall.yml

```yaml
---

- name: Stop and remove elastalert container
  docker_container:
    name: elastalert-{{ base_environment }}
    state: absent

- name: Delete files and configs
  file:
    path: '{{ elastalert_mount_path }}'
    state: absent
```

elastalert/templates/config.yaml.j2

```yaml
---

rules_folder: ../elastalert-files/rules

run_every:
  minutes: 1

buffer_time:
  minutes: 30

es_host: {{ elasticsearch_ip }}
es_port: {{ elasticsearch_port }}

writeback_index: {{ elasticsearch_write_index }}

alert_time_limit:
  days: 2
```

elastalert/templates/rule-alerting-template.yaml.j2

```
include: ["LIST", "OF", "FIELDS"]

realert:
  {{ elastalert_realert_time }}

alert:
- debug
{% if slack_enabled %}
- slack
{% endif %}

{% if slack_enabled %}
slack:
slack_webhook_url: "{{ slack_hook_url }}"
slack_username_override: "Elastalert-{{ base_environment }}"
{% endif %}
```

elastalert/files/start-elastalert.sh

```
#!/bin/sh
set -e

# Check that elastalert configs are available
if [ ! -f ${CONF_FILE} ]; then
  echo "ELASTALERT CONFIGS NOT FOUND!"
  echo "Config dir query path: $CONF_FILE"
  echo "Exiting..."
  exit 1
fi

# Check Elasticsearch connection before index checking
while ! wget -q -T 3 -O - "http://${ELASTIC_HOST}:${ELASTIC_PORT}"
2>/dev/null
do
  echo "Waiting for Elasticsearch..."
  echo "Connecting to:   http://${ELASTIC_HOST}:${ELASTIC_PORT}"
  sleep 5
done
echo "Connected to Elasticsearch !"

# Query elastalert index from elasticsearch
echo "Quering elastalert index from Elasticsearch..."
if ! wget -q -T 3 -O - "http://${ELASTIC_HOST}:${ELASTIC_PORT}/${ELAS-
TIC_INDEX}" 2>/dev/null
then
  echo "Creating index to Elasticsearch data..."

  elastalert-create-index \
    --host "${ELASTIC_HOST}" \
    --port "${ELASTIC_PORT}" \
    --config "${CONF_FILE}" \
    --index "${ELASTIC_INDEX}"

  echo "Elastalert index created."
else
  echo "Elastalert index already exists in Elasticsearch."
fi

echo "Starting Elastalert..."
echo "Elasticsearch IP: http://${ELASTIC_HOST}:${ELASTIC_PORT}"
echo "Elastalert config file:  $CONF_FILE"
cd /opt/elastalert && python elastalert/elastalert.py --config
${CONF_FILE}
```

Liite 2. Elastalert Docker lähdetiedostot

Dockerfile

```
ARG ARTIFACTORY_REGISTRY
FROM ${ARTIFACTORY_REGISTRY}/path/to/centos:7.4.1708
LABEL maintainer="Lauri Pudas"

# Timezone
ENV TZ=Europe/Helsinki
# Create working directory
RUN mkdir -p /opt
WORKDIR /opt

# Install and update yum packages
# Fetch, unzip and rename elastalert --- v0.1.28 ---
# Set timezone
RUN yum -y install epel-release && \
  yum update -y && \
  yum -y install python-devel python2-pip wget unzip gcc && \
  yum -y remove epel-release && \
  yum clean all && \
  rm -rf /var/cache/yum && \
  wget -O elastalert-master.zip https://github.com/Yelp/elastalert/ar-
chive/4eae2e27bd95e278d50f54acfdf6ee756450a072.zip && \
  unzip elastalert*.zip && \
  rm elastalert*.zip && \
  mv elastalert* elastalert && \
  ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && \
  echo $TZ > /etc/timezone

# Install elastalert
WORKDIR /opt/elastalert
RUN pip install "setuptools>=11.3" && \
  python setup.py install && \
  pip --no-cache-dir install -e .

# Certificates
COPY certs/Your_ceritificate_file.crt /etc/ssl/certs/
ENV REQUESTS_CA_BUNDLE=/etc/ssl/certs/Your_ceritificate_file.crt

ENTRYPOINT ["sh", "/opt/elastalert-files/start-elastalert.sh"]
```

docker-compose.yml

```yaml
---

version: '3'

services:
  elastalert:
    build:
      context: elastalert
      args:
        ARTIFACTORY_REGISTRY: ${ARTIFACTORY_REGISTRY}
    image: ${ARTIFACTORY_REGISTRY}}/PATH/TO/FOLDER/elastalert:0.1.28-2
```

Liite 3. Elastalert Jenkins lähdetiedosto

Jenkinsfile

```
#!groovy

properties([
  parameters([
    booleanParam(name: 'uninstallElastalert', defaultValue: false, de-
scription: 'Check this if you want to UNINSTALL elastalert'),
    booleanParam(name: 'installElastalert', defaultValue: true, descrip-
tion: 'Check this if you want to INSTALL elastalert')
  ])
])

node() {

  // Verify that at least one option is selected before continuing
  def installTags = ""
  if (!params.installElastalert && !params.uninstallElastalert) {
    error message: """
        You must specify at least one option.
        Uninstall and install can be defined at same time.
        If both options are defined uninstall will be executed first.
        """
  } else if (params.installElastalert && !params.uninstallElastalert) {
    installTags = "install"
  } else if (!params.installElastalert && params.uninstallElastalert) {
    installTags = "uninstall"
  } else {
    installTags = "install,uninstall"
  }

  builderStage('Clone YOUR_REPO_HERE') {
    hg('YOUR_REPO_HERE.SUB_REPO', 'master')
  }

  ansibleStage('Deploy or Uninstall elastalert') {
    def execPath = "YOUR_REPO_HERE.SUB_REPO/PATH/TO/ANSIBLE/FOLDER"
    sh "cd $execPath && \
        ansible-playbook -i inventories/YOUR_INVENTORY/ install-unin-
stall-elastalert.yml -t $installTags"
  }
}
```