jamk.fi

Klusterin metriikka- ja lokipohjainen monitorointi

Timo Huttunen

Opinnäytetyö
Marraskuu 2020
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Huttunen, Timo	Opinnäytetyö, AMK	12 2020
	Sivumäärä 63	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Klusterin metriikka- ja lokipohjainen monitorointi

Tutkinto-ohjelma
Tietotekniikan tutkinto-ohjelma

Työn ohjaaja(t)
Häkkinen Antti, Huotari Jouni

Toimeksiantaja(t)
Timo Huttunen

Tiivistelmä

Opinnäytetyön päätavoitteena oli syventää monitorointiosaamista ja kasvattaa ammattitaitoa monitorointijärjestelmistä teoriapohjalta ja käytännössä. Sekä oppia tuntemaan käytettyjä työkaluja kuten Prometheus, Grafana sekä Elasticsearch, sekä niihin liittyvistä lisäosista ja ominaisuuksista.

Tutkimuksessa käytiin läpi mitä monitorointi on teoriassa sekä sen osa-alueita, kuten metriikat ja lokit, mitä ne merkitsevät ja mitä niillä pystytään tekemään monitorointijärjestelmässä. Sekä miten hälytykset vaikuttavat ympäristön aktiiviseen ja passiiviseen valvontaan.

Työssä käytettiin kahta toteutustapaa, joilla lähestyttiin tavoitteita. Työssä esitetään teoriaa työkalujen sekä yleisen monitoroinnin osalta ja käytännönosuudessa, jossa teoreettisen tutkimuksen osa-alueita otettiin käytännössä käyttöön ympäristössä.

Tuloksena syntyi toimiva aktiivinen monitorointiratkaisu seitsemän palvelimen klusterille, jossa on toimiva visualisointi, joka mahdollistaa palvelimien yhtenäisen sekä yksittäisen tarkastelun ja valvomisen metriikkapohjaisena ratkaisuna. Sovelluksien lokiviesteihin perustuvan monitoroinnin. Sekä passiivisen monitoroinnin hälytyksien muodossa klusterin palvelimien resurssien käytölle. Sovelluksien erilaisten lokiviesteihin pohjautuvan hälytys ratkaisun.

Yhteenvetona tavoitteissa onnistuttiin kohtalaisen hyvin. Sekä tietämys että monitorointi osaaminen kasvoi työn aikana ja saimme aikaiseksi tavoitteiden mukaisen monitorointijärjestelmän klusterin metriikka- ja lokipohjaiselle monitoroinnille.

Avainsanat (asiasanat)

Monitorointi, Prometheus, Grafana, Elasticsearch

Muut tiedot (Salassa pidettävät liitteet)
-

jamk.fi

Description

Author(s)	Type of publication	Date
Huttunen, Timo	Bachelor's thesis	12 2020
		Language of publication:
	Number of pages 63	Permission for web publication: x

Title of publication
Metric- and log-based cluster monitoring

Degree programme
Information Technology

Supervisor(s)
Häkkinen Antti, Huotari Jouni

Assigned by
Huttunen Timo

Abstract

The main goal of the thesis was to deepen monitoring skills and increase professional skills in monitoring systems in theory and in practice. Secondary objective was to increase knowledge about the tools used such as Prometheus, Grafana and Elasticsearch, as well as related add-ons and features.

The study went through what monitoring is in theory as well as its components such as metrics and logs. What they mean and what they can do in a monitoring system and how alerts affect the active and passive monitoring of the system.

Two implementation methods were used in the work to approach the objectives. The implementation methods were theoretical research in terms of tools as well as general monitoring. The practical part, in which the aspects of theoretical research were put into practice in the environment to achieve the desired result.

The result was a working active monitoring solution for a cluster of seven servers with functional visualization that enabled unified as well as individual examination and monitoring of servers as a metric-based solution and applications as a log-based solution. Passive monitoring in form of alerts for cluster resources and as a log-based solution for applications.

In summary, the objectives were moderately successful, both knowledge and monitoring skills increased during the work, and we achieved a monitoring system in line with the objectives for metric- and log-based monitoring of the cluster.

Keywords/tags (subjects)
Monitoring, Prometheus, Grafana, Elasticsearch

Miscellaneous (Confidential information)

Sisältö

Kuviot

Sanasto

PaaS **(Platform as a Service)** eli sovellusalusta palveluna tarkoittaa nimensä mukaisesti alustapalvelua ohjelmistojen kehittämiseen, testaamiseen ja julkaisuun

AWS **(Amazon Web Service)** Amazonin tarjoama pilvipalvelu alusta

tsdb **(Time Series Database)** Prometheuksen käyttämä aikasarjatietokanta

PromQL **(Prometheus Query Language)** Prometheuksen oma kysely kieli

DQL **(dead letter queue plugin)** Logstashin käyttämä lisäosa. Tätä käytetään tapauksissa, jossa Logstash kohtaa kartoitusvirheen tai jonkin muun ongelman. Tällöin se säilöö ongelmallista tietoa tähän jonoon odottamaan prosessointia.

1 Johdanto

1.1 Tavoitteet

Opinnäytetyön tavoitteena oli kasvattaa omaa tietämystä klusterien monitoroinnista ennalta valituilla työkaluilla. Työkalut valittiin sen perusteella mitä nykyisellä työnantajalla on käytössä oman osaamisen ja ammattitaidon kasvattamiseksi. Työssä oli tavoitteena käydä teoria pohjalta, mitä on monitorointi ja työssä käytettävät työvälineet. Toteutusosiossa luodaan toimiva klusterin loki- ja metriikkapohjainen monitorointi hälytyksineen testiympäristöön.

1.2 Tutkimusmenetelmät

Tutkimusmenetelmänä työssä käytettiin pääasiallisesti empiiristä tutkimusta. Työn alkupuolella tutustaan ensin klustereiden metriikka- ja lokipohjaiseen monitorointiin tietopohjaisesti. Tällä haluttiin saada ymmärrys aiheesta, joka saavutettiin erilaisten aiheeseen liittyvän kirjallisuuden sekä oppaiden avulla.

Toteutusosiossa ennalta valikoiduilla monitorointivälineillä suoritettiin testiympäristöön työkalujen asennus sekä niiden asetusten määrittäminen toimivan monitorointijärjestelmän aikaansaamiseksi.

1.3 Työn rakenne

Työn tietoperustassa käydään ensin läpi yleisesti monitorointia, metriikoita, lokitusta, sekä niihin käytettäviä työkaluja. Tämän jälkeen käymme lyhyesti läpi ympäristön rakennetta. Viimeisenä vaiheen on itse toteutus, jossa ympäristöön asennetaan monitorointityökalut sekä suoritetaan niiden asetusten muokkaaminen ja testataan toimivuus. Toteutusvaiheen loppupuolella tehdään visualisointeja, sekä hälytyksiä kuvaamaan ympäristön tilaa monitorointi työkaluilla.

2 Monitorointi

Monitorointi yleisesti käsitetään, jonkun asian valvonta tämä pätee myös IT-ympäristöihin. Monitorointi varmistaa ylläpidettävän järjestelmän toiminnan ja antaa nopeat mahdollisuudet toimia vikatilanteissa. Monitorointi yleensä jaotellaan kahteen alaryhmään eli metriikka- ja lokipohjaiseen monitorointiin, joita käydään teoreettisesti läpi tässä osiossa.

Metriikat ja lokit edustavat järjestelmän tietoja. Monitorointi on näiden arvojen keräämisen, yhdistämisen ja analysoinnin prosessi järjestelmän ominaisuuksien ja käyttäytymisen parantamiseksi. Tiedot ympäristön eri osista kerätään monitorointijärjestelmään, joka vastaa tietojen tallennuksesta, visualisoinnista ja hälytyksistä, kun metriikoista saatavat arvot ylittävät tietyt rajat. (Justin Ellingwood 2017)

2.1 Monitorointijärjestelmä

Metriikat, lokit ja hälytykset ovat monitorointijärjestelmän keskeisiä osia. Metriikat ja lokit ovat syötteitä, raaka dataa, joita tarvitaan suorituskyvyn ja saatavuuden seurantaan. Monitorointi on yleisesti se, minkä päälle hälytysjärjestelmä on rakennettu. Yhdessä ne antavat käsityksen sovellusten ja infrastruktuurin toiminnasta. Ne havaitsevat suorituskyvyn tai käytön poikkeavuuksia, auttavat vianetsinnässä ja ongelmien tunnistamisessa, paljastavat käyttö- ja käyttäytymismalleja tai trendejä, jotka auttavat ymmärtämään sovelluksiin tai infrastruktuuriin tekemiesi muutosten vaikutuksen. (Sematext 2020)

2.2 Metriikat

Metriikat edustavat resurssien käytön tai käyttäytymisen mittauksia, jotka voidaan havaita ja kerätä koko järjestelmästä. Nämä voivat olla käyttöjärjestelmän tarjoamia matalan tason yhteenvetoja, tai ne voivat olla korkeamman tason dataa, jotka on sidottu komponentin tiettyyn toimintoon, kuten esimerkiksi kuinka paljon pyyntöjä sekunnissa komponentti on saanut. (Justin Ellingwood 2017)

Usein metriikoiden monitorointi on helpointa aloittaa niistä metriikoista, jotka käyttöjärjestelmä on jo paljastanut kuvaamaan taustalla olevien fyysisten resurssien käyttöä. Näitä ovat tiedot levytilasta, suorittimen kuormituksesta, muistin käytöstä jne. Nämä metriikat ovat yleisesti heti tarjolla ja antavat arvoa välittömästi. Yleisesti nämä voidaan välittää monitorointijärjestelmään ilman suurempaa vaivaa. Monet verkkopalvelimet, tietokantapalvelimet ja muut ohjelmistot tarjoavat myös omat metriikat, joita voidaan helposti tutkia. Muille komponenteille, etenkin omille sovelluksille, yleensä joudutaan lisäämään ohjelmoimalla tai käyttöliittymällä keinot paljastaa metriikat järjestelmälle. (Justin Ellingwood 2017)

Metriikat ovat hyödyllisiä, koska ne antavat käsityksen järjestelmän käyttäytymisestä ja terveydestä, varsinkin kun niitä analysoidaan kokonaisuutena. Ne edustavat raaka-ainetta, jota monitorointijärjestelmä käyttää rakentamaan kokonaisvaltaisen kuvan ympäristöstä, ja hälyttämään ihmisiä tarvittaessa. Metriikat ovat perusarvoja, joita käytetään pitkän aikavälin muutoksien tutkimiseen ja suorituskyvyn, kulutuksen tai virhetasojen muutosten mittaamiseen. (Justin Ellingwood 2017)

2.3 Lokit

Loki on järjestelmän luoma joukko dataa, joka kuvailee mitä tapahtumassa on tapahtunut. Lokiviesti sisältää lokitiedot, jotka ovat tapahtuman yksityiskohtia, kuten resurssi, jota käytettiin, kuka sitä käytti, ja aika, jolloin sitä käytettiin. Jokaisessa järjestelmän tapahtumassa tulee erilaista lokitietoa, joka kuvailee mitä on tapahtunut ja milloin. (Dan Reichert 2018)

Lokit antavat ylimääräisiä yksityiskohtia, joita tarvitaan vianmääritykseen, virheenkorjaukseen, tukeen ja auditointiin. Lähtökohtana saatat tietää monitorointijärjestelmästä tai hälytyksistä, että ongelma on olemassa ja milloin se alkoi. Tämä tieto ei usein ole riittävä ja niinpä, näissä tilanteissa yleensä joudutaan tarkastelemaan lokitietoja syvemmältä, jotta nähdään tarkalleen, mitä tapahtui ongelma tilanteen aikana ja missä. (Splunk 2016)

2.4 Hälytykset

Hälytys on komponentti monitorointijärjestelmässä, joka suorittaa toimenpiteitä metriikka datan muutosten perusteella. Hälytysten määritelmät koostuvat kahdesta osasta metriikkaan perustuvasta ehdosta tai raja-arvosta ja toiminnosta, joka suoritetaan, kun arvot jäävät hyväksyttävien arvojen ulkopuolelle. (Justin Ellingwood 2017)

Vaikka monitorointijärjestelmät ovat uskomattoman hyödyllisiä aktiivisessa tutkinnassa. Yksi hyvän valvontajärjestelmän ensisijaisista eduista on mahdollistaa järjestelmänvalvojien irroittautua järjestelmästä. Hälytysten avulla voit määritellä tilanteita, joita on järkevää hallita aktiivisesti, samalla kun luotat ohjelmiston passiiviseen seurantaan muuttuvien tapahtumien seuraamiseksi. (Justin Ellingwood 2017)

Hälytyksen päätarkoitus on tuoda ihmisten huomio järjestelmän nykyiseen tilaan. Varoituksen itsessään tulisi olla tietoa siitä, mikä on vialla ja mistä mennä etsimään lisätietoja. Hälytystä tutkiva henkilö voi sitten käyttää monitorointijärjestelmää ja siihen liittyviä työkaluja, kuten lokitiedostoja ongelman syyn selvittämiseen ja korjaamiseen. (Sematext 2020)

Hälytykset on yleensä jaoteltu tasoihin esimerkiksi info, varoitus tai kriittinen, joka mahdollistavat ongelman laajuuden helpon arvioinnin. Esimerkiksi lisääntynyt tallennustilan käyttö saattaa vaatia tiketin tai sähköpostin, kun taas virheiden lisääntyminen asiakkaihin liittyvissä palveluissa edellyttää nopeita toimenpiteitä päivystäville henkilöille. (Atlassian 2020)

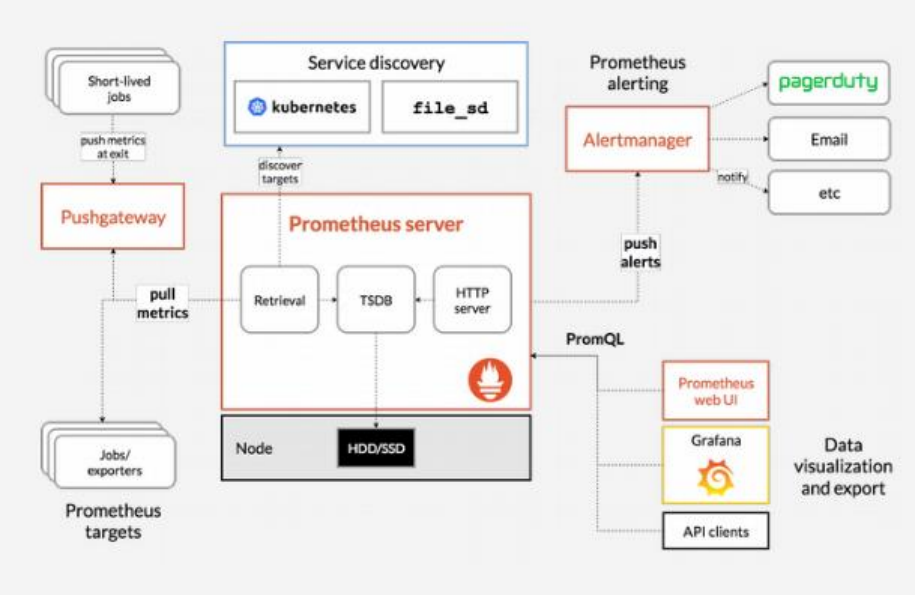
3 Työkalut

3.1 Prometheus

SoundCloud yrityksen tekemä Prometheus on monipuolinen avoimen lähdekoodin monitorointiin sekä hälytyksiin keskittyvä työkalu. Prometheus keskittyy lähinnä monitoroimaan mitä tapahtuu tällä hetkellä, kuin että mitä on tapahtunut viikkojen tai kuukausien aikana. Tämän vuoksi Prometheus onkin varsin mainion työkalu monitoroimaan hetkellistä tilannetta aina pienemmistä ympäristöistä suuriin palvelinkeskuksiin.

Prometheuksen toiminta perustuu aikasarjatiedon hakemiseen, joita kohteet paljastavat HTTP-rajapinnoilla. Tämä tapahtuu yleensä asiakaskirjastojen tai erilaisten exporter-palveluiden avulla. Kohde rajapinnat, joista Prometheus hakee tietoa voivat olla esimerkiksi prosesseja, palveluita, sovelluksia tai palvelimia. (John Turnbull 2020)

Seuraavassa kuviossa 1 käydään läpi Prometheuksen perusarkkitehtuuri ja miten sen osat toimivat keskenään. Tässä näkyy kuinka Prometheus server noutaa tietoa Prometheus kohteilta, jotka se varastoi tietokantaan. Tietokannasta niitä voidaan hakea PromQL-kielellä (Prometheus Query Language), joka Prometheuksen oma kysely kieli. Prometheus server on myös vastuussa hälytysten puskemisesta Alertmanager-komponentille, joka sitten voi edelleen lähettää niitä muille ohjelmille, kuten tässä työssä käytettävä Karma.



Kuvio 1 Prometheus arkkitehtuuri (Opensource Prometheus)

3.2 Prometheus exporter-lisäosat

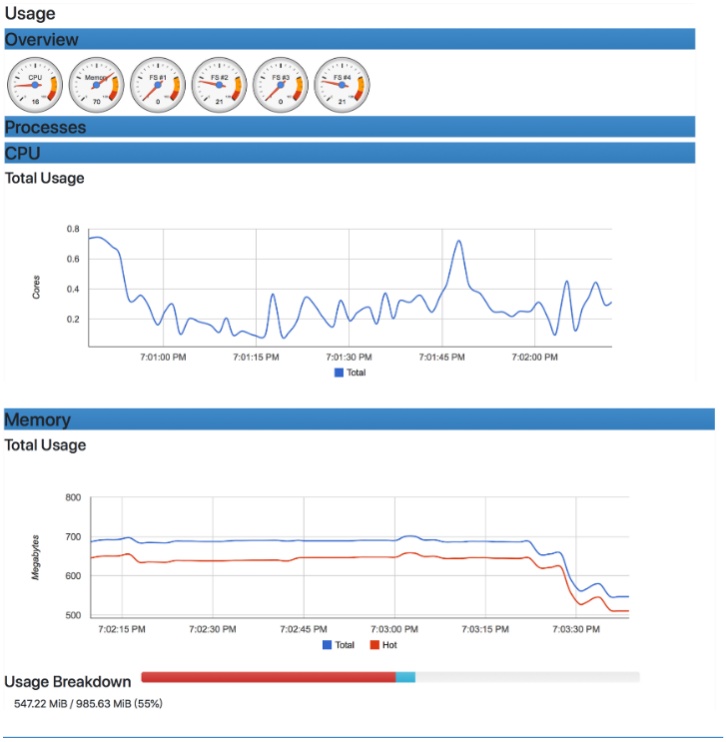
Prometheus exporter-lisäosat ovat monipuolisia lisäosia Prometheus monitorointiin. Exporter-lisäosat ovat kustomoitavia datan lähettäjiä, jotka yleisesti asennetaan kohdepalvelimille missä data sijaitsee keräämään metriikkaa ja tarjoamaan rajapinta, josta Prometheus voi niitä noutaa. Exporttereita on tarjolla Prometheuksen toimesta sekä yhteisön tekeminä moniin eri kategorioihin kuten esimerkiksi tietokantoihin, laitteistoon ja lähes kaikkeen mitä yleisesti monitoroidaan. Tässä työssä käytetään pääasiallisesti kahta Prometheus exportteria, jotka ovat seuraavat:

Node-exporter on yksi Prometheuksen perinteisistä data lähettäjistä, jota hyödynnetään tässäkin työssä. Node-exporter on tehty lähettämään monipuolisesti dataa Linux laitteistosta sekä kernelistä. Node-exportterin avulla saadaan tietoa liittyen palvelimen resurssien käyttöön, sekä sinne asennetuista prosesseista.

- **Elasticsearch- ja es-exporter** on toinen tärkeä tässä työssä käytetty lähettäjä, joka mahdollistaa Elasticsearch datan noutamiseen Prometheukseen ja Prometheuksella Elasticsearch lokien etsimisen ja sen muuttamisen metriikaksi, joka mahdollistaa loki pohjaisten hälytyksien tekemisen kuten työn toteutus osiossa demonstroidaan.

3.3 cAdvisor

cAdvisor kerää resurssien käyttöä, suorituskykyominaisuuksia ja niihin liittyviä tietoja ympäristössä toimivista Docker konteista. Se on Googlen tekemä avoimen lähdekoodin työkalu, joka auttaa Docker kontteihin liittyvien hyödyllisten tietojen keräämisessä, käsittelyssä ja yhdistämisessä. Työkalulla on natiivi tuki Dockerille ja sen avulla voi seurata Docker konttien resurssien käyttöä pitkältä aikaväliltä, histogrammeilla ja muilla työkaluilla. Esimerkkinä seuraavassa kuviossa 2 cAdvisorin tarjoama graafinen käyttöliittymä, jossa on näkyvissä erilaisia hetkellisiä mittareita sekä pidemmän aikavälin tietoja.

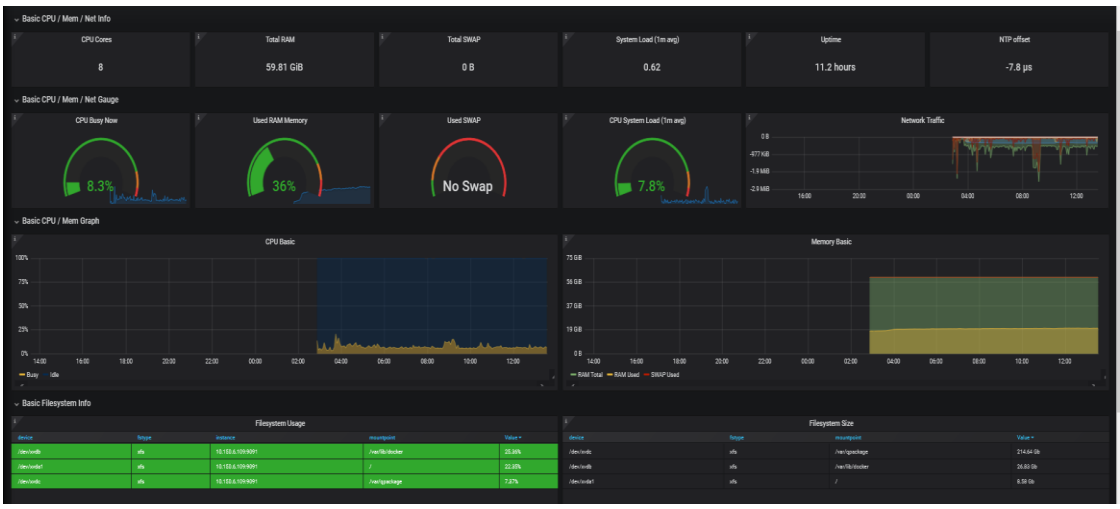


Kuvio 2 cAdvisor käyttöliittymä

3.4 Grafana

Grafana on avoimenlähde koodin datan visualisointiin ja analysointiin erikoistunut ohjelma Grafana Labs yritykseltä. Grafana antaa mahdollisuuden tiedustella, visualisoida, tutkia ja luoda hälytyksiä riippumatta siitä missä data sijaitsee. Toisin sanoen se tarjoaa työkalut aikasarjatietokanta datan muuttamiseksi hienoiksi graafeiksi sekä visualisoinneiksi. (Grafana 2020. What is Grafana 2020)

Grafana on tällä hetkellä yleisin visualisointi ohjelma Prometheus datan näyttämiseen ja niitä suositellaankin käytettäväksi yhdessä. Grafana mahdollistaa monien eri data lähteiden käyttämisen metriikka datan visualisointiin ja jopa hälytyksien tekemiseen. Seuraavassa esimerkki kuviossa 3 on visualisoitu Prometheus Node-exporter-lisäosan tarjoamaa perustason palvelin metriikkaa, josta on luotu erilaisia näkymiä prosessorin, muistin ja levynkäytöstä.



Kuvio 3 Grafanan esimerkki näkymä.

3.5 Elasticsearch

Elasticsearch on Apache Lucenella tehty avoimen lähdekoodin haku ja analysointi työkalu kaiken tyyppiselle datalle. Elasticsearch on tunnettu sen yksinkertaisista REST-sovellusliittymistään, hajautetusta luonteestaan, nopeudestaan ja skaalautuvuudesta. Se on keskeinen osa Elasticstack-työkaluja, joka on tietojenkeräämiseen, tallentamiseen, analysointiin ja visualisointiin kehitetty kokoelma ohjelmia paremmin tunnettuna nimellä ELK Stack, jonka nimi muodostuu sen pääkomponenttien nimistä Elasticsearch, Logstash ja Kibana. (Elasticsearch 2020 What is Elasticsearch)

Elasticsearchiä käytetään pääasiallisesti lokitietojen keskitettyyn keräämiseen ympäristöissä. Elasticsearchilla kerätään tietoja Beats-lisäohjelmien avulla, näitä lisäohjelmia on luotu moniin eri tarkoituksiin. Pääasiallisia näistä ovat Filebeat, joka lukee tiedostoja esimerkiksi lokitiedostoja ja lähettää siinä olevat tiedot eteenpäin suoraan Elasticsearchiin tai Logstashiin lisämuokkausta varten. Auditbeat, jonka käyttötarkoitus on kohde järjestelmän auditointi eli se lukee sisäänkirjautumisia, tiedostomuu-

toksia sekä vastaavia toimenpiteitä. Muita mainintoja Beats-ohjelmista ovat Metricbeat, joka tarjoaa metriikka dataa kohteista Prometheuksen kaltaisesti sekä Packetbeat, joka on puolestaan keskittynyt verkkotoiminnan tietojen keräämiseen.

Elasticsearch tallentaa saapuneet tiedot Index-nimisiksi kokonaisuuksiksi, joita voidaan nimetä ja määritellä erilaisin Logstash tai Filebeat tunnisteilla perustuen esimerkiksi mistä tiedostopolusta ne ovat kerätty, miltä palvelimelta ja minkä tyyppisiä tiedostoja ne ovat. Vastaavia määrittelyjä on mahdollista suorittaa myös niistä saadun tiedon lajittelemiseksi. Elasticsearch jakaa nämä indeksit vielä osiksi, joita kutsutaan nimellä Shard. Shardit koostuvat primääreistä sekä replikoista. Primäärit sisältävät alkuperäisen datan ja replikat sisältävät varmuuskopion alkuperäisestä.

3.6 Logstash

Logstash on osa Elasticsearch kokonaisuutta sen pääasiallinen tarkoitus on toimia eräänlaisena tietojenkäsittelyputkena, joka hankkii tietoa monista lähteistä, muuntaa ne ja lähettää ne käytettyyn varastoon. Yleisesti tämä varasto on Elasticsearch mutta Logstash tukee muitakin varasto vaihtoehtoja tiedon lähetykseen. (Elasticsearch 2020 Logstash)

Logstash käsittelee, muuntaa ja toimittaa tiedot dynaamisesti muodosta tai monimutkaisuudesta riippumatta. Tiedon rakenne joka Logstashin läpi kulkee, voidaan halutessa muokata grok-menetelmällä, josta lokitiedosta voidaan poistaa, muokata tai lisätä siihen tietoa sisältäviä kenttiä. Nämä kentät voivat olla IP-osoitteita tai palvelin nimiä tai esimerkiksi lokitiedon tyyppi. Tämä mahdollistaa niiden käyttämisen erilaisissa haku toiminnoissa Kibana-ohjelmalla tai muilla visualisointi työkaluilla. (Elasticsearch 2020 Logstash)

3.7 Kibana

Kibana on graafinen käyttöliittymä Elasticsearch palvelun hallintaan sekä sieltä saatavan datan visualisointiin, se mahdollistaa loki ja metriikka datan etsimisen kätevästi ja mahdollistaa erilaisten graafien ja muiden visualisointien luonnin Elasticsearch datan pohjalta. Kibanaa käytetään yleisesti keskitettyyn lokiviestien lukemiseen tai erillaisten Elastic klusterin hallintaan liittyvien toimenpiteiden suorittamiseen, kuten indeksien poistamiseen. Sillä voidaan myös luoda kätevästi raportteja lokitietoihin perustuen esimerkiksi asiakas määristä tai tilauksista, joita lokitiedoissa ilmenee.

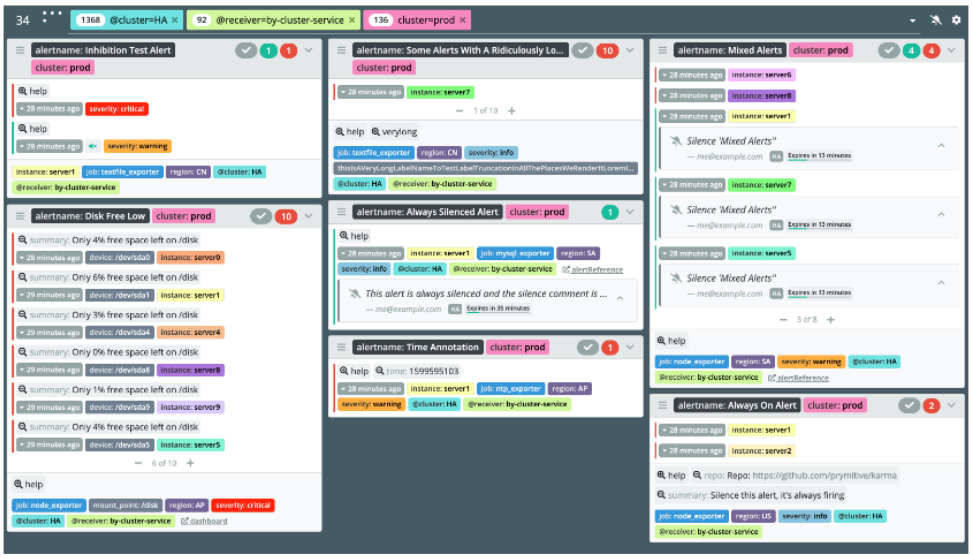
3.8 Alertmanager

Alertmanager käsittelee asiakasohjelmien, kuten Prometheus palvelimen lähettämät hälytykset ja huolehtii niiden kopioiden poistosta, ryhmittelystä ja reitittämisestä käytettyyn hälytys integraatioon kuten Karma, OpsGenie tai PagerDuty. Alertmanagerilla voidaan myös suorittaa hälytysten vaimentaminen ja estäminen. (Prometheus 2020 Alerting Alertmanager)

3.9 Karma

Karma on graafinen käyttöliittymä Prometheuksen hälytyksille ja antaa hyvän kuvan koko järjestelmän aktiivisista hälytyksistä ja tarjoaa samat työkalut kuin Alertmanager, mutta tarjoaa lisä arvoa kokonaiskuva näkymällä, joka auttaa isommissa ympäristöissä hälytysten hallitsemiseen ja visualisointiin. Seuraavassa kuviossa on esitetty

tästä esimerkki. Kuviossa 4 nähdään Karman tarjoama näkymä hälytysten visualisointiin.



Kuvio 4 Karma-näkymä

3.10 Ympäristön muut työkalut

Tässä osiossa käydään läpi ympäristön muut avustavat työkalut, jotka eivät suoranaisesti liity monitorointiin.

3.10.1 Consul

Consul on palveluverkkoratkaisu, joka tarjoaa täysin varustellut ohjaustason palvelujen löytämis-, määritys- ja segmentointitoiminnot. Kutakin näistä ominaisuuksista voidaan käyttää erikseen tarpeen mukaan tai niitä voidaan käyttää yhdessä täyden palvelun verkon rakentamiseen. (Consul 2020 Introduction to Consul)

Tässä työssä Consulia käytetään pääasiallisesti palvelujen yhdistämiseen toisiinsa sen tarjoaman palvelujen löytämistoiminnon (Service discovery) avulla. Consul tarjoaa

helposti sisäisen nimipalvelun Dnsmasq-ohjelman avulla sekä Registrator automaattisen rekisteröinnin Docker-konteille Consuliin. Rekisteröinnin jälkeen palvelut voivat käyttää Consul-osoitetta, joka on muotoa *palvelu.service.consul* toistensa kanssa kommunikointiin.

3.10.2 ZooKeeper

ZooKeeper on hajautettu koordinointipalvelu, jolla hallitaan suuria määriä palvelimia. Palvelun koordinointi ja hallinta hajautetussa ympäristössä on yleensä monimutkainen prosessi, johon ZooKeeper tarjoaa ratkaisun. Yksinkertaisella arkkitehtuurillaan ja rajapinnalla varustettu ZooKeeper antaa kehittäjille mahdollisuuden keskittyä itse sovelluksiin ilman että tarvitsee huolehtia miten ne on ympäristöön sijoitettu. (Tutorials Point n.d)

ZooKeeperin perustan on rakentanut alun perin Yahoo! Se kehitettiin helpottamaan sovelluksiin pääsemistä. Myöhemmin Apache ZooKeeperistä tuli standardi organisoiduille palveluille, jota Hadoop, HBase ja muut hajautetut alustat käyttivät. Esimerkiksi Apache HBase käyttää ZooKeeperia seuraamaan jaetun datan tilaa. (Tutorials Point n.d)

Tässä työssä ZooKeeperin rooli on pienempi ja sitä käytetään lähinnä Mesos/Marathonin vaatimana lisäosana suorittamaan sen master-/slave-valinnat ja koordinoimaan Marathonin päälle käynnistettävät sovellukset.

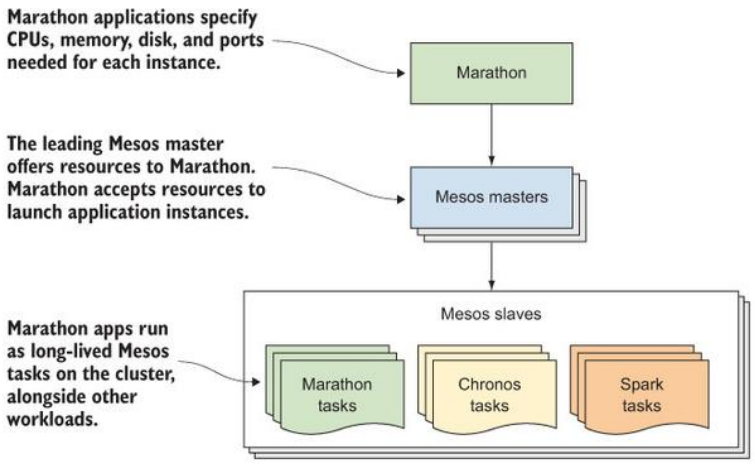
3.10.3 Mesos/Marathon

Mesos on klusterin hallintatyökalu, joka tarjoaa tehokkaan resurssien eristämisen ja jakamisen hajautettujen sovellusten tai alustojen kesken. Mesos on avoimen lähdekoodin ohjelmisto, joka on alun perin kehitetty Kalifornian yliopistossa Berkeleyssä.

Se on sovelluskerroksen ja käyttöjärjestelmän välissä ja helpottaa sovellusten käyttöönottoa ja hallintaa tehokkaammin suurissa ympäristöissä. (Sachin P Bappalige 2014)

Mesos hyödyntää moderneja kernelin ominaisuuksia kuten cgroup Linuxissa ja zonet Solariksessa eristääkseen prosessorin, muistin, tiedostojärjestelmän ja muita laitteisto tason resursseja. Ajatus Mesoksen takana on yhdistää ympäristön resurssit monelta palvelimelta ja jakaa niitä tarpeen mukaan. Mesos päättää, kuinka paljon resursseja jokaiselle kehykselle tarjotaan ja kehykset päättävät, mitkä resurssit hyväksytään ja mitä niillä tehdään. (Sachin P Bappalige 2014)

Marathon on sovellusten orkestrointijärjestelmä Mesoksen päällä. Marathon pyörittää sovelluksia Mesos-tehtävinä sekä Linux-alustan cgroup että Docker konteissa. Sitä voidaan pitää myös yksityisenä sovellusalusta palveluna (PaaS), johon sovelluksia voidaan ottaa käyttöön. Marathon tekee tämän käynnistämällä sovelluksen Mesos-tehtävinä, kuten voidaan nähdä kuviosta 5. (Roger Ignazio 2016)



Kuvio 5 Mesos/Marathon-sovelluksen asennuksen kulku

Marathon antaa sinun määrittää tarvittavat resurssit kullekin sovellukselle ja kuinka monta instanssia sitä asennetaan. Samoin kuin nykyaikaiset palvelunhallinta työkalut, kuten systemd ja Upstart, Marathon käynnistää uudelleen epäonnistuneet tehtävät automaattisesti, käytettävissä olevien klusteriresurssien avulla. Jos ympäristössä oleva palvelin esimerkiksi kaatuu tai sovellus epäonnistuu vastaamaan kutsuihin, Marathon aloittaa automaattisesti uuden instanssin korvaamaan epäonnistuneen sovelluksen toiselle palvelimelle, missä on vapaita resursseja. (Roger Ignazio 2016)

3.10.4 Docker

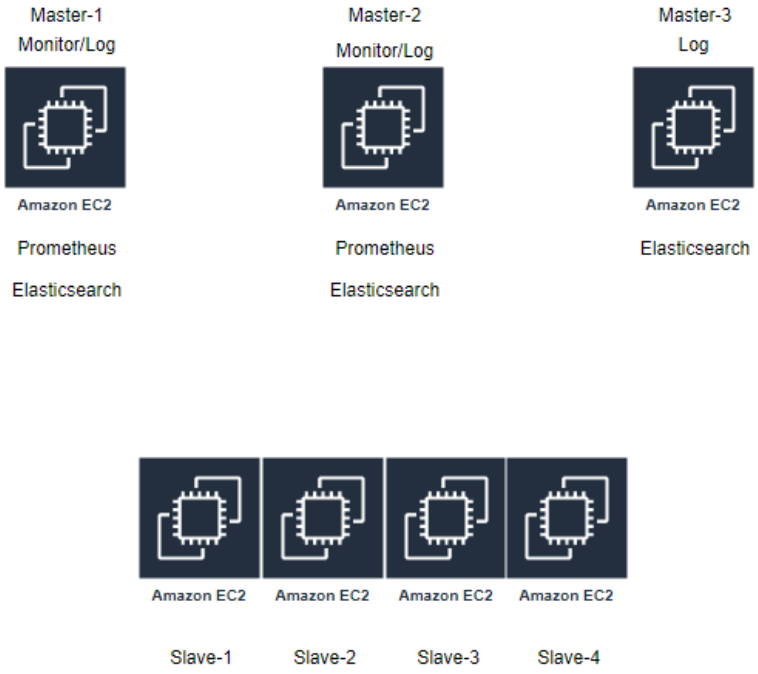
Docker on työkalu, joka on suunniteltu helpottamaan sovellusten luomista, käyttöönottoa ja ajamista konttien avulla. Konttien avulla kehittäjä voi pakata sovelluksen kaikki tarvitsemat osat, kuten kirjastot ja muut riippuvuudet, ja ottaa sen käyttöön yhtenä pakettina. Tekemällä Docker-kontin sovelluksesta kehittäjä voi olla varma, että se toimii missä tahansa muussa Linux-koneessa riippumatta koneen mahdollisesti mukautetuista asetuksista, jotka saattavat poiketa koodin kirjoittamiseen ja testaamiseen käytetystä koneesta. (Opensource n.d What is Docker)

Tavallaan Docker on kuin virtuaalikone. Mutta toisin kuin virtuaalikone, Docker antaa kokonaisen virtuaalisen käyttöjärjestelmän luomisen sijaan sovelluksille mahdollisuuden käyttää samaa Linux-ydintä kuin järjestelmässä, jossa he kehittivät sovelluksen. Tämä vaatii vain, että sovellukset toimitetaan sellaisten riippuvuuksien kanssa, jotka eivät ole käynnissä palvelimella missä konttia ajetaan. Tämä lisää merkittävästi suorituskykyä ja pienentää sovelluksen kokoa. (Opensource n.d What is Docker)

4 Ympäristö

Testiympäristönä työssä toimii pilvipalvelu AWSn (Amazon Web Service) tarjoamat EC2-instanssit, joita on yhteensä seitsemän kappaletta. Näillä koneilla käyttöjärjestelmänä toimii Centos 7 ja niillä ei ole käyttöjärjestelmän lisäksi asennettuina muita valmiita Amazonin tarjoamia komponentteja.

Tarkoituksen on rakentaa toimiva palvelintason metriikkapohjainen monitorointi ja lokipohjainen sovellusmonitorointi testiympäristöön. Tarkoituksena on tehdä kuviossa 7 esitetyn arkkitehtuurikuvion mukainen järjestelmä, joka sisältää kolme Master-palvelinta joihin, on myös sijoitettu monitorointi- ja lokityökalut, pääasiassa Prometheus, Grafana sekä Elasticsearch. Tämä malli on suositusten mukainen korkean saatavuuden sekä vikasietoisuuden vuoksi. Suositukset näille ovat kaksi kappaletta Prometheus-instansseja ja kolme kappaletta Elasticsearch-instansseja. Elasticsearch-instanssien suositukset voivat vaihdella riippuen minkä roolin Elasticsearch-palvelimia asennetaan, koska Elasticsearch mahdollistaa erilaisten roolien käytön sen asennuksissa. Esimerkiksi master, ingest tai data ovat yleisimpiä roolivaihtoehtoja. Pienissä ympäristöissä kuten tämä palvelimilla yleisesti on kaikki roolit, mutta suuremmissa tuotantoympäristöissä on hyvä erottaa niiden tehtäviä esimerkiksi tiedon säilyttämiseen ja tiedon käsittelyyn omat Elasticsearch-palvelimet.

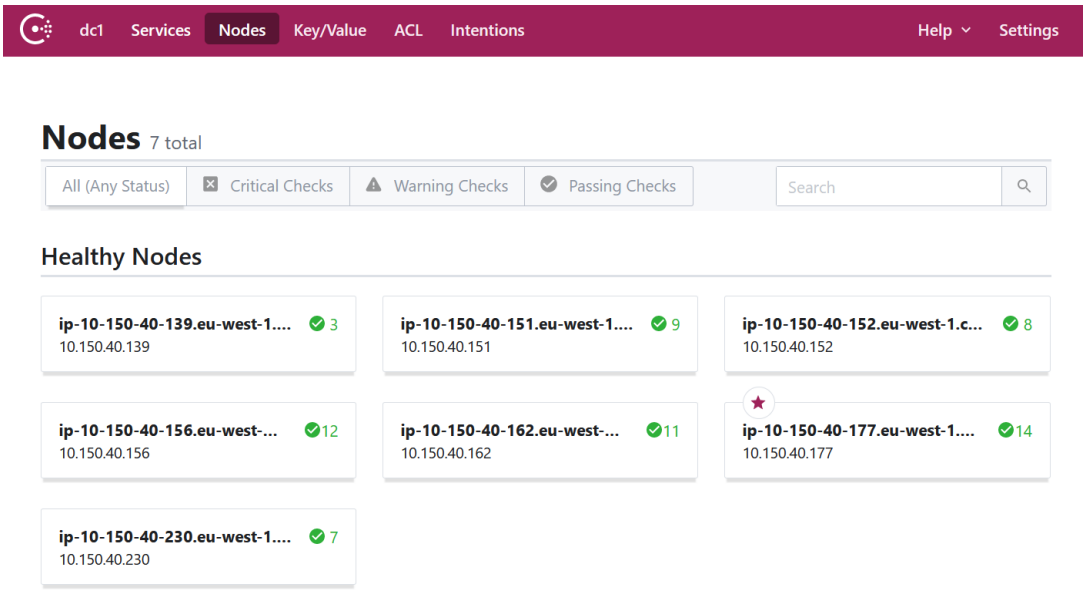


Kuvio 6 Monitorointiarkkitehtuuri AWS-alustalla

5 Toteutus

Koska Toteutus keskittyy monitorointiin, emme käy lävitse juuri muiden työkalujen asennus vaiheita vain keskitymme itse monitorointityökaluihin. Monitorointi työkaluissa on käytetty palveluntarjoajien omia Docker-paketteja, jotka on nimetty opinnäytetyöhön liittyen kuten useasta Dockerin käynnistyskomennosta ilmenee.

Ennen monitorointityökaluja ympäristöön on suoritettu muiden avustavien työkalujen asennus kuten Consul, jonka graafisesta käyttöliittymästä voimme tarkastella jo klusteriin kuuluvia palvelimia kuten oheisesta kuviosta 8 voimme nähdä.



Kuvio 7 Consul Nodes

5.1 Prometheus-palvelun Asennus

Prometheus on asennettu konttina Dockerin päälle seuraavassa kuviossa esitetyllä Docker-komennolla.

```
docker run -d --restart=unless-stopped --log-driver json-file  --log-opt max-file=3 --log-opt max-size=15m -m 2g
--memory-swap 4g --cpu-shares=2048 --user="64990" --net="bridge" -p 10.150.40.177:9090:9090
-v /var/qpackage/prometheus2/data:/prometheus -v /etc/qpackage/prometheus2/conf:/etc/prometheus
-v /certs/prometheus:/etc/prometheus/certs -v /etc/prometheus -v /etc/prometheus/certs
-v /prometheus -e "SERVICE_CHECK_HTTP=/metrics" -e "SERVICE_NAME=prometheus2" -e "SERVICE_IP=10.150.40.177"
-e "SERVICE_TAGS=metrics,monitor-1_monitor_prometheus2" -e "SERVICE_CHECK_TIMEOUT=5s" -e "SERVICE_CHECK_INTERVAL=60s"
--name monitor-1_monitor_prometheus2 prometheusct:2.10.0 --config.file=/etc/prometheus/prometheus.yml
--storage.tsdb.path=/prometheus --web.enable-admin-api --storage.tsdb.retention.time=24h
--web.console.libraries=/usr/share/prometheus/console_libraries --web.console.templates=/usr/share/prometheus/consoles
--storage.tsdb.max-block-duration=43h --query.timeout=2m --web.enable-lifecycle
```

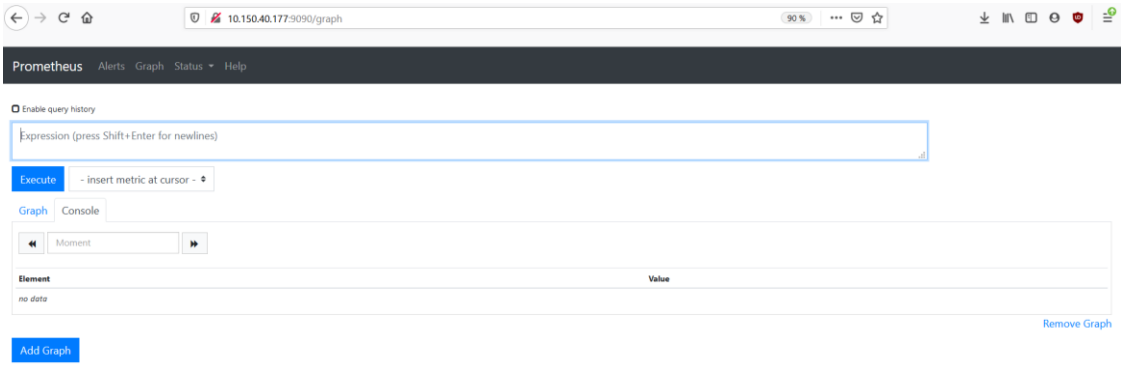
Kuvio 8 Prometheus Docker-käynnistyskomento

ässä komennossa on määritelty palvelimen ja kontin yhteiset tiedostokansiot sekä sille annettavat resurssit. Tällä komennolla myös määritetään IP-osoite, portti ja verkkotyyppi, Prometheus käyttää palvelimen IP-osoitetta ja porttia 9090. Komennossa myös asetetaan Prometheuksen käyttämän tsdb (Time Series Database) aikasarjatietokannan tiedostopolku ja kuinka kauan Prometheus säilyttää siellä dataa. Koska kyseessä on testaus ympäristö, tämä on asetettu 24 tuntiin. Prometheus-palvelun Docker-kontin käynnistys voidaan varmistaa kuvion 10 mukaisella komennolla, joka antaa vastaukseksi Docker-kontin tilan.

```
[root@ip-10-150-40-177 thuttunen]# docker ps | grep prometheus
abf3a0be1094        prometheusct:2.10.0-oppari                                      "/bin/prometheus --c…"   3 days ago
        Up 2 days             10.150.40.177:9090->9090/tcp                    monitor-1_Oppari_prometheus2
[root@ip-10-150-40-177 thuttunen]#
```

Kuvio 9 Prometheus-palvelun Docker tila

Tämän jälkeen voimme testata toimiiko Prometheuksen graafinen verkkokäyttöliittymä ottamalla yhteyden Docker run-komennossa määritettyyn IP-osoitteeseen ja porttiin. Kuten seuraavasta kuviosta 11 voidaan todeta Prometheus käyttöliittymään saatiin yhteys ja palvelu toimii. Asennusprosessi suoritettiin myös toiselle palvelimelle, korkean saatavuuden takaamiseksi.



Kuvio 10 Prometheus käyttöliittymä

5.2 Node-exporterin asennus ja asetukset

Prometheuksesta poiketen Node-exporter asennetaan suoraa palvelimelle Dockerin sijaan. Node-exporter-lisäosan asennukseen vaadittava paketti saadaan Github-palvelusta Prometheuksen virallisilta sivuilta käyttäen Linux-käyttöjärjestelmässä olevaa curl- komentoa. Seuraavan kuvion 12 mukaisesti curl-komento latasi paketin ja se näytettiin kansiossa.

```
[root@ip-10-150-40-177 thuttunen]# curl -LO https://github.com/prometheus/node_exporter/releases/download/v0.18.1/node_exporter-0.18.1.linux-amd64.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   661  100   661    0     0   2767      0 --:--:-- --:--:-- --:--:--  2765
100 7893k  100 7893k    0     0  7219k      0  0:00:01  0:00:01 --:--:--  201M
[root@ip-10-150-40-177 thuttunen]# ll
total 7896
-rw-r--r--. 1 root root 8083296 Nov 29 19:12 node_exporter-0.18.1.linux-amd64.tar.gz
```

Kuvio 11 Node-exporter-paketin noutaminen curl-komennolla

Tämän jälkeen käytettiin Linuxin tar-komentoa paketin purkamiseen kansioon /usr/bin/node_exporter. Tämän jälkeen luotiin konfiguraatiot tiedostoon /usr/lib/systemd/system/node_exporter.service perusasetuksia varten, jotta Node-

exporter-lisäosaa pystytään ajamaan palveluna systemd päällä. Kyseisen tiedoston sisältö näkyy kuviossa 13.

```
[Unit]
Description=Prometheus Node Exporter Service
After=network.target

[Service]
ExecStart=/usr/bin/node_exporter
User=prometheus
:
[Install]
WantedBy=multi-user.target
```

Kuvio 12 Node-exporterin systemd-konfiguraatio

Kuviossa 14 on esitetty konfiguraatiotiedoston /etc/systemd/system/node_exporter.service.d/override.conf sisältö, jossa määritellään itse Node-exporterin käyttämät tiedonkerääjät.

```
[Service]
ExecStart=
User=
User=node-exporter
ExecStart=/usr/bin/node_exporter --web.listen-address=10.150.40.177:9091 --collector.systemd --collector.ntp
 --no-collector.wifi --no-collector.ipvs --no-collector.arp --no-collector.edac --no-collector.infiniband --
no-collector.bcache --no-collector.zfs --no-collector.hwmon --no-collector.timex
```

Kuvio 13 Node-exporter override-konfiguraatiot

Tässä konfiguraatiossa olemme myös lisänneet normaalisti pois päältä olevia kerääjiä kuten systemd- ja ntp-kerääjät sekä ottaneet pois käytöstä turhia kerääjiä, jotka olisivat normaalisti käytössä. Tämän jälkeen voimme käynnistää Node-exporter-kerääjän ja tarkistaa sen tilan seuraavan kuvion 15 osoittamalla komennolla.

```
[root@ip-10-150-40-177 thuttunen]# systemctl status node_exporter
● node_exporter.service - Prometheus Node Exporter Service
   Loaded: loaded (/usr/lib/systemd/system/node_exporter.service; enabled; vendor preset: disabled)
  Drop-In: /etc/systemd/system/node_exporter.service.d
           └─override.conf
   Active: active (running) since Sat 2020-11-28 22:12:21 UTC; 21h ago
 Main PID: 16264 (node_exporter)
    Tasks: 13
   Memory: 12.4M
   CGroup: /system.slice/node_exporter.service
```

Kuvio 14 Node-exporter status

5.3 Elasticstack-palvelun asennus ja asetukset

Elasticsearch asennetaan samaan tyyliin kuin Prometheus pyörimään Docker-konttina. Dockerin vaatima asennustiedosto on saatu Docker-komennolla docker pull docker.elastic.co/elasticsearch/elasticsearch:7.9.2 Elasticsearchin virallisesta Docker-arkistosta ja itse asennuskomento on näkyvillä kuviossa 16.

```
docker run -d --restart=unless-stopped --log-driver json-file  --log-opt max-file=3 --log-opt max-size=15m -m 3.875g --cpu-shares=1843
--ulimit memlock=-1:-1 --ulimit nofile=262144:262144 --group-add="elasticsearch"
--user="elasticsearch" --net="bridge" -p 10.150.40.177:9200:9200 -p 10.150.40.177:9300:9300
-v /etc/qpackage/elasticsearch/conf/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml
-v /var/qpackage/elasticsearch/data:/usr/share/elasticsearch/data
-v /var/qpackage/elasticsearch/repo:/usr/share/elasticsearch/repo
-v /usr/share/elasticsearch/config/elasticsearch.yml
-v /usr/share/elasticsearch/data
-v /usr/share/elasticsearch/repo
-e "SERVICE_9200_NAME=elasticsearch"
-e "ES_JAVA_OPTS=-Xms1785m -Xmx1785m"
-e "SERVICE_9300_NAME=elasticsearch-node"
--name log-1_Oppari_elasticsearch docker-elasticsearch:oppari
```

Kuvio 15 Elasticsearch Docker-käynnistyskomento

Tällä komennolla Elasticsearchille määritellään yhteiset kansiot palvelimen kanssa, muistin sekä prosessorin käyttörajoitukset ja tietysti IP-osoite sekä portti. Itse Elasticsearchin asetukset löytyvät elasticsearch.yml-tiedostosta, joka sijaitsee sen kotikansiossa. Elasticsearch.yml on lisätty liitteenä työn loppuun (Liite 1, Liite 2). Varmistetaan vielä Docker-kontin toimivuus kuvion 17 komennolla.

```
[root@ip-10-150-40-177 thuttunen]# docker ps |grep docker-elasticsearch
f0fc7fc64f76        docker-elasticsearch:oppari                                   "/tini -- /usr/local…"   52 minutes ago      Up 52 minutes       10.150.40.177:9200->9200/tcp, 10.150.40.177:93
00->9300/tcp                                log-1_oppari_elasticsearch
```

Kuvio 16 Elasticsearch-kontin tila

Kuten kuviossa huomataan, Elasticsearch-kontti pyörii ilman ongelmia. Elasticsearch-kontin asennuksen jälkeen voimme siirtyä asentamaan lokitietojen keräämisestä vas-

taavaa Logstash-ohjelmaa. Kuten Elasticsearch myös Logstash on asennettu Elasticsearchin virallisesta Docker-arkistosta ja sen Docker-käynnistyskomento käydään läpi kuviossa 18.

```
docker run -d --restart=unless-stopped --log-driver json-file  --log-opt max-file=3 --log-opt max-size=25m -m 2g --memory-swap 4g --cpu-shares=2048
--net="bridge" -p 10.150.40.177:5044:5044 -p 10.150.40.177:9600:9600
-v /etc/qpackage/logstash/conf/logstash.yml:/usr/share/logstash/config/logstash.yml
-v /var/qpackage/logstash:/usr/share/logstash/backups
-v /var/qpackage/logstash/audit/log-1_log_logstash:/var/audit/spool
-v /var/qpackage/logstash/data/log-1_log_logstash:/usr/share/logstash/data
-v /etc/qpackage/logstash/conf/pipeline:/usr/share/logstash/pipeline
-v /usr/share/logstash/backups
-v /usr/share/logstash/config/logstash.yml -v /usr/share/logstash/data
-v /usr/share/logstash/pipeline
-v /var/audit/spool
-e "SERVICE_5044_NAME=logstash" -e "LS_JAVA_OPTS=-Xmx1638m -Xms1638m" --name log-1_oppari_logstash docker-logstash:oppari logstash -f /usr/share/logstash/pipeline
```

Kuvio 17 Logstashin käynnistyskomento

Docker-käynnistyskomennossa käydään läpi samat asiat kuin Elasticsearchin käynnistyskomennossa. Itse Logstash-konfiguraatiot löytyvät logstash.yml tiedostosta Logstashin kotihakemistosta ja ovat kuvion 19 mukaiset.

```
http.host: "0.0.0.0"
path.config: /usr/share/logstash/pipeline
dead_letter_queue.enable: false
```

Kuvio 18 Logstash.yml

Logstash konfiguraatiotiedosto on huomattavasti pienempi kuin Elasticsearchin vastaava. Kuviossa 19 määriteltiin vain IP-osoite, josta Logstash-rajapinta vastaa kyselyihin. Sekä poistettiin käytöstä DQL-lisäosa (dead letter queue plugin). Logstashin pääasialliset konfiguraatiot sijaitsevat edellisessä kuviossa mainitussa pipeline-tiedostoissa.

Pipeline-tiedostoissa määritellään mistä lokitiedostoja vastaanotetaan, mihin ne lähetetään sekä minkälaisia muokkauksia niille halutaan tehdä. Tässä työssä ei suori-

teta lokitiedostojen muokkausta. Vaan luotetaan Elasticsearchin tarjoamaan dynaamiseen kenttähakuun, joka automaattisesti lisää lokitietoihin kenttiä, joiden mukaan niitä voidaan suodattaa Kibana-ohjelman avulla.

Logstash kuitenkin vaatii vähintään kaksi pipeline-tiedostoa, jotka ovat 01-input.conf sekä 99-output.conf Näissä olevat asetukset määrittävät mistä lokitietoja otetaan vastaa ja mihin ne lähetetään. Numeroinnilla on väliä näissä tiedostoissa, koska Logstash lukee tiedostot numero järjestyksessä. Kuviossa 20 nähdään sisääntulokonfiguraatio. Tämä konfiguraatio kertoo, että tietoa vastaanotetaan Beats lisäohjelmista portista 5044

```
input {
  beats {
    port => 5044
    client_inactivity_timeout => 300
  }
}
```

Kuvio 19 Logstash 01-input.conf

Logstashin 99-output.conf-tiedostossa määritellään mihin Logstash-ohjelman keräämät tiedot lähetetään. Tässä työssä lähetyspaikkana toimii Elasticsearch ja vielä jaottelemme saapuvat lokitiedostot kahteen ryhmään, sovellus- sekä järjestelmälokeihin.

Näille luodaan omat indeksit Elasticsearchiin ja niiden jaottelu tapahtuu Filebeat-asetuksissa, jotka käydään läpi myöhemmin. Kuviossa 21 on lähetyksestä vastaavat asetukset Logstash-ohjelmalle.

```
output {
  # Sovellus lokit
    if ("oppari_sovellus" in [tags]) {
    elasticsearch {
    hosts => [ "elasticsearch.service.consul:9200" ]
    ilm_rollover_alias => "oppari_sovellus"
    ilm_pattern => "{now/d}-0001"
    }
    }
  # Systeemi lokit
    if ("oppari_systeemi" in [tags]) {
    elasticsearch {
    hosts => [ "elasticsearch.service.consul:9200" ]
    ilm_rollover_alias => "oppari_systeemi"
    ilm_pattern => "{now/d}-0001"
    }
    }
}
```

Kuvio 20 Logstash 99-output.conf

Asetus luo oppari_solvellus tai oppari_systeemi nimisen indeksin Elasticsearch-ohjelmaan ja lähettää lokeja siihen tiedostoon riippuen, miten saapuvat lokitiedostot on merkitty. Tämä merkitseminen tapahtuu Filebeat-ohjelmalla.

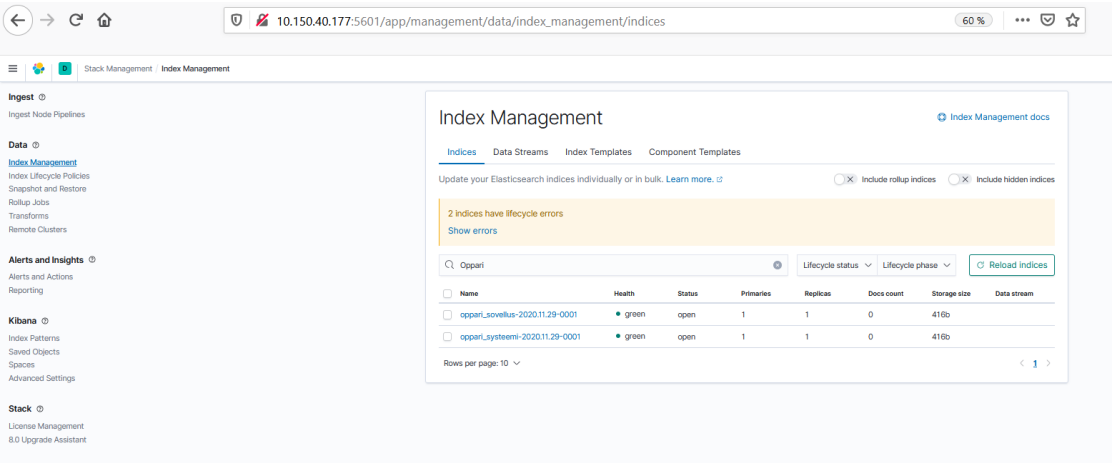
Elasticstackin viimeinen komponentti Kibana on graafinen käyttöliittymä Elastic-klusterin hallintaa ja on siis lokien tutkimiseen ja visualisointiin keskittyvä ohjelma. Kibanan Docker-kontin käynnistyskomento on kuvion 22 mukainen.

```
docker run -d --restart=unless-stopped --log-driver json-file  --log-opt max-file=3
--log-opt max-size=15m -m 1g --memory-swap 2g --cpu-shares=1024 --net="bridge" -p 10.150.40.177:5601:5601
-v /etc/qpackage/kibana/conf:/usr/share/kibana/config -v /usr/share/kibana/config/
-e "SERVICE_NAME=kibana" --name log-1_oppari_kibana docker-kibana:oppari
```

Kuvio 21 Kibanan Docker-kontin käynnistyskomento

Komento sisältää IP-osoitteen, portin sekä Kibanan asetus tiedostojen polun. Kibanan käynnistyksen jälkeen voimme testata verkkoselaimella sen toimivuutta käynnistyskomennossa käytetyssä IP-osoitteessa käyttäen porttia 5601 Samalla voimme tutkia

onko Logstash luonut Opinnäytetyössä käytettävät indeksit. Kibanan verookopohjainen käyttöliittymä on esitetty kuviossa 23.



Kuvio 22 Kibana käyttöliittymä testaus

Kuten edellisestä kuviosta näemme Kibanan käyttöliittymä toimii ja Logstash on luonut kaksi indeksiä. Nämä indeksit eivät vielä sisällä dataa koska emme ole asentaneet Filebeat-lokitiedostokerääjää palvelimille.

5.4 Filebeat-lisäosan asennus ja asetukset

Filebeat-lisäosan asentamiseen tarvitaan Filebeat-paketti. Se haettiin palvelimelle käyttäen curl-komentoa kuvion 24 mukaisesti. Elastiseachin tarjoamasta virallisesta pakettiarkistosta.

```
[root@ip-10-150-40-177 thuttunen]# curl -L -O https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-7.9.2-linux-x86_64.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 29.7M  100 29.7M    0     0  96.2M      0 --:--:-- --:--:-- --:--:-- 96.4M
```

Kuvio 23 Filebeat-paketin noutaminen curl-komentolla

Paketin noutamisen jälkeen se puretaan käyttäen tar-työkalua seuraavalla komennolla haluamaamme sijaintiin: *tar xzvf filebeat-7.9.2-linux-x86_64.tar.gz*. Luomme Filebeat-ohjelmalle systemd-tiedoston, jotta voimme ajaa sitä palveluna systemdn kanssa, jossa myös määritämme tiedostopolut missä asetukset ja ohjelma sijaitsevat kuvion 25 mukaisesti.

```
[Unit]
Description=filebeat
Documentation=https://www.elastic.co/guide/en/beats/filebeat/current/index.html
Wants=network-online.target
After=network-online.target

[Service]
LimitNOFILE=65536
ExecStart=/usr/share/filebeat/bin/filebeat -c /etc/filebeat/filebeat.yml -path.home /usr/share/filebeat -path.config /etc/
filebeat -path.data /var/lib/filebeat -path.logs /var/log/filebeat
Restart=always

[Install]
WantedBy=multi-user.target
```

Kuvio 24 Filebeat-palvelun systemd-asetukset

Itse Filebeat-ohjelman asetukset löytyvät asennuskansion tiedostosta filebeat.yml joka on kokonaisuudessaan nähtävillä opinnäytetyön liiteosiossa (Liite 3 ja Liite 4). Tärkeimpänä näistä liitteessä 3 esitetty filebeat.inputs, jossa määritetään mistä poluista lokitietoja noudetaan, sekä miten ne merkitään Logstashin käsittelyä varten. Filebeat-palvelun toimivuus voidaan tarkistaa komennolla *systemctl status filebeat*, joka antaa vastaukseksi kuvion 26 mukaiset tiedot. Kuten kuviosta näkyy, Filebeat-palvelu on alkanut jo keräämään tietoa määritellyistä poluista.

```
[root@ip-10-150-40-177 thuttunen]# systemctl status filebeat
● filebeat.service - filebeat
   Loaded: loaded (/usr/lib/systemd/system/filebeat.service; enabled; vendor preset: disabled)
   Active: active (running) since Sun 2020-11-29 23:23:15 UTC; 3 days ago
     Docs: https://www.elastic.co/guide/en/beats/filebeat/current/index.html
 Main PID: 1048 (filebeat)
    Tasks: 11
   Memory: 49.9M
   CGroup: /system.slice/filebeat.service
           └─1048 /usr/share/filebeat/bin/filebeat -c /etc/filebeat/filebeat.yml -path.home /usr/share/filebeat -path.c...

Dec 03 04:34:15 ip-10-150-40-177.eu-west-1.compute.internal filebeat[1048]: INFO [monitoring] log/log.go:145 Non-zero...65
Dec 03 04:34:45 ip-10-150-40-177.eu-west-1.compute.internal filebeat[1048]: INFO [monitoring] log/log.go:145 Non-zero...:6
Dec 03 04:35:15 ip-10-150-40-177.eu-west-1.compute.internal filebeat[1048]: INFO [monitoring] log/log.go:145 Non-zero...:6
Dec 03 04:35:45 ip-10-150-40-177.eu-west-1.compute.internal filebeat[1048]: INFO [monitoring] log/log.go:145 Non-zero...":
Dec 03 04:36:15 ip-10-150-40-177.eu-west-1.compute.internal filebeat[1048]: INFO [monitoring] log/log.go:145 Non-zero...65
Dec 03 04:36:45 ip-10-150-40-177.eu-west-1.compute.internal filebeat[1048]: INFO [monitoring] log/log.go:145 Non-zero...65
Dec 03 04:37:15 ip-10-150-40-177.eu-west-1.compute.internal filebeat[1048]: INFO [monitoring] log/log.go:145 Non-zero...:6
Dec 03 04:37:17 ip-10-150-40-177.eu-west-1.compute.internal filebeat[1048]: INFO log/harvester.go:290 Closing harvest...es
Dec 03 04:37:17 ip-10-150-40-177.eu-west-1.compute.internal filebeat[1048]: INFO log/harvester.go:326 Reader was clos...g.
Dec 03 04:37:17 ip-10-150-40-177.eu-west-1.compute.internal filebeat[1048]: INFO log/harvester.go:299 Harvester start...es
Hint: Some lines were ellipsized, use -l to show in full.
```

Kuvio 25 Filebeat-palvelun tila

5.5 Elasticsearch-exporter-lisäosan asennus

Elasticsearch- ja Filebeat-palveluiden asennuksen jälkeen voimme asentaa Elasticsearch-exporter-lisäosan, joka mahdollistaa metriikan saamisen Elasticsearch-klusterista. Elasticsearch asennetaan myös Docker-konttina kuvion 27 mukaisella komennolla. Siinä määritellään Elasticsearch osoite sekä itse lähettäjän IP-osoitteet, portit ja resurssirajat.

```
docker run -d --restart=unless-stopped --log-driver json-file  --log-opt max-file=3 --log-opt max-size=25m -
m 128m --memory-swap 256m --cpu-shares=204 --net="bridge" -p 10.150.40.156:9114:9108 -e "SERVICE_NAME=elasticsearch-exporter"
-e "SERVICE_TAGS=metrics" --name log-2_oppari_elasticsearch-exporter elasticsearch_exporter:oppari
-es.all -es.uri http://10.150.40.156:9200
```

Kuvio 26 Elasticsearch-exporter käynnistyskomento

5.6 Prometheus-es-exporter-lisäosan asennus

Prometheus-es-exporter eroaa hieman Elasticsearch-exportterista. Tämän lähettäjän avulla Prometheus pystyy suorittamaan Elasticsearch-hakupyyntöjä lokeihin ja muuttamaan ne metriikoiksi. Tämä mahdollistaa lokissa esiintyvien sanojen, kuten esimerkiksi ERROR-sanan yhtäjaksoisen hakemisen ja siitä hälytyksen luomisen. Tämä ohjelma on myös asennettuna Docker-konttina käyttäen kuvion 28 komentoa. Prometheus-elasticsearch-exporter asennettiin vain yhdelle palvelimelle vähentämään sen suorittamista hauista kertyvää kuormaa Elasticsearch-klusterille.

```
docker run -d --restart=unless-stopped --log-driver json-file  --log-opt max-file=3 --log-opt max-size=10m -m 170m
--memory-swap 341m --cpu-shares=170 --net="bridge" -p 10.150.40.156:9206:9206
-v /etc/qpackage/prometheus-exporters/conf/prometheus_es_exporter.cfg:/usr/src/app/exporter.cfg -v /usr/src/app/exporter.cfg
-e "SERVICE_9206_TAGS=metrics" -e "SERVICE_9206_NAME=prometheus-exporter-es"
--name monitor-oppari_exporters-es prometheus-es-exporter:oppari
-u /usr/local/bin/prometheus-es-exporter -e elasticsearch.service.consul:9200 --indices-stats-timeout 20 --log-level ERROR
```

Kuvio 27 Prometheus-es-exporter-lisäosan käynnistyskomento

Tiedosto prometheus_es_exporter.cfg sisältää asetukset, johon sijoitetaan Elasticsearch-haut. Esimerkkihaku käydään läpi toteutuksen hälytykset osiossa, jossa luodaan lokipohjainen hälytys käyttäen tätä työkalua.

5.7 Alertmanagerin asennus ja asetukset

Alertmanager on Prometheuksen hälytyksiin käytettävä työkalu ja tarpeellinen osa monitorointijärjestelmää. Alertmanager asennetaan Docker-konttina seuraavan kuvion 29 mukaisella komennolla

```
docker run -d --restart=unless-stopped --log-driver json-file  --log-opt max-file=3
--log-opt max-size=15m -m 250m --memory-swap 500m --cpu-shares=1024 --user="64990" --net="host"
--expose=9096/tcp -p 10.150.40.177:9096:9096 -v /var/qpackage/alertmanager/data:/alertmanager
-v /etc/qpackage/alertmanager/conf:/etc/alertmanager -v /alertmanager -v /etc/alertmanager
-e "SERVICE_9096_CHECK_TIMEOUT=5s" -e "SERVICE_9096_NAME=alertmanager" -e "SERVICE_IP=10.150.40.177"
-e "SERVICE_9096_CHECK_HTTP=/" -e "SERVICE_9096_CHECK_INTERVAL=60s" -e "SERVICE_9096_TAGS=metrics,node1"
--name monitor-1_oppari_alertmanager alertmanager:oppari
--config.file=/etc/alertmanager/config.yml --storage.path=/alertmanager/data --web.listen-address=:9096
```

Kuvio 28 Alertmanagerin Docker-kontin käynnistyskomento

Alertmanagerin asetukset sijaitseva Docker-kontin käynnistyskomennossa mainitussa config.yml-tiedostossa, jonka sisältö on esillä kuviossa 30. Prometheuksen ja Alertmanagerin väliset konfiguraatiot käydään tarkemmin läpi Prometheuksen asetuksia käsittelevässä luvussa.

```
global: {smtp_from: alertmanager@testi.com, smtp_smarthost: 'localhost:25'}
inhibit_rules: []
receivers:
  - name: default-receiver
route:
  group_by: [cluster, alertname]
  group_interval: 5m
  group_wait: 30s
  receiver: default-receiver
  repeat_interval: 4h
```

Kuvio 29 Alertmanager config.yml

5.8 Karma-työkalun asennus ja asetukset

Karma on hälytyksien helppoon visualisointiin tarkoitettu työkalu, jolla saadaan aikaiseksi helppo lukuinen graafinen käyttöliittymä hälytysten tutkimiseksi. Karma asennetaan Docker-konttina kuvion 31 mukaisella Docker-kontin käynnistyskomennolla. Käynnistyskomennossa määritellään palvelimen kanssa jaetut yhteiset kansiot, IP-osoitteet, portti sekä asetustiedosto karma.yml.

```
docker run -d --restart=unless-stopped --log-driver json-file  --log-opt max-file=3 --log-opt max-size=15m
 -m 512m --memory-swap 1g --cpu-shares=1024 --net="bridge"
--expose=9095/tcp -p 10.150.40.177:9095:9095 -v /var/qpackage/karma/logs:/logs -v /var/qpackage/karma/data:/data
 -v /etc/qpackage/karma/conf/karma.yml:/karma.yaml
-v /data -v /karma.yaml -v /logs -e "SERVICE_9095_CHECK_HTTP=/metrics" -e "SERVICE_IP=10.150.40.177"
-e "SERVICE_TAGS=metrics,monitor-1_monitor_karma" -e "SERVICE_9095_NAME=karma"
-e "SERVICE_9095_CHECK_TIMEOUT=20s" -e "SERVICE_9095_CHECK_INTERVAL=60s"
-e "KEEP_LABELS=node severity customer datacenter environment country" -e "CONFIG_FILE=/karma.yaml" -e "PORT=9095"
-e "COLOR_LABELS_UNIQUE=severity" -e "ANNOTATIONS_HIDDEN=summary" --name monitor-1_oppari_karma karma:oppari
```

Kuvio *30* Karma-työkalun Docker-kontin käynnistyskomento

Karman asennustiedostossa määritellään Alertmanagerin sijainti ja graafisen käyttöliittymän hälytyksien järjestelyyn vaikuttavia asetuksia. Sekä asetetaan valmis suodatin aktiivisen hälytysten tutkintaan, kuten voidaan todeta kuviosta 32.

```
alertmanager:
  interval: 10s
  servers:
    - name: "alertmanager-server-1"
      uri: "http://node1.alertmanager.service.consul:9096"
      timeout: 5s
      proxy: true

annotations:
  hidden:
    - summary

filters:
  default:
    - "@state=active"

grid:
  sorting:
    order: label
    reverse: false
    label: severity
    customValues:
      labels:
        severity:
          critical: 1
          warning: 2
          info: 3
```

Kuvio 31 Karma.yml-asetustiedoston sisältö

5.9 Grafana-palvelun asennus

Kuten suurin osa työssä käytetyistä työkaluista myös Grafana asennetaan Docker-konttina Grafanan käynnistyskomento on kuvion 33 mukainen. Käynnistyskomennossa Grafanalle annetaan muista työkaluista poiketen pääkäyttäjän salasana, jota tarvitaan graafiseen käyttöliittymään kirjautumisessa. Grafanan asetustiedosto on grafana.ini, mutta sen asetuksia emme käy tarkemmin läpi, koska käytämme kaikissa asetuksissa oletus arvoja.

```
docker run -d --restart=unless-stopped --log-driver json-file  --log-opt max-file=3 --log-opt max-size=25m -m 128m
--memory-swap 256m --cpu-shares=204 --net="bridge" -p 10.150.40.177:9114:9108
-e "SERVICE_NAME=elasticsearch-exporter" -e "SERVICE_TAGS=metrics"
--name log-1_oppari_elasticsearch-exporter elasticsearch_exporter:oppari -es.all -es.uri http://10.150.40.177:9200
```

Kuvio 32 Grafanan käynnistyskomento

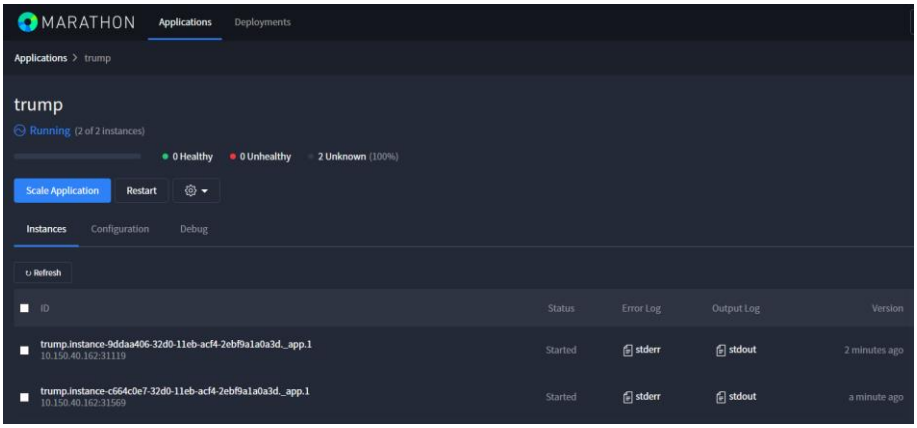
5.10 Prometheus-palvelun asetukset

Prometheusen asetukset löytyvät prometheus.yml-tiedostosta ja se sisältää pääasiassa asetukset mistä metriikoita noudetaan. Opinnäytetyötä varten loimme seuraavat asetukset, jotka kokonsa vuoksi ovat erillisenä liitteenä (Liite 5) työn lopussa. Edellisen liitteen asetuksilla löysimme seuraavat metriikkatietoa tuottavat kohteet, joita pystyy helposti tarkastelemaan Prometheuksen graafisesta käyttöliittymästä. Prometheuksen löytämät kohteet voidaan nähdä liitteistä 6 ja 7.

5.11 Testi sovelluksen asennus

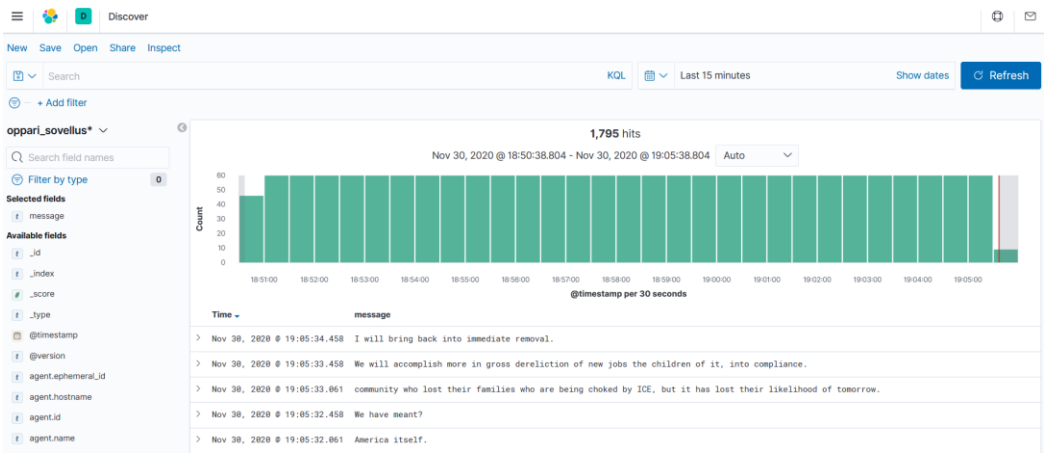
Testisovelluksena toimii Trump-niminen sovellus, joka on kehitetty lokituksen toimivuuden testaamiseen. Sovellus kokoaa tietokannassa olevista Trumpin Twitter-viesteistä satunnaisgeneraattorilla viestejä. Asennamme Trump-sovelluksen Mesos/Marathonin päälle pyörimään ja sitä kautta saamme sovellusloki tietoa. Kuten kuviosta

34 nähdään, asennettiin Trump-sovellukselle kaksi Docker-konttia Marathonilla käynnistäen, joka asensi sovellukset slave-palvelimelle. Sovellus skaalattiin kahteen, mutta ikävä kyllä se käytti vain yhtä slave-palvelinta kahden sijaan.



Kuvio 33 Trump-sovelluksen asennus

Kibanasta voimme todeta sovelluslokituksen toimivuuden tarkastelemalla sen Discover-toiminnolla saapuvia lokitietoja kuten kuvio 34 osoittaa. Discover-toiminto näyttää myös saapuneiden lokitiedostojen määrän, sekä kerää ne kätevästi luettavaksi.



Kuvio 34 Kibanan Discover-toiminto

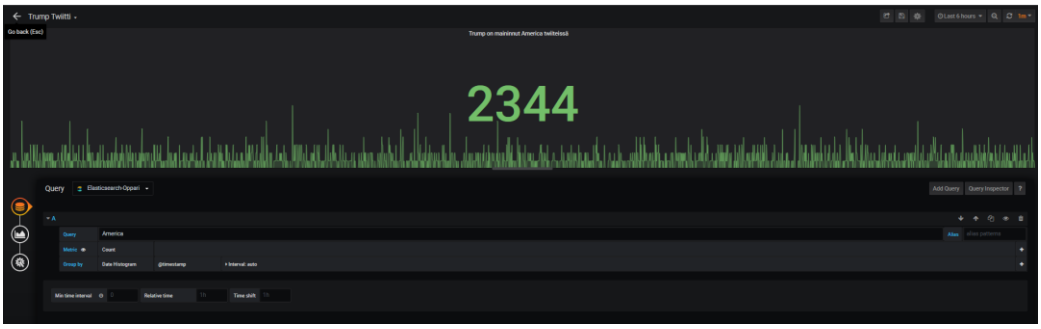
5.12 Visualisointi Grafanalla

Ennen Grafanalla erilaisten näkymien tekoa, tarvitsee sille määrittää datalähteet. Tässä työssä käytetään kahta datalähdettä: Prometheus sekä Elasticsearch, josta käytetään indeksiä oppari_sovellus. Näiden lisääminen onnistuu helposti Grafanan graafisen käyttöliittymän kautta. Molempien datalähteiden asetukset näkyvät liitteissä 8 ja 9. Kun datalähteet oli lisätty, laadittiin visualisointeja ympäristöstä kerätylle metriikalle.

Node-exporterin datan tuomaan palvelintason metriikkaan kannattaa käyttää valmiita näkymiä, joita on tarjolla Grafanan verkkosivuilla. Yleisesti ne toimivat suoraan ilman tarvetta muokata niitä. Työssä käytettiin liitteistä 10 ja 11 löytyvien kuvien mukaisia valmiita näkymiä palvelintason metriikoille. Kyseiset Grafana näkymät antavat hyvän yleiskuvan koko järjestelmästä sekä mahdollistavat yksityiskohtaisen tiedonhakemisen palvelinkohtaisesti.

Sovelluksen lokipohjaiseen visualisointiin loimme oman Grafana-näkymän käyttämällä datalähteenä Elasticsearchia ja aikaisemmin Mesos/Marathonin päälle käynnistettyä Trump-sovellusta. Tässä visualisoinnissa teimme paneelin, jotka kuvastavat tietyn arvon esiintymistä lokitiedoissa. Esitettynä arvona oli sana America. Tällä testauksella halusimme demonstroida kuinka lokitiedoista Grafanaa käyttäen on mahdollista saada tietoa sovelluslokeista ja luoda niistä visualisointeja isommissa ympäristöissä,

joissa käynnissä on paljon sovelluksia. Tämän tyylisiä hakuja käyttää esimerkiksi hakemaan virheitä, joita sovellukset lokitiedoissa lähettävät. Haku tuotti Grafana-näkymään kuvion 36 mukaiset tulokset.



Kuvio 35 Grafanan lokitietojen haku

5.13 Hälytykset

Hälytyksiä on mahdollista luoda, kun kaikki klusterin monitorointityökalut ovat paikallaan. Hälytysten päämääräinen tarkoitus on mahdollistaa, että ympäristöä ei tarvitse kokoaikaa aktiivisesti valvoa. Hälytyksiin käytettiin Prometheusta ja hälytykset sijoitettiin base_alerts-nimiseen kansioon, joka on mainittu Prometheus.yml-tiedostossa hälytysten sijainniksi. Perustason laitteistohälytyksiä varten luotiin tiedosto laitteisto.alerts.yml, joka on nähtävillä kokonaisuudessaan liitteissä 12,13 ja 14. Tämä ryhmä hälytyksiä kattaa suurimman osan yleisistä laitteistoon liittyvistä hälytyksistä liittyen prosessorin käyttöön, muisteihin, levytilaan ja verkon toimintaan.

Hälytysten toiminta testattiin ottamalla yhden näistä hälytyksistä tarkasteluun. Hälytykseksi valittiin prosessorin käyttöön liittyvä hälytys ja sen asetukset. Nämä on esitetty kuviossa 37.

```
- alert: CPUExcessiveUsage
  expr: 100 - (avg by (node) (irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) * 100) > 99
  for: 5m
  labels:
    severity: critical
  annotations:
    brief: "CPU usage has been over 99% on host {{ $labels.node }} for more than 5 minutes. Value: {{ $value|humanize }}"
```

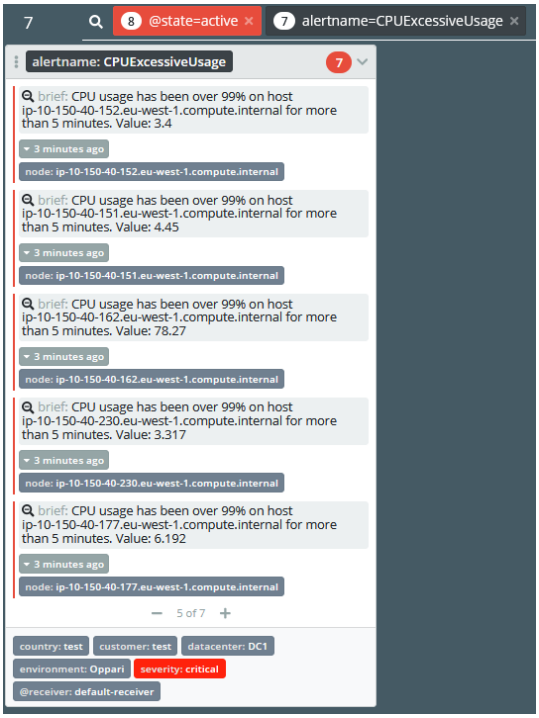
Kuvio 36 Prosessorin käyttöhälytys

Hälytystä muutettiin siten että, raja-arvo 99 prosentin keskimääräinen käyttö viimeisen viiden minuutin ajalta muutettiin yhden prosentin raja-arvoksi kuvion 38 mukaisesti.

```
- alert: CPUExcessiveUsage
  expr: 100 - (avg by (node) (irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) * 100) > 1
  for: 5m
  labels:
    severity: critical
  annotations:
    brief: "CPU usage has been over 99% on host {{ $labels.node }} for more than 5 minutes. Value: {{ $value|humanize }}"
```

Kuvio 37 Prosessorin käyttöhälytyksen testaus

Voimme todeta, että hälytys järjestelmä toimii Karma-työkalun kuten kuvion 39 näkymässä on esitetty. Kaikki klusteriin kuuluvat palvelimet hälyttävät korkeaa prosessorin käyttöä



Kuvio 38 Prosessorin käyttöhälytys Karmassa

Sovellus hälytyksiin käytämme lokipohjaista hälytystä, joka on luotu Prometheus-es-exporter-työkalun avulla. Kuviossa 40 on nähtävissä Prometheuksen käyttämä Elasticseach-haku tietojen hakemiseen.

```
[query_oppari_trump_sanoo]
QueryIntervalSecs = 300
QueryTimeoutSecs = 15
QueryIndices = <oppari_sovellus*>
QueryJson = {
  "size": 0,
  "query": {
    "bool": {
      "filter": [
          {"range" : {
                "@timestamp" : {
                  "gte" : "now-1d",
                  "lte" :  "now"
            }
          }
        },
        {
          "match_phrase": {
            "message": {
              "query": "Great again"
            }
          }
        }
      ]
    }}}
```

Kuvio 39 Prometheus-es-exporter-haku

Kuten kuviossa 41 nähdään Prometheus saa Elasticsearch-lokitiedosto haulla metriikkaa tutkimalla Prometheuksen käyttöliittymän metriikkahakutoimintoa. Tällä metriikkahaulla voimme tarkastella eri metriikoita, mitä Prometheus kerää. Kuten kuviosta 41 ilmenee, Prometheus saa metriikkaa nimellä oppari_trump_sanoo_hits.

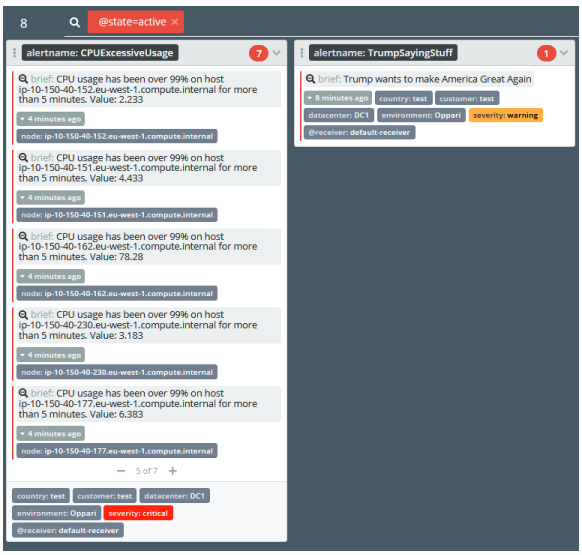
Kuvio 40 Metriikkahaku

Koska Prometheus saa metriikkaa lokitiedoista voimme tehdä kuvion 42 mukaisen hälytyksen perustuen lokitiedoista tehtyyn metriikkaan. Hälytys aktivoituu, kun lokitiedoissa esiintyy sana ”Great again”.

```
groups:
- name: loki.alerts
  rules:
  - alert: TrumpSayingStuff
    expr: sum(oppari_trump_sanoo_hits) > 1
    labels:
      severity: warning
    annotations:
      brief: Trump wants to make America Great Again
```

Kuvio 41 Lokitietopohjainen metriikkahälytys

Voimme todeta lokipohjaisen hälytyksen toimivan Karma-näkymästä, jossa se on tällä hetkellä aktiivisena hälytyksenä, kuten kuviosta 43 voidaan nähdä. Tämän pohjalta todettiin, että olimme saaneet suoritetuksi toimivan palvelin- ja sovellustason hälytykset ympäristössä ja monitorointi toimii halutulla tavalla.



Kuvio 42 Karma-näkymä lokipohjaisesta hälytyksestä

6 Yhteenveto ja pohdinta

6.1 Haasteet

Tutkimuksen haasteina olivat ehdottomasti monitorointiin liittyvä englanninkielinen sanasto ja termit, jotka eivät helposti taipuneet suomenkielisiksi vastineiksi. Nämä aiheuttivat huomattavan määrän päänvaivaa ja hankaluuksia etsiä oikea tapa sanoa asiat oikein.

Haasteena oli myös laaja aihealue, josta oli vaikeuksia tehdä päätöksiä mihin keskittyy. Sillä opinnäytetyön tekoaika oli rajallinen, joka olisi ollut ehkä helpompi toteuttaa paremmalla suunnittelulla.

Harmittamaan jäi jälkikäteen, että olisi voinut keskittyä enemmän Docker-konttien monitorointiin myös tämän työn yhteydessä.

6.2 Onnistumiset

Toteutuksen osuus sujui yllättävän vaivattomasti ja erilaisten asetusten toimivuus yllätti, sillä oletin että niiden luomiseen olisi kulunut huomattavasti enemmän aikaa.

Palvelintason metriikka ja sovellustason lokipohjaisessa monitoroinnissa onnistuttiin hyvin ja saimme tehtyä testi sovellukselle ja palvelimille Grafana-näkymät sekä hälytykset.

6.3 Yhteenveto

Työssä onnistuttiin kohtalaisella tasolla huomioiden asetettuihin tavoitteisiin eli monitorointiin ja monitorointityökaluihin liittyvän tietotaidon kasvattamiseen yleisesti. Monitorointi esitettiin tietoperustan kautta ja hieman katsottua syvemmin mitä metriikat, lokit, hälytykset sekä monitorointijärjestelmä termeinä tarkoittavat.

Saimme tehtyä käytännön osiossa toimivan monitorointi ratkaisun, jossa kaikkia klusterin palvelimia monitorointiin, jollain tavalla. Saimme myös aikaiseksi visualisointeja klusterin palvelimista, sekä käytetystä testi sovelluksesta. Saimme onnistuneesti testattua palvelintason sekä sovellustason hälytyksien luonnin.

Kaiken kaikkiaan työ suoritettiin onnistuneesti loppuun, vaikka henkilökohtaisesti olisi vielä halunnut saada tutkittua aihetta laajemmin. Muutamia ongelmia toteutusosuutta tehdessä tuli vastaan, mutta niistä suoriuduttiin kunnialla.

Lähteet

Atlassian 2020; viitattu 25.11.2020. Atlassian IT alerting. https://www.atlassian.com/incident-management/on-call/it-alerting.

Consul 2020 Introduction to Consul; Official Consul documentation. viitattu 16.11.2020. https://www.consul.io/docs/intro.

Grafana 2020 What is Grafana; official documentation of Grafana tool. viitattu 9.11.2020. https://grafana.com/docs/grafana/latest/getting-started/what-is-grafana/.

Elasticsearch 2020 What is Elasticsearch; official documentation of Elasticsearch. viitattu 9.11.2020. https://www.elastic.co/what-is/elasticsearch.

Elasticsearch 2020 Logstash; official documentation of Logstash. viitattu 9.11.2020. https://www.elastic.co/logstash.

Dan Reichert 2018; viitattu 25.11.2020. Dan Reichert Logs and Metrics: What are they and how do they help me. https://www.sumologic.com/blog/logs-metrics-overview/.

John Turnbull Monitoring with Prometheus 2018; julkaistu kirja Prometheus monitoroinnista. Viitattu 13.11.2020. Chapter 2 Introduction to Prometheus. https://books.google.fi/books/about/Monitoring_with_Prometheus.html?id=EtlfDwAAQBAJ&redir_esc=y.

Justin Ellingwood 2017; viitattu 23.11.2020. An Introduction to Metrics, Monitoring, and Alerting Digital Ocean. https://www.digitalocean.com/community/tutorials/an-introduction-to-metrics-monitoring-and-alerting.

Opensource n.d What is Docker; viitattu 27.11.2020. Opensource What is Docker https://opensource.com/resources/what-docker.

Opensource Prometheus; kuva viittaus 23.11.2020. Prometheus Arkkitehtuuri kuvio.

https://opensource.com/article/19/11/introduction-monitoring-prometheus.

Prometheus 2020 Alerting Alertmanager; Official Prometheus documentation. viitattu 13.11.2020. https://prometheus.io/docs/alerting/latest/alertmanager/.

Roger Ignazio 2016; Roger Ignazio Mesos in Action. kirja viitattu 25.11.2020. luku 7 Introduction to Marathon.

Sachin P Bappalige 2014; viitattu 25.11.2020. Open source datacenter computing with Apache Mesos. https://opensource.com/business/14/9/open-source-datacenter-computing-apache-mesos.

Splunk 2016; viitattu 25.11.2020 Splunk Metric and Log Monitoring: Do You Really Need Both. https://www.splunk.com/en_us/blog/devops/metric-log-monitoring-really-need.html.

Sematext 2020; viitattu 23.11.2020. Sematext Complete Guide to Metrics, Monitoring & Alerting. https://sematext.com/blog/monitoring-alerting/

Tutorials Point n.d; viitattu 25.11. Tutorials Point Zookeeper Overview. https://www.tutorialspoint.com/zookeeper/zookeeper_overview.htm.

Liitteet

Liite 1. Elasticsearch.yml osa 1

```
# ======================== Elasticsearch Configuration =========================
#
# NOTE: Elasticsearch comes with reasonable defaults for most settings.
#       Before you set out to tweak and tune the configuration, make sure you
#       understand what are you trying to accomplish and the consequences.
#
# The primary way of configuring a node is via this file. This template lists
# the most important settings you may want to configure for a production cluster.
#
# Please consult the documentation for further information on configuration options:
# https://www.elastic.co/guide/en/elasticsearch/reference/index.html
#
# ---------------------------------- Cluster -----------------------------------
#
# Use a descriptive name for your cluster:
#
cluster.name: Logging_elasticsearch
cluster.initial_master_nodes: ["10.150.40.177","10.150.40.156","10.150.40.151"]
#
# ------------------------------------ Node ------------------------------------
#
# Use a descriptive name for the node:
#
node.name: elasticsearch-1#
#
# Add custom attributes to the node:
#
node.attr.rack: rack1

# ---------------------------------- Paths ------------------------------------
#
# Path to directory where to store the data (separate multiple locations by comma):
#
path.data: /usr/share/elasticsearch/data
#
# Path to log files:
#
path.logs: /usr/share/elasticsearch/logs
#
# ---------------------------------- Memory -----------------------------------
#
# Lock the memory on startup:
#
bootstrap.memory_lock: true
#
# Make sure that the heap size is set to about half the memory available
# on the system and that the owner of the process is allowed to use this
# limit.
#
# Elasticsearch performs poorly when the system is swapping the memory.
```

Liite 2. Elasticsearch.yml osa 2

```
# ---------------------------------- Network -----------------------------------
#
# Set the bind address to a specific IP (IPv4 or IPv6):
#
network.publish_host: "10.150.40.177"
network.bind_host: _site_
# Set a custom port for HTTP:
#
#http.port: 9200
#
# For more information, consult the network module documentation.
#
# --------------------------------- Discovery ----------------------------------
#
# Pass an initial list of hosts to perform discovery when new node is started:
# The default list of hosts is ["127.0.0.1", "[::1]"]
#
discovery.seed_hosts: ["10.150.40.177","10.150.40.156","10.150.40.151"]

#
# Prevent the "split brain" by configuring the majority of nodes (total number of master-eligible nodes / 2 + 1):
#
discovery.zen.minimum_master_nodes: 2

# For more information, consult the zen discovery module documentation.

#---------------------------------- Repositories-----------------------------
path.repo: ["/usr/share/elasticsearch/repo"]
```

Liite 3 Filebeat.yml osa 1

```
###################### Filebeat Configuration Example #########################

# This file is an example configuration file highlighting only the most common
# options. The filebeat.reference.yml file from the same directory contains all the
# supported options with more comments. You can use it as a reference.
#
# You can find the full configuration reference here:
# https://www.elastic.co/guide/en/beats/filebeat/index.html

# For more available modules and options, please see the filebeat.reference.yml sample
# configuration file.

#============================ Filebeat processors ==============================

processors:
- add_docker_metadata:
    when:
      contains:
        tags: "container"

#============================ Filebeat inputs (formerly prospectors) ============================
filebeat.inputs:

- type: log
  paths:
    - '/var/lib/mesos/slaves/*/frameworks/*/executors/*/runs/latest/*'
  tags: ["oppari_sovellus"]
  close_inactive: 30s
  close_timeout: 5m
  clean_inactive: 168h
  scan_frequency: 1m
  ignore_older: 167h

- type: log
  paths:
    - '/var/log/messages*'
  tags: ["oppari_systeemi"]
  close_timeout: 5m
  clean_inactive: 168h
  scan_frequency: 1m
  ignore_older: 167h

- type: log
  paths:
    - '/var/lib/docker/containers/*/*.log'
  tags: ["container", "oppari_systeemi"]
  processors:
    - add_docker_metadata: ~
  close_inactive: 30s
  close_timeout: 5m
  clean_inactive: 168h
  scan_frequency: 1m
  ignore_older: 167h
```

Liite 4. Filebeat.yml osa 2

```
  #============================== General ===================================

# The name of the shipper that publishes the network data. It can be used to group
# all the transactions sent by a single shipper in the web interface.
name: ip-10-150-40-177

# Optional fields that you can specify to add additional information to the
# output.
fields:
  env: Oppari
  ip: 10.150.40.177
  version: Filebeat:7.9.2

#----------------------------- Logstash output --------------------------------

output.logstash:
  # The Logstash hosts
  hosts:
    - logstash.service.consul:5044
# send to all hosts
  loadbalance: true
  bulk_max_size: 1024

#================================ Logging =====================================t=

logging.to_syslog: true
```

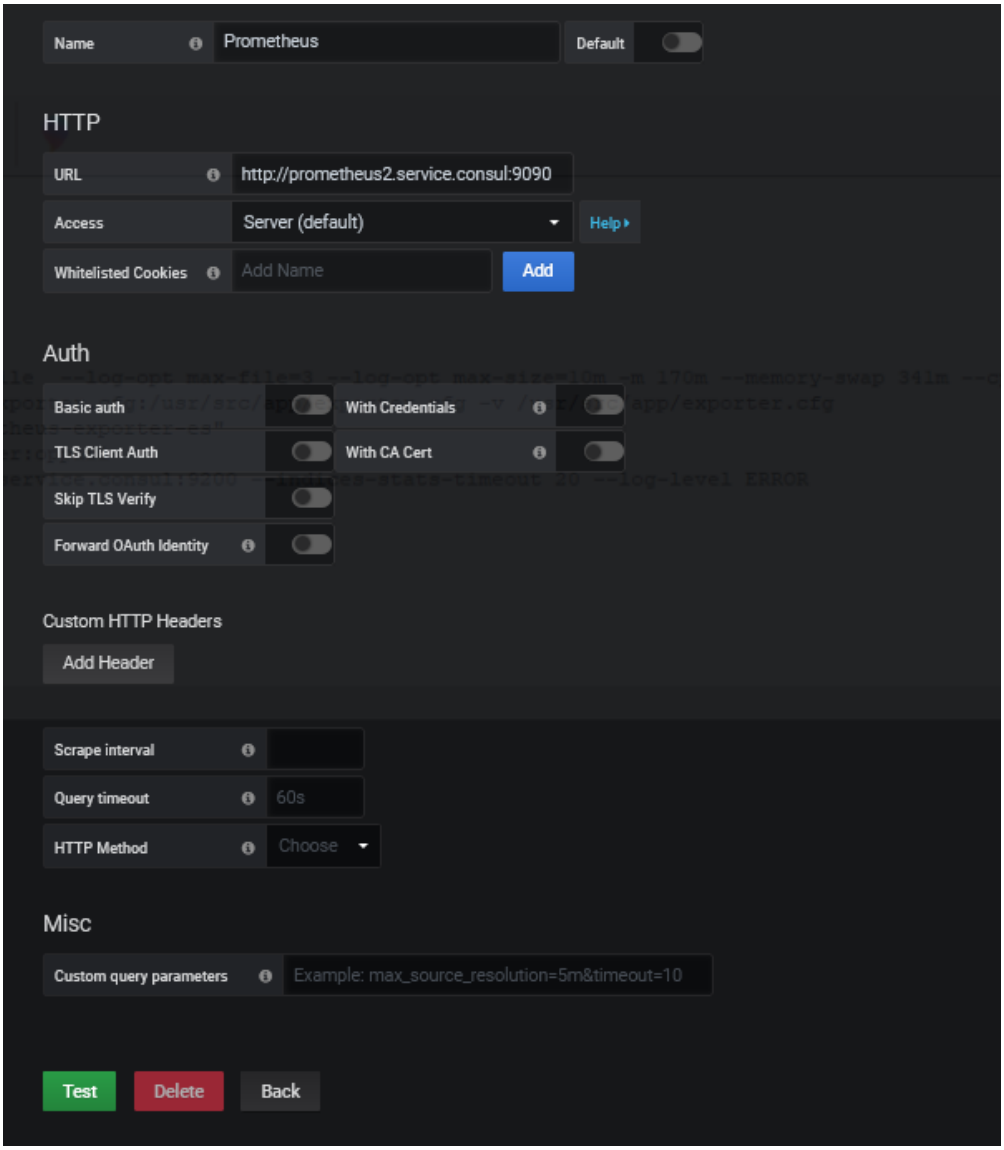
Liite 5 Prometheus.yml

```
global:
  scrape_interval: 30s
  scrape_timeout: 20s
  evaluation_interval: 45s
  external_labels: {country: test, customer: test, datacenter: DC1, environment: Oppari}

rule_files:
  - "base_alerts/*.alerts.yml"

alerting:
  alertmanagers:
  - scheme: http
    static_configs:
    - targets:
      - "alertmanager.service.consul:9096"

scrape_configs:
  - job_name: 'Oppari_metriikat'
    scrape_interval: 30s
    scrape_timeout: 15s
    consul_sd_configs:
      - server: consul.service.consul:8500
    relabel_configs:
      - source_labels: [__meta_consul_tags]
        regex: '.*metrics,.*'
        action: keep
      - source_labels: [ __meta_consul_service]
        target_label: 'job'
      - source_labels: [__meta_consul_node]
        target_label: 'node'
      - source_labels: [__meta_consul_tags]
        target_label: 'job'
        regex: '.node-metrics,.*'
        replacement: node
      - target_label: 'datacenter'
        replacement: 'dc1'
      - target_label: 'environment'
        replacement: 'Oppari'
```

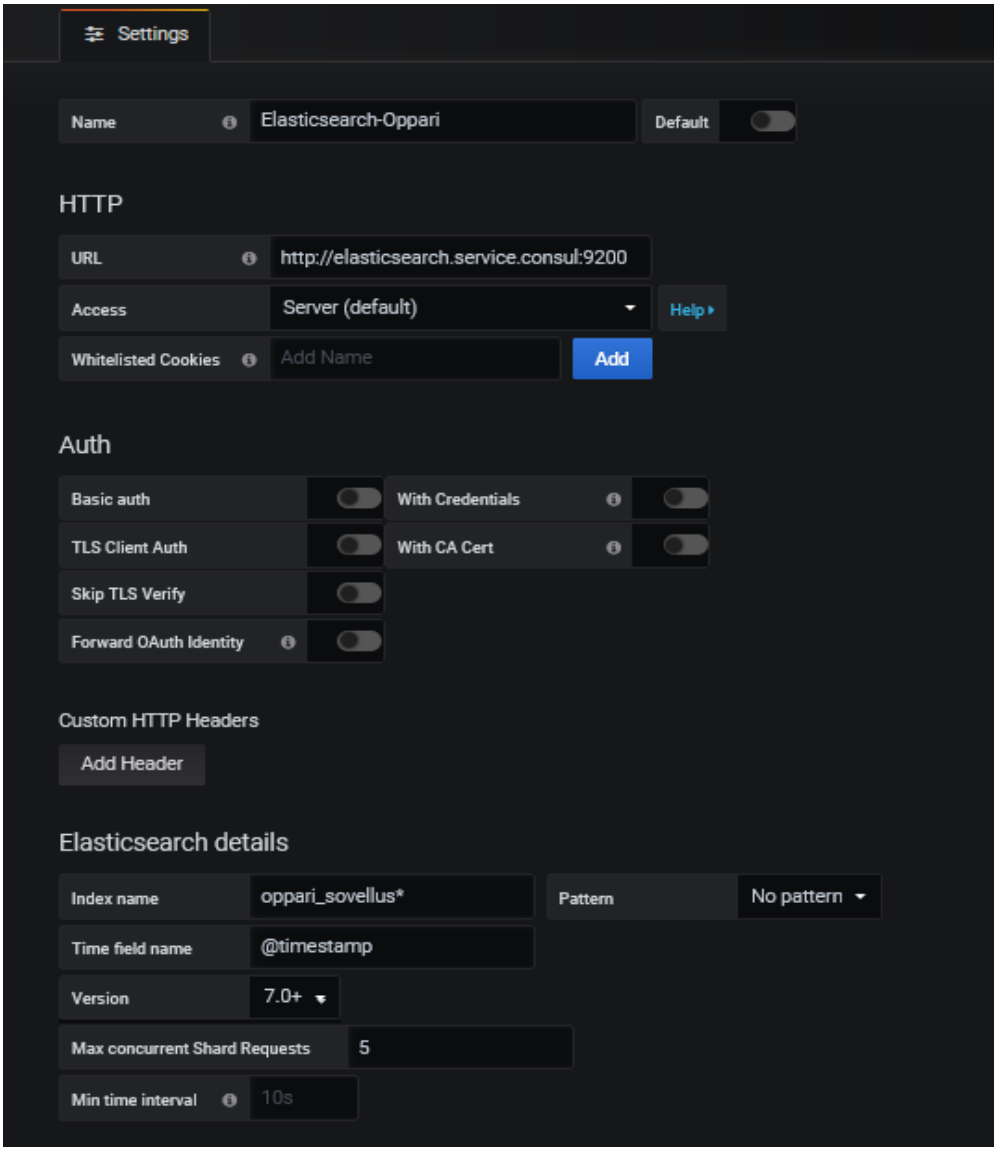
Liite 6 Prometheus target osa 1

Liite 7 Prometheus target osa 2

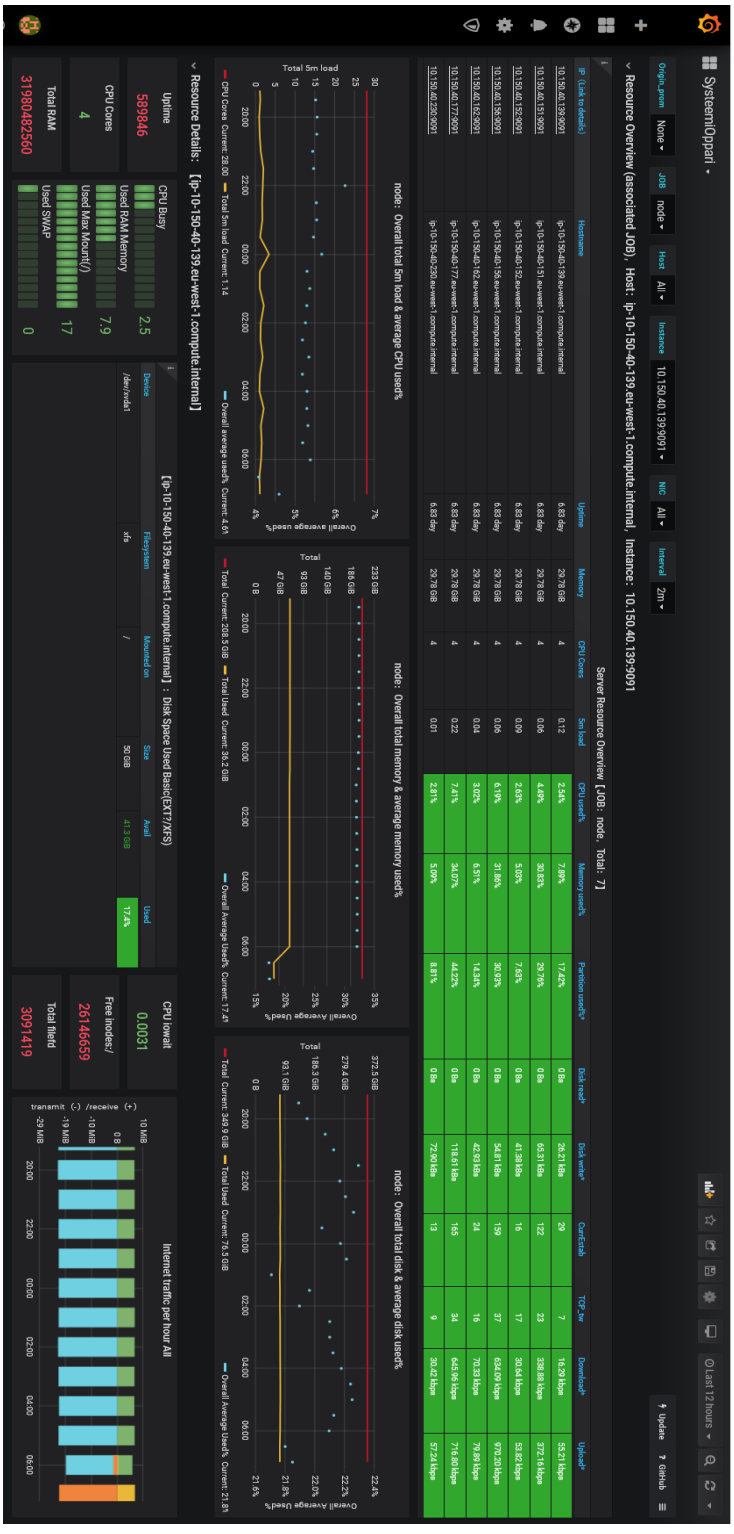
Liite 8 Grafana datasource Prometheus



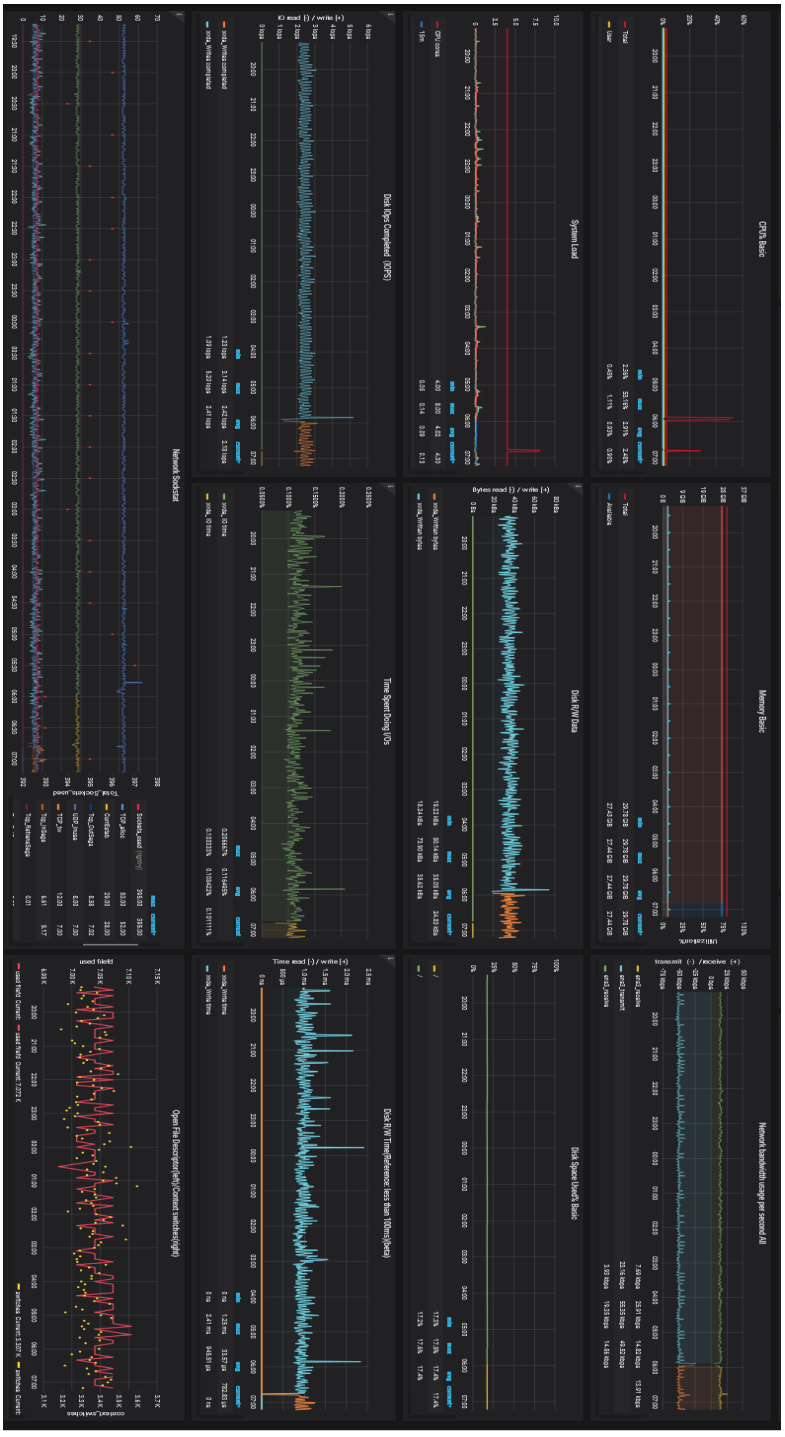
Liite 9 Grafana datasource Elasticsearch



Liite 10 Palvelin tason Grafana näkymä



Liite 11 Palvelin tason Grafana näkymä osa 2



Liite 12 Laitteisto hälytykset

```
groups:
- name: node.alerts
  rules:
  - alert: FilesystemUsedMajorityOfSpace
    expr: (1 - (node_filesystem_avail_bytes{fstype!~"(tmpfs|iso9660)",mountpoint!~"(/sys|/dev|/proc).*",device!="sunrpc"} /
    on(node,instance,mountpoint,device) node_filesystem_size_bytes) * on(instance,node) group_left(nodename) (node_uname_info)) * 100 > 75
    and (1 - (node_filesystem_avail_bytes{fstype!~"(tmpfs|iso9660)",mountpoint!~"(/sys|/dev|/proc).*",device!="sunrpc"} /
    on(device,instance,node, mountpoint) node_filesystem_size_bytes) * on(instance,node) group_left(nodename) (node_uname_info)) * 100 < 90
    for: 5m
    labels:
      severity: warning
    annotations:
      brief: "{{ $value|humanize }}% used space in filesystem {{$labels.mountpoint}} on host {{ $labels.nodename }} {{ $labels.instance }}"
  - alert: FilesystemUsedAlmostAllSpace
    expr: (1 - (node_filesystem_avail_bytes{fstype!~"(tmpfs|iso9660)"} / on(device,instance,node,mountpoint) node_filesystem_size_bytes) *
    on(instance,node) group_left(nodename) (node_uname_info)) * 100 > 90
    for: 5m
    labels:
      severity: critical
    annotations:
      brief: "{{ $value|humanize }}% used space in filesystem {{$labels.mountpoint}} on host {{ $labels.nodename }} {{ $labels.instance }}"
  - alert: FilesystemPredictedToBeFull
    expr: predict_linear(node_filesystem_free_bytes{job="node",fstype!~"(tmpfs|iso9660)",mountpoint!~"(/sys|/dev|/proc).*",device!="sunrpc"}[1h],
    8 * 3600) / 1073741824 < 0
    for: 5m
    labels:
      severity: warning
      level: infrastructure
    annotations:
      brief: "Filesystem {{$labels.mountpoint}} on host {{ $labels.node }} {{ $labels.instance }}
      will be out of free space in 8 hours according to the last hours growth rate"
  - alert: InodeHighUsage
    expr: node_filesystem_files_free{fstype!~"(tmpfs|iso9660)"} / node_filesystem_files{fstype!~"(tmpfs|iso9660)"}* 100
    <= 20 and node_filesystem_files_free{fstype!~"(tmpfs|iso9660)"}
    / node_filesystem_files{fstype!~"(tmpfs|iso9660)"}* 100 > 10
    for: 15m
    labels:
      severity: warning
    annotations:
      brief: 'Only {{ $value | printf "%.2f" }} free inodes on filesystem {{ $labels.mountpoint }} on host {{ $labels.node }}'
      procedure: TBD
```

Liite 13 Laitteisto hälytykset osa 2

```
- alert: RAMHighUsage
  expr: (node_memory_MemTotal_bytes - node_memory_MemFree_bytes - node_memory_Buffers_bytes - node_memory_Cached_bytes
  - node_memory_SReclaimable_bytes)
  / node_memory_MemTotal_bytes * 100 > 90 < 95
  for: 5m
  labels:
    severity: warning
  annotations:
    brief: "RAM usage is over 90% on host {{ $labels.node }}. Percentage of RAM used: {{ $value|humanize }}%"
- alert: RAMExcessiveUsage
  expr: (node_memory_MemTotal_bytes - node_memory_MemFree_bytes - node_memory_Buffers_bytes - node_memory_Cached_bytes
  - node_memory_SReclaimable_bytes)
  / node_memory_MemTotal_bytes * 100 >= 95
  for: 1m
  labels:
    severity: critical
  annotations:
    brief: "RAM usage is over 95% on host {{ $labels.node }}. Percentage of RAM used: {{ $value|humanize }}%"
- alert: CPUHighUsage
  expr: 100 - (avg by (node) (irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) * 100) > 90 and 100 - (avg by (node)
  (irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) * 100) < 95
  for: 5m
  labels:
    severity: warning
  annotations:
    brief: "CPU usage has been over 95% on host {{ $labels.node }} for more than 5 minutes. Value: {{ $value|humanize }}"
- alert: CPUExcessiveUsage
  expr: 100 - (avg by (node) (irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) * 100) > 99
  for: 5m
  labels:
    severity: critical
  annotations:
    brief: "CPU usage has been over 99% on host {{ $labels.node }} for more than 5 minutes. Value: {{ $value|humanize }}"
- alert: CPUHighIowait
  expr: (avg by(instance) (irate(node_cpu_seconds_total{job="node",mode="iowait"}[5m])) * 100) > 10 and (avg by(instance)
  (irate(node_cpu_seconds_total{job="node",mode="iowait"}[5m])) * 100) < 30
  for: 5m
  labels:
    severity: warning
  annotations:
    brief: "CPU IOWait is high on host {{ $labels.node }}. Value: {{ $value|humanize }}%"
```

Liite 14 Laitteisto hälytykset osa 3

```
- alert: CPUExcessiveLoad
  expr: avg by(node) (node_load15) / count by(node) (count by(cpu, node) (node_cpu_seconds_total)) * 100 > 99
  for: 5m
  labels:
    severity: critical
  annotations:
    brief: "CPU load (15 min.) has been over 99% for more than 5 minutes on host {{ $labels.node }}.
    Value:  {{ $value|humanize }}"
- alert: FiledescriptorsHighUsageByProcess
  expr: filedescriptor_open_per_process / filedescriptor_limit_per_process * 100 > 60 and filedescriptor_open_per_process
  / filedescriptor_limit_per_process * 100 < 80
  for: 5m
  labels:
    severity: warning
  annotations:
    brief: 'Percentage of filedescriptor used by a single process has reached {{ $value | printf "%.2f" }}%
    on host {{ $labels.node }}'
- alert: FiledescriptorsCloseToLimitForProcess
  expr: filedescriptor_open_per_process / filedescriptor_limit_per_process * 100 > 80
  for: 5m
  labels:
    severity: critical
  annotations:
    brief: 'Percentage of filedescriptor used by a single process has reached {{ $value | printf "%.2f" }}%
    on host {{ $labels.node }}'
- alert: ProcessCountOnNodeIsHigh
  expr:  node_processes_pids/node_processes_max_processes * 100 > 60 AND node_processes_pids/node_processes_max_processes * 100 < 80
  for: 5m
  labels:
    severity: warning
  annotations:
    brief: 'Percentage of max number of running processes has reached {{ $value | printf "%.2f" }}% on host {{ $labels.node }}'
- alert: NodeHasRebooted
  expr: (time () - node_boot_time_seconds) / 3600 < 2
  for: 5m
  labels:
    severity: warning
  annotations:
    brief: 'Host {{ $labels.node }} rebooted {{ $value | printf "%.0f" }}
    hours ago, this alert is raised if a node has been rebooted within the last 2h'
```