Tampereen yliopisto

Toni Riikonen

JÄRJESTELMÄMONITOROINNIN KEHITTÄMINEN

Palvelutasodatan keräämisen, säilyttämisen, käsittelemisen ja esittämisen uusiminen

Diplomityö
Tekniikan ja luonnontieteiden tiedekunta
Timo Mäkinen
Sami Hyrynsalmi
Joulukuu 2020

TIIVISTELMÄ

Toni Riikonen: Järjestelmämonitoroinnin kehittäminen: Palvelutasodatan keräämisen, säilyttämisen, käsittelemisen ja esittämisen uusiminen
Diplomityö
Tampereen yliopisto
Johtamisen ja tietotekniikan DI-ohjelma
Joulukuu 2020

Palvelutasosopimus on nykyään tavallinen osa asiakkaalle toimitetussa palvelussa. Sopimuksen kriteerien täyttymisen todistamiseen ja vikatiedoista nopean tiedon saamiseksi tarvitaan järjestelmämonitorointia. Tätä varten kohdeyrityksen SyncWare-ohjelmaan kuuluu siihen toteutettu oma järjestelmämonitorointi, joka on kuitenkin todettu puutteelliseksi. Dataa saadaan liian harvalla syklillä, kerätystä datasta muodostettava raportti sisältää käsin tehtäviä toimintoja, johon kuluu aikaa, uusien kerättävien raporttitietojen konfiguroiminen on monimutkaista ja ilmoituksen voi saada ainoastaan sähköpostilla tai tekstiviestillä.

Tässä diplomityössä tälle vanhalle järjestelmämonitoroinnille toteutettiin korvaava versio. Tutkimuskysymyksissä selvitettiin, millä tehokkaammilla järjestelmämonitorointityökaluilla uusi järjestelmämonitorointi toteutettaisiin ja paljonko vanhasta toiminnasta voidaan siirtää näille uusille työkaluille. Jäljelle jäävät toiminnot yhdistettiin uuden järjestelmämonitorointikokonaisuuden kanssa. Datasta piti myös pystyä muodostamaan kuvaajia niin, että niitä voi käyttää raporteissa. Lisäksi hälytyksistä pitää myös pystyä lähettämään ilmoitus keskusteluohjelma Slackiin ja tehtävä tehtävien hallinta järjestelmään Jiraan. Toteutusta varten perustettiin demo versio omalle työkoneelle ja uudelle palvelimelle, jolloin saatiin kerättyä dataa omasta koneesta ja siinä olevasta SyncWare-testiympäristöstä.

Käytettäviksi työkaluiksi valittiin datan keräämisen ja raportoinnin palvelu Telegraf, aikasarjatietokanta InfluxDB, datan prosessoinnin runko Kapacitor, käyttöliittymä ja haku- sekä visualisointityökalu Chronograf ja datan haku-, visualisointi- ja hälytystyökalu Grafana. Toteutuksessa saatiin yleinen ympäristöstä kerättävä data kuten prosessorin, kovalevyn ja muistin tilat sekä Java-koodin ajamiseen WWW-palvelimen ympäristössä tarkoitetun Tomcatin muistin ja säikeiden käyttö kerättäväksi Telegrafille. SyncWarelle itselleen jäi edelleen kerättäväksi tietokannan, klusterin, replikoinnin ja sanomien data. Lokin sisältö jätettiin toistaiseksi toteuttamatta, koska sillä ei ole väliä hälytysten kannalta ja keräämiselle joko ei ole toteutusta tai siinä on sopimuksellisia ongelmia. Palvelimien lokien ja tietokannan datan kerääminen odottaa sitä, että niille tulee toteutus Telegrafiin. Testauksen yhteydessä tarkistettiin, että data siirtyy InfluxDB:hen, sitä saadaan nopealla syklillä ja arvot vastaavat muiden monitorointi keinojen antamia arvoja samasta asiasta. Samalla testattiin raporttien muodostamista kerätystä datasta. Lisäksi katsottiin, että ilmoitus Slackiin ja tehtävä Jiraan toiminnot toimivat hälytystilanteissa. Uudesta toteutuksesta tehtiin konsepti toteutus mitä voidaan käyttää pohjan uuden järjestelmämonitoroinnin asentamiseen asiakkailla.

Tulevaisuudessa tämä uusi järjestelmämonitorointikokonaisuus on tarkoitus ottaa käyttöön asiakkaiden ympäristöissä. Parannettavaa on hälytys komentosarjojen toteutuksessa niin, että ne olisivat helposti kaikkien määritettävissä. Mahdollisesti on luotava ympäristöstä yksi iso komentosarja, joka sitten on kopioitavissa uudelle ympäristölle ja otettavissa käyttöön säätämällä komentosarjan alussa olevia muuttujia. Alustavan version esittelyssä kuukausipalaverissa palautteena oli, että viestinnän täytyisi olla mahdollista myös tavalla, mikä on täysin erillään muista viestintäkanavista niin kuin tekstiviestit olivat vanhassa järjestelmässä. Tähän sopiva keino voisi olla Pushover-ilmoitussovellus. Järjestelmämonitorointi kokonaisuudesta on myös tulossa uusi versio InfluxDB 2.0, joka tulee tarkoittamaan isoja muutoksia, mikäli se pyritään ottamaan käyttöön tulevaisuudessa. Siinä on kuitenkin luvattu, että vanhan version hakujen ja hälytyskomentosarjojen migraatio olisi mahdollista.

Avainsanat: Järjestelmämonitorointi, Telegraf, InfluxDB, Kapacitor, Chronograf, Grafana.

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

ABSTRACT

Toni Riikonen: Development of system monitoring: Renewal of collecting, storing, handling and displaying service level data
Master of Science thesis
Tampere University
Master's Programme in Management and Information Technology
12 2020

Service level agreement is nowadays regular part of service provided to customer. To prove that this agreement has been fulfilled and to get information about error situations quickly, system monitoring is needed. For this reason, target company's program SyncWare includes its own system monitoring, which however has been found lacking. Data is collected in too sparse cycle, reports formed from collected data include manual labor which takes time, configuring new collected report data is complicated and notification from alarm can only be received as text message or e-mail.

In this Master of Science thesis new system monitor version was developed to replace the old one. During research questions it was resolved with which more effective system monitoring tools this new version would be made and how much of the old functions could be moved for these new tools. The remaining functions were connected to the new system monitor. One should also be able to create diagrams from the data so that they can be used in reports. Additionally, you must be able to send notification to chat program Slack and create task to task control system Jira from alarms. For the implementation, demo version was created to my own computer and to new server where upon data could be collected from my computer and from a SyncWare test environment in it.

Chosen tools were data collector and reporter service Telegraf, time series database InfluxDB, data processing engine Kapacitor, user interface and data query and visualization tool Chronograf and data query, visualization and alarm tool Grafana. In the implementation, general collected data like states of processor, hard drive and memory and provider of running Java code in HTTP web service environment Tomcat's memory and thread usage were able to be given to Telegraf to be collected, to the SyncWare itself was left collection of database, cluster, replicator and message data. Contents of logs was left unimplemented for now since it does not matter in alarms and there is no implementation for them, or they have agreement related problems. Server logs and database data collection wait for their Telegraf implementation. During testing it was made sure that the collected data is transferred, it is received in fast cycle and values equate to the ones received from other monitoring means from the same thing. At the same time creating reports from collected data was tested. Also, it was made sure that the notification is send to Slack and task created to Jira in alarm situations. New solution was created as a concept which can be used as base when new system monitoring is installed to customers.

In the future this new system monitoring entity is meant to be implemented in customer environments. Improvements are needed in alarm scripts so that their implementation is easy for everyone. Possibly a one big script for one environment which can then be copied for copied and adjusted by changing the variables in the beginning of the script. During the display of initial version in monthly meeting, feedback was that the notification should also be possible to be received in a way that is separated from all the other communication channels like the text messages were in the old system. For this possible solution could be Pushover notification service. There is also new version of the monitoring entity coming InfluxDB 2.0, which would mean big changes if it were to be implemented in the future. Though promise has been made that migration of old version's queries and alarm scripts should be possible.

Keywords: System monitoring, Telegraf, InfluxDB, Kapacitor, Chronograf, Grafana.

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

ALKUSANAT

Tämä työ on tehty Syncron Tech Oy:lle. Kiitos yritykselle mahdollisuudesta tehdä työ. Kiitokset Ari Suutarille työn ideasta ja hänelle sekä Mikko Pukille avusta toteutuksessa. Lopuksi kiitos Timo Mäkiselle tekstin parannusehdotuksista.

Porissa, 7.12.2020

Toni Riikonen

SISÄLLYSLUETTELO

1. JOHDANTO 1
1.1 Työn tausta 1
1.2 Työn tavoitteet ja rajaukset 2
1.3 Työn lähestymistapa ja rakenne 3
2. JÄRJESTELMÄMONITOROINTI 5
2.1 Järjestelmämonitoroinnista yleisesti 5
2.2 Järjestelmämonitorointi työkaluja 8
2.2.1 TICK 9
2.2.2 Prometheus 13
2.2.3 Applications Manager 18
2.2.4 Collectd 20
2.2.5 Grafana 21
3. SYNCWARE 23
3.1 SyncWaresta yleisesti 23
3.2 SyncWaren järjestelmämonitorointi 29
3.2.1 Järjestelmämonitorointikokonaisuus 29
3.2.2 Asiakkaan SyncWare 31
3.2.3 Pommac 33
4. SUUNNITELMA 38
4.1 Järjestelmämonitorointityökalujen valinta 38
4.2 Uusi järjestelmämonitorointi 40
5. TOTEUTUS 45
5.1 Telegrafin keräämä data 45
5.2 SyncWaren keräämä data 46
5.3 Hälytykset 52
6. TESTAUS 55
6.1 Testaamisesta yleensä 55
6.2 Telegrafin keräämän datan testaaminen 58
6.3 SyncWaren keräämän datan testaaminen 60
6.4 Hälytysten testaaminen 61
7. YHTEENVETO 64
LÄHTEET 67

KUVALUETTELO

Kuva 1. 10 suosituinta DB-Enginesin listaamaa aikasarjatietokantaa..... 7
Kuva 2. TICK-kokonaisuus 9
Kuva 3. Esimerkki yksittäisestä pisteestä census-mittauksessa..... 11
Kuva 4. Chronografin raporttinäkymä..... 12
Kuva 5. Prometheuksen arkkitehtuuri 14
Kuva 6. Applications Managerin käyttöliittymä 19
Kuva 7. SSC Serv konfiguraation käyttöliittymä 20
Kuva 8. Grafanan raportti näkymä 21
Kuva 9. Replikaattorin luokkakaavio 25
Kuva 10. Aikalistan luokkakaavio..... 26
Kuva 11. Viestinnän luokkakaavio 27
Kuva 12. Hälytyksen luokkakaavio..... 28
Kuva 13. MonitorData-luokka..... 29
Kuva 14. SyncWaren järjestelmämonitoroinnin käyttötapauskaavio..... 30
Kuva 15. SyncWaren järjestelmämonitorointi kokonaisuus 31
Kuva 16. Datan keräyksen luokkakaavio 33
Kuva 17. Uuden järjestelmämonitoroinnin käyttötapauskaavio..... 43
Kuva 18. Uuden järjestelmämonitoroinnin kokonaisuus 44
Kuva 19. Kuvaajan haun asetukset..... 56
Kuva 20. Kuvaajan visuaaliset asetukset 57
Kuva 21. Kuvaajan yleiset asetukset..... 58
Kuva 22. CPU:n kuormitus Grafanassa 58
Kuva 23. Windowsin tehtävien hallinnan antama tulos..... 59
Kuva 24. Tomcatin HeapMemory PS Old Gen..... 59
Kuva 25. Tomcatin muistin käyttö 60
Kuva 26. Värjätty Grafana taulukko taltion tiedoista 61
Kuva 27. Kapacitorin konfigurointi Chronografissa..... 62
Kuva 28. Viestejä #alert-testing kanavalla 62
Kuva 29. Kapacitorista luotu Jiran tehtävä 63

OHJELMALUETTELO

Ohjelma 1. Lisäosan määrittely Telegrafissa *10*
Ohjelma 2. Esimerkki TICKscriptistä *13*
Ohjelma 3. Laskuri toteutettuna Javalla *15*
Ohjelma 4. Prometheuksen konfiguraatio *16*
Ohjelma 5. Prometheuksen hälytyksen hallinnan konfiguraatio *18*
Ohjelma 6. Prometheuksen hälytyssäännön konfiguraatio *18*
Ohjelma 7. Lisäosan määrittely Collectd *21*
Ohjelma 8. Win_perf_countersi-lisäosan määrittely *45*
Ohjelma 9. CPU-lisäosan määrittely *45*
Ohjelma 10. Tomcat-lisäosan määrittely *46*
Ohjelma 11. HTTP Listener v2 -lisäosan määrittely *46*
Ohjelma 12. TelegrafSystemMonitorin main metodi *47*
Ohjelma 13. Telegrafille lähetys *48*
Ohjelma 14. Replikointidatan keräys *48*
Ohjelma 15. Device-raporttien datojen keräys *50*
Ohjelma 16. Tarkkailtavat taltioiden haku *51*
Ohjelma 17. Taltion arvon tarkkailu *52*
Ohjelma 18. Hälytyskomentosarjan esimerkki *53*
Ohjelma 19. HttpPost-lisäosan konfiguraatio *54*

LYHENTEET

API	Application programming interface
BSD	Berkeley Software Distribution
CPU	Central Processor Unit
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
GDPR	General Data Protection Regulation
ICMP	Internet Control Message Protocol
JDBC	Java Database Connectivity
JSON	JavaScript Object Notation
PromQl	Prometheys Query Language
REST	Representational State Transfer
SLA	Service Level Agreement
SQL	Structured Query Language
TICK	Telegraf InfluxDB Chronograf Kapacitor
TOML	Tom's Obvious, Minimal Language
TSDB	Time Series Database
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WWW	World wide web
YAML	YAML Ain't Markup Language

1. JOHDANTO

Johdannossa käydään läpi, mikä työssä on lähtökohtana. Esitellään työn tavoitteet ja rajaukset. Lisäksi käydään läpi käytettävä tutkimusmetodi, jolla työtä on lähestytty, ja esitellään työn rakenne.

1.1 Työn tausta

Palveluna hankittujen ohjelmistojen (engl. Software as a Service) käyttäjät vaativat käyttämältään palvelulta tiettyä saatavuutta. Tällä tarkoitetaan yhteyden saamista palveluun ja mahdollisuutta käyttää sen kaikkia osia. Tätä varten palvelun myyjä ja ostaja sopivat palvelutasosopimuksen (engl. Service Level Agreement; lyh. SLA). Saatavuuden todistamiseksi, vikatiedoista nopean tiedon saamiseksi ja niiden mahdolliseen ennakointiin tarvitaan järjestelmämonitorointia. Monitoroinnilla havaitaan, diagnosoidaan ja ratkaistaan teknologia ympäristössä esiintyviä vikoja ja mahdollisesti ympäristön käyttäjiä haittaavia muita ongelmia. Tässä työssä uusitaan Syncron Tech Oy:n (jäljempänä: yritys) käyttämä nykyinen järjestelmämonitorointi.

Yritys toteuttaa tuotannon hallinnan ratkaisuja ja palveluita teollisuuslaitoksille. Ratkaisut ja palvelut koostuvat tuotannon suunnittelu- ja ohjaus-, raportointi-, tiedonkeruu- ja energianhallintajärjestelmistä tai niiden yhdistelmistä. Yrityksen kaikkien ohjelmistojen yhteisenä alustana toimii SyncWare-ohjelmisto, jonka tärkein toiminto on tiedon kerääminen, käsittely ja esittäminen asiakkaalle. SyncWarea käytetään selaimen kautta. Toiminnallisuuden ja sitä kautta saatavuuden tarkkailemista varten SyncWareen on toteutettu järjestelmämonitorointi, joka jakaantuu kahteen osaan. (1) Asiakkaan ympäristössä oleva SyncWare kerää tietoa muistin, kovalevyn ja prosessorin käytöstä palvelinympäristöstään, itsensä aiheuttamasta kuormituksesta ja eri osiensa kuten klusterissa olevien palveluiden, replikoinnin, sanomaliikenteen ja taltioiden keräämisen toimivuuksista. (2) Yrityksen omalla palvelimella on myös SyncWare, johon data kerätään ja jossa se käsitellään. Tämän järjestelmämonitoroinnin toiminnassa on kuitenkin puutteita, joita tässä työssä on tarkoitus parantaa tai korvata uudella toiminnalla. Uudesta kokonaisuudesta tehdään konseptitoteutus mitä voidaan käyttää sitten pohjana, kun uusi versio asennetaan asiakkaille.

1.2 Työn tavoitteet ja rajaukset

Työssä uudistetaan SyncWaren järjestelmämonitorointi sopivilla Internetistä löytyvillä järjestelmämonitorointityökaluilla ja tarvittaessa muokataan SyncWaren koodia. Nykyisessä sitä saadaan liian harvalla syklillä, joten lyhyistä tapahtumista ei saada tietoa. Monitorointia ei voi myöskään nopeuttaa, koska se aiheuttaisi liikaa kuormitusta valvottavalle järjestelmälle. Uutta järjestelmämonitorointikokonaisuutta on jo kehitelty jotain alustavaa ideaa mahdollisesta toteutuksesta ja sitä varten on jo otettu käyttöön uusi palvelin, jonne ympäristöä on alettu perustamaan. Tämä kokonaisuus koostuisi seuraavista järjestelmämonitorointityökaluista: datan keräämisen ja raportoinnin palvelusta Collectd, aikasarjatietokannasta InfluxDB ja datan haku-, visualisointi- ja hälytystyökalusta Grafana. Myöhemmin Collectd:n tilalle alettiin kaavailla toista datan keräämisen ja raportoinnin palvelua Telegrafia. Vaihtona on myös toinen suosittu järjestelmämonitorointikokonaisuus Prometheus, joka sisältää keinot datan keräämiseen ja käsittelemiseen sekä aikasarjatietokannan. Viimeisenä vaihtoehtona on yksittäisen asiakkaan käyttämä ManageEnginen Applications Manager, joka toimii kokonaan oman käyttöliittymänsä kautta ja sisältää tietokannan ja mahdollisuuden kerättävien datojen määrittämiseen, käsittelemiseen ja esittämiseen. Ehdotetut työkalut ovat esitelty aliluvussa 2.2. Mikäli vaikuttaa, että yrityksen idea on toimiva ratkaisu, noudatetaan sitä. (T) Tutkimus kysymyksenä on, millä näillä ehdotetuilla yleisillä työkaluilla nykyinen SyncWaren järjestelmämonitorointi voidaan korvata ja miten tämä tehdään.

(V1) Nykyisen järjestelmämonitoroinnin toiminnat korvataan uusilla järjestelmämonitorointityökaluilla niiltä osin kuin se on mahdollista. Uusien järjestelmämonitorointi työkalujen pitää pystyä keräämään dataa nopeammassa syklissä ja vähentää SyncWaren kuormaa datan keräämisen osalta. (V2) SyncWarelle jäljelle jäävät toiminnot yhdistetään uuden kokonaisuuden kanssa. Nykyisessä järjestelmämonitoroinnissa on myös muita ongelmia, jotka on tarkoitus ratkaista uudessa. Raportin muodostaminen kerätystä datasta edellyttää paljon aikaa vaativia manuaalisia tehtäviä ja uusien kerättävien raporttitietojen konfiguroiminen on tarpeettoman hankalaa. (V3) Uudessa näiden täytyisi olla helpommin tehtävissä. Nykyisessä järjestelmämonitoroinnissa hälytyksen voi saada ainoastaan tekstiviestillä. (V4) Uudessa se täytyy saada myös muilla keinoin kuten pikaviestipaluun esim. Slackiin (Slack 2020). Parannusten lisäksi uuden järjestelmämonitoroinnin täytyy pystyä samaan kuin nykyinen. Nykyinen järjestelmämonitorointi luo Jira-tehtävienhallintajärjestelmään (Atlassian 2020) hälytyksistä tehtäviä, joten (V5) uuden täytyy pystyä samaan. Lisäksi yksittäinen asiakas on toivonut pystyvänsä seuraamaan dataa heille toimitetun järjestelmän toiminnasta, joten (V6) tähän on löydettävä keino.

Tavoitteina ovat:

- V1: Korvata SyncWaren järjestelmämonitorointi uusien työkalujen toiminnoilla
- V2: Jäljelle jäävien toimintojen yhdistäminen uuden järjestelmämonitorointikokonaisuuden kanssa
- V3: Kuvaajien muodostaminen datasta niin, että niitä voi käyttää raporteissa uudella työkalulla
- V4: Hälytysten muodostaminen ja ilmoituksen lähettäminen Slackiin keskusteluohjelmaan uudella työkalulla
- V5: Tehtävien luominen Jiraan hälytyksistä
- V6: Kerätyn datan lähettäminen tälle yksittäiselle asiakkaalle heidän monitorointityökaluunsa, Applications Manageriin

Tavoitteet V3-6 toimivat myös vaatimuksina järjestelmämonitorointityökalujen valinnassa.

1.3 Työn lähestymistapa ja rakenne

Tutkimus noudattaa Basilin (1992) esittämää konstruktiivista metodia:

1. Tutki olemassa olevaa ratkaisua.
2. Ehdota parempaa ratkaisua.
3. Toteuta/kehitä.
4. Mittaa ja analysoi.
5. Toista prosessi, kunnes parannusmadollisuuksia ei ole jäljellä.

Työn toisessa luvussa käydään läpi järjestelmämonitoroinnin teoriaa ja esitellään tätä varten kehitettyjä kokonaisuuksia ja yksittäisiä työkaluja. Kolmannessa luvussa esitellään monitorointiin liittyvistä SyncWaren osista ja korvattavasta järjestelmämonitorointitoteutuksesta (1. askel). Käydään läpi, miten se on toteutettu, käytettävät tietokannat ja niihin kohdistuvien transaktioiden tarkkailu. Neljännessä luvussa perustellaan työkalujen valinnat ja esitetään suunnitelma tavoitteiden saavuttamiseksi toteutuksessa (2. askel). Viidennessä luvussa käydään läpi uuden

järjestelmämonitoroinnin toteutus (3. askel). Kuudennessa luvussa käydään läpi toteutetun ratkaisun validointi omassa ympäristössä (4. askel). Seitsemännessä luvussa tehdään yhteenveto kokonaisuudesta ja mahdollisesta jatkokehityksestä (5. askel).

2. JÄRJESTELMÄMONITOROINTI

Luvussa käydään läpi järjestelmämonitorointia yleisesti, niin kuin Turnbull (2018) on sen kirjassaan Monitoring with Prometheus kuvannut. Läpi käytäviä asioita ovat: mihin järjestelmämonitorointia käytetään, mitä siinä tehdään ja avainkohtia yleisesti ja ilmoitusten sekä visualisoinnin kannalta. Lisäksi esitellään lyhyesti aikasarjatietokanta ja työssä läpikäydyt järjestelmämonitorointityökalut.

2.1 Järjestelmämonitoroinnista yleisesti

Järjestelmämonitoroinnilla on kahdenlaisia käyttäjiä: teknologia ja liiketoiminta. Teknologiapuolella järjestelmämonitoria käytetään ympäristön ylläpitoon ja selvittämään sen tila. Monitorointi tuottaa tärkeän osan tuote- ja teknologiapäätöksiin vaikuttavasta datasta ja se mittaa projektien onnistumista. Järjestelmämonitorointi toimii tuotteen elinkaaren ja sisäisten asiakassuhteiden perustana sekä auttaa osoittamaan että raha on hyvin käytetty. Liiketoimintapuolella järjestelmämonitorointi tukee ja auttaa jatkamaan liiketoimintaa. Monitorointi antaa palautetta, joka auttaa yritystä tekemään kannattavia tuote ja teknologia sijoituksia. Lisäksi se auttaa mittaamaan teknologian tuottamaa arvoa. (Turnbull, 2018, s. 7–8)

Järjestelmämonitoroinnin tulisi työkaluna olla ydinosa infrastruktuurin ja liiketoiminnan hallintaa. Sen tulisi olla pakollinen osa toteutettuna toimitettaviin sovelluksiin. Järjestelmämonitorointia on kuitenkin hankala toteuttaa oikein, joten virheiden välttämiseksi voi noudattaa joitakin avainkohtia monitoroinnin helpottamiseksi ja huonosti toimivien suunnitelmamallien välttämiseksi:

- Monitorointi jälkiajatuksena: Parasta olisi määritellä heti alussa ennen sovelluskokonaisuuden toteutusta, mitä tarvitaan. Jos monitorointia ajattelee vain pienenä lisäarvoa tuottavana osana eikä ydintoimintona ja sitä aletaan toteuttaa vasta projektin päätteeksi, on erittäin todennäköistä, että joku tärkeä osa jää monitoroimatta.

- Monitorointi mekaanisena toistona: On huono idea vain toistaa samoja valvottavia asioita, mitä muualla on käytetty, jos nykyisen sovelluksen avainpalveluita jää monitoroinnista pois. Sovelluksella ei saa olla mahdollisuutta lakata toimimasta ilman, että sitä huomataan.

- Ei monitoroida oikeellisuutta: Ei ole riittävää, että vain valvotaan yksittäisen osan toimivuutta, koska silloin ei saada tietoa virheellisestä toiminnasta. Parempi olisi ensisijaisesti valvoa, että osa toimii oikein. Tällöin saadaan ilmoitus mahdollisista virheellisistä konfiguraatiosta ja ohjelmointivirheistä itse toiminnan lakkaamisen lisäksi.

- Monitoroidaan staattisesti: Mielivaltaisesti määritetty raja, kuten hälytystilanne CPU:n kuormituksen ylittäessä kahdeksankymmentä prosenttia, on tarpeettoman kankea. Se ei ota mitenkään huomioon, onko tilanne poikkeuksellinen vai onko niin tapahtunut luonnollisesta syystä. Hyvään monitorointiin kuuluu koko aikajakson tarkkaileminen eikä vain sen yksittäisen pisteen.

- Ei monitoroida tarpeeksi tiheään: Monissa järjestelmämonitorointityökaluissa on ongelmia skaalauksen kanssa tai oletustarkistusjakso on asetettu korkeaksi, esimerkiksi viisitoista minuuttia. Tällöin kriittisiä tilanteita jää huomaamatta. Monitoroinnin tulisi olla tarpeeksi tiheää, että voidaan havaita virheet ja poikkeukset, pystytään saavuttamaan valvonnalta ongelmatilanteessa odotettu vastausaika ja saadaan tarpeeksi selvää dataa määrittämään toiminnallisuus ja suunnat.

- Ei automaatiota tai itsepalvelua: Yleinen syy, miksi monitorointi on huonolaatuista tai väärin määritettyä, on se, että se on liian hankalaa. Monitoroinnin lisääminen ja käyttöönotto tulisi automatisoida niiltä osilta kuin on mahdollista ja käyttäjien tulisi pystyä tekemään määritykset itse ilman ulkopuolista apua.

(Turnbull, 2018, s. 8–14)

Järjestelmämonitoroinnin keräämä data jakaantuu ensisijaisesti kahteen muotoon: metriikkoihin ja lokeihin. Metriikat ovat ohjelmistojen ja laitteistojen ominaisuuksista otettuja mittauksia. Jotta metriikasta saadaan hyödyllistä, täytyy sen tila olla seurattavissa. Tämä tehdään tavallisesti tallettamalla kerätty metriikka datapisteinä ajan kuluessa niin, että tallenne koostuu arvosta, aikaleimasta ja mahdollisesti mittausta kuvaavista ominaisuuksista, kuten lähteestä tai tageista. Lokit ovat sovelluksien tuottamaa, yleensä tekstimuotoista, dataa sovelluksen tapahtumista ja ne hyödyllisiä virhetilanteiden tutkimisessa ja selvittämisessä. (Turnbull, 2018, s. 17–20)

Sopiva tietokanta järjestelmämonitoroinnin keräämälle datan säilyttämiseen on aikasarjatietokanta. Aikasarjatietokanta on tietokanta, joka on varta vasten tarkoitettu

aikaleiman omaavalle datalle. Se on optimoitu mittaamaan muutosta ajan kuluessa. Ominaisuuksiin kuuluu aikaleimallisen datan varastointi ja tiivistäminen, datan elinkaaren hallinta, datan yhteenveto ja kyky käsitellä suuria aikasarjahakuja. (Goldberg 2019; InfluxData, Inc. 2020a) Suosituimmat aikasarjatietokannat ovat esitettynä kuvassa 1. (solid IT GmbH 2020)

☐ include secondary database models — 32 systems in ranking, January 2020

Rank			DBMS	Database Model	Score		
Jan 2020	Dec 2019	Jan 2019			Jan 2020	Dec 2019	Jan 2019
1.	1.	1.	InfluxDB	Time Series	21.14	+0.81	+5.78
2.	2.	2.	Kdb+	Time Series, Multi-model	5.50	+0.02	+0.29
3.	3.	↑ 5.	Prometheus	Time Series	4.08	+0.14	+1.74
4.	4.	↓ 3.	Graphite	Time Series	3.35	+0.00	+0.50
5.	5.	↓ 4.	RRDtool	Time Series	2.78	+0.04	+0.17
6.	6.	6.	OpenTSDB	Time Series	1.97	-0.02	-0.16
7.	↑ 8.	↑ 8.	TimescaleDB	Time Series, Multi-model	1.92	+0.13	+1.14
8.	↓ 7.	↓ 7.	Druid	Multi-model	1.87	+0.04	+0.46
9.	9.	↑ 12.	FaunaDB	Multi-model	0.80	+0.11	+0.50
10.	↑ 11.	↓ 9.	KairosDB	Time Series	0.56	0.00	+0.06

Kuva 1. *10 suosituinta DB-Enginesin listaamaa aikasarjatietokantaa.*

Hälytykset ja ilmoitukset ovat järjestelmämonitorointityökalujen pääasiallinen tuotos. Hälytys laukaistaan, kun jotain huomattavaa tapahtuu eli esimerkiksi jokin raja-arvo ohitetaan. Tästä lähetetään ilmoitus halutuille henkilöille tai muille kohteille esimerkiksi sähköpostin tai tekstiviestin muodossa. Tässä on hyvä harkita tiettyjä perusseikkoja:

- Mistä ongelmista tehdään ilmoitus.
- Kenelle tästä ongelmasta kerrotaan.
- Miten heille kerrotaan.
- Kuinka usein heille kerrotaan.
- Milloin lakataan kertomasta heille, tehdään jotain muuta tai eskaloidaan ilmoitus jollekin toiselle.

Täytyy olla varovainen, että ei lähetä ilmoituksia liian tiheään. Tällöin tahtiin ei pystytä reagoimaan, ilmoitusten saajat saattavat mykistää ne tai heiltä voi jäädä jotain kriittistä huomaamatta. Vielä tärkeämpää on miettiä, mitä ilmoituksessa kerrotaan. Siitä pitää

saada selville, mikä on pielessä ja mikä kaipaa huomiota. Asioita, joihin ilmoituksen määrittelemisessä tulisi keskittyä, ovat:

- Ilmoitusten tekeminen selkeiksi niin, että ihminen ymmärtää mitä niissä lukee.
- Kontekstin lisääminen ilmoituksiin niin, että niistä käy ilmi mitä asia tarkkaan ottaen koskee.
- Ilmoitusten lähettäminen ainoastaan silloin kun se on järkevää.

(Turnbull, 2018, s. 39–41)

Datan visualisointi on erittäin tehokas analysointi-, tulkitsemistekniikka- ja oppimistyökalu. Datan ja sen visualisoinnin tulkitseminen voi kuitenkin olla hankalaa. Ihmisellä on taipumus nähdä yhteyksiä siellä missä niitä ei ole ja näin tehdä vääriä syy-seurauspäätelmiä. Visualisoinnissa kannattaa noudattaa muutamia yleissääntöjä:

- Esitä data selkeästi.
- Johdata katsoja keskittyminen olennaiseen, älä ulkonäköön.
- Vältä datan vääristämistä.
- Tee isoista data kokonaisuuksista yhtenäisiä.
- Mahdollista tarkkuuden säätäminen ilman että sillä on vaikutusta ymmärrykseen.

(Turnbull, 2018, s. 41–42)

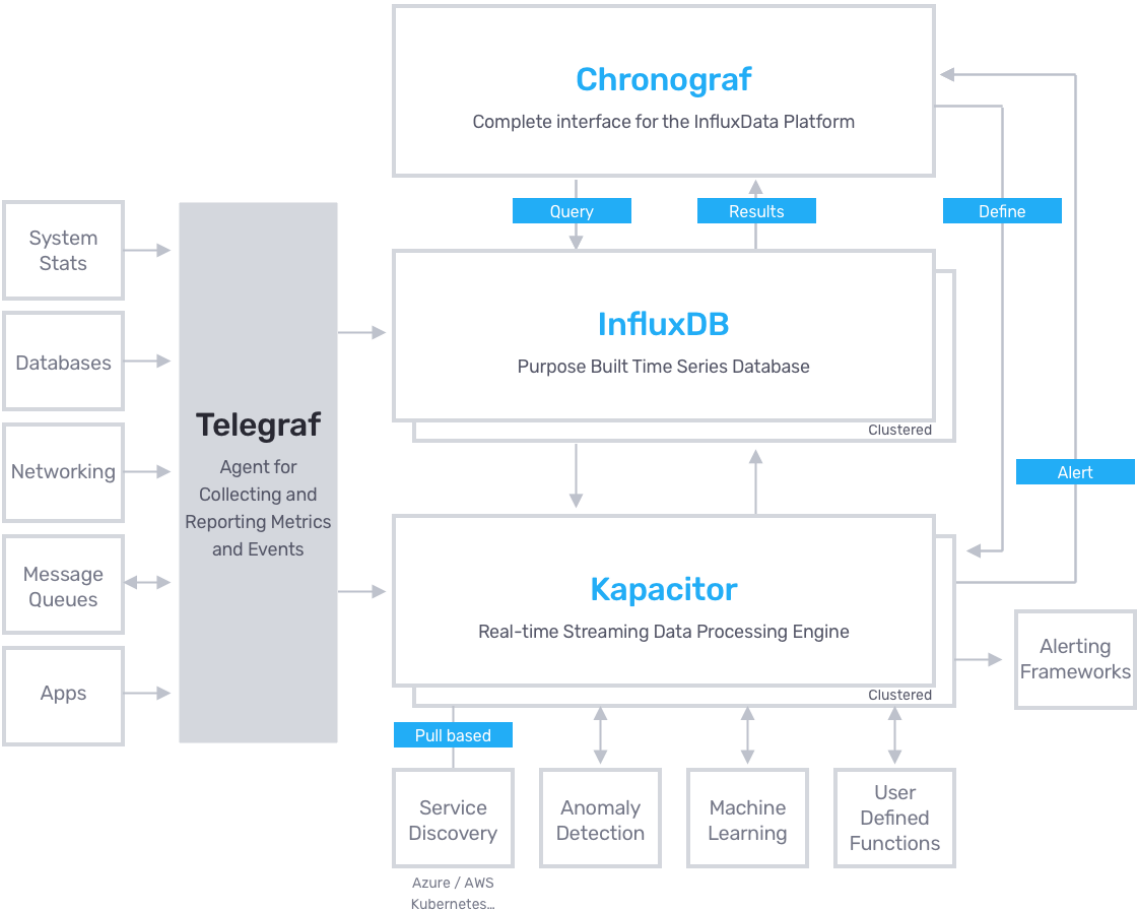
2.2 Järjestelmämonitorointi työkaluja

Valitut tarkasteltavat työkalut ovat TICK, Prometheus, Applications Manager, Collectd ja Grafana. Collectd oli ensimmäinen datan keräämiseen tarkoitettu työkalu, johon yritys törmäsi päättäessään alkaa uudistamaan järjestelmämonitorointia. TICK sisältää kuvassa 1 DB-Enginesin mukaan suosituimman aikasarjatietokannan InfluxDB, joka on ollut yrityksellä aikaisemminkin käytössä. Toisena oleva Kdb+ näytti omilla sivuilla antaman kuvauksen mukaan olevan pelkästään aikasarjatietokanta eikä sisältänyt mitään toimintoja datan keräämiseen, joten tämä ohitettiin. Prometheus oli listan kolmantena ja Turnbillin (2018) valitsema työkalu omassa toteutuksessaan. Grafana oli suositeltu työkalu Prometheuksessa, yrityksen visuaaliseen esittämiseen suunnittelema

työkalu ja myös yleisesti pidetty verkossa. Applications Manager on yksittäisen asiakkaan, joka myös käyttää paljon järjestelmämonitorointia, suosima työkalu.

2.2.1 TICK

TICK on InfluxDatan virallisesti tukema kokonaisuus, joka kattaa koko järjestelmämonitorointikokonaisuuden: datan keräämisen Telegraf, käsittelemisen Kapacitor, säilyttämisen InfluxDB ja esittämisen Cronograf. Kokonaisuus on esitetty kuvassa 2. (InfluxData Inc., 2020b)



Kuva 2. *TICK-kokonaisuus*

Datan kerääminen

Telegraf on datan keräämisen ja raportoinnin palvelu. Se on toteutettu kokonaan Go-ohjelmointikielellä ja kuormittaa minimaalisesti muistia. Sille on tehty monia sisääntulon lisäosia (engl. plugin), jotka mahdollistavat datan keräämisen suoraan järjestelmästä mihin se on asennettu, kolmannen osapuolien ohjelmointirajapinnoista (lyh. API) tai muilta datan kerääjiltä kuten statsd- tai Kafka Consumer -palveluista. Samoin sille on

ulostulon lisäosia, jotka mahdollistavat datan lähettämisen influxDB:hen ja muihin yhteensopiviin tietokantoihin. (InfluxData Inc., 2020b)

Telegraf toimii palvelimella palveluna. Sen asetusten säätö tapahtuu muokkaamalla telegraf.conf-tiedostoa käytettävien lisäosien määrittämiseksi. Määritys tapahtuu Tom's Obivious Minimal Language (lyh. TOML) formaatilla. Se koostuu kolmesta osasta:

1. Yleiset tagit (engl. global tags), jotka annetaan kaikille lähetettäville mittauksille ja joita voidaan käyttää tietokantahaussa.

2. Toimijan asetukset (engl. agent settings), jossa määritellään yleiset asetukset, kuten datan keräyksen tiheys aikaväli.

3. Lisäosat, jossa määritetään käytettävät lisäosat ja niiden asetukset.

Esimerkkinä datan lähetys Influxdb:hen sen lisäosalla on esitetty ohjelmassa 1. (InfluxData Inc. 2020c)

```
[[outputs.influxdb]]
  urls = [ "http://example.org:8086" ]
  database = "telegraf"
```

Ohjelma 1. *Lisäosan määrittely Telegrafissa*

Datan säilyttäminen

InfluxDB on avoimen lähdekoodin Go-kielellä tehty aikasarjatietokanta (InfluxData Inc. 2020d). Sen etuna muihin aikasarjatietokantoihin on, että se on alusta alkaen aikasarjatietokannaksi tarkoitettu (InfluxData Inc. 2020a). InfluxDB-tietokanta pohjautuu sarakeperhetietokannan tietorakenteeseen ja koostuu pisteistä (engl. point), jotka kuuluvat johonkin mittaukseen (engl. measurement). Mittaus kuuluu johonkin säilytyspolitiikkaan (engl. retention policy), joka määrää, kuinka kauan dataa ja kuinka monta kopiota datasta säilytetään. Pisteellä on aikaleima ja kentät, joissa on mitatut arvot, sekä mahdollisesti tageja. Kuvassa 3 on esitetty esimerkki yksittäisestä pisteestä census mittauksessa, jolla on aikaleima, kenttinä butterflies ja hoybees ja tageina location ja scientist. (Carnegie Mellon Database Group, 2020a; InfluxData Inc., 2020e)

```
name: census
-----------------
time                    butterflies honeybees   location    scientist
2015-08-18T00:00:00Z    1           30          1           perpetua
```

Kuva 3. *Esimerkki yksittäisestä pisteestä census-mittauksessa*

Tietokannan kyselykieli InfluxQL on pyritty tekemään samankaltaiseksi kuin relaatiotietokannan SQL. Samankaltaisilla SQL-tyylisillä komennoilla saa aikaiseksi samankaltaisen tapahtuman kuin SQL:ssä, joista mainittakoon:

- CREATE DATABASE: Tietokannan luonti.
- DROP DATABASE: Tietokannan poisto.
- DROP MEASUREMENT: Mittauksen poisto.
- DROP SERIES: Pisteiden poisto mittauksesta
- INSERT: Pisteen luonti mittaukseen.
- SELECT: Haku, jossa on määritetty minkä sarakkeiden arvot haetaan.
    - FROM: Mistä mittauksesta arvot haetaan.
    - WHERE: Haku ehdot.

Esimerkiksi

SELECT "host", "region", "value" FROM "cpu" WHERE "value" > 0.9

haulla haettaisiin host, region ja value kenttien arvot mittauksesta cpu niissä tapauksissa, kun value on yli 0,9. Datojen aikaleimat kuuluvat oletusarvoisesti hakutulokseen. (InfluxData Inc., 2020f; InfluxData Inc., 2020g; InfluxData Inc., 2020h)

Datan esittäminen

Chronograf on selaimen kautta käytettävä TICK:n käyttöliittymä, tosin tietokantaa ja hälytyksiä voi myös hallita käyttöjärjestelmän komentosyötteen kautta. Kun sille on määritelty käytettävät InfluxDB:t ja Kapacitorit, sitä voi käyttää tietokannan ja hälytysten hallintaan. InfluxDB:hen voi syöttää sen käyttämiä komentoja ja hakutuloksista voi muodostaa kuvaajia ja kuvaajista raporttinäkymiä (engl. dashboard). Kapacitorille voi

luoda ja muokata hälytyssääntöjä. Lisäksi hälytyksistä voi myös luoda raporttinäkymiä. Kuvassa 4 on esimerkki Chronografin raporttinäkymästä. (InfluxData Inc., 2020i; InfluxData Inc., 2020j; InfluxData Inc., 2020k)

Kuva 4. *Chronografin raporttinäkymä*

Datan käsittely

Kapacitor on avoimen lähdekoodin datan prosessoinnin runko, joka mahdollistaa hälytysten ja dataa muokkaavien prosessien luomisen sekä poikkeavuuksien havaitsemisen. Sillä voi prosessoida dataa virtana (engl. stream) tai erinä (engl. batch). Lisäksi se voi suorittaa hakuja InfluxDB:hen aikataulussa ja vastaanottaa dataa linjaprotokollana sekä muilla tavoilla, joita InlfuxDB tukee. Siihen voi myös lisätä käyttäjän määrittämiä toimintoja poikkeavuuksien havaitsemiseksi ja sen voi yhdistää Slack:in ja myös joidenkin muiden keskusteluohjelmien kanssa. (InfluxData Inc., 2020l)

Kapacitorin toiminnon voi määritellä siis toimimaan virtana, jolloin kaikki data, mikä kirjoitetaan InfluxDB:hen, peilataan myös Kapacitoriin, jonka seurauksena Kapacitorin on talletettava dataa kovalevylle. Toisena vaihtoehtona oli määritellä se käsittelemään eriä, jolloin hakuja suoritetaan InfluxDB:hen säännöllisin väliajoin ja data käsitellään sellaisena kuin se hausta saadaan.

Toiminto määritellään datan prosessointiputkina (engl. pipeline) käyttäen TICK:n omaa TICKscript kieltä kuten ohjelmassa 2.

```
dbrp "telegraf"."autogen"
```
2

```
stream
    // Valitse vain cpu measurement.
    |from()
        .measurement('cpu')
    |alert()
        .crit(lambda: int("usage_idle") <  70)
        // Hälytystilanteessa kirjoitetaan merkintä tiedostoon.

        .log('/tmp/alerts.log')
```

Ohjelma 2. *Esimerkki TICKscriptistä*

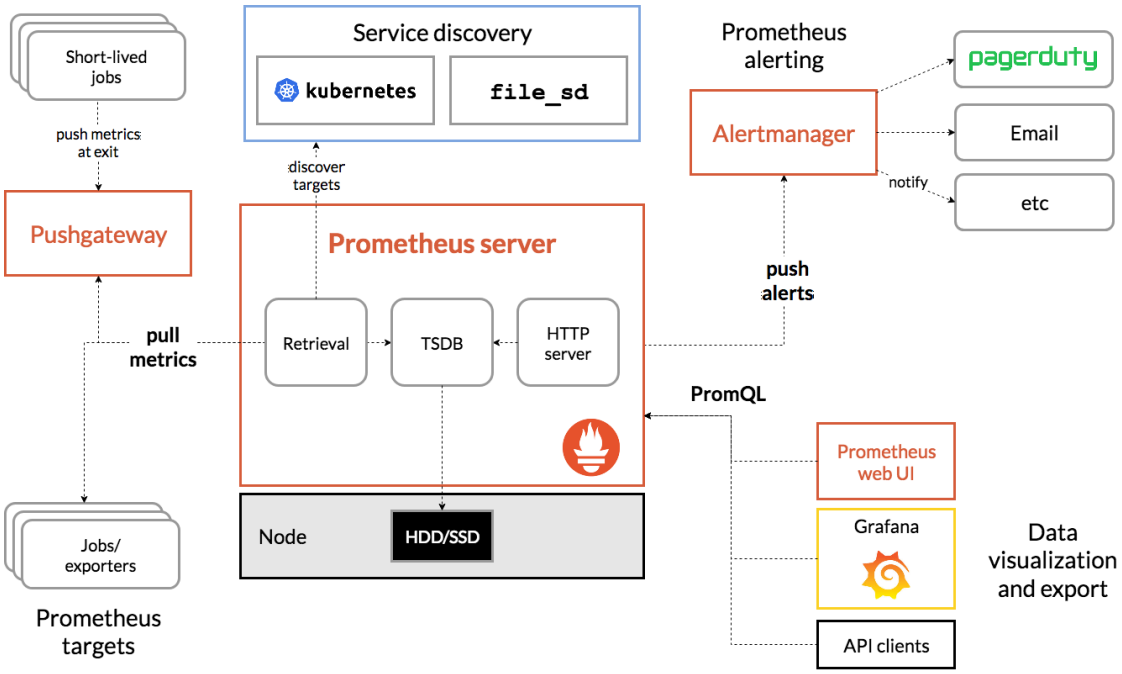
Esimerkissä haetaan telegraf taulusta virtana mittaus cpu. Saadusta datasta tarkkaillaan usage_idle arvoa ja silloin kun se on alle 70, kirjoitetaan hälytys tmp-kansiossa sijaitsevaan alerts.log-tiedostoon. (InfluxData Inc., 2020m)

2.2.2 Prometheus

Prometheus on itsenäinen avoimen lähdekoodin järjestelmämonitorointikokonaisuus. Alun perin Matt T. Proudin ja Julios Volzin aloittama projekti SoundCloudin sponsoroimana. Kokonaisuus on ensisijaisesti toteutettu Go-ohjelmointi kielellä mutta myös Javaa, Pythonia ja Rubyä on hieman käytetty. Kokonaisuuteen kuuluu:

- Prometheuksen pääpalvelin, joka kokoaa ja tallentaa aikasarjadatan.
- Asiakaskirjastot sovelluskoodin instrumentoimiseen.
- Työntöyhdyskäytävä (engl. Push gateway) lyhythetkisille töille.
- Erityistarkoituksen omaavat viejät datan siirtämiseksi toiseen aikasarjatietokantaan.
- Hälytysmanageri hälytyksien hallintaan.
- Muita mahdollisia tukityökaluja.

Kokonaisuus on esitelty kuvassa 5. Omaa datan esittämisen työkalua ei Prometheukseen kuulu vaan siihen suositellaan käytettäväksi Grafanaa. (Prometheus Authors, 2020a; Prometheus Authors, 2020b)



Kuva 5. *Prometheuksen arkkitehtuuri*

Datan kerääminen

Datan keräämiseksi täytyy monitoroitavan palvelun koodiin lisätä instrumentointi käyttäen Prometheuksen asiakaskirjastoja. Tuki löytyy Javalle, Scalalle ja muutamille muille ohjelmointikielille sekä epävirallisilla kolmannen osapuolen kirjastoilla muille. Kirjastoilla määritellään koodiin lähetettävä data ja vaihtoehtoina ovat:

- Laskuri, minkä arvoa voi kasvattaa tai nollata
- Mittari, minkä arvoa voi kasvattaa tai vähentää mielivaltaisesti
- Histogrammi, millä voi tallettaa kokoelman dataa jollain aikavälillä, datan koon ja niiden määrän ryhmissä (engl. buckets)
- Yhteenveto, millä voi tallettaa kokoilman dataa jollain aikavälillä, datan koon, datojen määrän ja määritetyt kvantiilit

(Prometheus Authors, 2020c; Prometheus Authors, 2020d)

Prometheuksen käydessä monitoroitavan kohteen läpi, kirjasto lähettää kaiken koodissa määritetyn datan. Esimerkiksi Javalle laskuri määriteltäisiin tyyliin niin kuin ohjelmassa 3. (Prometheus Authors, 2020e)

```
import io.prometheus.client.Counter;
class YourClass {
  static final Counter requests = Counter.build()
     .name("requests_total").help("Total requests.").register();

  void processRequest() {
    requests.inc();
    // Oma koodi.
  }
}
```

Ohjelma 3. *Laskuri toteutettuna Javalla*

Vaihtoehtoisesti lyhytaikaiset (engl. ephemeral) ja eränä (engl. batch) suoritettavat tehtävät voivat lähettää datansa työntöyhdyskäytävälle, mistä Prometheus sitten hakee ne.

Prometheuksen konfigurointi tapahtuu YAML Ain't Markup Language (lyh. YAML) kielellä. Konfiguroinnin määrittelyt toteutetaan esimerkiksi tiedostoon prometheus.yml ja tiedosto annetaan käynnistyksen yhteydessä komentoon:

./prometheus --config.file=prometheus.yml.

Määrittelyistä mainittakoon:

- global: Oletusasetukset.
- rule_files: Käytettävät säännöt.
- scrape_config: Määritellyt hakukohteet ja haun parametrit.
- static_config: Haku kohteen määrittely.
- aletmanager_config: Hälytyksen hallinnan konfiguraatio.

(Prometheus Authors, 2020f)

Esimerkkinä Prometheuksen itsensä kuunteleminen portista 9090 kuunteleva konfiguraatio on esitetty ohjelmassa 4. (Prometheus Authors, 2020g)

```
global:
  scrape_interval:     15s # Käy kohteet läpi 15 sekunnin välein.

  # Annetaan tämä aikaleima kaikille aikasarjoille tai hälytyksille,
kun
  # kommunikoidaan ulkopuolisten systeemien kanssa (keskusjärjestön,
  # etävarasto, Alertmanager)
  external_labels:
    monitor: 'codelab-monitor'

```

```
# Hakukonfiguraatio sisältää yhden päätepisteen, joka käydään läpi.
# Tässä tapauksessa Prometheus itse.
scrape_configs:
  # Tehtävän nimi lisätään leimana `job=<job_name>` kaikkiin
  # aikasarjoihin, jotka haetaan tässä konfiguraatiossa.
  - job_name: 'prometheus'

    # Yliaja globaali arvo ja suorita tehtävä 5s välein.
    scrape_interval: 5s

    static_configs:
      - targets: ['localhost:9090']
```

Ohjelma 4. *Prometheuksen konfiguraatio*

Käytettäviin sääntöihin voi lisätä YAML tiedostoon kirjotettuja tallennussääntöjä, joissa kerättävää dataa voi muokata toisenlaiseen tallennettavaan muotoon normaalin tallennuksen lisäksi. (Prometheus Authors, 2020h)

Datan säilyttäminen

Tietokanta pohjautuu avain-arvoparitietokannan logiikkaan. Datan nimi toimii avaimena. Arvot koostuvat aikaleimasta ja aikaisemmin mainitusta datatyypeistä eli laskureista, mittareista, histogrammeista ja yhteenvedoista. (Carnegie Mellon Database Group 2020b). Hakukielenä toimii Prometheus Query Langueage (lyh. PromQl), jossa haun määrittely jakaantuu kolmeen osaan:

- Pikavektorin (engl. instant vector) valinnat, jossa määritetään, mistä haetaan dataa ja tarkennettu haku annetaan aaltosulkujen {} sisään.
- Aluevektorin valinnat, jossa määritellään haun aikaväli hakasulkeiden sisään [].
    - Aikavälille voi haluttaessa antaa myös resoluutio kaksoispisteellä : erotettuna
- Vaihesiirron määrite, jossa voi siirtää aloitusaikaa nykyhetkestä lisäämällä komentoon offset.

Lisäksi käytettävänä on ohjelmointikielen operaatiot, kuten + summan laskemiseksi ja funktiot kuten abs() keskiarvon laskemiseksi. (Prometheus Authors 2020i) Esimerkiksi komennolla

http_requests_total{job="apiserver", handler="/api/comments"}[5m]

haetaan http_request_total datasta ne arvot, joilla on job apiserver ja handler /api/comments viimeisen viiden minuutin ajalta ja komennolla

rate(http_requests_total[5m])[30m:1m]

haettaisiin http_request_total datasta rate funktiolla pyyntöjen määrä per sekunti viiden minuutin aikana ja hakualueena on viimeiset 30 minuuttia ja resoluutiona minuutti. (Prometheus Authors, 2020k)

Datan käsittely

Prometheuksen hälytysrakenne koostuu kahdesta osasta: hälytyssäännöistä ja hälytyksen hallinnasta. Hälytyssäännöt Prometheus-palvelimella lähettää hälytykset hälytyksen hallintaan. Hälytyksen hallinta käsittelee hälytykset vaientaen, estäen tai kokoamalla ne ja lähettämällä ilmoituksen sähköpostiin, päivystävään ilmoitusjärjestelmään tai keskustelualustalle. Toimintaa varten on konfiguroitava hälytyksen hallinta, konfiguroitava edellä mainittu Prometheuksen yhteys hälytyksen hallintaan ja luotava hälytyssäännöt Prometheukselle. (Prometheus Authors, 2020l)

Samoin kuin Prometheuksen konfiguraatiossa hälytyksen hallinnan konfiguraatio tehdään YAML-tiedostoon, joka annetaan hälytyksen hallinnalle sen käynnistyksen yhteydessä komennolla

```
./alertmanager --config.file = alertmanager.yml.
```

Konfiguraatiotiedostossa määritellään polku, jossa määritellään, mitä saapuville hälytyksille tehdään. Ohjelman 5 esimerkissä *service* täytyy olla *mysql* tai *cassandra*, jotta se lähetettäisiin *database-pagerille* ja *team* täytyy olla *frontend*, jotta se *lähetettäisiin frontend-pagerille*.

```
# Juurireitti kaikkine parametreineen, jotka peritään lapsireiteissä
# ellei niitä ajeta yli.
route:
  receiver: 'default-receiver'
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 4h
  group_by: [cluster, alertname]
  # Kaikki hälytykset, jotka eivät vastaa seuraavia lapsireittejä,
  # pysyvät juuritasolla ja lähetetään ‘default-receiverille’,
  routes:
  # Hälytykset, joilla on service=mysql tai service=cassandra
  # lähetetään ‘database-pagerille’.
  - receiver: 'database-pager'
```

```
      group_wait: 10s
16    match_re:
        service: mysql|cassandra
18  # Kaikki hälytykset, joilla on team=frontend leima vastaavat tätä
    # alireittiä. Ne ryhmitetään productin ja enviromentin mukaan
20  # clusterin ja alertnamen sijaan.
    - receiver: 'frontend-pager'
22    group_by: [product, environment]
      match:
        team: frontend
```

Ohjelma 5. *Prometheuksen hälytyksen hallinnan konfiguraatio*

Näille hälytyksille määritellään receiver-määrittelyyn kyseiset vastaanottajat ja niiden asetukset. (Prometheus Authors, 2020m)

Samoin kuin datan keräämisen tallennussäännöt hälytyssäännöt kirjoitetaan YAML-tiedostoon ja annetaan *rule_files*-määritteessä. Esimerkkinä on ohjelmassa 6 esitetty hälytyksen luonti tilanteesta, jossa pyynnön kesto ylittää puoli sekuntia. (Prometheus Authors, 2020n)

```
  groups:
2 - name: example
    rules:
4   - alert: HighRequestLatency
      expr: job:request_latency_seconds:mean5m{job="myjob"} > 0.5
6     for: 10m
      labels:
8       severity: page
      annotations:
10      summary: High request latency
```

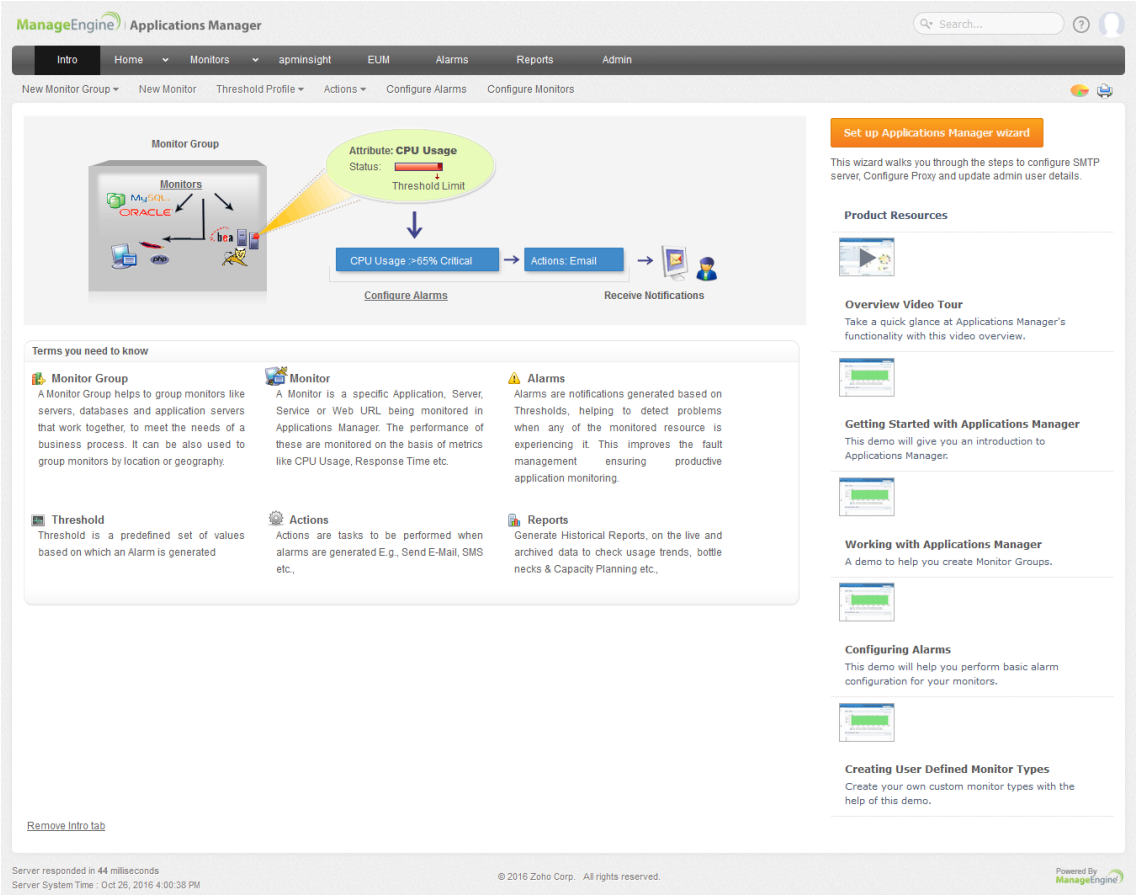
Ohjelma 6. *Prometheuksen hälytyssäännön konfiguraatio*

2.2.3 Applications Manager

Applications Manager on ManageEnginen kehittämä maksullinen järjestelmämonitorointityökalu. Myös ilmainen versio on olemassa, jossa voi valita viisi monitoroitavaa asiaa oletusarvoisesti monitoroitavien asioiden lisäksi.

Datan säilyttäminen ja esittäminen

Asennus tapahtuu asennuspaketin ajamisella, jonka yhteydessä valitaan Applications Managerin käyttämä tietokanta, joka voi olla PostgreSQL tai Microsoft SQL Server. (Zoho Corp, 2020a). Konfiguraatio tapahtuu ohjelman selainpohjaisen käyttöliittymän kautta, joka on esitetty kuvassa 6. Käyttöliittymässä nähdään myös kerätty data, kun kerääminen on määritetty. (Zoho Corp, 2020b; Zoho Corp, 2020c)



Kuva 6. *Applications Managerin käyttöliittymä*

Datan kerääminen

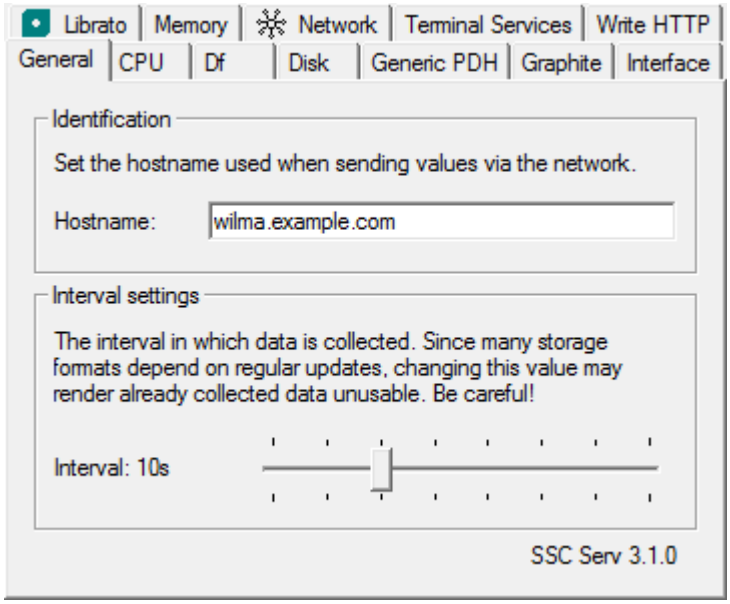
Monitoroinnin täytyy kuulua johonkin monitorointiryhmään. Ryhmä voi olla tyypiltään tavallinen monitorointiryhmä tai web-sovellusryhmä. Ryhmällä on nimi, kuvaus, käyttöoikeudet ja siihen voi kuulua aliryhmiä. Ryhmän määrittämisen jälkeen ryhmään voi lisätä monitorointeja. Monitoroitava kohde voi olla samalla palvelimella kuin itse Applications Manager tai eri palvelimella. Applications Managerilla on lista sen tukemia sovelluksia, pilvisovelluksia, tietokantoja, toiminnanohjausjärjestelmiä, postipalveluita, käyttöjärjestelmiä, väliohjelmistoja, virtuaalikoneita, konvergoituja infrastruktuureja, web-palveluita ja palveluita. Näistä voi valita jonkun monitoroitavaksi ja syöttää siihen tarvittavat asetukset. (Zoho Corp, 2020d; Zoho Corp, 2020e; Zoho Corp, 2020f) Järjestelmän valvoja pystyy luomaan omia mukautettuja monitoreita. Mukautetussa monitoroinnissa monitorointi tapahtuu ajettavalla Linux Shell Script tai Windows Batch File -komentosarjoilla. (Zoho Corp, 2020g)

Datan käsitteleminen

Hälytyksiä varten ovat raja-arvoprofiilit ja toiminnot. Raja-arvoprofiilissa määritetään kriittisen, varoituksen ja ohi menemisen rajat ja tarkkailtavaksi arvoksi voi ottaa keskiarvon pidemmällä aikavälillä. Toiminnoissa määritellään mahdolliset tehtävät asiat, mitä voivat olla muun muassa: ilmoituksen lähetys sähköpostitse tai tekstiviestillä, ohjelman suorittaminen tai verkkokoukun (engl. webhook) toiminto. Verkkokoukulla voi käyttää toimintoja toisessa verkkosovelluksessa. Tämän jälkeen monitorointiryhmässä voi monitoroinnilta saatavalle arvolle määrittää käytettävät raja-arvoprofiilit ja raja-arvon ohittamisesta suoritettavat toiminnot. (Zoho Corp, 2020h; Zoho Corp, 2020i; Zoho Corp, 2020j; Zoho Corp, 2020k)

2.2.4 Collectd

Collectd on datan keräämiseen ja raportointiin tarkoitettu palvelu. Se on toteutettu C-ohjelmointikielellä ja omaa kattavan määrän mahdollisia lisäosia. (collectd, 2019a). Windows-versiossa asetusten määrittelyn voi tehdä SSC Serv nimisen käyttöliittymän kautta, joka on esitetty kuvassa 7. Tarkkailtavien kohteiden asetusten määritykset löytyvät omilta välilehdiltään. (collectd, 2019b)



Kuva 7. *SSC Serv konfiguraation käyttöliittymä*

Unix-versiossa asetusten määrittäminen tapahtuu collect.conf-tiedostossa. Lisäosan saa käyttöön komennolla LoadPlugin ja lisäosan asetuksia voi muuttaa komennossa plugin. Ohjelmassa 7 on esitetty esimerkkinä CPU-lisäosan käyttöönotto niin, että

oletusasetusten lisäksi käytettävissä olevien CPU:den lukumäärä raportoidaan. (colelctd, 2019c; *collectd, 2019d*)

```
LoadPlugin cpu
<Plugin cpu>  ReportNumCpu true  </Plugin>

```

Ohjelma 7. *Lisäosan määrittely Collectd*

2.2.5 Grafana

Grafana on avoimen lähdekoodin datan haku-, visualisointi- ja hälytystyökalu. Se on yhteensopiva useiden eri tietokantojen kanssa kuten InfluxDB:n, Prometheuksen tietokannan ja monen muun. Useita tietokantalähteitä voi käyttää samassa kuvaajassa. (Grafana Labs, 2020) Kuvassa 8 on esimerkki Grafanan raporttinäkymästä. (Grafana Labs, 2020)

Kuva 8. *Grafanan raportti näkymä*

Tietokantoihin suoritetusta hausta saadun datan visualisointiin löytyy useita vaihtoehtoja lämpökartasta histogrammiin ja lisää vaihtoehtoja on saatavissa lisäosien kautta. Näistä kuvaajista muodostetaan raporttinäkymiä, jolle voi antaa muuttujia kaikkien kuvaajien hakuehtojen nopeaan muuttamiseen ja näkymän voi jakaa kahtia eri hakujen vierekkäistä vertailua varten. (Grafana Labs, 2020)

Lokilisäosalla voi tarkastella hauista muodostuneita lokeja. Kuvaajiin voi myös kirjoittaa omia ilmoituksia ja kuvaajille voi visuaalisesti antaa hälytysehtoja ja ilmoituksen hälytyksistä pystyy lähettämään yhteensopiviin keskusteluohjelmiin kuten Slack. (Grafana Labs, 2020)

3. SYNCWARE

Luvussa esitellään yrityksen tuote SyncWare niiltä osin kuin se on merkittävä järjestelmämonitoroinnin kannalta. Käydään läpi, miten se on toteutettu, käytettävät tietokannat ja niihin kohdistuvien transaktioiden tarkkailu. Käydään läpi myös malliluokkien ja taltioiden toiminta sekä raportit. Malliluokka mahdollistaa Java-luokkien toiminnan tietokannan kanssa. Näistä malliluokista käydään läpi olennaisimmat. Taltiot ovat oma toteutuksena suuria mittausmääriä varten. Lisäksi esitellään SyncWareen toteutettu järjestelmämonitorointi. Käydään läpi, miten se kerää dataa asiakkaan palvelimella ja miten yrityksen palvelimella (”Pommac”) toimiva SyncWare hakee ja käsittelee kyseisen datan sekä vanhoja toteutuksia datan hausta.

3.1 SyncWaresta yleisesti

SyncWaren käyttöliittymä on toteutettu Apache Wicket -käyttöliittymäkirjastolla, joka mahdollistaa Java-ohjelmointikielen käyttämisen yhdessä HTML-kuvauskielen ja CSS-tyylimääritysten kanssa. Ohjelmisto toimii asiakkaan palvelimella, joka voi olla käyttöjärjestelmältään Windows, Unix tai Linux, Apache Tomcat palvelussa ja sitä käytetään selaimen kautta. Sen keskeisin toiminto on tiedon kerääminen, käsittely ja esittäminen asiakkaan haluamassa muodossa. (Helenius, 2010, s. 20)

SyncWare voi käyttää datan säilyttämiseen ja käsittelyyn yhtä tai useampaa kolmesta eri relaatiotietokannasta: PostgreSQL, Oracle tai Microsoft SQL Server. Tämän mahdollistaa Java Database Connectivity (lyh. JDBC) -rajapinta, jolla voi muodostaa ja suorittaa SQL lauseita Java koodista. Tällä tehtyä kaksi tärkeintä tietomallia SyncWaressa ovat malliluokat (engl. model) ja taltiot (engl. period). (Helenius, 2010, s. 21, 24)

Malliluokkaa käytetään Javan luokkien tallentamiseen ja lukemiseen tietokannasta. Itse luokan nimi toimii tauluna ja luokan muuttujat, jotka ovat annotaatiolla (engl. annotation) sellaiseksi määritetty, toimivat sarakkeina. Vastaavalla nimellä tehtyä malliluokan kokoelman (engl. model collection) luokkaa käytetään tietojen haussa. Annotaatioilla määritellyt muuttujat toimivat hakuvaihtoehtoina. Vastaavalla nimellä tehty palvelinluokka puolestaan sisältää tietokannan hakukomennot, joissa on toteutettu Javan muuttujien sijoittaminen niihin ja hakutulosten poiminta niistä. Malliluokalla on aina yksilöivä tunnus *oid*, jolla siihen viitataan sen sisältyessä toiseen malliluokkaan. (Helenius, 2010, s. 21-22)

Taltiot ovat suuria mittausmääriä varten, joille on toteutettu tietokantaan oma säilytys- ja hakulogiikkansa. Taltioiden tauluihin kuuluu aikaleiman sarake ja jokainen siihen tallentuva arvo muodostaa oman sarakkeensa. Taltiot jakaantuvat kevyisiin *LightPeriod* ja raskaisiin *HeavyPeriod*. Raskaat taltiot ovat jatkuvaa esim. automaatiosta minuutin välein tulevaa dataa varten, joita voidaan koostaa korkeammille resoluutiotasoille, kuten tunti ja päivä, summana, pienimpänä tai suurimpana arvona tai keskiarvona. Kevyet taltiot ovat satunnaisin aikavälein tulevalle ja myös tekstimuotoiselle datalle. Taltiolle on määritelty sille tulevan datan lähteet, jotka se tallentaa kevyiksi muuttujiksi *LightVariable* tai raskaiksi muuttujiksi *HeavyVariable* malliluokkaan. Muuttujalla on tieto, missä kohtaa sen arvoja säilytetään taltioiden tietokantatauluissa. Lisäksi on olemassa *VirtualVariable*-ohjelmaluokka, jossa voi käyttää näitä kevyitä ja raskaita muuttujia laskentakaavoissa matemaattisilla funktioilla. Kokonaisuus toimii tavallaan relaatiotietokantaan toteutettuna aikasarjatietokannan virityksenä. (Helenius, 2010, s. 23)

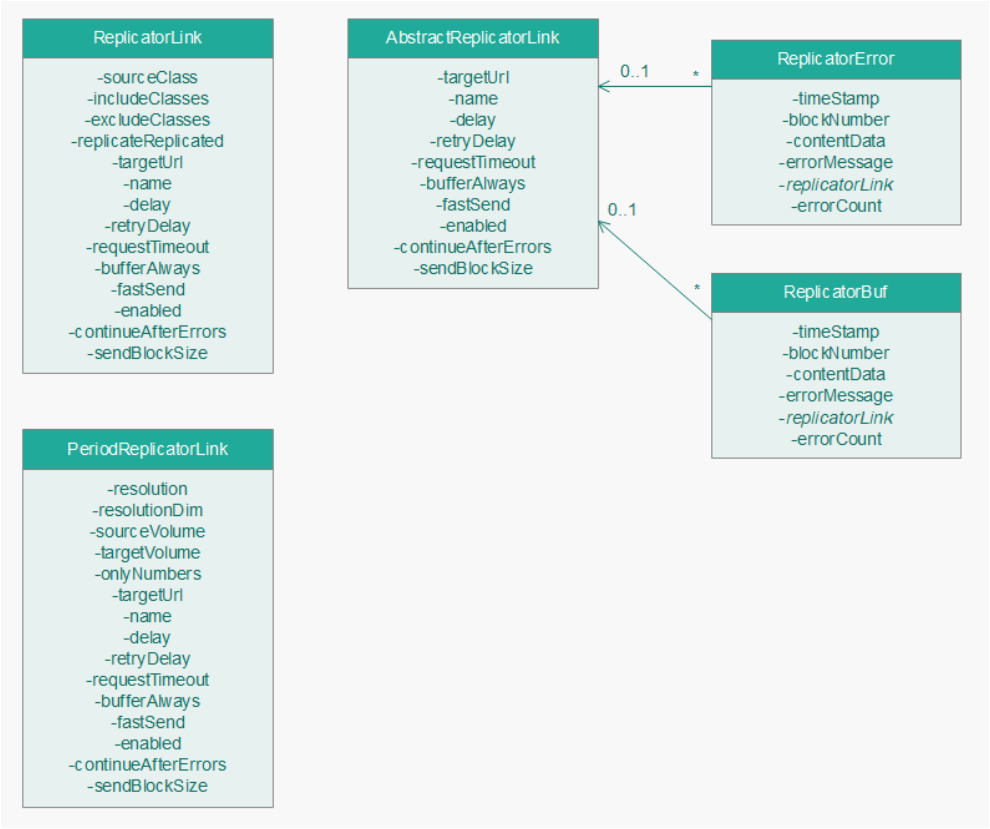
Kaikista SyncWaren tietokantaan muodostamista transaktioista tehdään *TxnInfo*-objekti, joita säilytetään *TxnNaming*-luokassa muistissa. *TxnInfo* säilyy siihen asti, kunnes transaktio on kokonaan suoritettu loppuun. (Pukki, 2020; Syncron Tech Oy, 2020)

SyncWaressa on raportointitoteutus, joka pohjautuu pääasiassa kuvaajien esittämiseen tehtyyn AmChartin JavaScript-kirjastoon. Raportissa määritetään sen muuttujat, joissa voi käyttää kevyitä, raskaita ja virtuaalimuuttujia sekä suorittaa niillä laskuja, ja esitetäänkö niitä kuvaajassa. Valittavissa on myös esitettävän kuvaajan tyyppi ja sen asetukset sekä raportin muuttujien hakuaikaväli erilaisissa muodoissa. (Pukki, 2020; Syncron Tech Oy, 2020)

SyncWaressa on oma klusterointimekanismi. Sille määritellään yrityksen asiakkaan ympäristössä käytössä olevat sovelluspalvelut. Ajossa voi olla kaksi instanssia, joista toinen toimii ensisijaisena toisen ollessa valmiustilassa. Jokainen klusterin palvelu kuuluu johonkin ryhmään. Ryhmälle määritetään, millä instansseilla ne ovat käytössä, ja vähintään yksi ensisijainen instanssi. Klusteroinnista vastaa Tomcatin käynnistyksen yhteydessä ladattava *ClusterManagerImpl*-Java-luokka. *ClusterManagerImpl* pitää yllä instanssien tilatietoja. Mikäli ensisijainen instanssi jostain syystä katoaa, aktivoituu valmiustilassa oleva ja ryhmän palveluiden toiminta siirtyy sille. Kun ensisijainen instanssi palaa takaisin, se ilmoittaa siitä aktiiviselle instanssille ja ensisijainen instanssi aktivoituu ja toissijainen siirtyy takaisin valmiustilaan. (Nenonen, 2010, s. 15-16)

Tietokannan kahdentamisesta huolehtii replikointipalvelu *ReplicatorService*, joka lähettää ja vastaanottaa kahdennussanomia. Palvelulla on käytössään malliluokat:

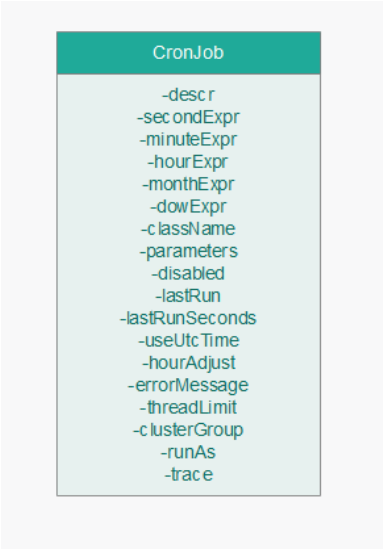
ReplicatorLink, PeriodRelpicatorLink, AbstractReplicatorLink, ReplicatorBuf ja RepicatorError, joiden luokkakaavio on esitetty kuvassa 9.



Kuva 9. *Replikaattorin luokkakaavio*

ReplicatorLink ja *PeriodRelpicatorLink* -objektelle on määritelty kahdennettavat malliluokat antamalla luokan polku (*sourceClass)* ja taltiot antamalla taltion aikamääre (*resolution*) ja muuttuja (*sourceVolume*). *ReplicatorBuf*-tauluun annetaan kahdennussanomat jonoon, josta ne sitten onnistuneen lähetyksen jälkeen poistetaan. *ReplicatorError*-tauluun lisätään tieto lähetyksessä tapahtuneesta virheestä. Viitteenä käytetään abstraktia malliluokkaa *AbstractReplicatorLink*, joka yhdistää *ReplicatorLinki*-objektin ja *PeriodReplicatorLink*-objektin mutta ei ole varsinainen taulu. (Nenonen, 2010, s. 17; Pukki, 2020; Syncron Tech Oy, 2020)

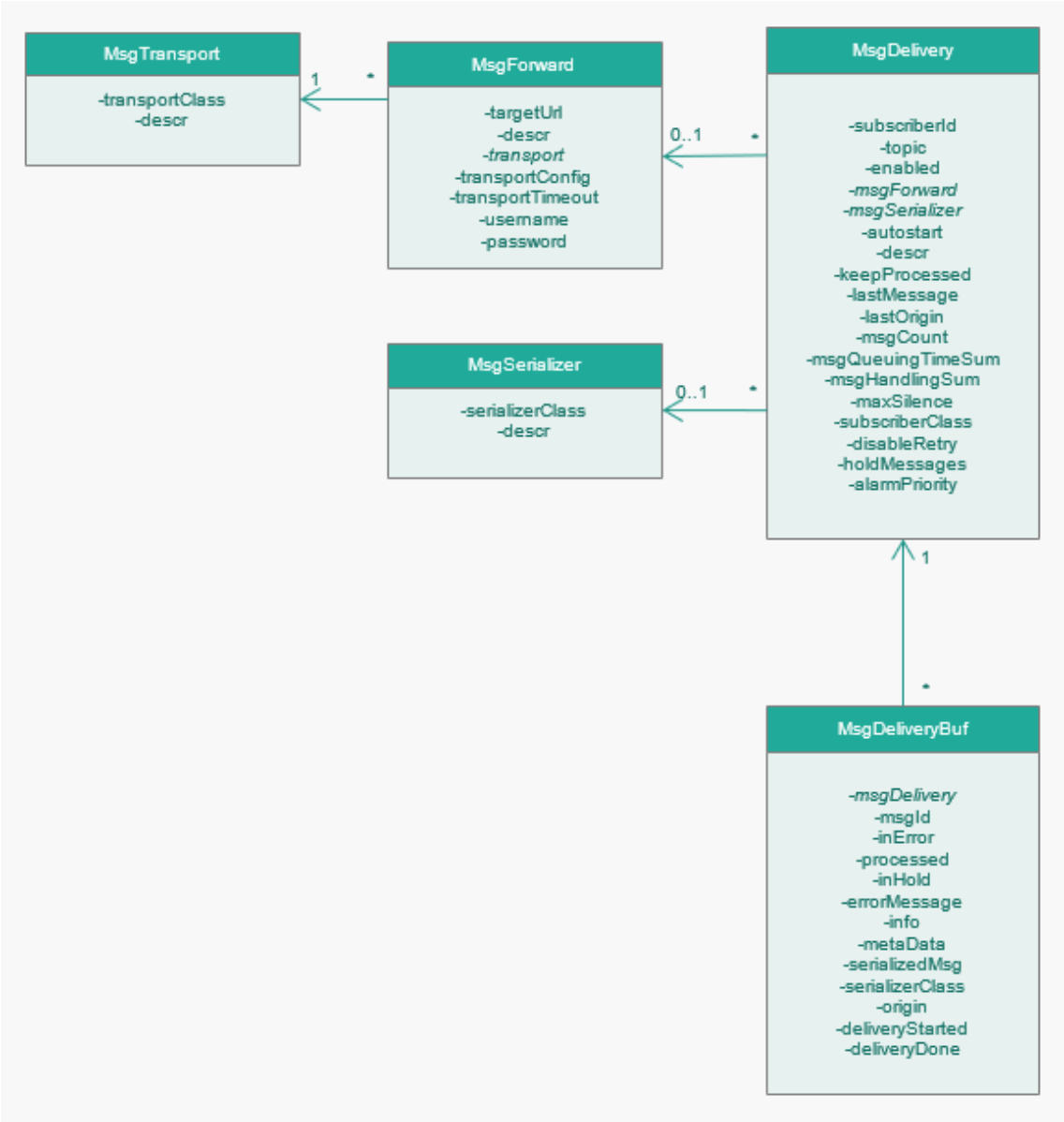
Säännöllisin väliajoin tehtäviä suorituksia varten SyncWaressa on aikalistapalvelu Cron. Tämä mahdollistaa Java-luokan käynnistyksen tiettyinä ajanhetkinä. Palvelulla on käytössään *CronJob*-malliluokka, joka on esitetty kuvassa 10.



Kuva 10. *Aikalistan luokkakaavio*

CronJob-objektille on annettu suoritettavan tehtävän nimi (*descr*), ajettava Java-luokka (*class-name*), mahdolliset luokan ajossa käytettävät argumentit (*parameters*) ja aikamääreet siihen, milloin ajo tehdään (*secondExps*, *minuteExpr*, *hourExps*, *monthExpr* ja *dowExpr*). (Pukki, 2020; Syncron Tech Oy, 2020)

Sanomien lähetys ja vastaanotto koostuvat *SerializerEngine* ja *DeliveryEngine* -palveluista. Palveluilla on käytössään malliluokat *MsgTransport, MsgSerializer, MsgForward, MsgDelivery* ja *MsgDeliveryBuff*, joiden luokkakaavio on esitetty kuvassa 11. (Pukki, 2020; Syncron Tech Oy, 2020)

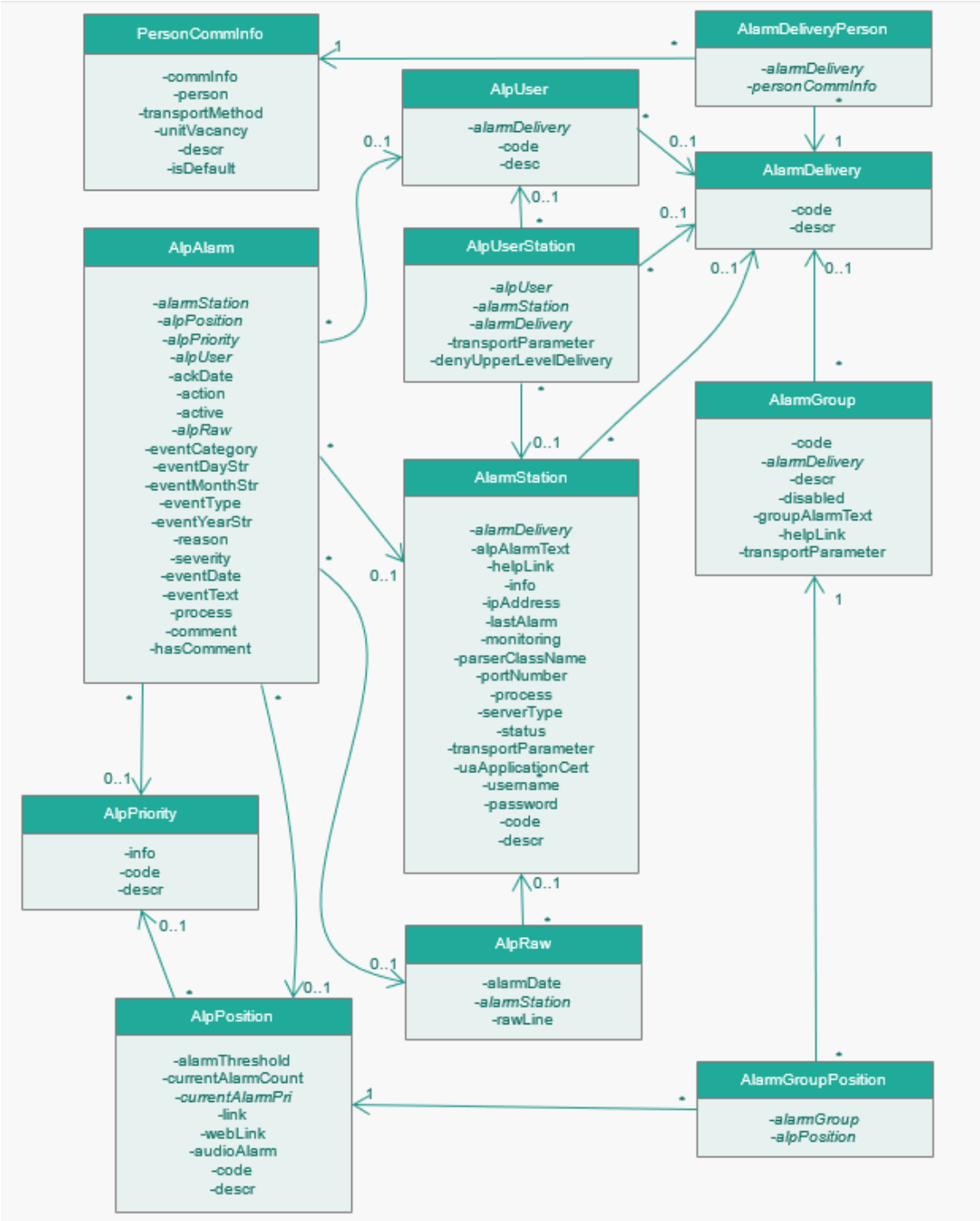


Kuva 11. *Viestinnän luokkakaavio*

MsgTransport-objektille on määritetty kuljetustaso (*transportClass*). *MsgSerializer*-objekti on sovitin kolmannen osapuolen sanomarakenteiden ja Java-objektien välillä. *MsgForward*-objektille on määritelty sanoman lähetyskohde (*targetUrl*). *MsgDelivery*-objektille on määritelty sanoman luokka (*subsciberClass*), mitä *MsgForward*-objektia se käyttää ja säilytetäänkö sitä lähetyspuskurissa (*MsgDeliveryBuffer*) käsittelyn jälkeen ja onko sillä maksimihiljaisuusaikaa (*maxSilence)*. Maksimihiljaisuusajalla tarkoitetaan aikaa, joka saa kulua ennen uuden sanoman lähetystä ennen kuin oletetaan sanoman lähetyksessä olevan jotain vialla. (Pukki, 2020; Syncron Tech Oy, 2020)

Hälytykset koostuvat *AlpAlarmServer* ja *AlpPositionServer* -palveluista. Palveluilla on käytössään malliluokat *AlpAlarm, AlpUser, AlarmStation, AlpUserSation, AlpPosition*,

AlarmGoup ja *AlarmGroupPosition*, joiden luokkakaavio on esitetty kuvassa 12. (Pukki, 2020; Suutari, 2020; Syncron Tech Oy, 2020)



Kuva 12. *Hälytyksen luokkakaavio*

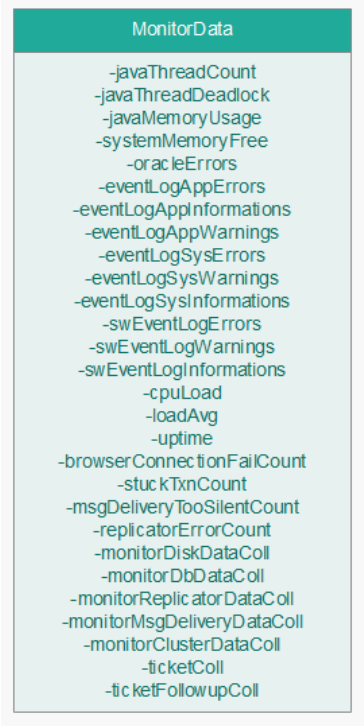
AlpAlarm-taulu sisältää luodut hälytykset. *AlpAlarm*-objektilla on hälytysaika (*ackDate*), onko se aktiivinen (*active*), hälytystaso (*severity*), hälytysasema (*alarmStation*), hälytyspositio (*alpPosition*) ja hälytyskäyttäjä (*alpUser*). *AlarmStation*-taulu sisältää hälytysasemat. *AlarmStation*-objektille voi määrittää jakelulistan (*alarmDelivery*).

AlpUser-taulu sisältää hälytyskäyttäjät. *AlpUser*-objektille voi määrittää jakelulistan (*alarmDelivery*). *AlpUserStation*-taulu sisältää hälytyskäyttäjien ja hälytysasemien yhdistelmät. *AlpUserStation*-objektille voi määrittää jakelulistan (*alarmDelivery*). *AlarmPosition*-taulu sisältää hälytyspositiot. *AlarmGroup*-taulu sisältää hälytysryhmät. *AlarmGroup*-objektille voi määrittää jakelulistan (*alarmDelivery*). *AlarmGroupPosition*-taulu yhdistää hälytyspositiot hälytysryhmiin. *AlarmDelivery*-taulu sisältää jakelulistat. Kun hälytys luodaan, siitä lähetetään ilmoitus kaikille eri jakelulistojen henkilöille. Hälytyksiä varten on myös tehty oma erillinen sivu hälytystietojen hakemiseksi, jonne pääsee käyttöliittymästä. (Rikkonen, 2020; Syncron Tech Oy, 2020)

3.2 SyncWaren järjestelmämonitorointi

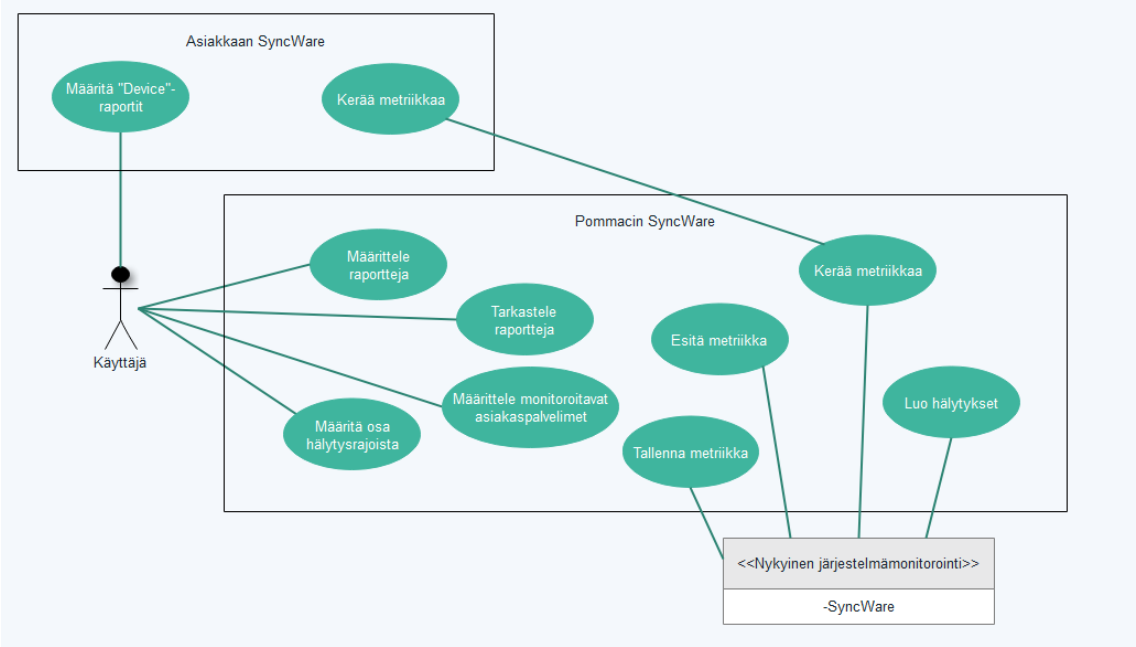
3.2.1 Järjestelmämonitorointikokonaisuus

SyncWaren järjestelmämonitorointi jakaantuu yrityksen omassa palvelimessa (”Pommac”) toimivaan SyncWareen ja asiakkaiden palvelimella olevan SyncWaren keräämään ja säilyttämään dataan Device-raporteissa ja MonitorData-malliluokassa. Device-raporteiksi kutsutaan raportteja, joiden nimessä on oltava DEVICE. MonitorData on esitetty kuvassa 13. (Suutari, 2020; Syncron Tech Oy, 2020)



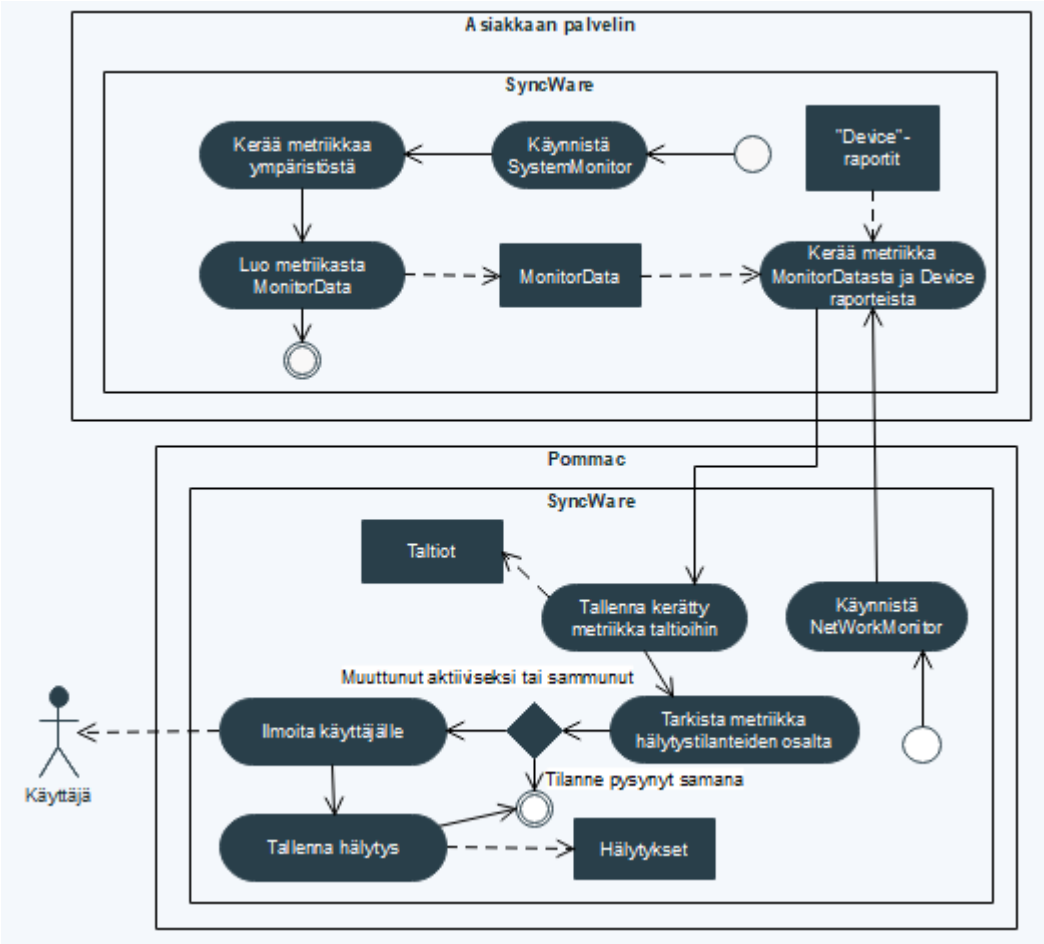
Kuva 13. *MonitorData*-luokka

Pommac hakee asiakkaiden SyncWareilta monitoroitavan datan ja huolehtii sen esittämisestä ja hälytyksistä. Kuvassa 14 on esitetty järjestelmämonitoroinnin käyttötapauskaavio, jossa on esitettynä, miten monitorointia pystyy säätämään ja käyttämään ilman SyncWaren koodiin tehtäviä muutoksia. Käyttäjä voi luoda asiakkaan ympäristössä Device-raportteja, joiden data kerätään. Hän voi luoda kerätystä datasta raportteja ja tarkastella niitä. Lisäksi hän voi märittää tarkkailtavat asiakaspalvelimet ja osan kerätyn datan käsittelyyn liittyvistä hälytysrajoista. Kuvassa näkyy myös se, mitä järjestelmä monitorointi tekee. Pommacin SyncWare hakee asiakkaan SyncWaren keräämän datan. Lisäksi se käsittelee kerätyn datan ja luo tarvittaessa hälytyksiä sekä tallettaa kerätyn datan. Kerättyä dataa ja hälytyksiä se esittää sille luoduissa raporteissa. (Suutari, 2020; Syncron Tech Oy, 2020)



Kuva 14. *SyncWaren järjestelmämonitoroinnin käyttötapauskaavio*

Kuvassa 15 on puolestaan tämä järjestelmämonitorointikokonaisuus esitettynä edelleen käytössä olevilta osilta aktiviteettikaaviona. Siinä näkyy datan keräämisen, tallentamisen ja käsittelyn osuudet.



Kuva 15. *SyncWaren järjestelmämonitorointi kokonaisuus*

3.2.2 Asiakkaan SyncWare

Asiakkaan SyncWaressa *MonitorData*-objektin sisältö kootaan, kun aikalistalla suoritetaan tehtävä *SystemMonitor*, joka ajetaan noin 2.5–10 minuutin aikavälein riippuen asiakkaasta. (S1) Ensiksi haetaan hyödyntäen Javan omia luokkia ympäristön käyttöjärjestelmä, tiedot Javan säikeistä ja tiedot Javan käyttämästä muistista. Javan käyttämästä muistista käydään läpi sen historia kymmenen minuutin ajalta ja otetaan pienin muistin käyttö, jonka näin ollen oletetaan olevan Javalla pysyvässä käytössä. Javan säikeiden tiedoista otetaan säikeiden lukumäärä. (Suutari, 2020; Syncron Tech Oy, 2020)

(S2) Käydään läpi SyncWaren transaktiot *TxnNaming*-luokasta. Lasketaan yli kaksi minuuttia aktiivisina olleiden transaktioiden lukumäärä, koska ne ovat täten tulkittu jumiutuneiksi. (S3) Palvelimen tilanteesta haetaan vapaa muisti, kovalevyjen käytöt, prosessorin kuormitus ja käynnissä oloaika käyttöjärjestelmästä riippuvilla menetelmillä. Käynnissä oloaikaa ei selvitetä Windowsin tapauksessa. UNIX:n tapauksessa tiedot

käydään hakemassa tietostoista, joissa käyttöjärjestelmä niitä säilyttää. Windowsin tapauksessa käytettäisiin Performance Counters -rajapintaa tietojen hakemiseen mutta toteutus jäi kesken. (Suutari, 2020; Syncron Tech Oy, 2020)

(S4) Tietokannan tila selvitetään tietokantahaulla käyttäen *DbStatus*-malliluokkaa, jolle on palvelinluokkaan määritelty kaksi tilanteen selvittämiseen liittyvää tietokantahakua. Sieltä haetaan jokaisen tietokantaosion käyttämä kovalevyn tila. SQL Serverin ja Oraclen tapauksessa haetaan myös käytetty tila, jolla myös siitä lasketaan käytetty tila prosentteina. Lisäksi Oraclen tapauksessa haetaan täyttöaste, vapaa tila ja vapaa tilan seuraavan tilan allokoinnin jälkeen. (Suutari, 2020; Syncron Tech Oy, 2020)

(S5) Mikäli replikointipalvelu on käytössä, saadaan siltä replikointilinkkien tilanteet. Näiltä poimitaan talteen linkin nimi, linkin sanomajonon koko ja onko linkki alhaalla. (S6) Mikäli sanomien lähetys- ja vastaanottopalvelu on käytössä, haetaan käytössä olevat *MsgDelivery*-malliluokat ja poimitaan niiden tilaajien tunnukset, ovatko ne ylittäneen sallitun maksimihiljaisuusajan ja käsittelemättömien sanomien määrä *MsgDeliveryBuff*-talusta. Samalla lasketaan maksimihiljaisuusajan ylittäneiden sanomakonfiguraatioiden lukumäärä. (Suutari, 2020; Syncron Tech Oy, 2020)

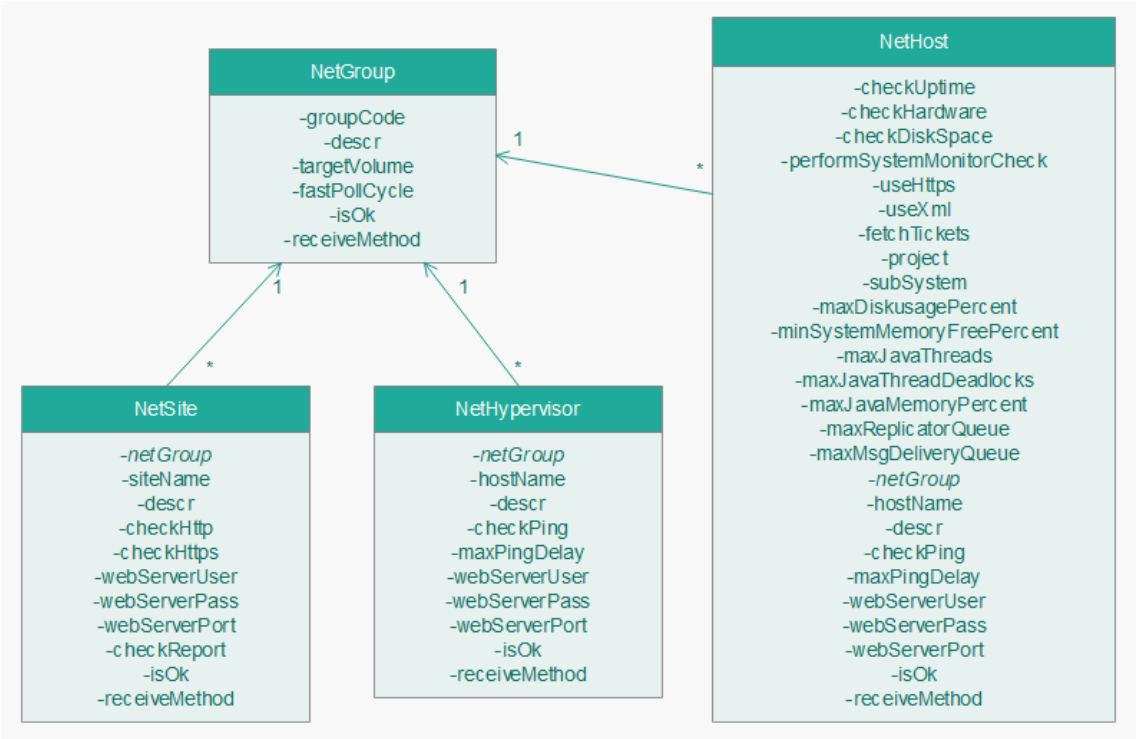
(S7) Haetaan lokiin kirjoitettujen virhe-, varoitus- ja infotekstien lukumäärät viimeisen viiden minuutin aikana SyncWarelta. Erikseen haetaan järjestelmän ja sovellusten lokien osalta käyttöjärjestelmältä Windowsin tapauksessa ja tietokannalta Oraclen tapauksessa, josta otetaan vain virheiden määrä. SyncWaren lokikirjoitusten määrät saadaan käymällä läpi sen tekemät error.log-kirjoitukset. (S8) Windowsilta käytäisiin järjestelmän tarjoamaa ohjelmointirajapintaa järjestelmä- ja palvelulokin läpikäyntiin mutta tämä on estetty, koska se oli liian raskasta ja näin ollen lukumääriksi saadaan vain nollia. Oraclen lokikirjoitusten määrä saadaan käymällä läpi sen lokikansioon luodut tiedostot. (Suutari, 2020; Syncron Tech Oy, 2020)

(S9) *ClusterManagerImpl*-luokalta haetaan käytössä olevat palvelut. Niiltä poimitaan ryhmän nimi, onko se aktiivinen ja onko se varapalvelimelle määritelty. (S5) Haetaan vielä erikseen *ReplicatorError*-malliluokalta sillä olevien virheiden määrä. Lopuksi vielä, mikäli aikalistan parametreihin on niin määritetty, haetaan myös automaattisesti luodut tiketit, joita säilytetään tietokannassa omassa taulussaan. Tiketit kertyvät automaattisesti virheistä, joita ei ole erikseen kooditasolla käsitelty. Mikäli käyttäjän näytölle tulee virheilmoitus, tehtävään kirjautuu virheen aiheuttanut tilannekuva. MonitorData saa aikaleimakseen sen muodostamisen hetken kellonajan. (Pukki, 2020; Suutari, 2020; Syncron Tech Oy, 2020)

(S10) Device-raportit ovat taltioiden arvojen tarkkailua varten. Ensisijaisesti halutaan tietää, onko johonkin muuttujaan saatava arvo mahdollisesti jumissa sillä tavalla, että sieltä ei ole pitkään aikaan saatu mitään arvoa tai on saatu vain samaa arvoa. Tällaisen raportin on oltava tyypiltään ”Teksti, kään.”, jossa raportin viimeisen aikaleiman tulos esitetään yhteen menoon kirjoitettuna tekstinä. Lisäksi muuttujien selitteen on alettava tekstillä DEVICE ja muuttujan laskukaava on muodostettava siten että tulokseksi tarkkailtavan arvon vikatilanteessa tulee tyhjä (null). (Suutari, 2020; Syncron Tech Oy, 2020)

3.2.3 Pommac

Pommacilla on tarkkailtavien kohteiden määrittelyyn malliluokat *NetSite*, *NetHost* ja *NetHyperVisor*, jotka kuuluvat johonkin malliluokkaan *NetGroup*. Näiden luokkakaavio on esitetty kuvassa 16.



Kuva 16. *Datan keräyksen luokkakaavio*

NetGroup-objektille on määritelty ryhmän tunnus (*groupCode*). *NetHost*-objektille on määritelty paikan nimi (*hostName*), tarkistetaanko ping (*checkPing*), maksimi ping-viive (*maxPingDelay*), käytetäänkö SystemMonitoria (*performSystemMonitorCheck*), yhteyden muodostamiseen käytettävä käyttäjätunnus (*webServerUser*) ja salasana (*webServerPass*), käytettävä portti (*webServerPort*). Erikseen on määrittelyt sille, tarkistetaanko päällä oloaika (*checkUptime*), laitteisto (*checkHardware*) ja levytila

(*checkDiskSpace*). Lisäksi on mahdollisuus määrittää hälytysrajat levytilankäytölle (*maxDiskusagePercent*), järjestelmän vapaalle muistille (*maxSystemMemoryFreePercent*), Javan säikeiden määrälle (*maxJavaThreadDeadlock*), Javan muistin käytölle (*maxJavaMemoryPercent*), replikointijonoille (*maxReplicatorQueu*) ja sanomajonoille (*maxMsgDeliveryQueue*). *NetSitelle*-objektille on määritelty isännän nimi (*siteName*), tarkistetaanko HTTP-yhteys (*checkHttp*), tarkistetaanko HTTPS-yhteys (*checkHttps*), yhteyden muodostamiseen käytettävä käyttäjätunnus (*webServerUser*) ja salasana (*webServerPass*) ja tarkastettavien Device-raporttien polku (*checkReport*). *NetHypervisor*-objektille on määritelty kohteen nimi (*hostName*), tarkistetaanko ping (*checkPing*), maksimi ping-viive (*maxPingDelay*) sekä yhteyden muodostamiseen käytettävä käyttäjätunnus (*webServerUser*) ja salasana (*webServerPass*). Hälytyksissä käytetään ainoastaan hälytysryhmien jakelulistoja (Kuva 12). (Rikkonen, 2020; Suutari, 2020; Syncron Tech Oy, 2020)

Aikalistalla on tehtävä *NetWorkMonitor*. Kun se suoritetaan, haetaan ensin kaikki *NetGoup*-objekteille kuuluvat *NetHost*-objektit sekä *NetHyperVisor*-objektit ja suoritetaan ping niihin, joille se on määritelty tarkistettavaksi. Eli lähetetään Internet Control Message Protocol -datapaketteja (lyh. ICMP) verkon kautta ja katsotaan, saadaanko niihin vastauspaketti takaisin. Lokiin kirjoitetaan tiedot epäonnistumisista. Sitten haetaan myös NetGrouppeihin kuuluvat NetSitet ja suoritetaan NetHyperVisoreille, NetSiteille ja NetHosteille tarkistukset. Tarkistusten yhteydessä hälytystilanteissa aktivoidaan SyncWaren *AlpAlarm*-hälytys ja muuten se deaktivoidaan. Lisäksi samalla kerätään tulokset tallennetavaksi taltioihin tarkistusten päätteeksi muodossa

ryhmätunnus.HOST, SITE tai HYPERVISOR.isännän tai paikan nimi.tarkistuskohde.

NetHost ja *NetHyperVisor* -malliluokilla hälytystilanteita ping-suorituksen tuloksista ovat tarkistuksen yhteydessä tyhjä (null) tai yli kolmenkymmenen prosentin pakettien menetys sekä maksimiviiveajan ylittänyt ping. Maksimiviiveaika on oletuksena 150 millisekuntia. (Suutari, 2020; Syncron Tech Oy, 2020)

NetHyperVisor-objektin tarkistuksessa käydään läpi sille märitellyt VMware-virtuaalijärjestelmät. Tarkistuksen kaatuminen virhetilanteeseen tulkitaan hälytystilanteeksi. Järjestelmiin muodostetaan yhteys ja kirjaudutaan sisään. Tämän jälkeen muodostetaan lista siitä, mitä tietoja järjestelmältä pyydetään ja saadaan järjestelmään määritetyt virtuaalikoneet. Näiden koneiden tilat käydään läpi, mikä voi olla

vihreä, keltainen tai punainen. Punainen eli kriittinen ja keltainen eli varoitus tila tulkitaan hälytystilanteiksi. (Suutari, 2020; Syncron Tech Oy, 2020)

NetSite-objektin tarkistuksessa, jos HTTP-yhteys on määritetty tarkistettavaksi, muodostetaan sillä yhteys asiakkaan palvelimeen. Jos HTTP-yhteys on määritetty tarkistettavaksi tarkistettavaksi, muodostetaan sillä yhteys asiakkaan palvelimeen. Hälytystilanteita ovat ok-statuksesta 200 poikkeavat statukset. Tämän jälkeen muodostetaan yhteys objektille määritettyihin raportteihin. Yhteyden muodostamisen epäonnistuminen tulkitaan hälytystilanteeksi. Jos yhteyden muodostus onnistui, luetaan raportin DEVICE-alkuisten muuttujien arvot. Tyhjä (null) arvo tulkitaan hälytystilanteeksi. (Suutari, 2020; Syncron Tech Oy, 2020)

NetHost-objektin läpikäynnissä, ensiksi suoritetaan Perl-ohjelmointikielellä toteutetuilla komentosarjoilla tarkistukset asiakkaan palvelimessa objektin määritysten mukaan. Jos portti on annettu, tarkistetaan komentosarjalla portin tilanne. Hälytystilanne on ok-statuksesta 200 poikkeava status. Jos ”tarkista käynnistäoloaika” on päällä, tarkistetaan komentosarjalla käynnissäoloaika. Hälytystilanteita ovat äskettäin käynnistynyt järjestelmä, jolla on yli viisi prosessorin ydintä käytössä. Jos ”tarkista levytila” on päällä, tarkistetaan komentosarjalla kovalevyjen tilanteet. Hälytystilanteita ovat yli 95 % täynnä oleva kovalevy ja epäonnistunut levyjen tilanteiden selvittäminen. Jos ”tarkista laitteisto” on päällä, tarkistetaan komentosarjalla laitteiston lämpötila. Hälytystilanne on yli 35℃ lämpötila. Perl-komentosarjojen tarkistukset ovat kuitenkin jäänne vanhasta toteutuksesta. SystemMonitor-objektien sisällön tarkkaileminen on nykyinen järjestelmämonitorointimenetelmä. Jos ”käytä SystemMonitoria” on päällä, haetaan asiakkaan SyncWaresta sen sisältö. Sisältö haetaan pääasiassa suoraan verkon kautta mutta on yksittäinen poikkeustapaus, jossa tiedot saadaan sähköpostiin ja haetaan sieltä. Yhteyden muodostamisen epäonnistuminen ja SystemMonitor-objektin puuttuminen tulkitaan hälytys tilanteeksi. SytemMonitor-objektin sisällössä hälytystilanteiksi tulkitaan:

- Järjestelmän käynnissä oloaika on alle kolmekymmentä minuuttia
- Järjestelmän vapaa muisti alle *NetHost*-objektille määritetyn rajan, joka on oletuksena 10 %
- Javan säikeiden määrä yli *NetHost*- objektille määritetyn rajan, joka on oletuksena 200
- javaThreadDeadlock yli 0 (Arvo on kuitenkin aina 0)

- Javan muistin käyttö yli *NetHost*- objektille määritetyn rajan, joka on oletuksena 80 %
- CPU:n jonossa olevien prosessien määrä on viiden minuutin ajan yli 5
- Jumiutunut transaktio
- Maksimi hiljaisuusajan ylittäneiden sanomien määrä yli nolla
- Replikointivirheiden määrä yli nolla
- Kovalevyn käyttö yli *NetHost*- objektille määritetyn rajan, joka on oletuksena 95 %
- Tietokanta, joka on merkattu olevan täynnä (Arvo on kuitenkin aina epätosi)
- Replikointijono, joka on ylittänyt *NetHost*- objektille määritetyn rajan, joka on oletuksena sata
- Sanomajono, joka on ylittänyt *NetHost*- objektille määritetyn rajan, joka on oletuksena sata
- Maksimi hiljaisuusajan ylittänyt sanoma
- Klusterisolmu, joka ei ole aktiivinen ja ei ole varasolmu tai solmu, joka on aktiivinen ja on varasolmu

SystemMonitor-objektilta saadut tiedot tallennetaan erikseen taltioihin samalla nimeämistyylillä. (Suutari, 2020; Syncron Tech Oy, 2020)

Pommacissa olevan SyncWaren pääsivulla on nähtävissä raportti, missä näytetään aktiiviset hälytykset ja viimeaikaiset hälytykset ja myös muutama viivadiagrammi yleisistä tiedoista mutta aktiivisten hälytysten raporttitaulu on oikeastaan ainoa mistä välitetään. Yksittäisen asiakkaan datasta on myös erillinen raporttisivu heiltä kerättyyn dataan kuukausittaiseen raportointiin. Lisäksi hälytyksillä on oma erillinen sivu, josta voidaan hakea tietyn järjestelmän kaikki hälytykset joltain ajalta ja näin laskea niistä tiedoista, kuinka kauan jokin tilanne on ollut päällä. Tulosta verrataan SLA-sopimukseen, jossa on määritelty, kuinka paljon suhteellisesti järjestelmä on pystyttävä pitämään toimintakunnossa esimerkiksi yhdeksänkymmentäviisi prosenttia ajasta. Muilla raporttisivuilta löytyy vanhoja raportteja, joissa esitetään kerättävää dataa viivadiagrammeina. (Rikkonen, 2020; Syncron Tech Oy, 2020)

Automaattitiketit tallentuvat Pommac-palvelimella ticket-malliluokalle ja uusista luoduista tiketeistä lähetetään Jiran rajapinnalle sanoma, joka kopioi tiketin tiedot Jira-tehtävälle. Myös yksittäisitä hälytyksistä voi muodostaa manuaalisesti Jira-tehtävän. Näin asiakas sekä yrityksen asiakasvastaava näkevät tätä kautta onko sovelluksessa jotain hallitsemattomia ongelmia tai ohjelmointi virheitä. Automaattisesti luotu Jira-tehtävä antaa usein kehittäjälle selkeän kuvan missä ongelma on ja kuinka sen voi korjata. Muille voi ilmoittaa tehtävän kautta, milloin sen tutkinta on otettu työn alle ja milloin se on saatu korjatuksi. (Pukki, 2020; Syncron Tech Oy, 2020)

4. SUUNNITELMA

Luvussa käydään läpi suunnitelma tavoitteiden toteuttamiseen. Ensiksi käydään läpi mahdolliset järjestelmämonitorointityökalut ja uusien järjestelmämonitorointityökalujen valinta. Sitten miten tavoitteisiin päästään näillä työkaluilla ja millaisen kokonaisuuden ne tulevat muodostamaan.

4.1 Järjestelmämonitorointityökalujen valinta

Yrityksen alkuperäisessä ideassa Collectd, myöhemmin Telegraf, keräisi datan, InfluxDB toimisi säilytys paikkana ja Grafana datan esitystyökaluna ja sitä käytettäisiin myös hälytyksiin. Käydään läpi kuitenkin kaikki tutkitut osat ennen lopullisen kokonaisuuden muodostamista, jotta nähdään, miltä osin tätä ideaa kannattaisi mahdollisesti muuttaa. Valinnasta pyritään tekemään mahdollisimman yhtenäinen ja siinä noudatetaan yrityksen asettamia vaatimuksia ja mielipidettä.

Prometheus vastaisi vetopainotteiselta tyyliltään enemmän alkuperäistä SyncWarea mutta jättäisi suoralla kirjastojen käytöllä datan keräämisen edelleen kokonaan SyncWarelle ja SyncWaren kuormitusta haluttaisiin nimenomaan vähentää. InfluxDB:llä on myös helpompi hakukieli, koska se muistuttaa SQL:ää. Lisäksi se on suosituin tietokanta DB-Enginesin listalla. Prometheuksen omissa vertailudokumenteissa sitä kuitenkin pidetään parempana metriikan kerääjänä ja kyselykieltään monipuolisempana kuin TICK mutta huonompana lokien käsittelyssä (Prometheus Authors 2020o). Ei löydy syytä, miksi yrityksen ideasta poikettaisiin.

ManageEnginen Applications Managerissa olisi helposti lisättävissä eri monitoroitavat osat tunnettujen yleisten ohjelmien ja ympäristöjen kannalta. SyncWaren tiedot saisi sitten mukautetun monitoroinnin kautta. Ilmainen versio tästä ei kuitenkaan riitä ja näin ollen toteutus tulisi yritykselle liian kalliiksi. Lisäksi on epäselvää, miten tietokanta on siinä toteutettu tehokkaalle aikasarjadatan käsittelylle sillä dokumenteissa ei ollut muuta mainintaa käytettävän relaatiotietokannan määrittämisen lisäksi.

Telegraf on jo valmiiksi osa TICK:ä kun taas Collectd olisi erillisen ryhmän toteuttama osa. Collectd on kyllä ollut olemassa pidempään kuin Telegraf, joten siihen on ollut enemmän aikaa tehdä lisäosia. Esimerkiksi Oracle-tietokannan datan lisäosaa ei olla vielä Telegrafiin tehty. Nämä voidaan kuitenkin toteuttaa muilla keinolla. Lisäksi kaikki

toistaiseksi puuttuvat lisäosat varmaan tulee lopulta aktiivisen ja kasvavan yhteisön ansiosta.

Grafana on datan esittämisessä hyvä ja monipuolinen työkalu. Kuvaajien valintavaihtoehtoja on enemmän kuin Chronografilla ja se on yhteensopiva muidenkin tietokantojen kuin InfluxDB:n kanssa. Yrityksellä on myös yksi MySQL-tietokanta ja sen tietojen esittäminen onnistuu näin ollen myös. Siinä olisi kyllä myös hälytysten luontimahdollisuus ja ilmoitusten lähettäminen niistä mutta Kapacitor vaikuttaisi olevan tässä parempi. Kapacitorilla on oma komentosarjakielensä datan keräämiseen ja ehtojen päättämiseen siitä tehtävälle hälytykselle. Grafanan hälytysehdoille ei voi määrittää muuta kuin rajaehdon, milloin hälytys tehtäisiin. Helpointa Kapacitoria olisi käyttää Chronografin kautta.

Lopullinen kokonaisuus olisi siis koko TICK:n ja Grafanan yhdistelmä. SyncWare kerää edelleen sen datan, mitä ei voi muuta kautta saada ja siihen kehitetään systeemi sen saamiseksi InfluxDB:hen. Tämä tehdään mahdollisesti Telegrafin kautta. Telegraf toimii datan kerääjänä. InfluxDB toimii datalle aikasarjatietokanta. Grafanaa käytetään tämän datan esittämisen. Kapacitor toimii hälytystyökaluna ja sitä käytetään Chronografin kautta ja samalla Chronografia voidaan myös käyttää InfluxDB:n hallitsemiseen. Taulukkossa 1 vielä yhteenveto päätökseen vaikuttaneista työkalujen ominaisuuksista.

Taulukko 1: *Työkalujen ominaisuudet*

TICK	**Prometheus**	**Applications Manager**
- Omaa suosituimman aikasarjatietokannan DB-Engines Ranking of Time Series DBMS listalla - Ollut yrityksellä aikaisemminkin osittain käytössä - Kyselykieli muistuttaa SLQ-kieltä	- Tietokanta kysely hankalampaa mutta monipuolisempaa - Jättää datan keräämisen edelleen SyncWarelle normaalissa toteutuksessa	- Datan keräys helposti määritettävissä tunnetuiden ohjelmien ja ympäristöiden kannalta - Tuntematon tietokanta toteutus - Maksullinen
Collectd	**Grafana**	

- Sisältää monipuolisen valikoiman lisäosia	- Sisältää monipuolisen valikoiman lisäosia erilaisia datan esittämisen tyylejä varten - Yhteen sopiva monien tietokantojen kanssa	

4.2 Uusi järjestelmämonitorointi

Nyt kun käytettävät järjestelmämonitorointityökalut ovat valittu, tutkitaan, miltä osin Telegraf voi hoitaa SyncWaren datan keräysosuuden, ja pyritään samalla tavoitteisiin V1 ja V2. Tarkoitus olisi pyrkiä siihen, että saadaan kerättyä kaikki sama data kuin vanhassakin järjestelmässä. Virtuaalikoneista saatava tieto on nykyään merkityksetöntä ja Perl-komentokielen käyttö oli käytöstä poistunut datan keräysmenetelmä, joten tavoitteena ovat SystemMonitorin keräämät datat. Ainakin niiltä osin mitä ei ole ajan saatossa otettu pois käytöstä mutta katsotaan, jos poistettujakin osuuksia saataisiin toteutettua uudessa versiossa.

Tavoitteen V1 saavuttamiseksi käytetään Telegrafin lisäosia. Itse palvelimen tarkkailu, SyncWaren ominaisuus S3, voidaan korvata Windows-käyttöjärjestelmissä win_perf_counters-lisäosalla. Unix palvelimilla puolestaan käytettäisiin laitteen komponenteille tehtyjä omia lisäosia cpu, disk, mem, swap ja system. Koska SyncWare toimii Tomcat-ympäristössä, voidaan ohjelman oma kuormitus ja näin ollen SyncWaren ominaisuuden S1 hakea siltä. Tämä tapahtuu Apache Tomcat -lisäosalla, jossa haetaan Tomcatin tilasivulta siinä olevat tiedot ja näin saadaan SyncWaren käyttämä heap-muisti ja säikeet. Tietokannasta saataviin tietoihin, SyncWaren ominaisuus S4, olisi käytettävissä lisäosat PostgreSQL ja Microsoft SQL Server tietokantoja varten. Tästä kuitenkin puuttuu Oracle. Toinen mahdollinen vaihtoehto olisi Generic SQL -lisäosa, jolla voisi ajaa tietokantakomentoja kaikissa kolmessa tietokannassa mutta tämän työn aikana sitä ei ollut vielä liitetty Telegrafiin. Keskustelu lisäosasta ja sen tilanteesta on nähtävissä lähteen osoitteessa (InfluxData Inc., 2020n). Näin ollen annetaan SyncWaren toistaiseksi kerätä tämä data, kunnes kaikkien kolmen tietokantojen tietojen keräys saadaan siirrettyä itse Telegrafille. Järjestelmän lokien keräys, SyncWaren ominaisuus S8, oli Unixin osalta toteuttamatta ja Windowsin osalta poistettu käytöstä. Nämä voitaisiin mahdollisesti toteuttaa uudessa järjestelmässä Syslog-lisäosalla Unixin osalta ja vastaavalla Windowsin osalla taas Windowsin osalta. Tällöin saataisiin pelkkien eri

tyyppien määrien sijaan itse lokit nähtäville yhteen paikkaan ja määriä voitaisiin laskea haluttaessa Grafanassa erillisillä hauilla. Tätä Windows osaa ei kuitenkaan oltu vielä työn aikana liitetty Telegrafiin. Keskustelu lisäosasta ja sen tilanteesta on nähtävissä osoitteessa (InfluxData Inc., 2020o). Samoin kuin järjestelmän lokien kanssa SyncWaren lokien keräys, SyncWaren ominaisuus S7, on pelkästään lokikirjoitustyyppien määriin pohjautuvaa eikä sitä aina kerätä. Nämä määrät eivät oikein kerro mitään, joten tässä ainakin parempi olisi kerätä loki kokonaisuudessaan. Tähän voitaisiin käyttää Logparser-lisäosaa, jolla SyncWaren tekstitiedostoon kirjoitettava loki voidaan lukea. Ongelmana on kuitenkin se, että yleisen tietosuoja-asetuksen (engl. General Dara Protection Regulation, lyh. GDPR) vaatimus ei anna tehdä tätä kaikkien asiakkaiden osalta, joten tämäkin jäänee toistaiseksi toteuttamatta.

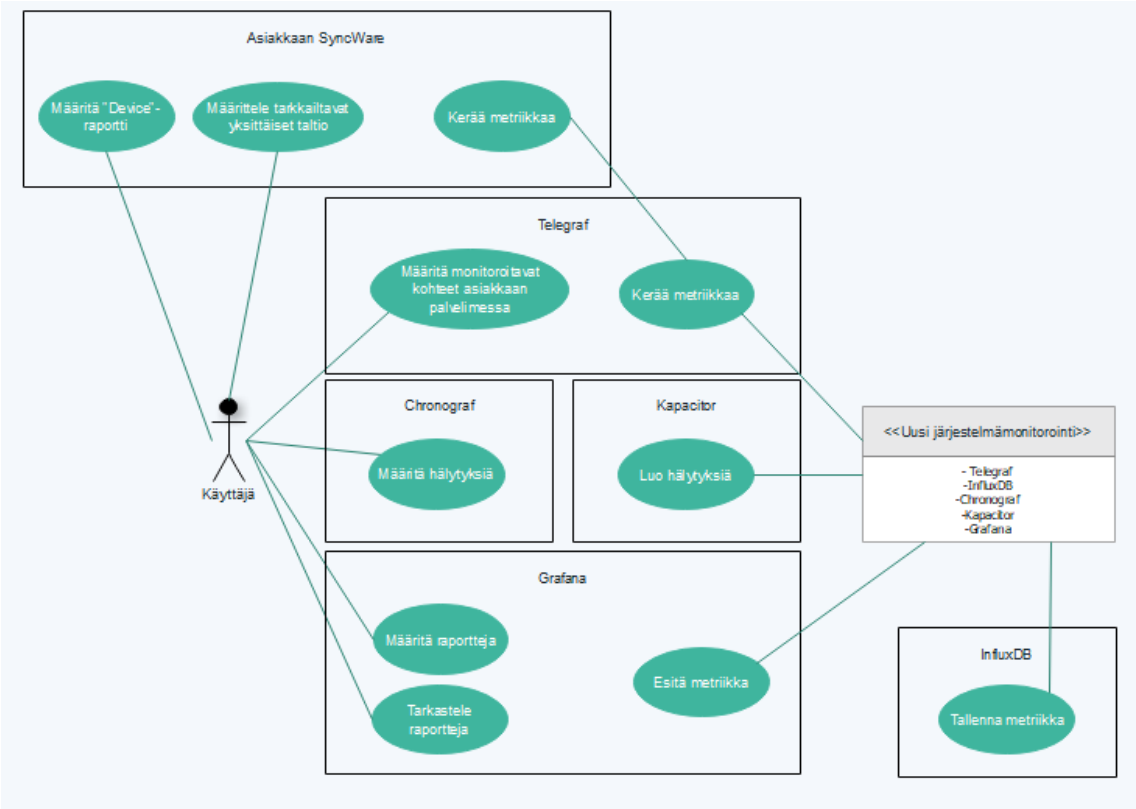
Loput datat jäisivät sitten tavoitteen V2 osalta SyncWaren kerättäväksi. Ensiksi tarvitaan SyncWaren ominaisuudet S5 ja S6 replikointi- ja sanomapalveluiden tilanteet, koska yleiset tiedot itse palveluista ovat nähtävissä vain SyncWarella. Toiseksi tarvitaan SyncWaren ominaisuudet S2 ja S9 transaktiot ja klustereiden tilat, koska niitä säilytetään ohjelman muistissa. Kolmanneksi tarvitaan tietokantojen tilanteet S4 ainakin toistaiseksi aikaisemmin mainituista syistä. Neljänneksi tarvitaan taltioiden tilanteet, eli SyncWaren ominaisuus S10. Device-raporteista ei haluta luopua, koska kaikkien parametrien uudelleen määrittäminen olisi liian iso työ, joten ne täytyy pystyä edelleen käymään läpi. Pyritään kuitenkin myös tekemään yksinkertaisempi ratkaisu, jossa tarkkailtavat taltiot voisi vain valita, eikä niitä varten täytyisi tehdä tietynlaisia kokonaisia raportteja. Datan siirtoon Telegrafille voidaan käyttää HTTP-yhteyttä. Vaihtoehtoina ovat HTTP-lisäosa, jolloin Telegraf itse käy pyytämässä dataa, tai HTTP Listener V2 -lisäosa, jolloin SyncWare lähettää datan Telegrafille. Valitaan HTTP Listener V2, jotta konfiguraatio on helpommin SyncWaren puolella säädettävissä ja toteutus voi vastata enemmän vanhaa toimintaa.

Tavoitetta V3 tarkkaillaan testauksen yhteydessä. Ensisijaisesti katsotaan, miten Grafanassa saisi aktiiviset hälytykset ja hälytyshistorian esitettyä, joista vanhassa järjestelmässä lopulta ainoastaan välitettiin, mutta katsotaan myös hieman, minkälaista kuvaajaa on mahdollista piirtää muusta datasta. Tavoitteessa V4 hälytysten osalta huolehditaan siitä, että ne menevät vaatimusten mukaan Slackiin. Tämä yhteen sopivuus on ainakin Kapacitorin dokumenteissa luvattu. Pyritään myös siihen, että hälytykset voisivat olla samantyylisiä kuin nykyisessä järjestelmässä. Tämä tehdään kuitenkin niin, että noudatetaan hyvien ilmoitusten määritteitä, eikä lähetetä ilmoituksia liian tiuhaan tahtiin tai epäselviä sellaisia. Tässä katsotaan vain toteutuskeino

Kapacitoriin. Lopulliset määrittelyt jätetään enemmän varsinaisen käyttöönottovaiheen suunnitteluun. Tavoitteen V5 saavuttamiseksi voitaisiin puolestaan yhdistää hälytykset Kapacitorin Slackin lisäksi sellaisen HTTP-protokollan POST-pyynnön lähettämiseksi, että Jira osaa tulkita sen rajapinnassaan tehtäväksi.

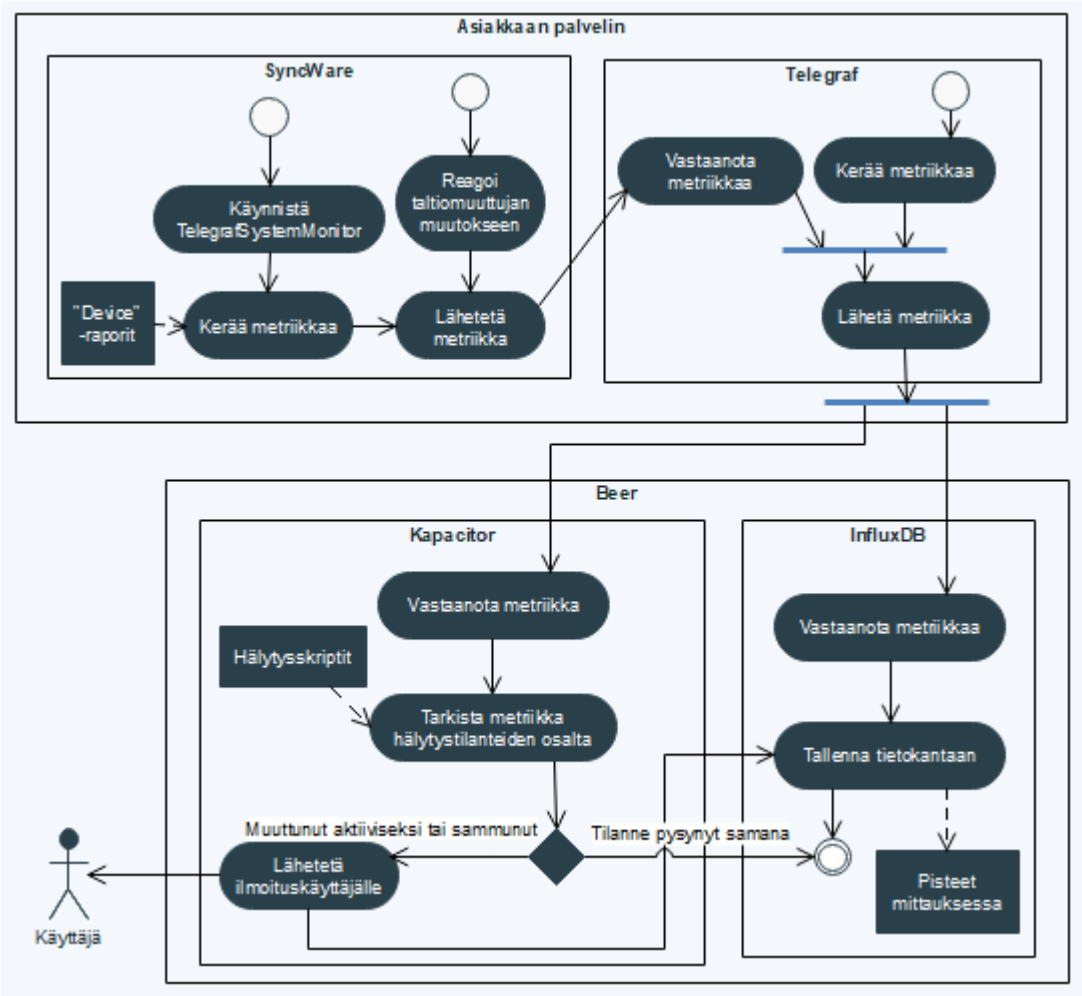
Tavoite V6:n osalta vaikuttaa, että johdannossa mainittu yksittäinen asiakas ei tarvitsisikaan tätä ominaisuutta. Ainakaan kerätystä datasta he eivät olleet kiinnostuneita vaan ainoastaan järjestelmän toimivuudesta. Tyydytään siis ilmoitusten lähettämiseen myös heille havaituista ongelma tilanteista. Tätä ei siis näin ollen ainakaan toistaiseksi toteuteta.

Toteutusta varten uutta ympäristöä varten on perustettu uusi palvelin, jonka käyttöjärjestelmänä on FreeBSD. FreeBSD on ilmainen avoimen lähdekoodin käyttöjärjestelmä, joka on johdettu BSD:stä, joka puolestaan pohjautuu UNIX-käyttöjärjestelmään. (FreeBSD 2019) Siihen kuuluu Grafana ja InfluxDB valmiina asennuspaketteina ja ne ovat palvelimelle jo asennettu. Kapacitoristakin löytyi nopeasti valmis asennuspaketti mutta Chronografin saamisessa sinne oli hieman ongelmia. Kuvassa 17 on esitetty uuden järjestelmämonitoroinnin käyttötapauskaavio käyttäjän ja uuden järjestelmämonitoroinnin osalta. Toiminnot ovat nyt jakautuneet eri komponenteille ja käyttäjällä määriteltävissä Device-raporttien lisäksi uusi keino taltioiden tarkkailun määrittämiseen. Hälytykset ovat myös määritettävissä kokonaan. Tämä tapahtuu Chronografin kautta. Asiakkaita ei konfiguroida järjestelmämonitoroinnin palvelimella vaan se tapahtuu asentamalla Telegraf asiakkaan palvelimelle ja määrittämällä sille kerättävä data.



Kuva 17. *Uuden järjestelmämonitoroinnin käyttötapauskaavio*

Kuvassa 18 on tämän uuden järjestelmämonitoroinnin toiminta kokonaisuus esitettynä aktiviteettikaaviona. Esitettynä on datan keräämisen, tallentamisen ja käsittelyn osuudet. Toiminnot ovat nyt suurimmalta osin siirtyneet pois SyncWarelta. Datan keräämisen suunta on vaihtunut vetomuotoisesta työntömuotoiseksi eli dataa ei haeta vaan se lähetetään. Perusidea on silti säilynyt samana.



Kuva 18. *Uuden järjestelmämonitoroinnin kokonaisuus*

5. TOTEUTUS

Luvussa käydään läpi suunnitelman toteutus. Ensiksi käsitellään datan kerääminen Telegrafin ja SyncWarelle jäävältä osilta. Lopuksi käsitellään hälytystilanteissa ilmoitusten lähettäminen ja JIRA-tehtävän luonti.

5.1 Telegrafin keräämä data

Win_perf_counters-lisäosan määrittely toimii luomalla objekti (*object*) ja antamalla sille objektin nimeen (*ObjectName*) tarkkailtava kohde, instansseihin (*Instances*) tarkkailtavat kohteen osien ilmentymät, laskureihin (*Counters*) tarkkailtavan kohteen osat ja mittaukseen (measurement), IsfluxDB:n mittaus johon data tallennetaan. Prosessorin kuormituksen datan kerääminen onnistuu siis ohjelmassa 8 esitetyllä tavalla.

```
[[inputs.win_perf_counters]]
  [[inputs.win_perf_counters.object]]
    # Prosessorin käyttö, raportoi per ydin.
    ObjectName = "Processor"
    Instances = ["*"]
    Counters = [
      "% Idle Time",
      "% Interrupt Time",
      "% Privileged Time",
      "% User Time",
      "% Processor Time",
      "% DPC Time",
    ]
    Measurement = "win_cpu"
    # Erillinen _Total instanssi kokonaisarvolle, kun tarkastellaan
    # kaikkia instannesja (*).
    IncludeTotal=true
```

Ohjelma 8. *Win_perf_countersi-lisäosan määrittely*

Cpu, disk, mem, swap ja system -lisäosilla on jokaisella omat parametrin ja esimerkiksi prosessorin määritys tehdään niin kuin ohjelmassa 9.

```
[[inputs.cpu]]
  ## Annetaanko arvo per CPU.
  percpu = true
  ## Annetaanko kaikki kokonaisarvot.
  totalcpu = true
  ## Kerätäänkö CPUn raaka aikadata.
  collect_cpu_time = false
  ## Lasketaanko ja kerätäänkö summa kaikista aktiivisista CPU
tiloista.
  report_active = false
```

Ohjelma 9. *CPU-lisäosan määrittely*

Muistin osalta Windowsin versio kuitenkin vaikuttaa puutteelliselta. Sen keräämästä datasta ei saa selville, kuinka suuri osa muistista on käytetty. Mem-lisäosa kerää sen suoraan pisteeseen used_percent ja se vaikuttaa toimivan myös Windowsissa, joten käytetään sitä win_perf_countersin sijaan tässä yhteydessä.

Tomcatin lisäosaa varten tarvitaan käyttäjätunnus, jolle on annettu manager-gui käyttöoikeudet. Lisäosa on määritelty niin kuin ohjelmassa 10.

```
# Kerää dataa Tomcatin tilasivulta.
[[inputs.tomcat]]
  ## URL eli osoite Tomcatin tilasivulle.
  url = "http://localhost/manager/status/all?XML=true"

  ## Tunnukset
  username = "käyttäjänimi"
  password = "salasana"

  ## Pyynnön aikakatkaisu.
  timeout = "5s"
```

Ohjelma 10. *Tomcat-lisäosan määrittely*

5.2 SyncWaren keräämä data

Käytettävä HTTP Listener v2 on määritelty ohjelmassa 11 esitettyyn tyyliin.

```
[[inputs.http_listener_v2]]
  ## Osoite ja portti HTTP kuuntelijalle.
  service_address = ":8080"

  ## Polku, jota kuunnellaan.
  path = "/telegraf"

  ## HTTP metodit, joita tarkastellaan.
  methods = ["POST"]

  ## Data formaatti, joka vastaanotetaan.
  ## Jokaisella formaatilla on omat kofiguraatio valinnat.
  ## Lisätietoa löytyy osoitteesta:
  ## https://github.com/influxdata/telegraf/blob/master/docs/DATA_FORMATS_INPUT.md
  data_format = "json"
  json_name_key = "measurement"
  json_time_format = "02-01-2006 15:04:05"
  json_timezone = "UTC"
  tag_keys = ["name"]
```

Ohjelma 11. *HTTP Listener v2 -lisäosan määrittely*

Telegraf kuuntelee siis osoitetta localhost/telegraf:8080 odottaen siihen JSON-muotoista POST-dataa. Lisäksi mittauksen, mihin data tallennetaan, voi määritellä antamalla JSON-avaimen *measurement* arvoksi jotain ja tagi vastaavasti antamalla avaimen *name* arvoksi jotain. Lähetys tapahtuu SyncWaressa siis uudella aikalistan tehtävällä *TelegrafSystemMonitor*, mikä on esitetty ohjelmassa 12.

```
// Suoritetaan, kun TelegrafSystemMonitor käynnistyy aikalistalla
public static void main(String[] args)
{
      // TelegrafSystemMonitorin konstruktorissa sille lisätään
      // kaikki mittaus arvot
      TelegrafSystemMonitor monitor = new TelegrafSystemMonitor();
      // Lähetys Telegrafille
      TelegrafConnection.sendToTelegraf(monitor.telegrafMonitor-
      DataJson.toString());
}
```

Ohjelma 12. *TelegrafSystemMonitorin main metodi*

Ohjelmassa 13 on esitetty Telegrafille lähetys, joka on toteutettu erikseen.

```
public static void sendToTelegraf(String data)
{
      // Lähetyksen muodostaminen Telegrafin kuuntelemaan
      // osoitteeseen
      CloseableHttpClient httpClient =
      HttpClientBuilder.create().build();
      try {
            HttpPost request = new HttpPost("http://lo-
            calhost:8080/telegraf");
            request.addHeader("content-type", "application/json");
            StringEntity params = new StringEntity(data);
            request.setEntity(params);
            CloseableHttpResponse response = httpClient.execute(re-
            quest);
            int status = response.getStatusLine().getStatusCode();
            // Ilmoitus epäonnistumisesta
            if (status != 204) {
                  Ooops.error(TelegrafConnection.class, null, "Sending
                  data to Telegraf failed with status: " + status);
            }
            response.close();
      }
      // Ilmoitus koodin kaatumisesta
      catch (Exception e) {
            Ooops.error(TelegrafConnection.class, e, "Sending data to
            Telegraf failed");
      }
      finally {
            try {
                  httpClient.close();
            }
            // Ilmoitus koodin kaatumisesta
            catch (IOException e) {
                  Ooops.error(TelegrafConnection.class, e, "HttpClient
                  close failed");
```

```
36              }
        }
38  }
```

Ohjelma 13. *Telegrafille lähetys*

Aikalistaa varten tehdyn luokan konstruktorissa luodaan *TelegrafMonitorDataJson*, joka sisältää *SystemMonitor*-luokan toiminnasta edelleen jäljelle jääneet haut ja muodostaa niistä JSON muotoisen datan Telegrafille lähetettäväksi. Ohjelmassa 14 on esitetty esimerkkinä replikointidatan keräys.

```
    public void addReplicatorData()
 2  {
        // Katsotaan onko replikointi palvelua olemassa
 4      if (ReplicatorService.getInstance() != null && ReplicatorServ-
        ice.getInstance().getReplicators() != null) {
 6
            // Käydään palveluiden repliointi määritteet läpi
 8          Iterator<AbstractReplicatorLink> iter = ReplicatorServ-
            ice.getInstance().getReplicators().iterator();
10          while (iter.hasNext()) {
                AbstractReplicatorLink replicatorLink = iter.next();
12
                // Muodostetaan JSON eri replikointien jonoista ja
14              // onko ne toiminnassa
                TelegrafJsonObject replicatorData = new
16              TelegrafJsonObject("replicator", replica-
                torLink.name);
18              replicatorData.put("queueSize", replica-
                torLink.getQueueSize());
20              replicatorData.put("isDown", booleanToInt(replica-
                torLink.isLinkDown()));
22              add(replicatorData);
            }
24
            // Replikointivirheen malliluokan läpikäynti
26          ReplicatorErrorColl replicatorErrorColl = new Replica-
            torErrorColl();
28          int errorCount = (int)replicatorErrorColl.fillCount();

30          // JSON replikointivirheiden määrästä
            TelegrafJsonObject replicatorErrorData = new
32          TelegrafJsonObject("replicator", null);
            replicatorErrorData.put("errorCount", errorCount);
34          add(replicatorErrorData);
        }
36  }
```

Ohjelma 14. *Replikointidatan keräys*

TelegrafMonitorDataJson on *JSONArray*-luokan aliluokka ja sisältää listan *TelegrafJsonObject*-objekteja, joka puolestaan on *JSONObject*-luokan aliluokka. Konstuktorissa annetaan *TelegrafJsonObject*-objektille *measurement* ja *tag*. Replikoinnin data kerätään siis replicator-mittaukseen ja replikointilinkkien dataan

merkitään tagiin nimi, kun taas virheiden määrä on omana erillisenä arvonaan. Datan olisi voinut lähettää myös suoraan InfluxDB:n komentomuodossa mutta koin tämän mielekkäämmäksi. Koodin luku edellyttää tosin tietoa Telegrafin asetusten määrittelystä, jota on pyritty helpottamaan käyttämällä *json_name_key* ja *tag_key* -attribuuttien arvoina sitä miksi ne on tarkoitettu käytettäviksi ja tekemällä ne automaattisesti määriteltäviksi arvoiksi *TelegrafJsonObject*-objektille.

Taltioiden tarkkailu olisi mahdollista tehdä niin, että kaikki data kerättäisiin InfluxDB-tietokantaan, ja hälytyskomentosarjoihin sitten määriteltäisiin, millaiset tilanteet tulkitaan virheellisiksi. Tämä ei kuitenkaan ole mahdollista, koska sopimukset kieltävät pääasiassa tämän datan keräämisen asiakkaan palvelimen ulkopuolelle. Näin ollen virhetilanteen määritteleminen on tehtävä asiakkaan palvelimella ja tieto toiminnallisuudesta on se data, mitä kerätään InfluxDB:hen. Device-raporttien logiikka tätä jo omalla tavallaan tekikin ja logiikka tulee jäämään asiakkaiden palvelimiin, koska niiden määrittelyjen siirtäminen muualle olisi liian työlästä. Toteutetaan kuitenkin myös uusi versio tarkkailtavien taltioiden määrittämisestä, jossa toiminta olisi yksinkertaisempaa ja käyttäjäystävällisempää määritellä.

Device-raporttien tarkkailua varten uudessa järjestelmämonitoroinnissa on luotu malliluokka *TelegrafDeviceReport*, jossa määritellään raportin sijainti kyseisessä järjestelmässä. *TelegrafSystemMonitor*-luokassa sitten käydään hakemassa tulokset taas *TelegrafMonotorDataJson*-objektiin. Tämä onnistuu hyödyntämällä *FormatterEngine*-luokkaa, joka on tehty raporttien käsittelemiseen koodissa varten. Se hakee raportille annetut määritykset ja suorittaa muuttujien tietojen haun raportin aikavälillä. Nämä tiedot poimitaan nyt Telegrafille lähetettäväksi ohjelmassa 15 esitettyyn tyyliin.

```
public void addDeviceReportData()
{
      // Haetaan käyttäjän määrittämät Device-raporttien polut
      TelegrafDeviceReportColl<TelegrafDeviceReport> telegrafDeviceRe-
      portColl = new TelegrafDeviceReportColl();
      telegrafDeviceReportColl.fillCollection();
      for (TelegrafDeviceReport telegrafDeviceReport : telegrafDeviceRe-
      portColl) {
            // Suoritetaan polun perusteella raportin haku
            String reportPath = telegrafDeviceReport.report;
            Hashtable params = new Hashtable();
            params.put("report", reportPath);
            params.put("class", "syncrontech.series.Result");
            FormatterEngine engine = new FormatterEngine();
            engine.parseAllParameters(params);

            // Arvojen haku, josta otetaan uusin
```

```
            Vector dataColl = engine.getFreshData();
20          Object data = dataColl.lastElement();

22          // Poimitaan raportilta laskujen nimet
            Any titles = engine.getVisiblePropertyParameters("titles");
24          int index = 0;
            // Arvojen tunnisteet
26          Vector<PropertyLink> props = engine.getPropertyLinks();

28          for (PropertyLink pl : props) {

30                // Poimitaan raportin yksittäisen laskun nimi ja sen
                  tulos
32                // ensimmäinen arvo vastaa ensimmäistä nimeä jne.
                  String title = titles.getProperty(index).toString();
34                String value = pl.getProperty(data).toString();
                  int status = 1;
36                // Laskut suoritetaan raportissa niin, että null
                  // palautetaan virhetilanteissa
38                if (value == null) {
                        status = 0; // Device ei ole ok
40                }

42                // Raportin arvot JSONiin
                  TelegrafJsonObject deviceData = new TelegrafJsonOb-
44                ject("device", title);
                  deviceData.put("status", status);
46                add(deviceData);

48                index++;
50          }
        }
52  }
```

Ohjelma 15. *Device-raporttien datojen keräys*

Device-raportit ovat silti tarpeettoman monimutkainen tapa toteuttaa taltion tarkkailu, kun tarkkailtava asia on aina sama: onko arvo pysynyt tyhjänä (null) tai samana tietyn ajan. Lisäksi olisi hyvä, jos voisi määrittää rajat, joiden välissä arvojen on oltava. Tätä ei raporteissa ei sitä tehdä, vaikka mahdollista olisikin. Uudeksi versioksi näiden tietojen keräämiseksi ensimmäinen ajatus oli tehdä yksinkertaisempi versio Device-raportista. Kokonaisen raportin sijaan olisi malliluokka, jolle voi määrittää muuttujan, tarkkaillaanko muuttujan pysymistä tyhjänä tai samana, tarkkaillaanko ylittääkö tai alittaako muuttuja jonkin arvon ja tarkkailun aikaväli. Tämä sitten tehtäisiin aina *TelegrafSystemMonitor*-luokan ajon yhteydessä. Lopulliseksi toteutukseksi kuinkin päätettiin, että tehdään tätä varten palvelu, joka tarkkailee taltioihin tallennettavia arvoja. Tämä onnistuu luomalla aliluokan *TriggerService*-luokalle, joka olisi *TelegrafPeriodTriggerService*, ja määrittämällä sille käytettävät *TriggerServer*-luokan aliluokat, joita tässä tapauksessa tarvitaan yksi *TelegrafPeriodTrigger*. Uusi palvelu on myös muistettava lisätä klusteriin. *TelegrafPeriodTrigger*-malliluokalle puolestaan voi määritellä tarkkailtavat taltiot, jolloin jokaisesta taltioon menevästä datasta saadaan sen arvo ja samalla myös edellinen

tallennettu arvo. Näistä sitten lähetetään tieto HTTP Listener v2 -lisäosan kautta InfluxDB:hen niin että dataan kuuluu tiedot: onko arvo tyhjä, onko arvo sama kuin edellisellä kerralla, onko arvo ylittänyt raja-arvon ja onko arvo alittanut raja-arvon. Hälytyskomentosarjassa pystyy sitten määrittelemään, kuinka kauan arvon täytyy olla tyhjä tai sama, ennen kuin siitä tehdään hälytys. Taltioiden poimiminen tarkkailtaviksi tapahtuu palvelun käynnistymisen yhteydessä, mikä on esitetty ohjelmassa 16.

```
private HashMap<String, TelegrafPeriodVariable> telegrafPeriodConditionMap;

// Suoritetaan palvelun käynnistyessä
@Override
public String[] getSourceVariables()
{
      // Haetaan käyttäjän määrittämät tarkkailtavat taltiot
      TelegrafPeriodVariableColl<TelegrafPeriodVariable>
      telegrafPeriodVariableColl = new TelegrafPeriodVariableColl();
      telegrafPeriodVariableColl.fillCollection();
      ArrayList<String> periodList = new ArrayList();
      // Kerätään ne palvelulle listaksi, jossa säilytetään taltion tietoja
      telegrafPeriodConditionMap = new HashMap();
      for (TelegrafPeriodVariable telegrafPeriodVariable : telegrafPeri-
      odVariableColl) {
            periodList.add(telegrafPeriodVariable.variable);
            telegrafPeriodConditionMap.put(telegrafPeriodVariable.variable,
            telegrafPeriodVariable);
      }
      Object[] ojectArray = telegrafPeriodConditionMap.keySet().toArray();
      // Palautetaan taulukko, joka sisältää tarkkailtavat taltiot
      return Arrays.asList(ojectArray).toArray(new String[ojectAr-
      ray.length]);
}
```

Ohjelma 16. *Tarkkailtavat taltioiden haku*

Arvon tulo taltioon käsitellään puolestaan ohjelmassa 17 esitetyllä tavalla.

```
// Suoritetaan, kun tarkkailtavassa taltiossa tapahtuu muutos.
// Sisältää aikaleiman, taltion tunnuksen, taltion vanhan ja uuden arvon
// kyseisellä aikaleimalla.
@Override
public void processChange(Date timeStamp, String var, Object oldVal, Object
val)
{
      // Heataan taltio listasta
      TelegrafPeriodVariable telegrafPeriodVariable =
      telegrafPeriodConditionMap.get(var);
      // Varmistetaan että taltion arvo on uudempi kuin viimeksi saatu
      if (telegrafPeriodVariable.oldTimeStamp == null || timeStamp.af-
      ter(telegrafPeriodVariable.oldTimeStamp)) {
            // Aletaan muodostamaan JSONia
            TelegrafJsonObject periodJsonData = new TelegrafJsonObject("pe-
            riod_condition", telegrafPeriodVariable.name, timeStamp);
            // Onko arvo tyhjä
            periodJsonData.put("valueNull", val == null ? 1 : 0);
```

```
20
            Double valDouble = null;
22          if (val != null) {
                // Onko arvo sama kuin viimeksi saatu arvo
24              if (telegrafPeriodVariable.oldValue != null) {
                    periodJsonData.put("valueSameAsPrevious",
26                  val.equals(telegrafPeriodVariable.oldValue) ? 1 :
                    0);
28              }

30              if (val instanceof Double) {
                    valDouble = (Double)val;
32                  // Onko arvo ylittänyt ylärajan
                    if (telegrafPeriodVariable.upperLimit != null) {
34                      periodJsonData.put("valueOverLimit",
                        valDouble > telegrafPeriodVariable.upperLimit
36                      ? 1 : 0);
                    }
38                  // Onko arvo alittanut alarajan
                    if (telegrafPeriodVariable.lowerLimit != null) {
40                      periodJsonData.put("valueUnderLimit",
                        valDouble < telegrafPeriodVariable.lowerLimit
42                      ? 1 : 0);
                    }
44              }
            }
46          // Telegrafille lähetys
            TelegrafConnection.sendToTelegraf(periodJsonData.toString());

50          telegrafPeriodVariable.oldTimeStamp = timeStamp;
            telegrafPeriodVariable.oldValue = val;
52      }
    }
```

Ohjelma 17. *Taltion arvon tarkkailu*

5.3 Hälytykset

Hälytysten määrittämiseen löytyy Chronografista niiden yksinkertainen luomisosio, jossa voi määrittää, mitä mittausta tarkkaillaan, jonkin rajamäärityksen sille, liitettävät hälytystoiminnot ja ilmoituksen mikä hälytyksestä tehdään. Mutta kuten testaus tuloksesta näkyy, tämä lähettää ilmoituksen aina kun mittaustulos luetaan ja se on rikkonut raja-arvoa, mikä tarkoittaa tiuhaa ilmoitustahtia. Tämä ei ole hyvän ilmoitusperiaatteen mukaista eikä pelkkä staattinen raja-arvo noudata hyvän monitoroinnin periaatetta. Tästä hälytysmäärittelystä Chronograf luo Kapacitorille hälytyskomentosarjan, jota pystyy erikseen muokkaamaan. Tähän komentosarjaan voidaan lisätä muutama komento ongelmien korjaamiseksi. Lisäämällä ennen Slackiin lähetystä *.slack()* komennon *.stateChangesOnly()*, ilmoitus lähetetään ainoastaan tilanteen muuttuessa. Komennolla

|stateDuration(lambda: "value" > 50).unit(1m)

voidaan raja-arvolle rikkoutumiselle määritellä, että mitataan sen kestoa, mikä on tässä tapauksessa yli viisikymmentä ja mittausaika on minuuteissa. Hälytystasolle voi sitten määritellä

.crit(lambda: "state_duration" >= 10)

Tämä tarkoittaa, että kriittiseen hälytykseen tilanteen täytyy kestää kymmenen minuuttia. Hälytyksen määrittelyosio komentosarjassa olisi ohjelman 18 tyylinen.

```
var trigger = stream
    |from()
        .database('telegraf')
        .retentionPolicy('autogen')
        .measurement('win_cpu')
        .where(lambda: "host" == 'booze')
// Tarkkaillaan prosessorin kuormitusta ottamalla keskiarvo
// kaikista mittaushetkellä saaduista arvoista
    |mean('Percent_Processor_Time')
        .as('value')
    // Mitataan kuinka kauan kuormitus on yli 50%
    |stateDuration(lambda: "value" > 50)
        .unit(1m)
    |alert()
        // Varoitus tila 1 minuutin jälkeen
        .warn(lambda: "state_duration" >= 1)
        // Kriittinen tila 10 minuutin jälkeen
        .crit(lambda: "state_duration" >= 10)
        // Välitetään vain tilan muutoksista
        .stateChangesOnly()
        .message('CPUn kuormitus {{ index .Fields "value" }}%. Taso
{{.Level}} tilanne ollut päällä {{ index .Fields "state_duration" }}
minuuttia')
        // Lähetetään hälytys Slackiin
        .slack()
        .channel('#alert-testing')
```

Ohjelma 18. *Hälytyskomentosarjan esimerkki*

Hälytyksen määrittelyssä on siihen liitettävissä Slackin lisäksi myös POST-pyyntö. Tälle ei kuitenkaan pysty määrittämään muuta kuin otsakkeen osion (engl. header) ja sisältö täytyisi olla tietynlainen JSON kokonaisuus, että Jira sen ymmärtäisi. Itse Kapacitorin konfiguraatiossa pystyy sentään POST-pyynnön sisältöön vaikuttamaan. Tämä onnistuu määrittelemällä sen osio ohjelmassa 19 esitetyllä tavalla.

```
[[httppost]]
   endpoint = "JIRA"
   url = "Jiran osoite"
   headers = { Content-Type = "application/json" }
   basic-auth = { username = "käyttäjätunnus", password = "salasana"
}

   # Määritä hälytyspohja, josta koostetaan HTTP body.
   # Hälytyspohjia käytetään vain hälytykset lähettäjien kanssa, koska
   # ne käyttävät hälytysdataa.
```

```
    # Pohja käyttää https://golang.org/pkg/text/template/ ja sillä on
12  # pääsy seuraaviin kenttiin:
    #    * .ID – Hälytyksen tunnus.
14  #    * .Message – Hälytyksen viesti.
    #    * .Details – Hälytyksen yksityiskohdat.
16  #    * .Time – Hälytyksen aikaleima.
    #    * .Duration – Hälytyksen kesto.
18  #    * .Level – Hälytyksen taso (INFO, WARN tai CRITICAL).
    #    * .Data – Hälytyksen käynnistänyt data.
20  #
    # Pohjan määrittely.
22  alert-template = "{{.Details}}"
```

Ohjelma 19. *HttpPost-lisäosan konfiguraatio*

Kapacitor käyttää hälytyksen yksityiskohtia (engl. details) POST-pyynnössä, jos sanoman päätepisteeksi (engl. endpoint) määritettään JIRA. Hälytyksen yksityiskohtiin voidaan sitten määrittää JIRA:lle kelvollinen JSON.

6. TESTAUS

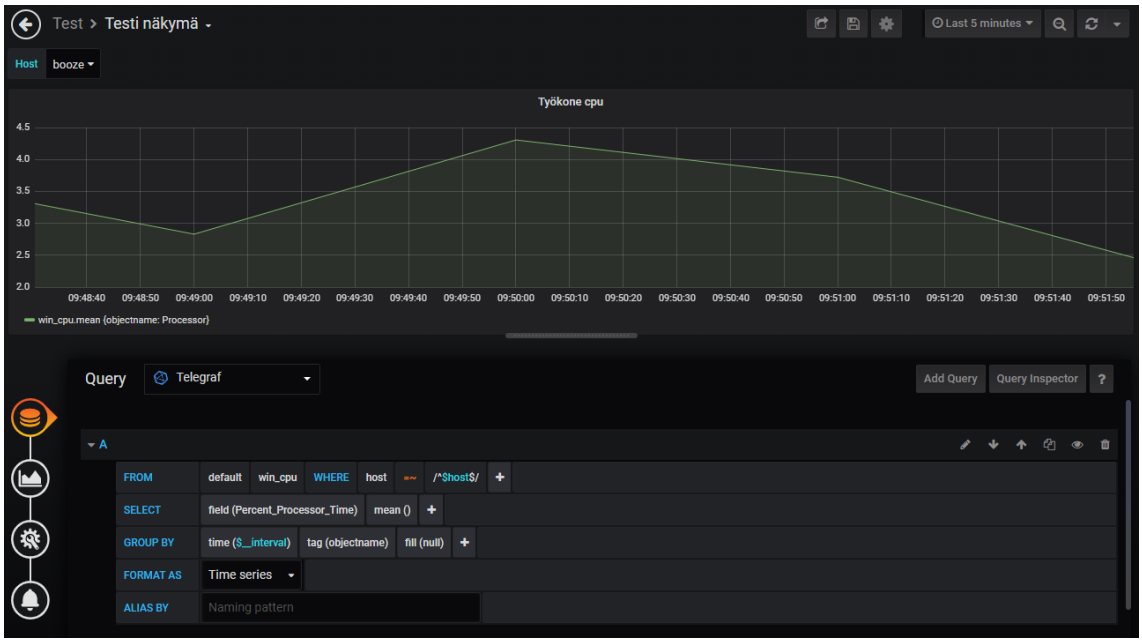
Luvussa käydään läpi toteutuksen testaus yleisiltä piirteiltään. Läpikäytäviä asioita ovat: yleiskuvaus testausympäristöstä, varmistukset että data siirtyy ja näyttää oikeanlaiselta sekä ilmoitusten lähettämisen onnistuminen ja JIRA tehtävän luonti.

6.1 Testaamisesta yleensä

Testaus suoritetaan käyttämällä omaa työkonetta testausympäristönä. Koneelle on asennettu Telegraf ja Tomcatissa ajetaan yksittäisen asiakkaan SynWare koodia. Testissä pyritään ensisijaisesti tarkistamaan, että data siirtyy onnistuneesti ja nopealla syklillä sekä mahdollisuuksien mukaan myös siirtyneen datan oikeellisuus tekemällä kerätystä datasta Grafanaan kuvaaja.

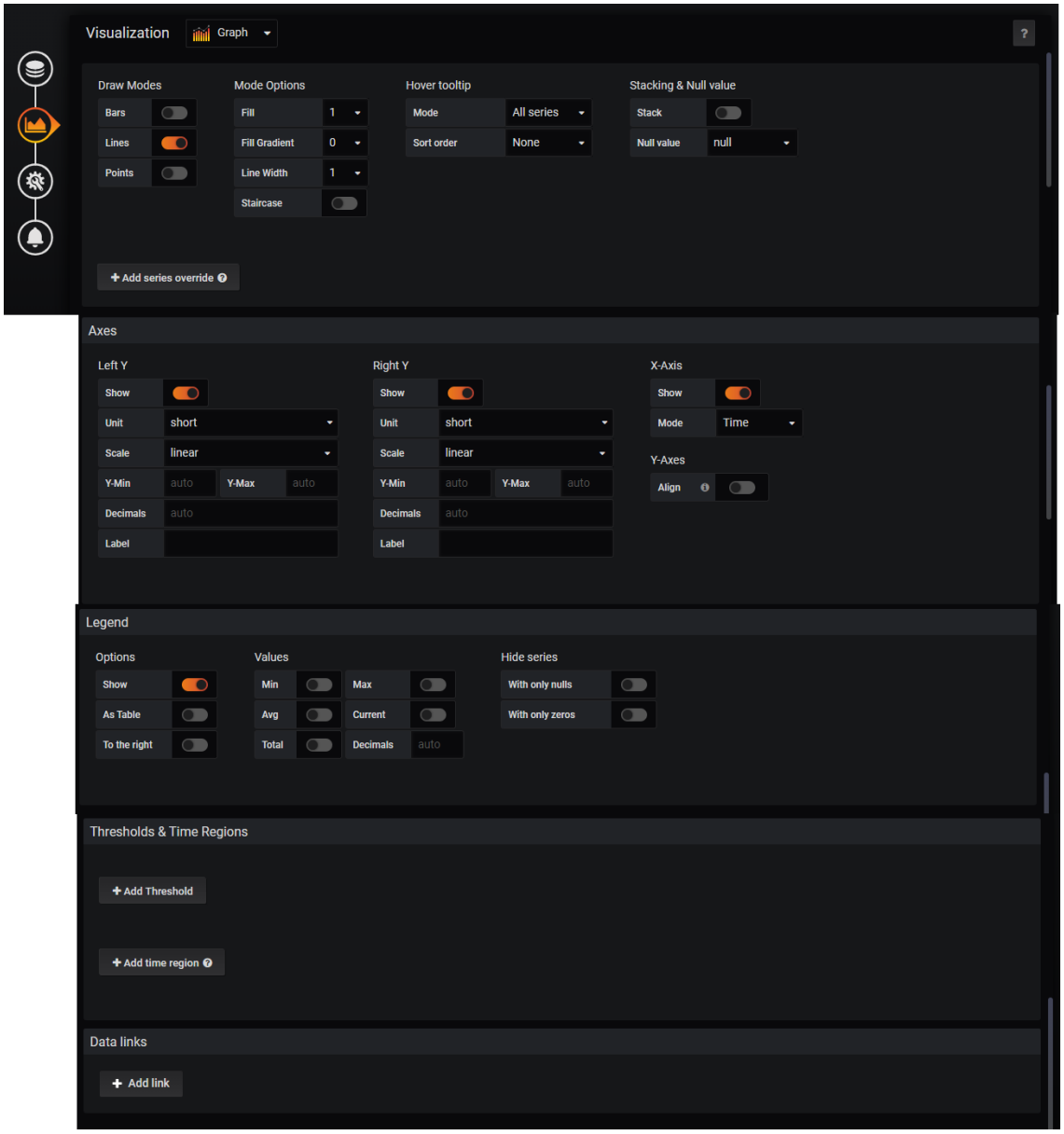
Grafanassa on luotu testi niminen raporttinäkymä, johon tämän koneen mittauksista tehdään kuvaajia. Raporttinäkymän yleisiin asetuksiin on määritelty muuttujaksi työkoneen nimi host=booze, jota voi käyttää haussa koneelta saadun datan hakemiseksi. Raporttinäkymälle määritellään myös miltä ajalta data haetaan.

Itse kuvaajan hakuasetuksissa määritellään InfluxQL:llä kuvaajaan haettava data. Kuvassa 19 esitettynä prosessorin kuormituksen haku. Vaihtoehtoisesti hakukomennon voi kirjoittaa kokonaan itse klikkaamalla kynäkuvaketta hakuikkunan oikeassa yläkulmassa.



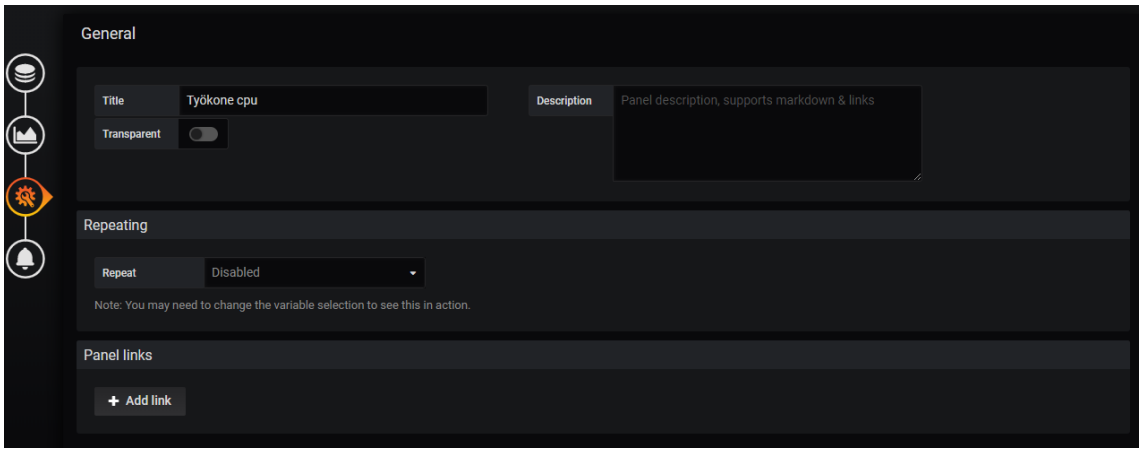
Kuva 19. *Kuvaajan haun asetukset*

Kuvaajan visuaalisissa asetuksissa ovat puolestaan itse kuvaajan tyypin asetukset ja valitun tyypin omat asetukset. Kuvassa 20 perusgraafi, josta voi päättää muun muassa esitetäänkö data viivoina, palkkeina tai pisteinä ja mikä on datan mittayksikkö.



Kuva 20. *Kuvaajan visuaaliset asetukset*

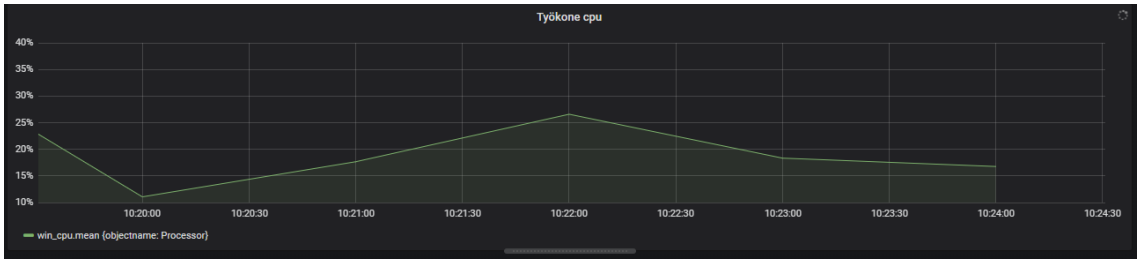
Yleisistä asetuksista, joka on esitetty kuvassa 21 on sitten kuvaajan nimen ja kuvauksen määrittäminen.



Kuva 21. *Kuvaajan yleiset asetukset*

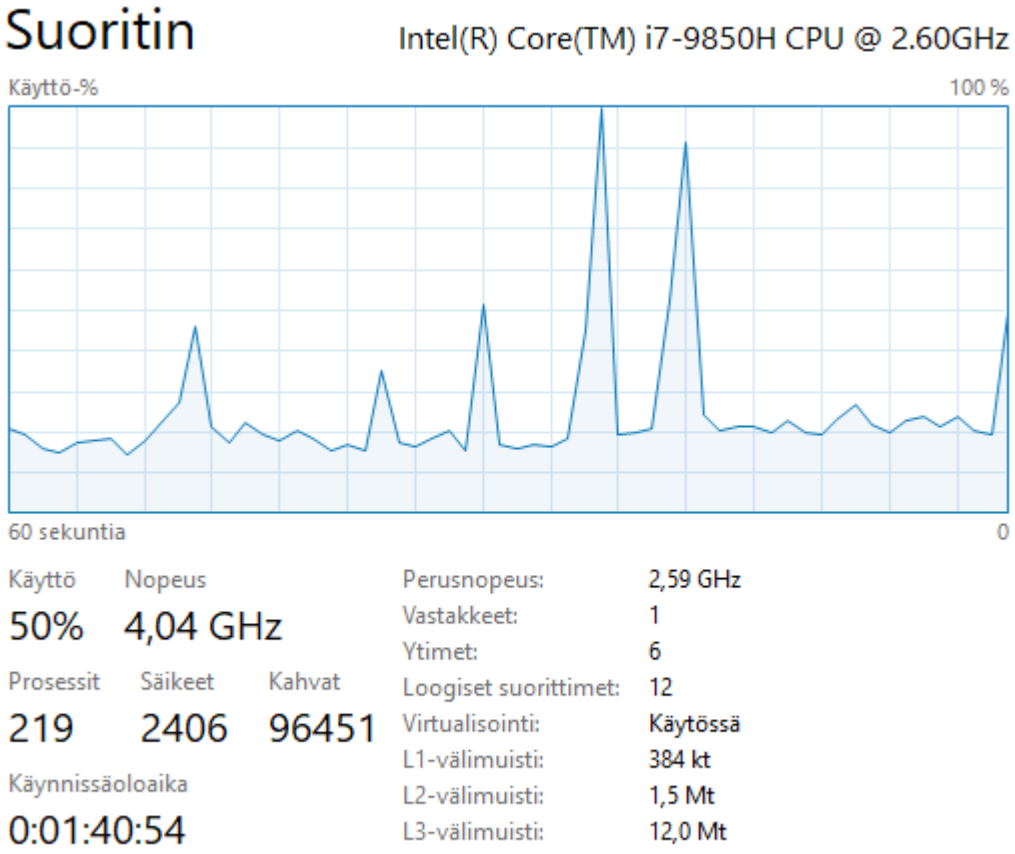
6.2 Telegrafin keräämän datan testaaminen

(V1) Datan keräämiseen ei vaadita muuta kuin että Telegraf on päällä ja lisäosat ovat määritettyinä. Kaikista monitoroitavista kohteista näyttää kertyvän dataa ja sitä minuutin syklillä, mikä tahti oli tämänhetkinen tavoite. Jos verrataan esimerkiksi CPU:n kuormitusta koneen ollessa lievässä kuormituksessa käynnissä olevista sovelluksista Windowsin oman järjestelmämonitoroinnin antamaan tulokseen, data näyttää siltä osin oikeilta. Molemmissa kuormitus on 15–25 % luokkaa tosin Windowsin lyhyemmän aikavälin monitoroinnissa näkyy myös hetkittäiset kuormituspiikit. Kuvassa 22 CPU:n kuormituksen data viimeiseltä viideltä minuutilta. Yksittäisen minuutin arvon on minuutin aikana mitattujen arvojen keskiarvo.



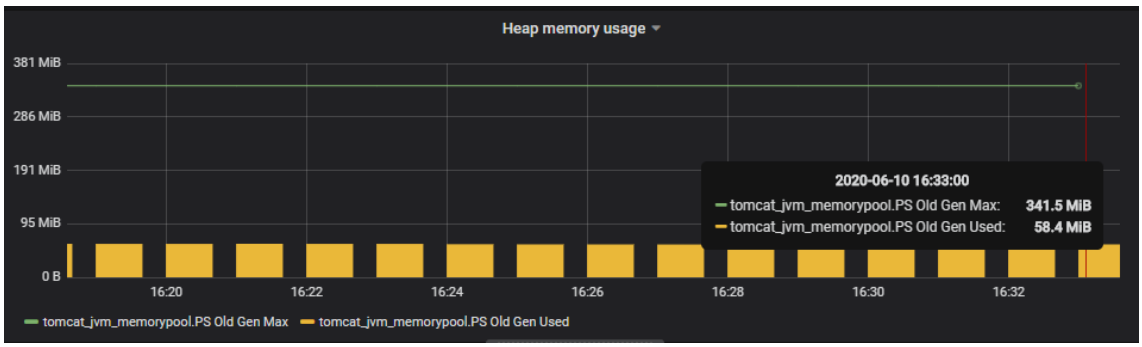
Kuva 22. *CPU:n kuormitus Grafanassa*

Kuvassa 23 Windowsin tehtävien hallinnasta otettu tilanne kuudenkymmenen sekunnin ajalta.



Kuva 23. *Windowsin tehtävien hallinnan antama tulos*

Samoin Tomcatin datan kanssa. Tarkkailtavan muistin PS Old Gen arvot vastaavat toisiaan 58,4 MB käytetty ja 341,5 MB on maksimi. Kuvassa 24 kerätty data PS Old Gen maksimista ja käytetystä muistista.



Kuva 24. *Tomcatin HeapMemory PS Old Gen*

Sama data Tomcatin tilanne sivulta kuvassa 25.

JVM

Free memory: 147.63 MB Total memory: 207.00 MB Max memory: 455.50 MB

Memory Pool	Type	Initial	Total	Maximum	Used
PS Eden Space	Heap memory	128.00 MB	37.50 MB	167.50 MB	0.93 MB (0%)
PS Old Gen	Heap memory	340.00 MB	168.00 MB	341.50 MB	58.42 MB (17%)
PS Survivor Space	Heap memory	21.00 MB	1.50 MB	1.50 MB	0.00 MB (0%)
Code Cache	Non-heap memory	2.43 MB	51.18 MB	240.00 MB	49.21 MB (20%)
Compressed Class Space	Non-heap memory	0.00 MB	7.00 MB	1024.00 MB	6.57 MB (0%)
Metaspace	Non-heap memory	0.00 MB	54.75 MB	-0.00 MB	52.93 MB

Kuva 25. *Tomcatin muistin käyttö*

6.3 SyncWaren keräämän datan testaaminen

(V2) SyncWaren keräämän datan saamiseksi täytyy Telegrafin käynnissä olon lisäksi ajaa paikallisessa SyncWaren ympäristössä aikalistalta TelegrafSystemMonitor. Paikallisesta ympäristöstä on kuitenkin hankala sanoa oikeellisuudesta kaikilta osilta tarkemmin, koska replikoitavaa dataa tai sanomaliikennettä ei ole ja klusterissa ei ole varmistusinstansseja, joten palveluita ei tarkkailla. Datan keräys on kuitenkin toteutettu samalla tavalla kuin ennenkin, joten jos se siirtyy InfluxDB:hen, sen pitäisi olla oikein. Jumiutuneiden transaktioiden määrän voidaan todeta pysyneen nollana, joten data ainakin siirtyy. Tätä voidaan paremmin testata sitten kun järjestelmämonitorointikokonaisuus saadaan toteutettua johonkin isompaan ympäristöön.

Taltioiden muutosteen reagoimista voidaan testata paremmin luomalla paikalliseen ympäristöön muokattava raportti, jossa on kenttänä tarkkailtava taltio. Tässä tapauksessa kyseinen taltio voisi olla vaikka jonkin laitteen lämpötila, jonka ylärajaksi määritellään tarkkailtavissa taltioissa 100 °C astetta. Taulukossa 2 esitetty taltioon syötetyt arvot aikaleimoineen:

Taulukko 2: *Taltioon syötetyt arvot*

Aikaleima	Arvo
8.5.2020 12:03	95
8.5.2020 12:04	96
8.5.2020 12:05	101
8.5.2020 12:06	102
8.5.2020 12:07	105
8.5.2020 12:08	105
8.5.2020 12:09	105
8.5.2020 12:10	90

Kyseisitä arvoista on saatu tiedot oikein, mikä on esitetty kuvassa 26 järjestyksessä uusimmasta vanhimpaan.

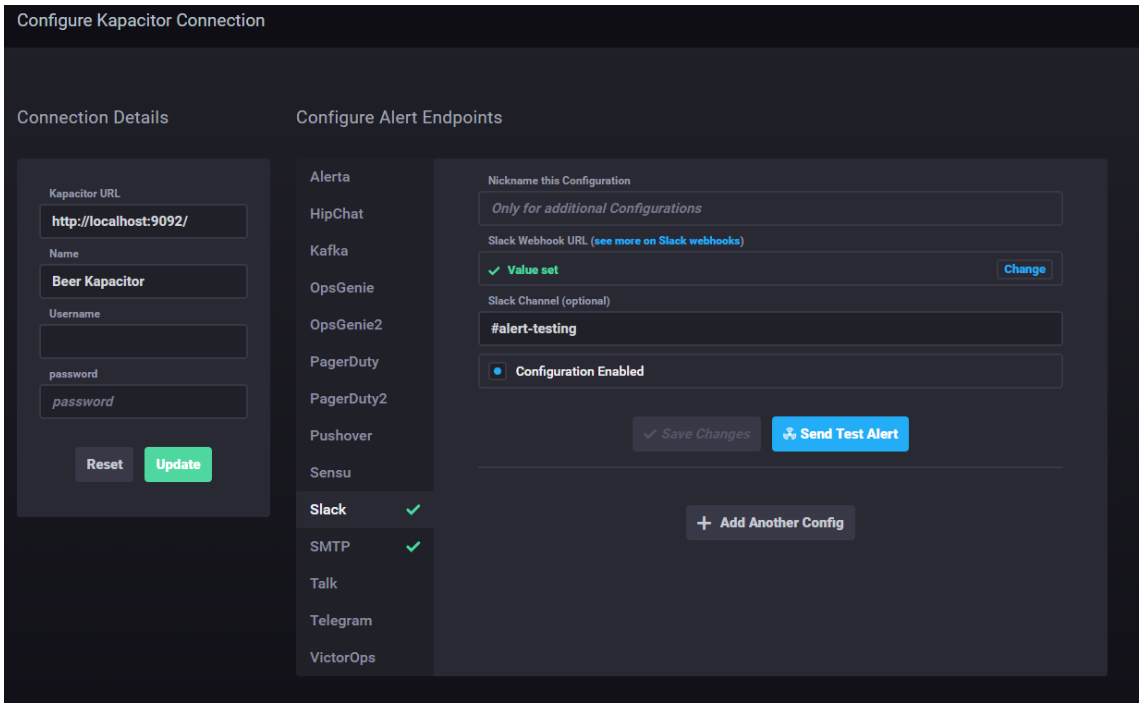
Lämpötilataltio			
Time ▾	Tyhjä	Pysynyt samana	Yläraja ylitetty
2020-05-08 12:10:00	0	0	0
2020-05-08 12:09:00	0	1	1
2020-05-08 12:08:00	0	1	1
2020-05-08 12:07:00	0	0	1
2020-05-08 12:06:00	0	0	1
2020-05-08 12:05:00	0	0	1
2020-05-08 12:04:00	0	0	0

Kuva 26. *Värjätty Grafana taulukko taltion tiedoista*

(V3) Kaikkia eri kuvaajia ei yritetty datan esittämiseen käyttää tosin vaikutti siltä, että muutama tuntui kaatuvan helposti kuten tilannepaneeli, jolla voisi esittää dataa vähän samaan tyyliin kuin värjätyssä taulukossa. Joka tapauksessa valmiin asiakkaalle lähetettävän raportin muodostaminen kerätystä datasta näyttäisi onnistuvan. Ehtona on, että käytetään sellaisia tageja ja arvoja, joista saadaan valmis laskennat tekevä tietokannan hakukomento luotua.

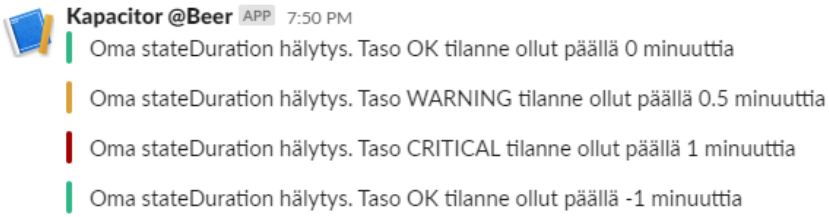
6.4 Hälytysten testaaminen

(V4) Slackin osalta hälytyksiä varten on luotu kanava #alert-testing. Tälle on lisätty itse tehty Slack sovellus Kapacitor @Beer, jolle on luotu vastaanottoa varten verkkokoukun osoite. Kyseinen osoite on annettu Chronografissa Kapacitorille, mikä on esitetty kuvassa 27.



Kuva 27. *Kapacitorin konfigurointi Chronografissa*

Vähentämällä toteutuksessa mainitussa hälytyskomentosarjassa prosessorin kuormitus raja normaalia tilannetta pienemmäksi saadaan siitä aikaseksi hälytys, josta lähtee ilmoitus Slackiin, joka on esitetty kuvassa 28.



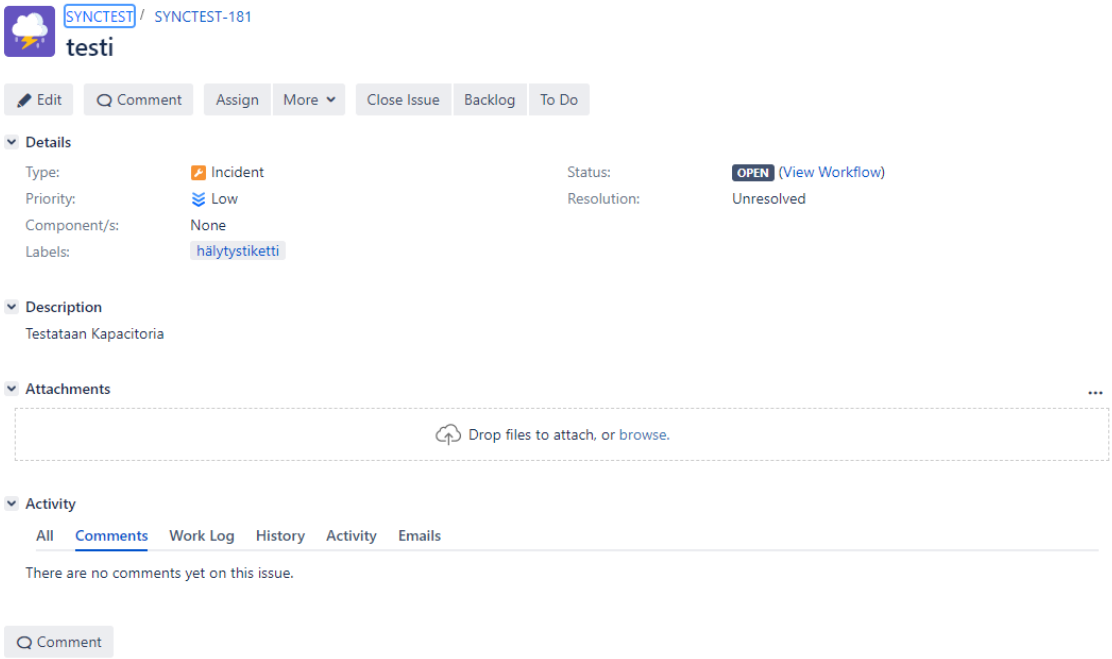
Kuva 28. *Viestejä #alert-testing kanavalla*

Kapacitorin dokumentointi väittää, että Slackiin lähetettävään viestiin voisi määrittää myös näytettävän käyttäjänimen komennolla .username('nimi') ja nimen perässä mahdollisesti käytettävän hymiön komennolla .iconEmoji('hymiön tunnus') mutta ominaisuus ei vaikuta toimivan.

(V5) Jiraan kirjoitusta varten puolestaan lähetettiin HTTPOST-lisäosalla Jiran osoitteeseen JSON, jonka Jiran käyttöliittymä pitäisi osata tulkita tehtäväksi. Lähetetty viesti oli testi projektille ja se oli muotoa

{"fields": {"summary": "testi", "project": {"key": "SYNCTEST"}, "issuetype": {"id": "10003"}, "description": "Testataan Kapacitoria", "reporter": {"name": "helpdesk"}, "labels": ["hälytystiketti"]}}

ja Jira kirjasi sen tehtäväksi, mikä on esitetty kuvassa 29.



Kuva 29. *Kapacitorista luotu Jiran tehtävä*

7. YHTEENVETO

Diplomityössä oli tarkoituksena toteuttaa konsepti uudelle järjestelmämonitorointikokonaisuudelle, joka tulee korvaamaan yrityksen nykyisen järjestelmämonitoroinnin SyncWaressa. Täksi uudeksi järjestelmämonitoroijaksi valittiin TICK tuotekokonaisuus, joka koostuu Telegrafista, InfluxDB:stä, Chronografista ja Kapacitorista sekä graafinen esitystyökalu Grafana. Vaatimuksina toteutukselle oli, että uusi järjestelmämonitorointi pystyy olennaisilta osin samaan kuin vanhakin niin että toteutus kuormittaa mahdollisimman vähän SyncWarea. Lisäksi uusien ilmoituksien hälytyksistä täytyisi mennä keskustelu ohjelma Slackiin ja itse kerätty data olisi lähetettävissä yksittäisen asiakkaan omaan järjestelmämonitorointiin. Tutkimuskysymyksen vastaus sekä vaatimukset ja niiden täyttämisistä ovat esitetty taulukossa 3. Viittauksina on käytetty ensimmäisessä luvussa määritettyjä tunnuksia.

Taulukko 3: *Tutkimuskysymys ja vaatimukset*

Tutkimuskysymys	**Vastaus**
T	Yrityksen alkuperäisestä ideasta saatiin hieman sitä muokkaamalla toimiva ratkaisu. Sen muodostaa TICK tuotekokonaisuus, joka koostuu Telegrafista, InfluxDB:stä, Chronografista ja Kapacitorista sekä graafinen esitystyökalu Grafana. Replikointi- ja muistipalvelun tilanteiden, transaktioiden ja klustereiden tilanteiden sekä Device-raporttien ja taltioiden monitorointi jää edelleen SyncWarelle. Samoin tietokantojen tilanne toistaiseksi ja loki kirjoitusten tarkkailu jätetään kokonaan pois. Näiden monitorointi pyritään toteuttamaan, kun siihen sopivat lisäosat tulevat Telegrafilla käytettäviksi. Palvelimen yleinen tarkkailu ja SyncWaren muistin käyttö voidaan tehdä Telegrafin lisäosilla.
Vaatimus	**Vaatimuksen täyttö**
V1	Telegraf korvaa osittain SyncWaren datan kerääjänä ja muuten SyncWare lähettää tälle keräämänsä datan. Telegrafin omaan datan keräykseen käytettävät lisä osat ovat: win_perf_counters, cpu, disk, mem, swap ja system sekä apache_tomcat Kapacitorista tulee uusi hälytysten lähettäjä ja Chronograf toimii siihen käyttöliittymänä. InfluxDB:stä tulee uusi datan ja hälytysten säilytyspaikka. Grafanasta tulee uusi hälytysten ja datan visuaalinen esittäjä.
V2	SyncWare lähettää keräämänsä datan Telegrafin HTTP Listener V2 lisäosaan. Taltioiden tarkkailulle tehtiin myös oma yksinkertaisempi toteutus Device-raporteista.
V3	Onnistuu Grafanan raporttinäkymissä. Tämä todettiin testausta varten luodussa rapottinäkymässä.

V4	Onnistuu Kapacitorin hälytyskomentosarjoilla, joiden määritys onnistuu helpoiten Chrongrafin kautta. Slackiin lähetys onnistuu käyttäen Kapacitorin Slack lisäosaa, jolle on annettu Slackkiin luodun sovelluksen verkkokoukun osoite.
V5	Toteutettiin käyttäen Kapacitorin HttpPostia, jolle lähetetään tietynlainen JSON muotoinen viesti hälytys tilanteissa kirjoittamalla viesti hälytyksen yksityiskohtiin.
V6	Jätettiin toteuttamatta asiakkaan kiinnostuksen puutteen vuoksi

Tulevaisuudessa koodi ja Telegrafin toteutus on tarkoitus alkaa ottaa asiakkaiden ympäristöissä käyttöön, jotta toiminnasta saadaan enemmän tietoa ja voidaan paremmin tutkia tarvittavia käytännöllisiä ominaisuuksia. Itse Telegrafien asennuksissa ei pitäisi olla ongelmaa, koska konfiguraatio tiedoston voi suurimmassa osassa tapauksista vain kopioida. Tarkkailtavien taltioiden määritteleminen on myös yksinkertaisempaa kuin ennen tosin uudessa versiossa ei tällä hetkellä ole mahdollista määrittää laskukaaviota tarkkailtavan taltion arvolle niin kuin vanhassa. Tämä on ominaisuus, mitä täytyy harkita uudellekin versiolle mutta toistaiseksi jos se on jostain syystä tarpeellista, se onnistuu Device-raportilla. Riippuen siitä, miten hälytysten halutaan toimivan, jokaisen hälytyksen määrittäminen komentosarjoissa voidaan kokea liian työlääksi. Chronografin hälytyksen luonnin käyttöliittymä, kun ei vaikuta riittävän Jiran kanssa yhteistoimintaan ja tarpeeksi monipuolisten ehtojen määrittämiseen, tosin sitä voi käyttää komentosarjan luonnin pohjana. Tästä täytyy kuitenkin mahdollisesti vielä selvittää miten sitä voisi yksinkertaistaa. Paras ehdotus tällä hetkellä on, että yksittäisen tarkkailtavan koneen kaikki hälytysehdot kirjoitetaan yhteen komentosarjaan, josta se on kopioitavissa ja muutettavissa säätämällä komentosarjan alussa olevia muuttujia uutta ympäristöä vastaaviksi. Lisäksi mahdollisesti tikettisysteemin voisi yrittää toteuttaa myös uudelle järjestelmälle. Se jätettiin tällä kertaa pois tavoitteista, koska vaikka ominaisuus on olemassa se ei ole kuin muutamalla asiakkaalla käytössä. Alustavan version esittelyssä kuukausipalaverissa palautteena oli, että viestinnän täytyisi olla mahdollista myös tavalla, mikä on täysin erillään muista viestintäkanavista niin kuin tekstiviestit olivat vanhassa järjestelmässä. Tulevaisuudessa Slackin ja sähköpostin lisäksi tätä varten sopiva keino voisi olla Pushover niminen ilmoitussovellus.

Työn aikana InfluxLabin InfluxDB 2.0 siirtyi testausvaiheeseen. Tässä uudessa versiossa Chrografia ja Kapacitoria ei enää ole erikseen vaan niiden ominaisuudet on siirretty itse InfluxDB:lle. Tässä uudessa versiossa tietokanta haku ei ole enää samalla tapaa samankaltainen SQL-kielen kanssa niin kuin InfluxQL vaan sitä varten on luotu uusi Flux niminen kieli. Myös TICKScript hälytyskomentosarjakieltä ei enää ole vaan myös hälytykset toteutetaan Flux-kielellä. Tämä tulee tarkoittamaan isoja muutoksia,

mikäli se pyritään ottamaan käyttöön tulevaisuudessa ja saattaisi tarkoittaa, että olisi parempi odottaa sen julkaisua ennen kuin toteutusta viedään pidemmälle. Uudessa versiossa on kuitenkin luvattu, että vanhan version hakujen ja hälytyskometosarjojen migraatio olisi mahdollista.

LÄHTEET

Atlassian (2020) *Jira Software*. Saatavissa: https://www.atlassian.com/software/jira (Viitattu 1.12.2019)

Basili, V. R. (1992) 'The Experimental Paradigm in Software Engineering', *In Experimental Software Engineering Issues: Critical Assessment and Future Directions, International Workshop, Germany, H D Rombach and V R Basili and R W Selby (Eds*.), LNCS 706. Springer-Verlag, s. 3-12.

Carnegie Mellon Database Group (2020a) *InfluxDB.* Saatavissa: https://dbdb.io/db/influxdb (Viitattu 24.1.2020).

Carnegie Mellon Database Group (2020b) *Prometheus*. Saatavissa: https://dbdb.io/db/prometheus (Viitattu 7.2.2020).

collectd (2019a*) collectd – The system statistics collection daemon*. Saatavissa: https://collectd.org/ (Viitattu 21.2.2020).

collectd (2019b) *Powerful, unobtrusive Windows statistics*. Saatavissa: https://ssc-serv.com/ (Viitattu 21.2.2020).

collectd (2019c) *First steps.* Saatavissa: https://collectd.org/wiki/index.php/First_steps (Viitattu 21.2.2020)

collectd (2019d) *Manpage collectd.conf (5).* Saatavissa: https://collectd.org/documentation/manpages/collectd.conf.5.shtml (Viitattu 21.2.2020)

FreeBSD (2019) *About FreeBSD*. Saatavissa: https://www.freebsd.org/about.html (Viitattu 28.2.2020).

Goldberg, J. (2019) *Time Series Databases, the Future of Monitoring and Logging.* Saatavissa: https://securityboulevard.com/2019/12/time-series-databases-the-future-of-monitoring-and-logging/ (Viitattu 21.2.2020).

Grafana Labs (2020) *Grafana features*. Saatavissa: https://grafana.com/grafana (Viitattu 21.2.2020).

Helenius, T. (2010) *Tietokantojen kopioinnin toteutus verkkosovelluksessa, diplomityö*, Lappeenrannan teknillinen yliopisto.

InfluxData Inc. (2020a) *Time series database (TSDB) explained*. Saatavissa: https://www.influxdata.com/time-series-database/ (Viitattu 21.2.2020).

InfluxData Inc. (2020b*) Telegraf 1.11 documentation*. Saatavissa: https://docs.influxdata.com/telegraf/v1.11/ (Viitattu 24.1.2020).

InfluxData Inc. (2020c) *Telegraf Configuration*. Saatavissa: https://github.com/influxdata/telegraf/blob/master/etc/telegraf.conf (Viitattu 31.1.2020).

InfluxData Inc. (2020d) *InfluxDB 1.7 documentation.* Saatavissa: https://www.influxdata.com/time-series-platform/ (Viitattu 24.1.2020).

InfluxData Inc. (2020e) *InfluxDB key concepts*, Saatavissa: https://docs.influxdata.com/influxdb/v1.7/concepts/key_concepts/ (Viitattu 24.1.2020).

InfluxData Inc. (2020f) *InfluxDB compared to SQL databases*. Saatavissa: https://docs.influxdata.com/influxdb/v1.7/concepts/crosswalk/ (Viitattu 24.1.2020).

InfluxData Inc. (2020g) *Getting started with InfluxDB OSS*. Saatavissa: https://docs.influxdata.com/influxdb/v1.7/introduction/getting-started/ (Viitattu 24.1.2020).

InfluxData Inc. (2020h) *Database management using InfluxQL*. Saatavissa: https://docs.influxdata.com/influxdb/v1.7/query_language/database_management/ (Viitattu 31.1.2020).

InfluxData Inc. (2020i) *Chronograf 1.7 documentaition*. Saatavissa: https://docs.influxdata.com/chronograf/v1.7/ (Viitattu 31.1.2020)

InfluxData Inc. (2020j) *Installing Chronograf*. Saatavissa: https://docs.influxdata.com/chronograf/v1.7/introduction/installation (Viitattu 31.1.2020)

InfluxData Inc. (2020k) *Getting started with Chronograf*: Saatavissa: https://docs.influxdata.com/chronograf/v1.7/introduction/getting-started/ (Viitattu 31.1.2020).

InfluxData Inc. (2020l) *Getting started with Kapacitor*. Saatavissa: https://docs.influxdata.com/kapacitor/v1.5/introduction/getting-started/ (Viitattu 31.1.2020).

InfluxData Inc. (2020m) *Kapacitor 1.5 documentation*. Saatavissa: https://docs.influxdata.com/kapacitor/v1.5/ (Viitattu 31.1.2020).

InfluxData Inc. (2020n) *Genericsqlplugin*. Saatavissa: https://github.com/influxdata/telegraf/pull/2785 (Viitattu 1.3.2020).

InfluxData Inc. (2020o) *Windows Event Log Analytics*. Saatavissa: https://github.com/influxdata/telegraf/issues/4525 (Viitattu 1.3.2020).

Nenonen, V. (2010) *Käyttäjien uudelleenohjauksen toteuttaminen hajautetussa sovelluspalveluklusterissa*, diplomityö, Lappeenrannan teknillinen yliopisto.

Prometheus Authors (2020a) *FAQ*. Saatavissa: https://prometheus.io/docs/introduction/faq/ (Viitattu 7.2.2020).

Prometheus Authors (2020b) *Overview – What is Prometheus?*. Saatavissa: https://prometheus.io/docs/introduction/overview/ (Viitattu 7.2.2020).

Prometheus Authors (2020c) *Client libraries*. Saatavissa: https://prometheus.io/docs/instrumenting/clientlibs/ (Viitattu 7.2.2020).

Prometheus Authors (2020d) *Metric types – Prometheus*. Saatavissa: https://prometheus.io/docs/concepts/metric_types/ (Viitattu 7.2.2020).

Prometheus Authors (2020e) *Prometheus JVM Client*. Saatavissa: https://github.com/prometheus/client_java#exporting-to-a-pushgateway (Viitattu 7.2.2020).

Prometheus Authors (2020f) *Configuration*. Saatavissa: https://prometheus.io/docs/prometheus/latest/configuration/configuration/ (Viitattu 7.2.2020).

Prometheus Authors (2020g) *Getting started*. Saatavissa: https://prometheus.io/docs/prometheus/latest/getting_started/ (Viitattu 7.2.2020).

Prometheus Authors (2020h) *Defining recording rules*. Saatavissa: https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/ (Viitattu 7.2.2020).

Prometheus Authors (2020i) *Querying Prometheus*. Saatavissa: https://prometheus.io/docs/prometheus/latest/querying/basics/ (Viitattu 14.2.2020).

Prometheus Authors (2020k) *Query examples*. Saatavissa: https://prometheus.io/docs/prometheus/latest/querying/examples/ (Viitattu 14.2.2020).

Prometheus Authors (2020l) *Alerting overview*. Saatavissa: https://prometheus.io/docs/alerting/overview/ (Viitattu 14.2.2020).

Prometheus Authors (2020m) *Configuration (Alerting)*. Saatavissa: https://prometheus.io/docs/alerting/configuration/ (Viitattu 14.2.2020).

Prometheus Authors (2020n) *Alerting rules*. Saatavissa: https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/ (Viitattu 14.2.2020).

Prometheus Authors (2020o) *Comparison to alternatives*. Saatavissa: https://prometheus.io/docs/introduction/comparison/#prometheus-vs-influxdb (Viitattu 22.7.2020).

Pukki, M. diplomi-insinööri, projektipäällikkö, Syncron Tech Oy, Lappeenranta, Haastattelu, 29.5.2020

Rikkonen, T. insinööri, tekninen tuki, Syncron Tech Oy, Lappeenranta, Haastattelu, 29.5.2020

Slack (2020) *Slack*. Saatavissa: https://slack.com/intl/en-fi/?eu_nc=1 (Viitattu 1.12.2019).

solid IT GmbH (2020) *DB-Engines Ranking of Time Series DBMS*. Saatavissa: https://db-engines.com/en/ranking/time+series+dbms (Viitattu 21.2.2020)

Suutari, A. diplomi-insinööri, teknologia- ja kehitysjohtaja, Syncron Tech Oy, Lappeenranta, Haastattelu, 29.5.2020

Syncron Tech Oy, SyncWaren koodi (2020)

Turnbull, J. (2018) *Monitoring with Prometheus*. Turnbull Press.

Zoho Corp (2020a) *Installation guide*. Saatavissa: https://www.manageengine.com/products/applications_manager/help/installing-application-manager.html (Viitattu 17.4.2020)

Zoho Corp (2020b) *Getting Started*. Saatavissa: https://www.manageengine.com/products/applications_manager/help/getting-started-applications-manager.html (Viitattu 17.4.2020).

Zoho Corp (2020c) *Introduction*. Saatavissa: https://www.manageengine.com/products/applications_manager/howtodemos/getting-started/introduction.html (Viitattu 17.4.2020).

Zoho Corp (2020d) *Creating a New Monitor*. Saatavissa: https://www.manageengine.com/products/applications_manager/help/discovery-configuration.html (Viitattu 17.4.2020).

Zoho Corp (2020e) *Creating Monitor Groups*. Saatavissa: https://www.manageengine.com/products/applications_manager/help/creating-monitor-groups.html (Viitattu 17.4.2020).

Zoho Corp (2020f) *Working with Applications Manager*. Saatavissa: https://www.manageengine.com/products/applications_manager/howtodemos/MonitorGroup/creategroup.html (Viitattu 17.4.2020).

Zoho Corp (2020g) *Custom Monitor Type*, Saatavissa: https://www.manageengine.com/products/applications_manager/help/custom-monitor-types.html (Viitattu 17.4.2020)

Zoho Corp (2020h) *Associating Threshold and Action with Attributes*. Saatavissa: https://www.manageengine.com/products/applications_manager/help/associating-threshold-and-action-with-attributes.html (Viitattu 17.4.2020).

Zoho Corp (2020i) *Alarm Configuration Demo*. Saatavissa: https://www.manageengine.com/products/applications_manager/howtodemos/configure_alerts/introduction.html (Viitattu 17.4.2020).

Zoho Corp (2020j) *Creating Threshold Profile*. Saatavissa: https://www.manageengine.com/products/applications_manager/help/creating-threshold-profile.html (Viitattu 17.4.2020).

Zoho Corp (2020k) *Creating Actions*. Saatavissa: https://www.manageengine.com/products/applications_manager/help/creating-actions.html (Viitattu 17.4.2020).