Haaga-Helia
ammattikorkeakoulu Oy

Laiterekisterin luku Dockerin ja Grafanan avulla

Axel Rusanen

Opinnäytetyö
Tietojenkäsittely
2020

Tiivistelmä

Haaga-Helia ammattikorkeakoulu Oy

Tekijä(t) Axel Rusanen	
Koulutusohjelma Tietojenkäsittelyn koulutusohjelma	
Raportin/Opinnäytetyön nimi Laiterekisterin luku Dockerin ja Grafanan avulla	**Sivu- ja liitesivumäärä** 24 + 0

Opinnäytetyöni oli tuotekehitystä Helsingin kaupungille eli toiminnallinen opinnäytetyö. Opinnäytetyössä pystytettiin toimeksiantajan palvelimelle helposti skaalautuva ja siirrettävä visualisaatioratkaisu. Tarkoituksena oli saada selville eri koulujen tietokoneiden käyttöasteet ja työ oli rajoitettu yhden palvelimen asennukseen toimeksiantajan sisäisessä verkossa.

Työ toteutettiin marras- ja joulukuussa 2019 toimeksiantajan tiloissa Töysänkadulla Helsingissä. Projekti toteutettiin itseohjautuvana opiskeluna ja toimintana, jossa projektin asiakas oli Helsingin kaupungin kasvatus- ja koulutustoimialan IT-osasto.

Toimeksiantajalla oli ongelmana saada selville laitteiden käyttöasteita. Ratkaisun toteutustapaan minulle oli annettu täysin vapaat kädet, joten projektin tuloksena syntyi toimiva graafinen ratkaisu annettuun ongelmaan, jossa on vielä jatkokehitysmahdollisuuksia. Toimeksiantajan testipalvelimelle asennettiin Grafana Docker-konttina ja käytettiin Docker-composea määrittelemään haluttu lopputulos. Halutut tiedot joita Grafana luki tulivat MySQL-dumpista, joka tehtiin suoraan laiterekisteristä samalle palvelimelle.

Asiasanat
ICT, Visualisaatio, Palvelinten hallinta, Pilvipalvelut

Sisällys

1 Johdanto

Aloitin projektin ollessani toimeksiantajalla eli Helsingin kaupungin kasvatus- ja koulutusalan IT-puolella työharjoittelussa. Toimeksiantaja hallinnoi kaikkia Helsingin kaupungin koulujen laitteita ja heillä on laajamittainen laiterekisteri, johon tulee hyvin paljon tietoa kaikista heidän laitteistaan. Toimeksiantajan johtoportaalla oli halu saada selville koulujen laitteiden käyttöasteita ja he etsivät ratkaisua tähän ongelmaan. Kysyin sen hetkiseltä esimieheltäni, että voisinko tehdä opinnäytetyöni tämän ongelman ratkaisusta. Minulle annettiin täysin vapaat kädet teknisestä toteutuksesta, joten päädyin valitsemaan käytettäviksi ohjelmistoiksi Dockerin ja Grafanan, joista molemmista minulla oli jo aikaisempaa projektikokemusta koulun kursseilta.

Helsingin kaupungin kasvatuksen- ja koulutustoimialan toimipaikkana toimi Töysänkadulla sijaitseva toimistotila, jossa toimeksiantajalla oli hyvin varusteita melkein minkä tahansa projektin läpivientiin. Tämä projekti toteutettiin käyttämällä kannettavaa tietokonetta, johon olin asentanut uusimman stabiilin version Ubuntusta ja jolla otin etäyhteyden heidän palvelimeensa, jossa laiterekisteri sijaitsi. Palvelin oli Debian Linux, joten sen käyttäminen ei ollut hankalaa, koska Ubuntu on melkein samanlainen kuin Debian. Tämä laiterekisteri oli pelkästään opetuksen verkon puolella ja se oli testiversio varsinaisesta laiterekisteristä, mutta toimintaperiaate oli sama kuin oikeassa. Projekti toteutettiin testiympäristössä, jotta pystytään arvioimaan sen luotettavuus ja toiminta ennen käyttöönottoa.

Projektin alkaessa minulle selvisi, että toimeksiantajan käyttämässä laiterekisterissä oli jo tämän kaltaisia visuaalisia toimintoja, mutta ne eivät olleet tarpeeksi riittäviä. Helsingin kaupungin kasvatuksen- ja koulutuksentoimialan hallinnon haluama ytimekäs tiivistelmä laiterekisterin sisällöstä käsitti eri koulujen käyttöasteet ja tietoturvapäivitykset. Aktiivisten koneiden osuus saataisiin näytettyä helposti valmiiden ratkaisujen, kuten grafanan avulla. Tiesin myös, että Docker ja Grafana toimisivat todella hyvin toistensa kanssa ja Docker mahdollistaisi tulevan ratkaisun siirrettävyyden ja skaalautuvuuden, joka voisi olla tärkeää ottaen huomioon toimeksiantajan suuren laitekannan.
Työn tarkoituksena oli saada edellä mainittuun ongelmaan toimiva ratkaisu.

Nykyisissä laskentaympäristöissä virtualisointi mahdollistaa paikallisten ja pilvipalveluympäristöjen täyden hyödyntämisen. Docker on yksi suosituimmista virtualisointiympäristöistä, jossa jokainen palvelu ja sen vaatimat ohjelmistot määritellään konteissa ja näitä kontteja voi pyörittää millä tahansa alustalla, jota Docker tukee. Tämä mahdollistaa saumattoman siirtymän alustalta toiselle, jos palvelun skaalaaminen vaatii tätä.

Grafana on avoimen lähdekoodin aikasarjojen visualisointiin erikoistunut ohjelmisto, jolla on valmiina oma docker konttinsa, joka voidaan liittää useisiin tietokantoihin helposti. Grafanan kehittäjät ovat myös dokumentoineet hyvin erilaisia käyttötarkoituksia ja muokkausmahdollisuuksia. Täten käyttäjä voi keskittyä oman näkymänsä eli dashboardin rakentamiseen. Graafit ovat myös interaktiivisia ja mahdollistavat myös sen, että loppukäyttäjä voi muokata niitä ilman ohjelmointikokemusta. Grafana myös tarjoaa useita automaatiomahdollisuuksia, kuten hälytyksiä.

Organisaation käyttämä laiterekisteri on ranskalaisvalmisteinen GLPI. Sitä käytetään agentin kanssa, joka asennetaan laitteeseen ja se hakee laitteista niiden ominaisuustietoja ja käyttötietoja hyvin yleisellä tasolla. Tästä syntyy noin sata tietopistettä laitetta kohti ja rekisterin piiriin kuuluu satojatuhansia laitteita. Tämän tietomäärän tehokas hyödyntäminen vaatii tiedon tiivistämistä kuhunkin käyttötarkoitukseen. Yleisin tiivistämistapa on graafit, joita tässäkin projektissa käytetään.

Toimeksiantaja käyttää agentti ohjelmistoa, joka on jokaisessa heidän laitteessaan. Tämä agentti ohjelmisto tuo helposti kaiken datan saataville laiterekisteriin. Agentti ohjelmiston nimi on FusionInventory.

Valitsin projektissa käytetyt työkalut omien kokemuksien takia ja en periaatteessa miettinyt tai suunnitellut käyttäväni mitään muuta. Grafana olisi toiminut yksin ilman Dockeria, mutta silloin projektista olisi tullut liian suppea ollakseen opinnäytetyö ja Docker mahdollistaa myös skaalautuvuuden ja helpon ylläpidon, joten se tuntui luontevalta lisäosalta projektin kannalta. Docker myös mahdollistaa tehokkaamman IT-infran käytön, säästön IT-kuluissa ja nopeamman sovelluskehityksen.

Normaalisti IT-infrassa on perinteisesti käytetty virtuaalipalvelimia, jossa on tietty sovellus jokaisessa. Tämä käytäntö tarkoittaa että jokainen sovellus tarvitsee erillisen käyttöjärjestelmän alleen, joka saattaa vaatia lisenssin, mutta vaatii ainakin resursseja. Dockerin käytöllä voidaan käyttää mahdollisesti vain yhtä käyttöjärjestelmää, josta voidaan ajaa kontteja. Kontit eivät varaa passiivisesti resursseja, vaan ne pyytävät käyttöjärjestelmältä tarvitut resurssit kun niitä ajetaan.

2 Tietoperusta

2.1 Docker

Docker on platform as a service (PaaS) tuote, joka tarkoittaa palvelualustan ulkoistamista. Dockerin avulla voidaan pakata applikaatio ja sen tarvitsemat lisäosat virtuaaliseen konttiin ja tätä tekniikkaa kutsutaan konttiteknologiaksi. Näitä kontteja voidaan sitten pyörittää haluamassaan käyttöjärjestelmässä helposti ja kätevästi. Tämä teknologia mahdollistaa skaalautuvuuden, siirrettävyyden, nopeuden saada applikaatio pyörimään ja helpomman ylläpidon (Vase 2015). Jokainen näistä ominaisuuksista tuo jotain hyötyä varsinkin isommille yrityksille. Dockeria on käytetty esimerkiksi suosituissa applikaatioissa kuten Spotifyssä, Yelpissä ja Ebayssä. (Bui 2015).

Avaan vähän tuota konttiteknologiaa, eli kontit ovat periaatteessa kokoelma sovelluksista ja niiden tarvitsemista lisäosista esimerkiksi kirjastoista ja konfiguraatioista, jotta applikaatiot toimisivat aina saman lailla riippumatta paikasta. Yksinkertaistettuna esimerkiksi kehittäjä A saattaa saada tietyn sovelluksen toimimaan omassa testiympäristössään ja kehittäjä B ei saa sitä toimimaan mitenkään omassa tuotantoympäristössään. Tähän konttiteknologia tuo ratkaisun, koska kontit ovat aina pakattu saman lailla ne toimivat saman lailla suurimmassa osassa ympäristöjä, tietty poikkeuksiakin varmasti löytyy. (Rubens 2017). Docker on vain alusta, joka hyödyntää ja on kehittänyt konttiteknologiaa eteenpäin ja tulen puhumaan opinnäytetyössäni Docker-konteista enkä juurikaan yleisesti konttiteknologiasta.

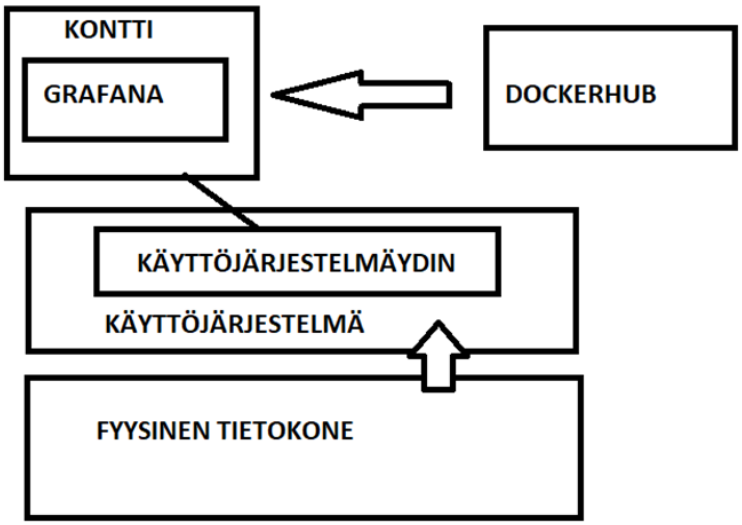
Docker tarvitsee alleen käyttöjärjestelmän, kuten Linuxin tai Windowsin, jolla se voi ajaa konttejaan. Dockerin-konttien ajoa varten tarvitsee myös mahdollistaa virtualisointi, joka onnistuu melkein jokaisella nykyaikaisella prosessorilla.

Docker on periaatteessa rakennettu viidestä osasta, joista kolme pääosaa ovat Docker Engine, Docker Daemon ja Docker Hub. Ensimmäinen on avoimen lähdekoodin virtualisointiratkaisu, joka pyörittää palvelimella ajattavat kontit ja viimeinen on (SaaS) Software as a Service alusta, jossa jaetaan Docker imageja. (Bui 2015).

Dockerin osia on myös Docker Image ja Docker File. Docker Image on tiedosto, joka sisältää sovelluksen ja sen tarvitsemat muut osat. Se on periaatteessa luettavissa oleva runko ja sen luomiseen tarvitaan Docker File. Grafanan kehittäjät ovat jakaneet tälläisen Imagen Docker Hubiin, josta sen saa ladattua. Docker File on konfiguraatiotiedosto, joka kertoo, miten Docker Image rakennetaan. (Docker 2019b)

Docker-kontti on käynnistettävissä oleva instanssi Docker Imagesta. Käyttäjä voi ohjata sen toimintaa, kuten käynnistyksiä, siirtoja ja pysäytyksiä Dockerilla itsellään komentokehoitteen kautta. Kontit määräytyvät niille asetetuista Docker Fileistä ja projektiin liittyen, jos tahtoisi liittää kontin useampaan verkkoon samanaikaisesti, niin se toimisi. (Docker 2019b)

Docker daemon pyörii käyttöjärjestelmässä kernelissä taustalla ja vastaanottaa komentoja joko komentoriviltä tai ohjelmistokirjastosta. Daemon hallinnoi erilaisia abstraktioita eli ohjelmistojen kuvia, kontteja, erilaisia nimiavaruuksia ja kytköksiä kuten cgroups linuxilla. (Combe, Martin & Pietro 2016).
Käyttäjän ei tarvitse tietää daemonin toiminnasta mitään, vaan hän voi toimia konttiabstraktion tasolla, mutta kokeneet käyttäjät ja ylläpitäjät voivat tarkastella helposti daemonin toimintaa esimerkiksi docker status komennolla.



Kuva 1. Docker-kontin arkkitehtuurikuvaus. (Rusanen 2020)

Docker Hub on konttien jakelu- ja säilytyspalvelu, johon eri ohjelmistojen kehittäjät ovat pistäneet omia kontteja, joita voi käyttää vapaasti. Tämän ansiosta esimerkiksi turvallisuuspäivytykset ovat helposti saatavilla kontin uudelleenkäynnistyksen yhteydessä.

Dockerin turvallisuus perustuu prosessien eristämiseen käyttäjätasolla Docker daemonin avulla, konttien eristys pohjautuu Linux kernelin ominaisuuksiin, kuten namespaces, cgroups ja kapasiteetin hallintaan. Namespace eristys ja kapasiteetin hallinta on vakiona kytkettynä päälle kun ollaan tekemisissä Docker konttien kanssa, mutta cgroups asetukset eivät ole ja ne pitää ottaa käyttöön joka kontissa erikseen. (Combe, Martin & Pietro 2016).

Dockerin vianselvityksessä tuli myös käytettyä runsaasti komentoa ”systemctl status docker”, joka näyttää docker.servicen tämän hetkisen tilan. Docker.service on periaatteessa Docker Engine.

2.2 Docker-compose

Docker-compose on työkalu millä voi määritellä ja pyörittää monikonttisia Docker-ohjelmia. Docker-compose käyttää YAML-tiedostoja määrityksien tekemiseen. Tämän jälkeen kaikki palvelut jotka ovat määritelty, pyöräytetään olemassaoloon ja käyntiin yhdellä komennolla ”docker-compose up”. Docker-composen käyttämiin YAML tiedostoihin voi määritellä kaiken porttimäärittelyistä, versioihin ja konttien nimiin. (Docker 2019a)

Alkuperäinen Docker-compose tiedosto on tyhjä tiedosto ja kaikki projektia varten vaaditut määrittelyt on tehty katsomalla ohjeita ja soveltamalla.

Compose-tiedoston tarkoituksena on määrittää halutut ohjelmistot, verkot ja säilytystilan pysyvyys. Verkkojen tehtävänä on määritellä mitkä kontit voivat kommunikoida keskenään ja isäntäkoneen välillä. Säilytystilan pysyvyyttä käytetään ohjelmistojen tilojen säilyttämiseen.

```
1  version: '3.3'
2  services:
3      grafana:
4          image: grafana/grafana:latest
5          restart: always
6          ports:
7              - "3000:3000"
8          environment:
9              - GF_SECURITY_ADMIN_PASSWORD=erittainsalainen
10         volumes:
11         - grafana_storage:/var/lib/grafana
12 volumes:
13   grafana_storage:
14
```

Kuva 2. Opinnäytetyössä käytetty grafana.yml pohja, josta on vaihdettu salasana. (Rusanen 2020)

Ylläolevassa kuvassa näkyy projektissa käytetyn Docker-compose tiedoston rakenne, joka luo stabiilin Grafana docker-kontin ja hakee esimerkiksi tietoturvapäivitykset jokaisella uudelleenkäynnistyskerralla. Tiedostossa määritellään aluksi versio, jota se käyttää. Imagen versio, joka on pistetty ”latest”, jotta saadaan kaikki tietoturvapäivitykset heti saataville. Portti on määritelty 3000:3000 mikä on default gateway Grafanalle. Enviroment kohdassa on määritelty käyttäjän salasana. Tämän voisi tehdä myös erillisenä env tiedostolla, jota Docker-compose tiedosto lukisi. Volumes määrittelee Grafanan tallennuspaikan, jotta kontin tiedot eivät katoa aina kun kontti käynnistetään uudelleen tai se kaatuu. Toinen volumes on tarvittava osa, jos tahdotaan vaikka käyttää monella eri kontilla samaa volumea. Verkkoa ei tarvinnut määritellä tässä projektissa Docker-composeen, koska projekti ja testilaiterekisteri olivat samalla palvelimella. Tästä olisi tullut yksi lisäosa, jos projekti olisi otettu tuotantopalvelimille käyttöön jo minun ollessani projektissa mukana.

```
1   influx:
2     image: influxdb
3     container_name: influx
4     ports:
5       - "8086:8086"
6     volumes:
7       - influxdb:/var/lib/influxdb
8
9   grafana:
10    image: grafana/grafana
11    container_name: uganda
12    ports:
13      - "3000:3000"
14    env_file:
15      - 'env.grafana'
16    links:
17      - influx
```

Kuva 3. Esimerkki kuvankaappaus docker-compose tiedostosta (Rusanen 2018)

Ylläolevassa kuvassa on aikaisemmin kouluprojektia varten tekemäni Docker-compose tiedosto, josta näkyy samoja piirteitä kuin projektia varten tehdyssä, mutta tässä käytetään

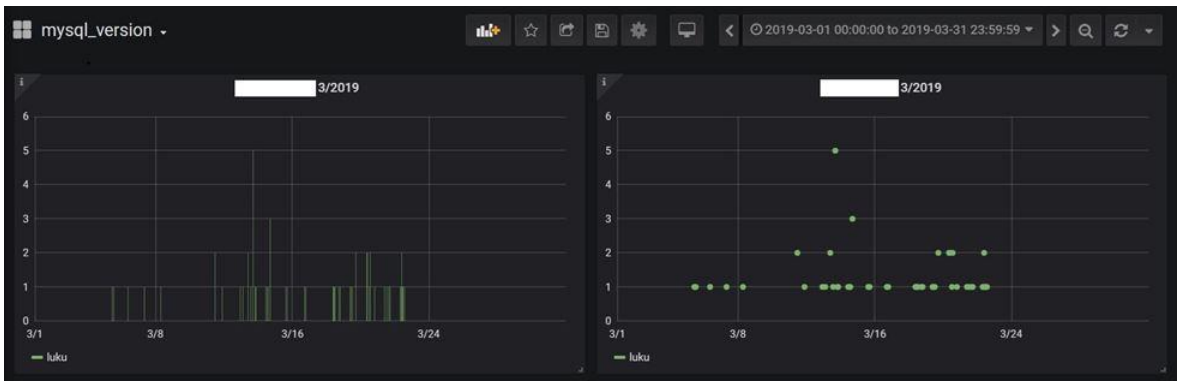
”environment” sijaan ”env_file”, joka on tiedosto johon on määriteltynä samat asiat kuin environment osioon. Tässä tiedostossa on myös käytetty ”container_name”, jolla voidaan määritellä docker-konteille omia nimiä, jotka helpottavat esimerkiksi ylläpitoa. Docker nimeää kontit ilman tuota asetusta satunnaiseksi sarjaksi numeroita ja kirjaimia.

Tahdoin tuoda kuvalla 3 esille eroavaisuuksia kahdesta eri Docker-compose tiedostosta ja lähestymistavasta.

2.3 Grafana

Grafana on avoimen lähdekoodin analytiikan ja interaktiivisen visualisoinnin ohjelmisto. Sillä voidaan piirtää graafeja, karttoja ja laittaa hälytyksiä tiettyihin datapisteisiin kun se on yhdistetty johonkin datalähteeseen. Grafanaa voi laajentaa lisäämällä siihen yhteisön ja kehittäjien tekemiä lisäosia, kuten projektissa käytettyä GLPI API:a. Grafana on hyvin yleinen tieteellisissä ja lääketieteellisissä piireissä. (Grafana Labs 2020).

Grafana valikoitui projektin osaksi, koska olin aikaisemmin käyttänyt sitä omissa projekteissani ja en tiennyt muitakaan visualisointiohjelmistoja. Kibana olisi voinut olla yksi vaihtoehtoinen ratkaisu, mutta en ollut perehtynyt siihen ennen projektin alkua ja tiesin että projektin voisi toteuttaa Grafanalla. Minulla oli myös tieto siitä, että Grafana toimisi saumattomasti Dockerin kanssa, koska ohjelmistojen kehittäjät tekevät myös yhteistyötä Docker Hubin kautta. Grafanalle on oma konttikonfiguraatio siellä, jota ylläpidetään Grafanan omien kehittäjien toimesta.



Kuva 4. Ruudunkaappaus esimerkki graafeista projektin ajalta (Rusanen 2019)

Kuvassa näkyy Grafanan ”dashboard” johon käyttäjä voi rakentaa omia näkymiä käyttämällä erilaisia graafeja ja määritellemällä niille aikavälejä. Kyseisessä kuvassa on otettu tietystä koulusta tietyltä aikaväliltä tietoja ja näytetty ne erinäköisissä graafeissa, kuten pylväinä ja pisteinä.

2.4 MySQL ja API

MySQL on relaatiotietokantaohjelmisto. Se on todella suosittu web-palveluiden tietokanta. Sitä käytetään usein LAMP-stackissä, mikä on Linux, Apache, MySQL ja PHP/Python/Perl yhdistelmä. MySQL sisältää rajapinnan useille eri ohjelmointikielille.

API eli ohjelmointirajapinta on määritelmä, jonka mukaan ohjelmat voivat keskustella, eli vaihtaa tietoja keskenään. (Haglund 2018).

2.5 YAML

YAML on ihmisystävällinen merkintäkieli, jota käytetään usein konfiguraatiotiedostoissa, kuten Docker-composessa ja Ansiblessa. Se periytyy Jsonista (=ohjelmointikieli) ja se kuvaa tekstin rakennetta tavalla, jolla pyritään erottamaan tekstin looginen rakenne sisällöstä. YAML tiedosto koostuu tekstistä, jossa voidaan määritellä erilaisia tietorakenteita, kuten listoja. (Ben-Kiki & Evans 2009)

2.6 PaaS

Tarkoittaa ”Platform as a Service” eli palvelualustan ulkoistamista. Etuina muun muassa mahdollistaa ohjelmistokehityksen ja pilvimallin mukaisen teknisen kehityksen. Antaa kehittäjille mahdollisuuden ja välineet ladata omia sovelluksiaan osaksi kokonaisuutta. Kehitysmalli mahdollistaa sen, että kehittäjien ei tarvitse huolehtia ohjelmiston skaalautuvuudesta, koska alustaa on mahdollista laajentaa tarpeen mukaan. ”PaaS voi myös olla kehittäjän itse tarpeen mukaan pilveen käynnistämä virtuaalikone, jossa on valmiiksi asennettuna ja konfiguroituna kaikki tarvittavat komponentit sekä integraatio kehitysympäristöön.” (Ahonen 2014)

2.7 SaaS

”Software as a Service” eli ohjelmiston hankkimista palveluna (Verneri 2017). Tarkoittaa periaatteessa kokonaista ohjelmistoratkaisua, jonka ostat pilvipalveluntarjoajalta. Otat käyttöön esimerkiksi jonkin applikaation yrityksessäsi ja sinun yrityksesi henkilöstö kirjautuu kyseiseen ohjelmistoon internetin välityksellä, yleensä internetselaimella. Kaikki ohjelmiston pohjalla oleva infrastruktuuri ja data sijaitsevat palveluntarjoajan datakeskuksessa. (Microsoft 2020)

3 Empiirinen osa

Työtapakuvaus: Itseohjautuva työtaitojen harjoittelu internetistä löytyneen materiaalin kanssa ja työkalujen testaus ennen varsinaista projektia.

3.1 Testaus

Projektissa käytettiin manuaalista testausta aina kun rakennettiin jotain uutta. Ensimmäisenä testinä oli pystyttää grafana ilman mitään muita ohjelmistoja. Tämä toteutettiin Ubuntun ohjelmistonhallinnalla ja tämän tarkoituksena oli selvittää Grafanan perustoimintoja ja konfigurointia.

Seuraavana testinä oli siirtää tämä konfiguraatio dockerin päälle ja siinä luotiin konfiguraatiotiedosto ja katsottiin että syntynyt kontti vastasi odotuksia.

Kolmantena testinä haluttiin pysyvä tila, niin muokattiin konfiguraatiotiedostoa niin, että siellä oli pysyvä säilytystila Grafanalle. Katsottiin että asennuksen tila pysyi samana kahden uudelleenkäynnistyksen aikana.

Grafanan sisällä tehtiin testejä että oli käyttäjätili, johon pystyi kirjautumaan aiemmin konfiguroidulla salasanalla ja käytettiin Grafanan testitoimintoa määritelläksemme että tietokantayhteys toimii odotetusti. Testasimme että taulukoita voi yhdistellä Grafanan SQL toiminnoilla, jota testasimme tekemällä yhdistetyn taulukon ja katsomalla sen tuloksia Grafanalla. Testattiin tehdä graafeja tästä tiedosta ja tämä testattiin luomalla graafi ja katsomalla että se toimii millä tahansa aikajanalla.

Tahdottiin että tiedot olisivat näkyvissä kouluittain, joten muokkasimme Grafanan piirtomäärittelyitä ja katsottiin että se näytti halutun tiedon halutulla tavalla. Testattiin systeemin uudelleenpystytettävyyttä ja se testattiin sillä että käynnistettiin uudelleen dockerin kautta ja tila säilyi samana.

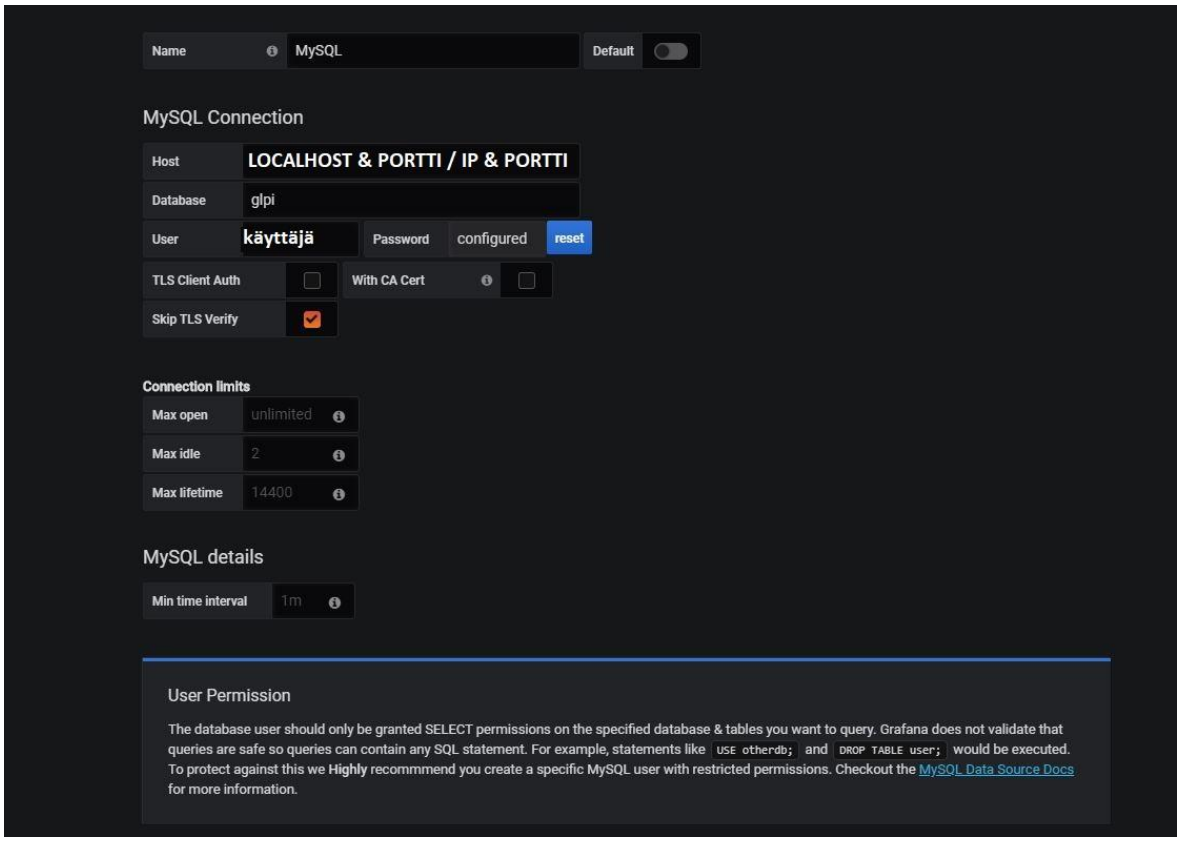
3.2 Projektin vaiheet

Projektin ideana oli toimittaa loppukäyttäjille, eli johtotasolle, toimiva visualisaatioratkaisu joka toimisi kaupungin sisäisessä verkossa. Minulle annettiin melko vapaat kädet päättää teknisestä toteutuksesta. Mitään vaatimuksia ei ollut, joten päätin kuitenkin, koska kyseessä on lopputyöni, että minimivaatimuksena ratkaisu tarjoaisi toimeksiantajalle tavan nähdä haluttu data helposti ymmärrettävinä graafeina.

Työ alkoi testialustan asentamisella. Tätä varten minulle annettiin käyttöön kannettava tietokone. Tähän asennettiin alustaksi Ubuntu, koska minulla oli aikaisempaa kokemusta käytöstä ja halutut ohjelmistot asentuvat ja toimivat hyvin tällä alustalla. Tämä vaihe testattiin kokeilemalla Ubuntua ja terminaalin toimintaa peruskäytössä. Mikään ei ollut muuttunut siitä ajasta kun käytin Ubuntua koulun tunneilla ja kotona eri projekteissa, koulutehtävissä ja muissa asioissa. Täten pystyttiin todentamaan Ubuntun toimivuus ja siirtymään seuraavaan vaiheeseen.

Tämän jälkeen alkoi projektin tiedonhakuvaihe. Tiesin aikaisemmasta kokemuksesta että Grafana toimii haluttujen kaltaisten visualisaatioiden luomiseen. Työn aikana asennus muuttui monta kertaa ja selvitimme mikä asennuksen malli toimisi parhaiten tämän kokoisessa organisaatiossa. Aluksi ohjelmat asennettiin suoraan Ubuntun työkaluilla, mutta toimeksiantajan käyttötarpeiden mukaan haluttiin varmistaa että asennus skaalautuisi ja se olisi helposti siirrettävissä. Tähän lupaava teknologia oli Docker, josta minulla oli myös aikaisempaa kokemusta. Selvitin miten asennuksen saisi automatisoitua konfiguraatiotiedostojen ja Docker-composen avulla.

Toinen aiheeseen liittyvä ongelma oli tietokannan käyttö. Selvitin ensin mikä tietokanta oli kyseessä GLPI:n dokumentaatioista ja konsultoimalla järjestelmän ylläpitäjää. Selvisi että tietokanta oli MySQL ainakin tässä testialustan versioissa, tuotantoversiossa se olisi MariaDB. Lyhyen selvityksen perusteella teknisesti tämän projektin tarpeisiin nämä tietokannat ovat yhteensopivia, joten tästä ei aiheutunut ongelmia. Grafanasta löytyy myös valmiiksi lisäosa kyseisten tietokantojen lukuun.



Kuva 5. Ruudunkaappaus tietokannan yhdistämisestä (Rusanen 2020)

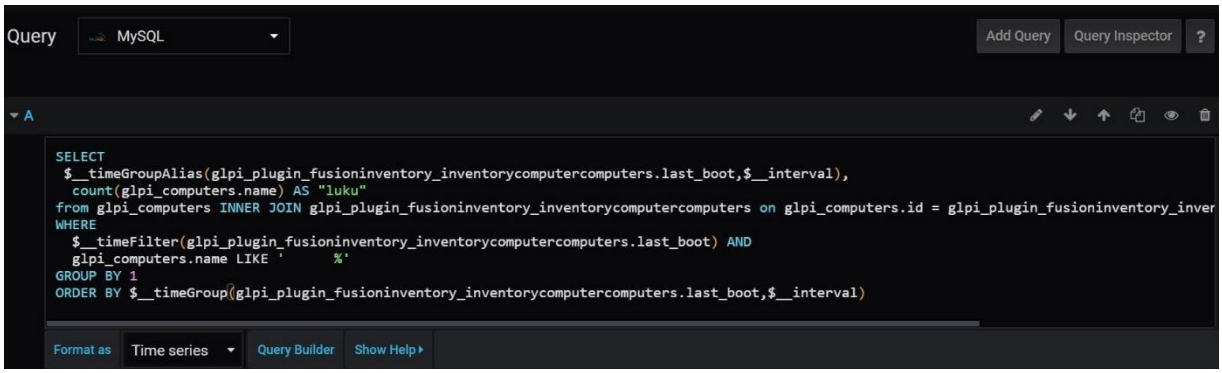
MySQL-tietokannan yhdistämiseen tarvitsee avata jokin portti tietokannalle esim. localhost:9534 tai laittaa toisen tietokoneen IP ja portti. Database kohtaan laitetaan tietokanta, esimerkiksi projektissa käytetty MySQL-dumppi, joka oli samalla palvelimella kuin Grafana. User kohtaan kannattaa tehdä käyttäjätili, joka pystyy vain lukemaan kyseistä tietokantaa eikä muuttamaan sitä ollenkaan.

Ongelmaksi kävi kuitenkin tietokannan laajuus ja käytettävän dokumentaation puute.

```
CREATE TABLE `glpi_plugin_fusioninventory_inventorycomputercomputers` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `computers_id` int(11) NOT NULL DEFAULT '0',
  `operatingsystem_installationdate` datetime DEFAULT NULL,
  `winowner` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
  `wincompany` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
  `last_fusioninventory_update` datetime DEFAULT NULL,
  `remote_addr` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
  `serialized_inventory` longblob,
  `is_entitylocked` tinyint(1) NOT NULL DEFAULT '0',
  `oscomment` text COLLATE utf8_unicode_ci,
  `hostid` varchar(255) COLLATE utf8_unicode_ci DEFAULT NULL,
  `last_boot` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `computers_id` (`computers_id`),
  KEY `last_fusioninventory_update` (`last_fusioninventory_update`)
```

Kuva 6. Ruudunkaappaus yhden taulun kolumneista (Rusanen 2019)

Täten suuri osa työstä oli GLPI:n tietokantataulujen läpikäymistä, jotta kiinnostavan tiedon hakemiseen tarvittavat SQL haut saatiin rakennettua. Opin tässä paljon taulujen yhdistelemisestä, asiatiedon kuten nimeämiskäytäntöjen hyödyntämisestä ja tiedon hakemisesta SQL tietokannoista. Lopputuloksena saatiin aikaan tarvittavat etsintäfunktiot, että projektin tavoitteet voitiin toteuttaa. Lopputulos on myös helposti integroitavissa tuotantoversioon, jos niin päätetään tehdä.



Kuva 7. Ruudunkaappaus MySQL-kyselystä (Rusanen 2019)

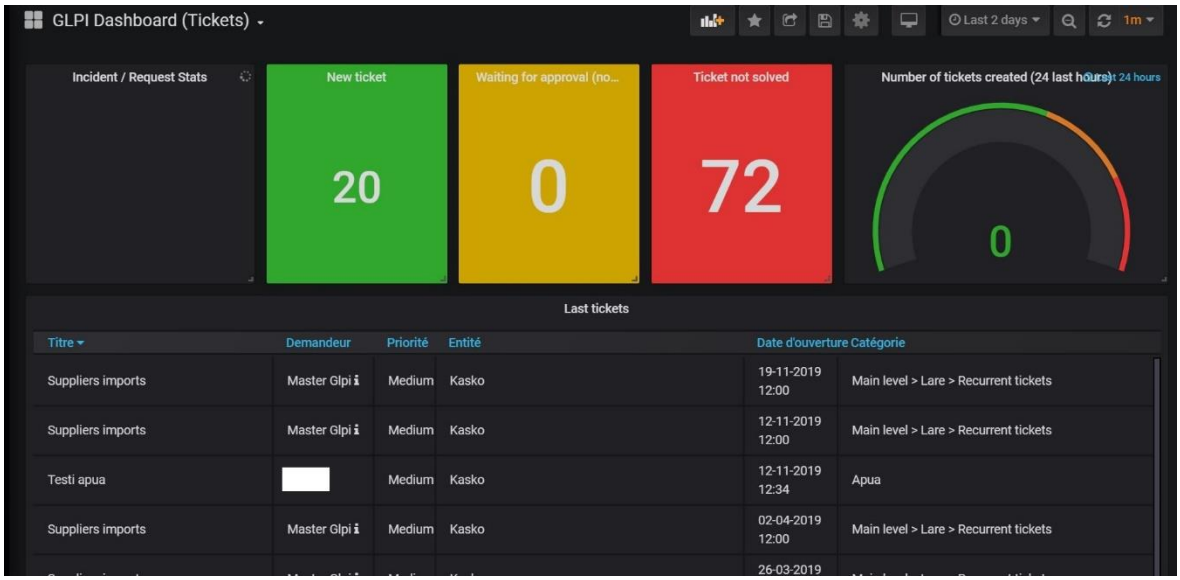
Kuvassa 7 näkyy halutun lopputuloksen MySQL-kysely, jossa on käytetty hyväksi toimeksiantajan käyttämää nimeämiskäytäntöä ja yhdistelty eri taulujen tietoja toisiinsa ”INNER JOIN” funktiolla. Tämä kysely luo halutusta tiedosta loppukäyttäjän haluamia graafeja. Määrittely ”luku” kertoo laitteiden määrän ja se voidaan vaihtaa siihen miksi loppukäyttäjä tahtoo kutsua graafissa esiintyvää tietoa.

Kuva 8. Ruudunkaappaus Laiterekisterin SQL-dumpin teosta (Rusanen 2019)

Laiterekisteri mahdollisti omien SQL ja XML-dumppien tekemisen suoraan palvelimelle, josta oli helpohkoa alkaa käymään läpi tietoja vaikuttamatta laiterekisterin päivittäiseen toimintaan. Tämä ominaisuus voitaisiin esimerkiksi automatisoida tulevaisuudessa projektia ajatellen, esimerkiksi 1-2 kertaa vuorokaudessa.

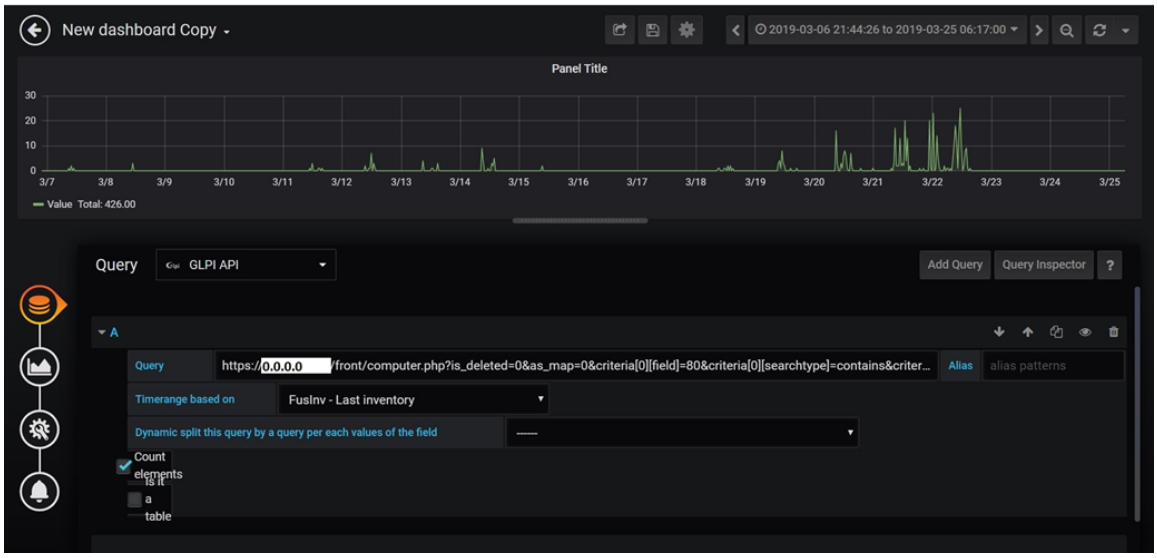
Alun perin tavoitteena oli siirtyä tuotantoon, mutta organisaatiossa päätettiin että systeemiä testattaisiin ensin testipalvelimella, johon vaiheeseen projekti lopulta päättyi omalta osaltani. Tämä tarkoitti asennuksen siirtoa testipalvelimelle, jossa sen käyttäytymistä seurataan ja ominaisuuksia testataan. Testipalvelimella oli staattinen kopio varsinaisesta laiterekisteristä, mutta ominaisuudet ja tiedon määrä vastasivat todellista käyttötapausta. Kokeilimme ensin Grafanaa suoraan dockerin päällä, mutta huomasimme että tämä konfiguraatio ei toimisi, koska sillä ei ole pysyvää tilaa. Toteutimme lopulta pysyvän tilan Docker-composen avulla, säätämällä konfiguratiotiedostoille pysyvän tallennustilan.

Huomasin myös että GLPI:llä on API jota Grafana voi hyödyntää. Tämän lisäosan mukana tuli myös muutama valmiiksi rakennettu näkymä teknisen tuen tikettien visualisointiin. Loppukäyttäjät pitivät tätä ominaisuutta erittäin hyödyllisenä ja tutkimme toiminnallisuutta, vaikka se ei varsinaisesti kuulunut projektin piiriin.



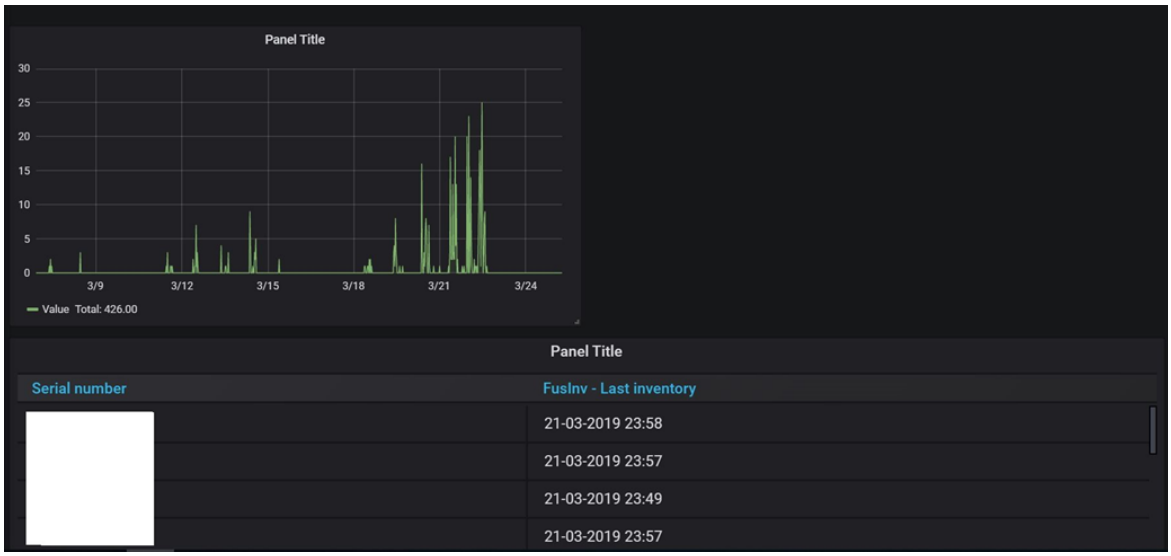
Kuva 9. Ruudunkaappaus GLPI API:n tikettien näkymästä (Rusanen 2019)

Ongelmaksi muodostui lisäosan ajoittainen hitaus. Yksinkertaisetkin haut saattoivat satunnaisesti käyttää tunteja laskenta-aikaa, mikä teki lisäosasta käyttökelvottoman loppukäyttäjille.



Kuva 10. Ruudunkaappaus haun tekemisestä GLPI API:n avulla (Rusanen 2019)

Kuvassa 10 näkyy ”Query” kohta jossa voi muuttaa eri tietueita sitä eteenpäin selaamalla ja esimerkiksi täsmentää mistä koulusta tahdotaan tietoja, mutta tämä on hyvin hankalaa varsinkin, jos ei ole käyttänyt kyseisiä ohjelmistoja aikaisemmin. Tässä voi myös täsmentää aikaskaalan jota käytetään näyttämään tietoja ylläolevassa graafissa. Testasin ”FusInv – Last Inventory” aikatietuetta tässä kyseisessä kuvassa, koska projektin tavoitteena oli saada selville käyttöasteet. FusInv on toimeksiantajan käyttämän agentin lyhenne, joka kerää tietoa laitteista.



Kuva 11. Ruudunkaappaus GLPI API:n tuomista tietokoneista ja niiden agenttien viimeisimmistä yhteyksistä listana (Rusanen 2019)

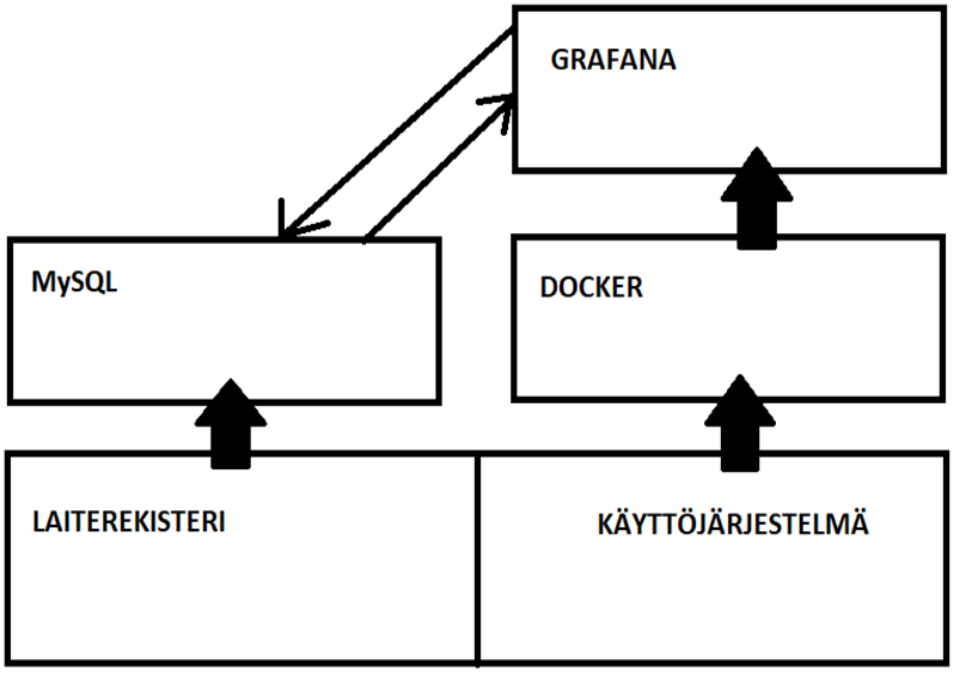
Kuvassa 11 näkyy minkälaisena listana GLPI API pystyi tuomaan tietoa laiterekisteristä. ”FusInv – Last Inventory” osio kertoo milloin viimeksi laitteella ollut agentti on lähettänyt tietoja laiterekisteriin, yleensä se lähettää kerran päivässä jos ei tapahdu mitään ihmeellistä ja muuten kun tietueita vaihdetaan tai konetta käytetään niin useammin.

Kuitenkin tästä lisäosasta opin lisää visualisaatioiden luomisesta Grafanassa. Tärkeimpänä oppina oli ehkä tietojen suodattaminen, jotta visualisaatioista sai selvää, eikä tiedon määrä ollut häiritsevä.

Tässä vaiheessa huomattiin että meillä oli mahdollinen luotettavuusongelma. Tietokanta katoaisi jos Docker kontti kaatuisi. Yhtenä ratkaisuna olisi se että tietokanta laitetaan pyörimään suoraan palvelimelle jossa Docker pyörii. Tällöin kontit pääsevät suoraan tietokantaan ja ei synny tietoturva aukkoa, koska tietokantaa ei tarvitse avata koneen ulkopuolelle. Toisena vaihtoehtona projektin loppupuolella selvitin myös miten järjestelmästä saisi vikasietoisemman ja helpommin ohjattavan. Tähän on olemassa konttiorkestrointityökaluja, kuten Kubernetes. Tällaisessä työkalussa voi määritellä useita instansseja ohjelmistoja ajettavaksi, jolloin yhden kaatuminen ei kaada koko järjestelmää. Järjestelmän eteen voidaan myös rakentaa työnjakajia, jotka parantavat skaalautumista useiden konttien avulla. Tätä ei kuitenkaan toteutettu, vaan jätettiin jatkokehitysideaksi.

Työn lopuksi toteutettiin vielä muutama tietoturvallisuusmuutos, kuten pelkästään tietokannan lukuoikeuden omaavan käyttäjän luominen Grafanaa varten sekä Grafanan sisäinen käyttäjänhallinta organisaatiota varten. Lisäksi luotiin esimerkki dashboard

tietokoneiden käyttöasteesta koulukohtaisesti. Yleisesti Grafanan käyttäjät pystyvät luomaan haluamiaan graafeja itse, jos heille on annettu oikeudet siihen.



Kuva 12. Lopputuloksen arkkitehtuurinen kuva (Rusanen 2020)

Kuvassa 12 näkyy projektin lopputuloksen arkkitehtuuri. Laiterekisterin tiedot päivitettiin MySQL-tietokantaan, joka kommunikoi suoraan grafanan kanssa ja grafana asennettiin docker alustan päälle, jolloin se on helposti siirrettävissä eri käyttöjärjestelmien päälle. MySQL oli asennettu testipalvelimelle ja ei ollut siirrettävissä. Toimeksiantajan tuotannon laiterekisteri on erillisellä palvelimella.

3.3 Lopputulos

Yhteenvetona lopputulokseksi muodostui valmis ohjelmistoratkaisu, joka vastasi toimeksiantajan ongelmaan. Ohjelmistoratkaisussa on vielä tilaa parantaa, mutta se muodosti itsenäisen kokonaisuuden.

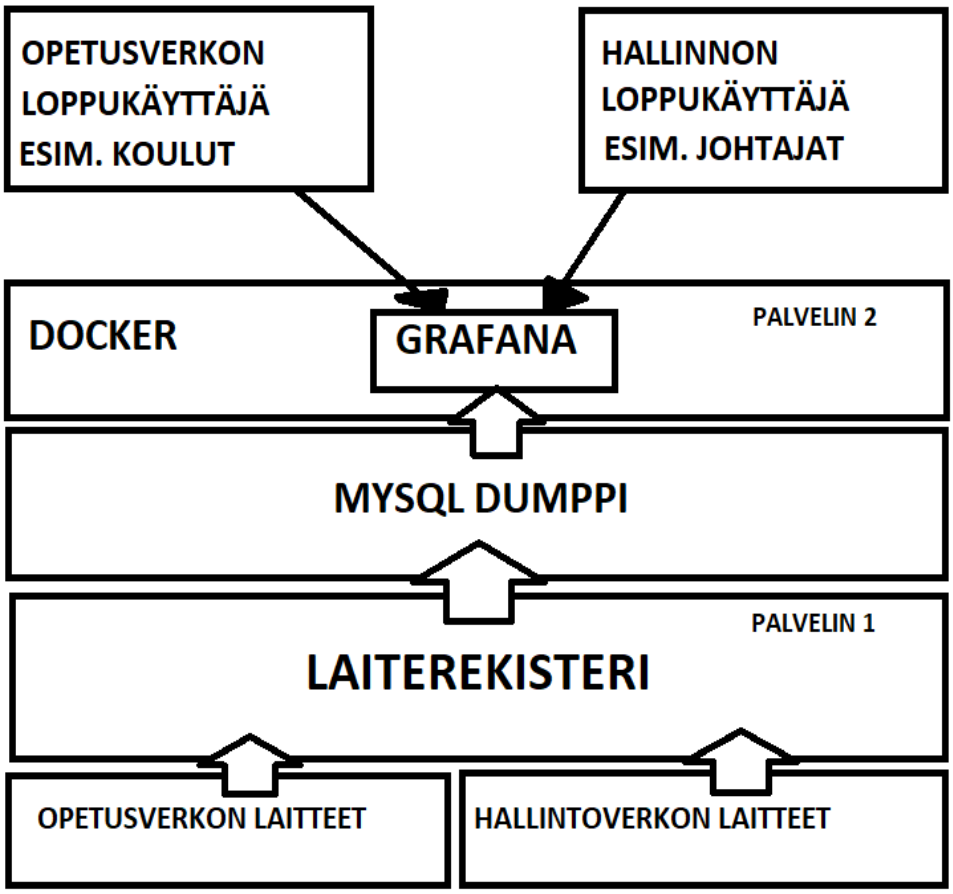
Lopputulos hyväksytettiin toimeksiantajan edustajilla. Dokumentoin työni ja esitin sen ohjaajalle ja toimeksiantajalle. Kirjoitin myös käyttö sekä asennusohjeet projektin jatkoa varten. Toimeksiantajalle toimitettu asennusohjeistus ei kuulu opinnäytetyön julkiseen sisältöön, koska se sisältää operatiivisia yksityiskohtia toimeksiantajan tietojärjestelmistä.

4 Pohdinta

Opin projektin aikana hyvin paljon eri asioita tällaisen mittakaavan projektin toteutuksesta, kuten sen, että kannattaa suunnitella enemmän ja hakea tiedot käyttämistään työkaluista etukäteen. Suunnittelussa kannattaa ottaa myös huomioon muut käyttäjät ja ylläpitäjät mikä jäi hyvin vähäiseksi tässä kyseisessä projektissa. Esimerkiksi en edes ajatellut että joku toinen voisi käyttää testilaiterekisteriä, joten sen kaatuminen saattoi vaikuttaa negatiivisesti muihin käyttäjiin. Suunnittelusta vielä sen verran, että mielestäni olisi hyvä tehdä jonkinlainen ”roadmap” koko projektin ajalle tai seuraavaan ohjauskokoukseen asti ja tarkastella projektin varrella, että mennään suunniteltua polkua pitkin.

Projektin työkalujen testaukseen ja vertailuun lähdin myös katsomatta toisia valmiita ratkaisuja, koska tiesin jo kyseisistä ohjelmista ja oli tiedossani, että ne voisivat olla toimiva ratkaisu laajaan organisaatioon. Tässäkin olisi voinut käydä huonommin, jos ei olisi vaikka jostain syystä toimineet ollenkaan.

Opin dockerin käytöstä sen, että ei kannata tunkea esimerkiksi koko tietokantaa yhteen konttiin, koska jos se kaatuu tai sille käy jotain niin koko tietokanta katoaa. Tämä pointti on myös kaikissa ohjelmistoissa yms. Tein tämän virheen kesken projektin, mutta onneksi vältyimme liiallisilta takaiskuilta. Tähän ratkaisuna on esimerkiksi docker-composen tai orkesterointityökalun kuten kuberneteksen käyttö. Kubernetes olisi jälkiviisaudessa ollut todella hyvä ratkaisu toimeksiantajalle ja kerroin siitä myös loppukokouksessa että kannattaa jatkokehittää siihen suuntaan, jos he laittavat ratkaisun lopulta tuotantoon. Mielestäni projektin rajaus onnistui hyvin, se ei ollut liian laaja eikä liian suppea. Projektin aikainen toimeksiantajan kanssakäyminen sujui myös mutkitta. Ottaen huomioon että tämä oli koulua varten tehty projekti, olisin voinut pitää ohjaajaa ja toimeksiantajaa enemmän mukana asioissa, mutta sain sellaisen työaallon päälle, että tein paljon itsenäisesti ajattelematta muuta maailmaa.



Kuva 13. Lopullinen toteutuskaavio mielikuva (Rusanen 2020)

Kuvassa 13 on kuvattu millainen ratkaisuni olisi tuotannossa. Siinä on kuvattu Helsingin kaupungin käyttämien verkkojen laitteiden informaation tuleminen laiterekisteriin, joka sijaitsee palvelimella yksi. Tästä tiedosta tehtäisiin kerran tai kahdesti päivässä MySQL-dumppi, jota luettaisiin Grafanan toimesta. Grafana sijaitsee palvelimella 2 ja sen pohjana toimii docker. Tähän grafanaan pääsisi käsiksi kaikki loppukäyttäjät molemmista verkoista, esimerkiksi koulun johtajat voisivat katsoa graafeja päivän käyttöasteista tai etsiä jos jotain laitteita on kadoksissa tai poistettu käytöstä. Kun taas hallintoverkossa työskentelevä johto näkee halutun lopputuloksen, eli käyttöasteet. Loppukäyttäjillä olisi oikeudet tehdä omia näkymiä grafanaan ja katsoa tuloksia, mutta ei mitään poisto- tai ylläpito-oikeuksia millä voisi saada järjestelmää nurin.

Uskon täysin että projektista on hyötyä toimeksiantajalle, jos he päättävät ottaa siitä kaiken irti.

Lähteet

Ben-Kiki, O. & Evans, C. 2009. YAML Ain't Markup Language (YAML™) Version 1.2. Luettavissa: https://yaml.org/spec/1.2/spec.html#id2759572. Luettu 26.4.2020.

Ahonen, T. 2014. Mikä ihmeen PaaS? Luettavissa: https://www.cybercom.com/fi/Suomi/Yritys/Blogit/Blogit/Mika-ihmeen-PaaS/. Luettu 26.4.2020.

Bui, T. 2015. Analysis of Docker Security. Luettavissa: https://arxiv.org/pdf/1501.02967.pdf. Luettu 25.3.2020.

Matthias, K. & Kane, S. P. 2015. Docker: Up & Running: Shipping Reliable Containers in Production. Luettavissa: https://books.google.fi/books?id=IDvcCQAAQBAJ&lpg=PP1&ots=3BKBO-9BHw. Luettu 25.3.2020.

Vase, T. 2015. Advantages of Docker. Luettavissa: https://jyx.jyu.fi/handle/123456789/48029. Luettu 25.3.2020.

Dua, R., Kohli, V. & Konduri, S. K. 2016. Learning Docker Networking. Luettavissa: https://books.google.fi/books?id=Dm1LDAAAQBAJ&lpg=PP1&ots=6cY-biL8OPc. Luettu 25.3.2020

Combe, T. & Martin, A. & Pietro, R. Di. 2016. To Docker or Not to Docker: A Security Perspective. Artikkeli IEEE Cloud Computing, versio 3, numero 5, sivut 54-62. Julkaistu Syys-Lokakuussa 2016. Luettavissa: https://www.researchgate.net/profile/Roberto_Pietro/publication/309965523_To_Docker_or_Not_to_Docker_A_Security_Perspective/links/5bd2f7c1a6fdcc3a8da6c537/To-Docker-or-Not-to-Docker-A-Security-Perspective.pdf. Luettu 26.3.2020.

Chelladhurai, J. S., Singh, V. & Raj, P. 2017. Learning Docker Second Edition. Luettavissa: https://books.google.fi/books?id=qHc5DwAAQBAJ&lpg=PP1&ots=3XpcMGdqAq. Luettu 25.3.2020

Rubens, P. 2017. What are containers and why do you need them? Luettavissa: https://www.cio.com/article/2924995/what-are-containers-and-why-do-you-need-them.html. Luettu 16.5.2020.

Smith, R. 2017. Docker Orchestration. Luettavissa: https://books.google.fi/books?id=4U8oDwAAQBAJ&lpg=PP1&ots=rRZpU6J80W. Luettu 25.3.2020.

Verneri, P. 2017. Mitä tarkoittaa SaaS?. Luettavissa: https://www.inderes.fi/fi/mita-tarkoittaa-saas. Luettu 26.4.2020.

Rusanen, A. 2018. docker-compose.yml. Luettavissa: https://github.com/AxelRusanen/grafanadocker/blob/master/srv/salt/docker-compose.yml. Luettu 29.3.2020.

Haglund, J. 2018. API:t käytännössä - selkokielinen katsaus. Luettavissa: https://www.alfame.com/blog/apit-kaytannossa-selkokielinen-katsaus. Luettu 26.4.2020.

Docker. 2019a. Docker Compose. Luettavissa: https://docs.docker.com/compose/. Luettu 30.3.2020.

Docker. 2019b. Docker Overview. Luettavissa: https://docs.docker.com/get-started/overview/. Luettu 30.3.2020

Grafana Labs. 2020. What is Grafana? Luettavissa: https://grafana.com/docs/grafana/latest/guides/what-is-grafana/. Luettu 30.3.2020

Microsoft. 2020. What is SaaS? Luettavissa:https://azure.microsoft.com/en-us/overview/what-is-saas/. Luettu 15.5.2020