Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Jonnie Käyhty

Lokienhallinnan täydentäminen lokienhakupalvelimella

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

13.11.2019

Tekijä Otsikko Sivumäärä Aika	Jonnie Käyhty Lokienhallinnan täydentäminen lokienhakupalvelimella 25 sivua 13.11.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Simo Silander Lauri Salmio

Insinöörityön tavoitteena oli luoda lokienhallintajärjestelmän ohelle prototyyppipalvelin, joka pystyy hakemaan lokeja verkkolaitteilta. Lokien haun lisäksi palvelimen on muokattava lokit sisältämään tarvittavia verkkolaitteen tietoja ennen niiden lähettämistä lokienhallintajärjestelmään. Prototyypin tavoitteena on selvittää palvelimeen liittyviä ongelmia ja dokumentoida mahdollisia ratkaisuja, joiden avulla voitaisiin kehittää tuotantoversio.

Työ sisälsi HTTP-rajapinnan ohjelmointia Python-kielellä ja lokienhallintaa Rsyslog-ohjelmiston avulla. Työ kehitettiin Linux-käyttöjärjestelmällä, joka vaati prosessien ja tiedostojen käyttöoikeuksien asettamista. Työn haastavin osa oli lokienhakualgoritmin kehittäminen, joka ratkaisi ison ongelman prototyypissä parantamalla lokien eheyttä huomattavasti.

Työn lopputulos oli toimiva palvelin, joka haki lokeja verkkolaitteilta, muokkasi ne oikeaan muotoon ja lähetti ne lokienhallintajärjestelmään. Prototyyppi todettiin toimivaksi manuaalisten integraatiotestien avulla kahden eri verkkolaitteen kanssa.

Avainsanat	Lokienhallinta, Python-rajapinta, Rsyslog

Author	Jonnie Käyhty
Title	Supplementing Log Management with a Log Retrieval Server
Number of Pages	25 pages
Date	13 Novermber 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Software Development
Instructors	Simo Silander, Senior Lecturer Lauri Salmio

The goal of the thesis was to develop a prototype server that could retrieve logs from network devices to supplement an existing log management system. In addition to retrieving logs the server would have to modify the logs to contain necessary information about the network device before sending them back to the log management system. The goal of the prototype was to solve problems related to the server and document the possible solutions. The thesis could be used as a reference to develop the production version of the server.

The study included the development of an HTTP server in Python and log management with Rsyslog. The software was implemented on a Linux operating system, so it also included configuring permissions of the processes and files. The most challenging aspect of the development was the log retrieval algorithm that solved a big problem for the prototype by increasing log integrity dramatically.

The result of the study was a working server that could retrieve logs from network devices and modify them to the correct format before sending them to the log management system. The prototype was found to be working by running manual integration tests with two different network devices.

Keywords	Log management, python API, Rsyslog

Sisällys

Lyhenteet

USB	Universal Serial Bus. Tietokoneoheislaitteiden liitinstandardi.
WSGI	Web Server Gateway Interface. Python-standardi web-palvelimien kutsujen käsittelylle.
HTTP	Hypertext Transfer Protocol. WWW-palvelimien ja selaimien käyttämä tiedonsiirtoprotokolla.
IP	Internet Protocol. Internet-verkon tietoliikennepakettien toimintaprotokolla.
UDP	User Datagram Protocol. Yksinkertainen tiedonsiirtoprotokolla ilman viestin vastaanottovarmistusta.
TCP	Transmission Control Protocol. Luotettava tiedonsiirtoprotokolla, joka varmistaa pakettien saapumisjärjestyksen.
SFTP	Secure File Transfer Protocol. Salattu tiedostojensiirtoprotokolla.
cURL	Client URL. Komentoriviohjelma HTTP-kutsujen lähettämiseen.

1 Johdanto

Insinöörityön tavoitteena on kehittää prototyyppi jo olemassa olevan ohjelmiston ja lokienhallintajärjestelmän ohelle. Prototyypin avulla on tarkoitus kerätä ja lähettää edelleen lokeja verkkolaitteilta, jotka eivät pysty niitä itse lähettämään oikeassa muodossa oikeaan paikkaan. Ajatuksena on, että insinöörityön tuloksia ja prototyyppiä voidaan käyttää hyväksi tuotantoversion kehittämisessä. Prototyyppiä kutsutaan tässä työssä Satelliitiksi.

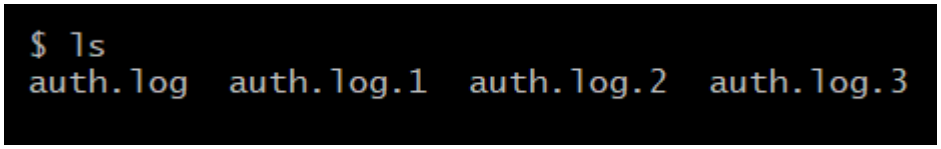
Insinöörityön tilaaja on Ericsson, joka on vuonna 1876 perustettu telekommunikaatioyritys [1]. Ericsson on toiminut Suomessa vuodesta 1918 lähtien. Ericsson siirtyi 70-luvulla Kirkkonummen Jorvakseen, jossa nykyäänkin tehdään tutkimus- ja tuotekehitystyötä.

Työ on tarkoitus ohjelmoida pääosin Python-ohjelmointikielellä, koska sitä on käytetty muissa projektiin liittyvissä palvelimissa. Työn tilaaja on valinnut työn kehitysympäristöksi virtuaalikoneen Red Hat Enterprise Linux-käyttöjärjestelmällä, joka on suosituin palvelinkäyttöjärjestelmä yritysmaailmassa [2].

2 Lokienhallinta

Lokienhallinta on konetuotetun lokidatan käsittelyä, tallennusta ja analysointia. Lokienhallinnan ensisijainen päämäärä on järjestelmien tietoturvan ja laitteiden toimintaan liittyvien ongelmien arkistointi. Lokien tarkastelu jälkeenpäin voi auttaa tietoturvamurtojen tai vikatilanteiden ratkaisemisessa. Niiden avulla voidaan myös analysoida trendejä tai pitkäaikaisia toistuvia ongelmia. Yritykset voivat toiminnastaan riippuen olla lain mukaan velvollisia tallentamaan tietynlaisia lokitietoja [3, s. 21]. Lokitietoja tallennettaessa on myös otettava huomioon sitä rajoittavia lainsäädäntöjä. Yksi lokitietoihin vaikuttava lainsäädäntö on GDPR eli EU:n yleinen tietosuoja-asetus, joka pyrkii yhtenäistämään Euroopan unionin jäsenmaiden tietosuojaa koskevaa lainsäädäntöä. GDPR-rikkeestä voidaan sakottaa jopa 20 miljoonaa euroa tai 4 % vuosittaisesta liikevaihdosta. Henkilötietoja sisältävä lokidata on käsiteltävä GDPR-asetusten mukaan. Henkilötietoja sisältävät lokit voidaan myös joko jättää tallentamatta tai anonymisoida, jolloin GDPR-asetuksista ei tarvitse huolehtia [4].

Lokitiedosto on yksinkertainen tekstitiedosto, jossa lokit, eli jotakin tapahtumaa kuvailevat tekstinpätkät, on eritelty riveittäin. Tiedosto on järjestetty nousevassa järjestyksessä ajan mukaan, eli uudet lokit kirjoitetaan tiedoston loppupäähän. Lokitiedostoilla ei käytännössä ole mitään tiedostokokorajaa, mutta liian suuret tiedostot voivat olla ympäristöstä riippuen ongelmallisia. Esimerkiksi ison tiedoston lukeminen tai siihen kirjoittaminen vaatii tietynlaisia lähestymistapoja. Lokitiedosto voi myös vikatilanteessa kasvaa yllättävän nopeasti niin suureksi, että järjestelmän levytila loppuu, josta seuraa lisää ongelmia. Lokien arkistointi säännöllisesti on järjestelmän hyvinvoinnin kannalta oleellista. Yksi arkistointistrategia on niin sanottu lokien kierrätys, joka onnistuu Logrotate-nimisellä ohjelmalla. Logrotate-ohjelma ajetaan yleensä automatisoidusti kerran päivässä. Logrotate arkistoi lokitiedoston, jos konfiguraatiossa määritelty tiedostokokoraja on ylittynyt. Konfiguraatiossa määritellään myös, kuinka monta arkistotiedostoa tallennetaan. Arkistotiedostot nimetään <arkisto>.1, <arkisto>.2 jne. Uusin arkistotiedosto nimetään pienimmällä järjestysluvulla, ja uuden arkiston syntyessä vanha nimetään uudelleen toiseksi uusimmaksi. Uudelleennimeämistä jatketaan, kunnes arkistotiedosto ylittää annetun rajan, jolloin se poistetaan. Kuvassa 1 on esimerkki lokitiedostosta, jota kierrätetään kolme kertaa, jolloin tiedostopolussa on nykyisen lokitiedoston lisäksi kolme vanhaa lokitiedostoa, joista vanhin on ”.3”-loppuinen.



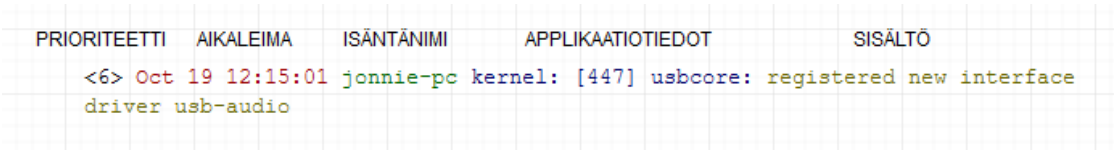
Kuva 1. Esimerkki kolmesti kierrätetystä lokista

Seuraavalla lokikierrätyksellä kuvan nykyinen ”auth.log.3”-tiedosto poistetaan. Lokienkierrätyksen vuoksi tämä lokitiedosto ei tule kasvamaan ajan myötä suuremmaksi ja suuremmaksi. Logrotate-ohjelmaa konfiguroitaessa on otettava huomioon odotettu lokitiedoston aktiviteetti ja ongelman reagointiin vaativa aika. Ongelmatilannetta selvittäessä olisi ikävää, jos relevantit lokit olisi jo poistettu.

2.1 Syslog

Loki-standardeja on useita erilaisia eri käyttötapauksia varten, mutta yksi yleisimmistä standardeista, erityisesti unix-käyttöjärjestelmissä ja verkkolaitteissa on Syslog. Syslog

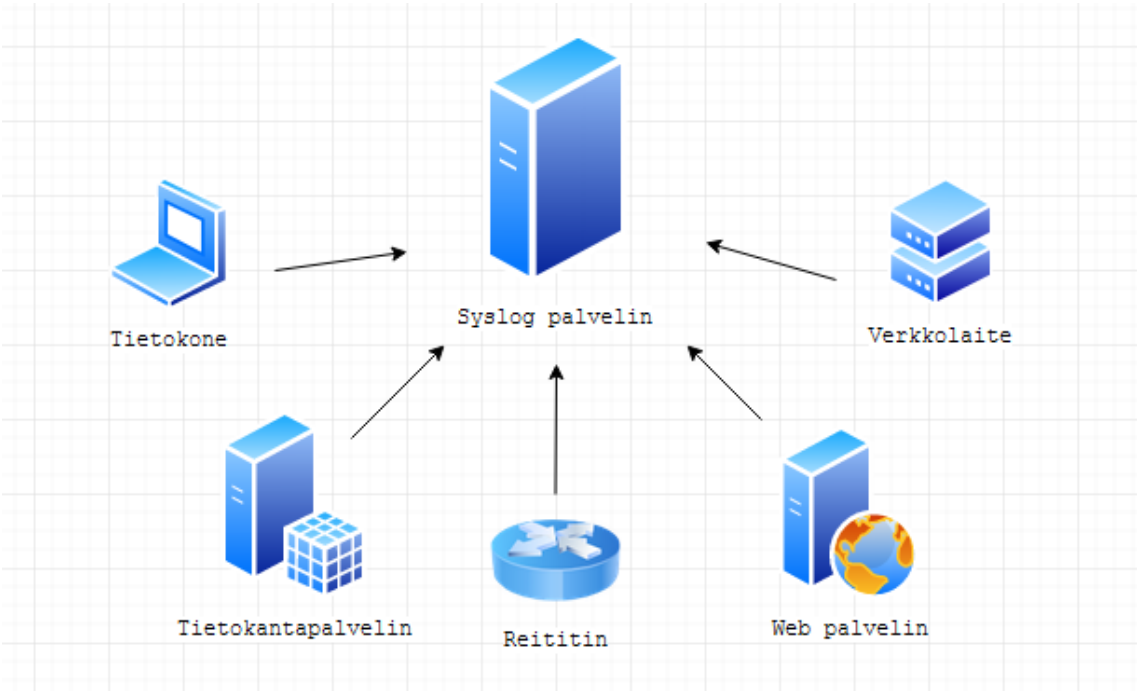
on kehitetty vuonna 1980 Eric Allmanin toimesta ja myöhemmin useaan kertaan standardisoitu, koska eri Syslog-implementaatiot olivat usein yhteensopimattomia. Viimeisin standardi on RFC 5424 vuodelta 2009, jota useimmat Syslog-implementaatiot seuraavat. Syslog-lokiin kuuluu lokiviestin eli sisällön lisäksi muutama datatietue, joista tärkeimmät ovat prioriteetti, aikaleima, isäntänimi ja sovellustiedot. Prioriteetti on kokonaisluku, joka lasketaan lokin osa-alueen ja vakavuuden perusteella. Aikaleima on käyttöjärjestelmän aika lokin kirjoitushetkellä, ja se voi olla kirjoitettu useissa eri muodoissa. Isäntänimi on usein laitteen nimi tai IP-osoite. Sovellustiedot sisältävät yleensä sovelluksen nimen ja prosessin. Suurin osa Syslog-lokin datatietueista on vapaaehtoisia, josta johtuen Syslog-lokit voivat näyttää hyvin erilaisilta olosuhteista riippuen [7]. Kuvassa 2 on esimerkki yksinkertaisesta Syslog-lokista, josta ilmenee, että tietokoneeseen nimeltä ”jonnie-pc” on liitetty USB-laite lokakuun 19. päivä. Prioriteetti kuusi tarkoittaa kernel-alueen infolokia.



Kuva 2. Tyypillinen Syslog-lokin rakenne

Lokitiedostoja on yleensä järjestelmässä useita, ja ne on eritelty sisällön mukaan. Jotkut lokit ovat laajempia koko järjestelmän kattavia lokeja, ja toiset ovat hyvin täsmällisiä tietyn osa-alueen tai sovelluksen lokeja.

Laitteet ja tietokoneet kirjoittavat lokitiedostoja omaan tiedostojärjestelmäänsä, mutta ne voivat myös lähettää lokit verkon yli Syslog-standardin mukaisesti. Syslog-lokin lähetys verkon yli toimii asiakas-palvelinarkkitehtuurilla, jota on visualisoitu kuvassa 3.



Kuva 3. Syslog-palvelin, johon eri asiakkaat lähettävät lokeja.

Syslog-palvelin kuuntelee tunnettua porttia 514, johon asiakas lähettää lokeja UDP-protokollaa käyttäen [8, s. 3]. UDP-protokolla on tähän käyttötapaukseen kuitenkin huono, koska se ei takaa viestien vastaanottoa, ja se on altis ruuhkautumiselle. Suurin osa Syslog-implementaatioista tukee UDP:n lisäksi salattua TCP-yhteyttä portilla 6514. Lokien lähetys verkon yli on oleellinen osa lokienhallintaa, jotta koko verkon kattavat lokit saadaan kerättyä yhteen paikkaan, jota kutsutaan yleensä lokienhallintajärjestelmäksi.

2.2 Lokienhallintajärjestelmät

Lokienhallintajärjestelmä on keskitetty ohjelmisto, joka yleensä kerää lokeja useista eri lähteistä pitkäkestoista varastointia varten. Lokeja pystyy etsimään ja suodattamaan eri parametrein, jotta ison lokimäärän seasta voidaan löytää tarvittava informaatio. Usein lokeja analysoidaan myös ohjelmallisesti, jolloin näiden analyysien tulokset ovat näkyvissä lokienhallintajärjestelmässä. Kaupallisia ja avoimen lähdekoodin lokienhallintajärjestelmiä on hyvin monia erilaisia, sillä ne ovat monille ohjelmistoprojekteille välttämättömiä. Lokienhallintajärjestelmät ovat myös sen verran monimutkainen kokonaisuus, että hyvin harvan ohjelmistoprojektin on taloudellisesti kannattavaa kehittää kokonainen järjestelmä itse.

Suosittuja lokienhallintajärjestelmiä ovat muun muassa Splunk, Logstash, ja SolarWinds. Logstash on Elastic yrityksen kehittämä lokienhallintajärjestelmä, joka on hyvin konfiguroitava yli 200 lisäosan ja rajapinnan avulla. Se jäsentelee ja muokkaa dataa monista eri lähteistä automaattisesti haluttuun muotoon, esimerkiksi lisäämällä maantieteellisen sijainnin IP-osoitteen perustella. Logstash integroituu Elasticin toisen tuotteen ElasticSearchin kanssa, joka voi tallentaa ison määrän dataa myöhempää analysointia ja hakua varten [9]. Splunk ja SolarWinds toimivat hyvin pitkälti samojen periaatteiden mukaisesti.

3 Työn vaatimukset

Lokienhallintajärjestelmä, jonka ohelle tätä insinöörityötä tehdään, toimii nykyisellään niin, että iso osa ohjelmiston kattavista verkkolaitteista lähettää jatkuvasti lokejansa tallennettavaksi lokienhallintajärjestelmään. Näitä lokeja voi jälkeenpäin etsiä erinäisillä hakusanoilla ja suodattimilla. Ohjelmisto analysoi myös lokeja määriteltyjen sääntöjen mukaisesti ja tekee ilmoituksen normaalista poikkeavien lokien tapauksessa. Ohjelmiston nykyinen tila on lokienhallinnan kannalta toimiva, mutta ei täysin kattava. Osa verkkolaitteista ei ole tarpeeksi joustavia, jotta ne pystyisivät lähettämään lokinsa lokienhallintajärjestelmään. On kuitenkin lokienhallintajärjestelmän periaatteen kannalta oleellista, että lokit kaikilta verkkolaitteilta pystytään keräämään. Olennaista on myös se, että lokeista ilmenee kaikki tarvittava tieto. Sulautetut verkkolaitejärjestelmät eivät täytä lokien osalta Syslog RFC -vaatimuksia, tai täyttävät ne näennäisesti, esimerkiksi käyttämällä järjestelmän sisäverkon nimiä tai osoitteita lokikentissä.

Tällä insinöörityöllä on ohjelmiston toiminnan kannalta kolme kriittistä vaatimusta. Ensimmäinen vaatimus on se, että Satelliitti pystyy hakemaan geneeriseltä verkkolaitteelta lokitiedostoja SFTP-protokollaa käyttäen. SFTP-protokolla mahdollistaa tiedostojen siirtoa ja kopiointia salatun verkkoyhteyden yli kahden tietokoneen välillä. Toinen vaatimus on lokitietojen rikastus, eli ohjelmiston on lisättävä metadataa lokiin verkkolaitteen tunnistamista varten. Ilman rikastusta lokienhallintajärjestelmä ei pysty erottamaan lokeja eri verkkolaitteilta. Käytännössä tämä tarkoittaa vähintään verkkolaitteen IP-osoitteen lisäämistä lokiin. Kolmas vaatimus on lokien edelleen lähettäminen lokienhallintajärjestelmään. Kriittisten vaatimusten lisäksi taulukossa 1 on listattu ohjelmiston muita vaati-

muksia. Yksi vaatimus on HTTP-rajapinnan implementointi, jonka kautta Satelliitin toimintaa ohjataan. Muihin vaatimuksiin kuuluvat muun muassa lokien eheys ja levytilan puhdistus.

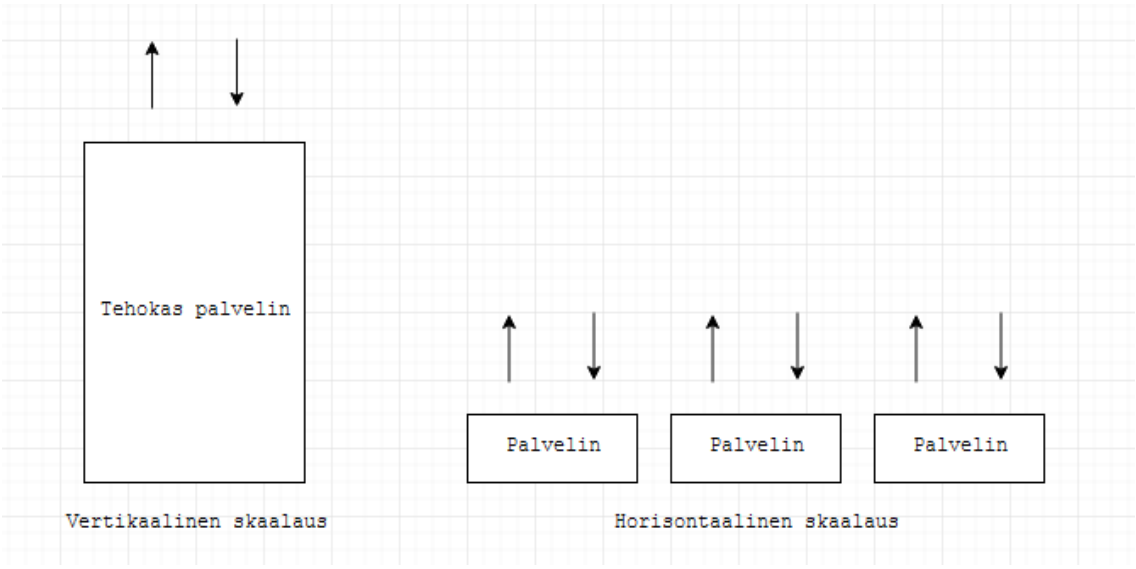
Taulukko 1. Insinöörityön vaatimukset

Vaatimus	**Toiminnallinen**	**Kuvaus**	**Onnistumiskriteerit**	**Prioriteetti**
Lokitiedostojen haku verkkolaitteelta	Kyllä	Satelliitin on pystyttävä hakemaan lokitiedostoja verkkolaitteelta SFTP-protokollan avulla	Satelliitti pystyy hakemaan tiedostoja.	Hyvin suuri. Tämä on kriittinen vaatimus.
Lokien rikastus	Kyllä	Satelliitti pystyy lisäämään lokeihin metadataa verkkolaitteesta.	Lokit sisältävät lisätietoja ennen niiden lähetystä lokienhallintajärjestelmään.	Hyvin suuri. Tämä on kriittinen vaatimus.
Lokien edelleen lähetys	Kyllä	Satelliitti lähettää lokit lokienhallintajärjestelmään.	Lokit lähetetään edelleen.	Hyvin suuri. Tämä on kriittinen vaatimus.
HTTP-rajapinta	Kyllä	Satelliitin ja pääohjelmiston välinen kommunikaatio käydään HTTP-rajapinnan kautta.	Toimiva rajapinta, joka hoitaa virhetilanteet	Hyvin suuri. Ohjelmisto ei voi toimia itsenäisesti ilman rajapintaa
Geneerinen	Ei	Toimii useiden verkkolaitteiden ja ympäristöjen kanssa	Toimii kaikkien testi verkkolaitteiden kanssa	Pieni. Prototyypin ei tarvitse olla täydellisen geneerinen
Levytilan puhdistus	Ei	Lokitiedostot eivät vie ajan mittaan enemmän ja enemmän levytilaa	Lokitiedostot eivät kasva tietyn tiedostokoon yli	Suuri. Suuret tiedostot hidastavat ohjelmistoa ja vaativat levytilan kasvattamista
Lokien eheys	kyllä	Lokienhallintajärjestelmään tulee lähettää edelleen jokainen loki vain ja ainoastaan kerran. Lokeja ei saa jättää lähettämättä eteenpäin.	Ohjelmisto lähettää kaikki lokit lokienkierrätyksen rajatapauksissa sekä levytilan puhdistuksen yhteydessä	Hyvin suuri. On erittäin tärkeää, että lokeja ei katoa.

4 Arkkitehtuuri

Ohjelmistoarkkitehtuuri koostuu kolmesta osasta: verkkolaitteista, tässä insinöörityössä toteutetusta Satelliitista ja pääohjelmistosta, joka sisältää lokienhallintajärjestelmän. Satelliitti implementoidaan työn tilaajan toiveen mukaan irrallisena palvelimena muusta ohjelmistosta niin sanotun mikropalveluarkkitehtuurimallin mukaisesti, joka on monoliittisen arkkitehtuurimallin vastakohta. Mikropalveluarkkitehtuurissa palvelut jaetaan modulaarisiin osiin, ja ne keskittyvät vain oman tehtävänsä suorittamiseen. Palveluiden välinen kommunikaatio käydään prosessienvälisien protokollien kuten HTTP:n tai TCP:n avulla.

Erillisen palvelimen hyödyt tulevat ilmi lähinnä ohjelmiston käyttöönoton ja ylläpidon yhteydessä, mutta yksi oleellinen hyöty on niin sanottu horisontaalinen skaalaus. Se tarkoittaa sitä, että skaalautuvuuden saavuttamiseksi otetaan käyttöön useampi kopio palvelimesta sen sijaan, että kasvatettaisiin yhden palvelimen suorituskykyä. Horisontaalisen ja vertikaalisen skaalauksen ero on visualisoitu kuvassa 4.



Kuva 4. Horisontaalinen ja vertikaalinen palvelimien skaalaus

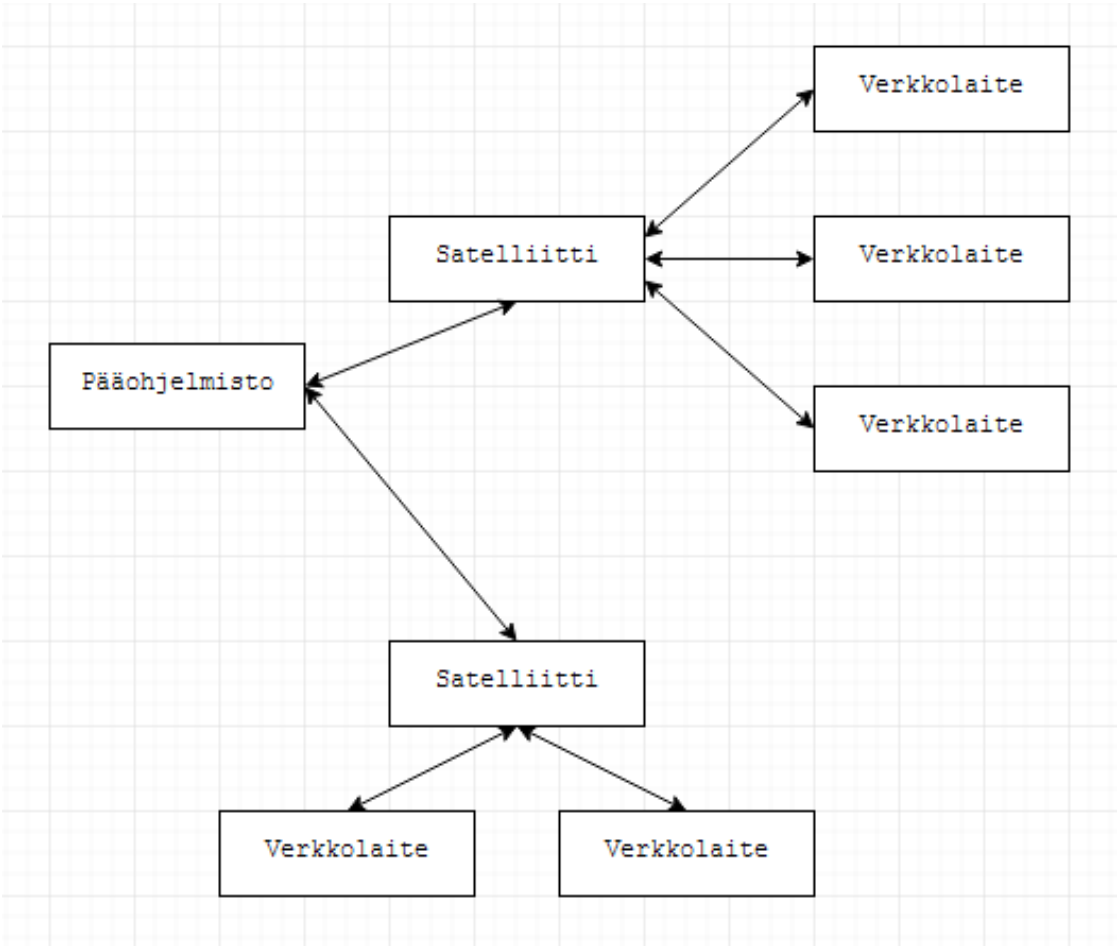
Horisontaalinen skaalaus toimii erinomaisesti tässä tapauksessa, sillä Satelliittien kopiot ovat toisistaan riippumattomia. Horisontaalinen skaalaus on hankalampi toteuttaa, jos kopioiden on jaettava jotain resurssia, kuten esimerkiksi tietokantaa.

Pääohjelmisto toimii koko järjestelmän ytimenä. Se mallintaa koko verkon rakennetta ja näin ollen ohjaa, miltä satelliitilta haetaan minkäkin verkkolaitteen lokeja. Pääohjelmisto kertoo satelliitille yhteystiedot sekä verkkolaitteelle, josta lokia haetaan, että lokienhallintajärjestelmään, jonne lokit lähetetään edelleen. Satelliitti tallentaa lokienhallintajärjestelmän yhteystiedot konfiguraatioonsa. Verkkolaitteen yhteystietoja ei kuitenkaan tallenneta satelliitille tietoturvasyistä, koska niiden avulla pääsee kirjautumaan sisään verkkolaitteeseen. Näin ollen satelliitti ei pysty itsenäisesti yhdistämään verkkolaitteeseen, joten pääohjelmiston on pyydettävä jokaisen verkkolaitteen lokeja tasaisin väliajoin.

Satelliitin tehtävä on vastaanottaa lokienhakupyyntöjä, ja niiden pohjalta hakea lokeja halututlta verkkolaitteelta. Satelliitin on myös lisättävä lokeihin verkkolaitteen tunnistamista varten metadataa. Lopuksi satelliitin on lähetettävä lokit lokienhallintajärjestelmään.

Verkkolaitteita on useita erilaisia. Niiden konfiguraatioita ei pysty muuttamaan eikä niille pysty asentamaan mitään ohjelmistoja. Työn tilaajan mukaan voidaan kuitenkin olettaa, että verkkolaitteelta voi aina hakea lokitiedostoja SFTP-protokollaa käyttäen. Lokit haetaan verkkolaitteen varmennetusta hallintajärjestelmästä. Varmennettu osa koostuu kahdesta sulautetusta Linux-tietokoneesta, jotka molemmat sisältävät kaikki järjestelmän sisäiset lokit.

Kuvassa 5 nähdään esimerkki pääohjelmiston, satelliittien ja verkkolaitteiden arkkitehtuurista.



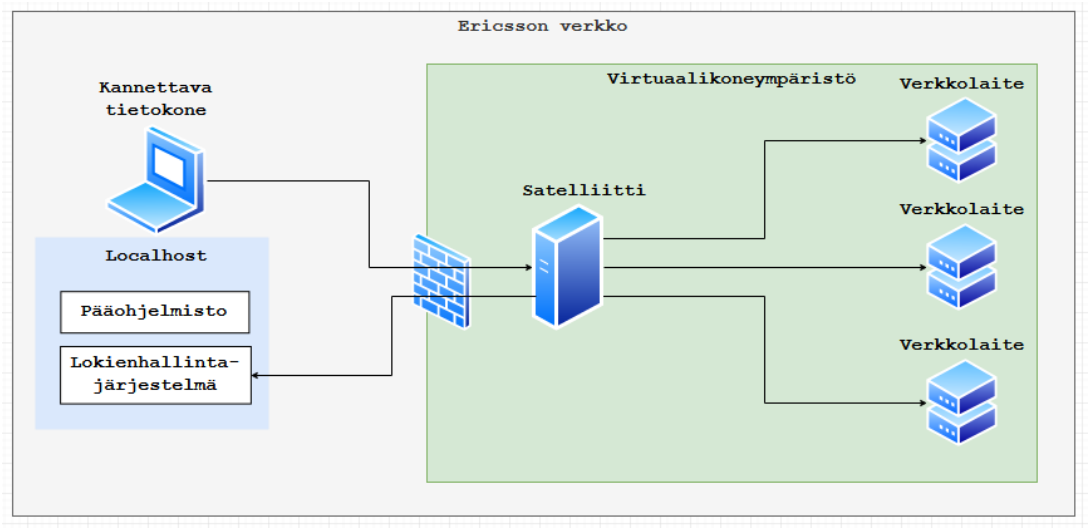
Kuva 5. Esimerkkijärjestelmä arkkitehtuurista kahdella satelliitilla ja viidellä verkkolaitteella.

5 Toteutus

Satelliitin tulee toimia yksinkertaisesti selitettynä niin, että Python HTTP-rajapinnan vastuulla on vastaanottaa lokienhakupyyntöjä, hakea lokeja verkkolaitteelta ja lisätä lokit Satelliitilla olevaan lokitiedostoon. Rsyslog, joka on lokienkäsittelyohjelma, pystyy lukemaan näitä lokitiedostoja, käsittelemään ne ja lähettämään ne edelleen lokienhallintajärjestelmään.

Ennen itse ohjelmiston implementointia luotiin testiympäristö. Lokienhallintajärjestelmää suoritettiin kannettavalla tietokoneella paikallisesti. Satelliittina toimi Ericssonin virtuaali-

konepalvelin, jonne asenettiin Red Hat Enterprise Linux-käyttöjärjestelmä. Tähän virtuaalikoneeseen otettiin etäyhteys kannettavan tietokoneen kanssa SSH-yhteyden yli. Virtuaalikonella oli suoraan verkkoyhteys jo olemassa oleville testiverkkolaitteille. Kuvassa 6 on hahmoteltu testiympäristön verkkoasetelmaa.



Kuva 6. Testiympäristö

Käyttöjärjestelmässä oli asennettuna Python 2.7 oletuksena, mutta halusin ohjelmoida rajapinnan Python 3:lla. Python 3.6 -versiota voi käyttää Red Hat -käyttöjärjestelmässä ajamalla koodiesimerkin 1 sisältämän komennon.

```
scl enable rh-python36 bash
```

Koodiesimerkki 1. Python 3 -version käyttöönotto

Pythonin kirjasto riippuvuudet ovat oletuksena globaaleja, joten asensin niin sanotun virtuaalisen ympäristön, jonka avulla projekti asentaa omat riippuvuutensa lokaalisti. Virtuaalisen ympäristön käyttöönotto onnistuu koodiesimerkki 2 sisältämien komentojen avulla.

```
python3 -m venv py36-venv
source py36-venv/bin/activate
```

Koodiesimerkki 2. Python 3 virtuaalisen ympäristön luonti ja käyttöönotto

5.1 HTTP-rajapinnan luonti

Pythonilla HTTP-rajapinnan luontiin löytyy useita avoimen lähdekoodin ohjelmistokehyksiä. Kehyksen valinnassa yksi tärkeä kriteeri on, että se implementoi WSGI-standardin. WSGI on websovellusstandardi, jonka avulla webohjelmistot voivat vastaanottaa ja lähettää HTTP-kutsuja välityspalvelimen kautta. Käytännössä se on siis adapteri palvelimen ja Python-ohjelman välillä, joka tarjoaa standardisoidun rajapinnan molemmille puolille [6]. Välityspalvelimena voi tässä tapauksessa toimia esimerkiksi Nginx tai Apache-webpalvelin. Tämä on ohjelmiston jatkokehityksen kannalta hyvä ominaisuus, koska sen avulla voidaan eritellä logiikka esimerkiksi palvelimen tietoturvalle, kuormituksen tasapainottamiselle ja kutsujen optimoinnille. Python-rajapinta voi näin ollen keskittyä täysin bisneslogiikan implementointiin. Django on yksi suosituimmista Python WSGI -webohjelmistokehyksistä, mutta en valinnut sitä tähän projektiin, koska se on suunniteltu suurikokoisille tietokantaa käyttäville projekteille. Tässä tapauksessa on parempi valita mahdollisimman ketterä ohjelmistokehys, koska kyseessä on yksinkertainen prototyyppi. Näin ollen valitsin Flask-nimisen Python-ohjelmistokehyksen, jolla on hyvin helppo ja nopea luoda rajapintoja. Koodiesimerkistä 3 nähdään yksinkertaisen rajapinnan luontiin vaadittava koodi.

```
from flask import Flask, request
app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello, world!'
```

Koodiesimerkki 3. Flask-kirjaston esimerkki HTTP-rajapinnan luomisesta

Ennen kuin Satelliitille voi lähettää lokienhakupyyntöjä, pitää se konfiguroida HTTP-rajapinnan kautta. Konfigurointi kirjoittaa annetut parametrit Rsyslogin konfiguraatiotiedostoon ja käynnistää uudelleen Rsyslog-prosessin. Konfiguraatiolle annetaan pakollisena parametrina lokienhallintajärjestelmän IP-osoite ja vaihtoehtoisina parametreina porttinumero ja protokolla, joka on TCP tai UDP. Koodiesimerkissä 4 on konfiguraatiorajapinnan POST-kutsun implementaatio, jossa parametrien validointi ja tiedoston formatointi on abstrahoitu.

```
class Configuration(Resource):
  def post(self):
    # parametrien validointi
    body = validators.configuration_parser.parse_args()
```

```
    try:
      # konfiguraatiotiedoston kirjoitus
      config_content = config.create_config(**body)
      with open(CONF_FILE, 'w') as file:
        file.write(config_content)

      # prosessin uudelleenkäynnistys
      pid = subprocess.Popen(['pgrep', 'rsyslog'],
              stdout=subprocess.PIPE).communicate()[0]
      os.kill(int(pid), signal.SIGTERM)
    except Exception as e:
      return {'message':
              'Error while writing configuration file'}, 500
    return body

api.add_resource(RequestLog, '/request_log')
```

Koodiesimerkki 4. Konfiguraatiorajapinnan implementaatio

Lokienhakupyyntö ottaa pakollisina parametreina verkkolaitteen isäntänimen eli yleensä IP-osoitteen ja listan lokitiedostoja, jotka sisältävät nimen ja tiedostopolun. Vapaaehtoiset parametrit ovat verkkolaitteeseen SFTP-yhteydenottoa varten portti, käyttäjänimi ja salasana. Lokienhakupyyntö tarkistaa, että konfiguraatiotiedosto on kirjoitettu ennen kuin se etenee itse lokien hakuun verkkolaitteesta.

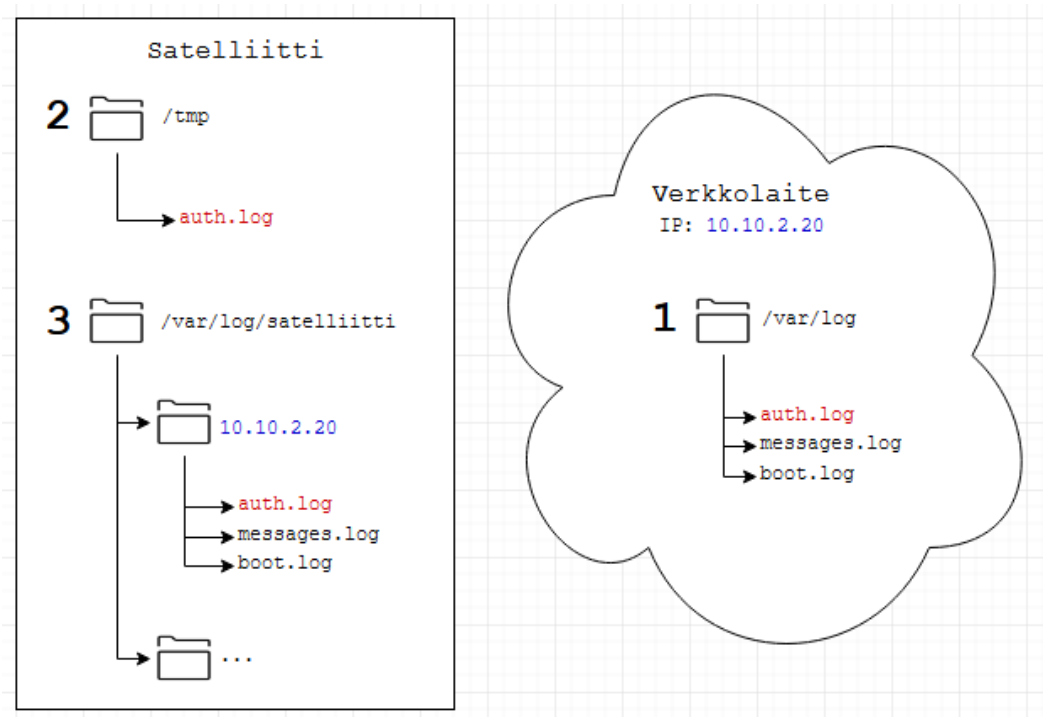
5.2 Lokien haku verkkolaitteesta

Lokien kopiointi verkkolaitteesta paikalliselle kiintolevylle onnistui yksinkertaisimmin Paramiko-kirjaston avulla. Paramiko-kirjastolla on useita rajapintoja tiedostojen kopiointiin, mutta tässä tapauksessa tarvitsimme vain STFP-rajapintaa. Ensimmäinen strategia lokien kopioimiseen oli yksinkertaisesti kopiointi verkkolaitteelta (kuva 7, numero 1) suoraan Satelliitin pysyvään lokitiedostohakemistoon (kuva 7, numero 3). Jos loki olisi seuraavalla kerralla pienempi tiedostokooltaan kuin aikaisemmin, tarkoittaisi se sitä, että loki on kierrätetty verkkolaitteessa. Silloin poistettaisiin vanha lokitiedosto, jotta Rsyslog ymmärtäisi, että uuden tiedoston kaikki rivit ovat uusia lokeja. Tässä ratkaisussa yksi ilmiselvä ongelma oli se, että lokikierrätyksen ohella kadotettiin tuntematon määrä lokeja riippuen kutsujen tiheydestä. Jokainen uusi loki viimeisen lokienhakupyynnön jälkeen, mutta ennen lokienkierrätystä olisi tämän strategian ulottumattomissa.

Toinen ongelma oli se, että testilaitteella SFTP-yhteys yhdisti aktiiviseen sulautettuun järjestelmään, joka vaihteli satunnaisin väliajoin. Sulautetuilla järjestelmillä oli sama uu-

sin loki samassa tiedostopolussa, mutta niillä oli eri lokikierrätysprosessit, joten lokitiedostojen koot poikkesivat toisistaan. Tämä rikkoi kopioinnin täysin, sillä satelliitti luuli välillä virheellisesti, että lokitiedosto oli kierrätetty. Se johti siihen, että samoja lokeja lähetetiin lokienhallintajärjestelmään useaan otteeseen. Näiden ongelmien vuoksi tarvittiin hieman fiksumpi algoritmi lokitiedostojen kopiointiin.

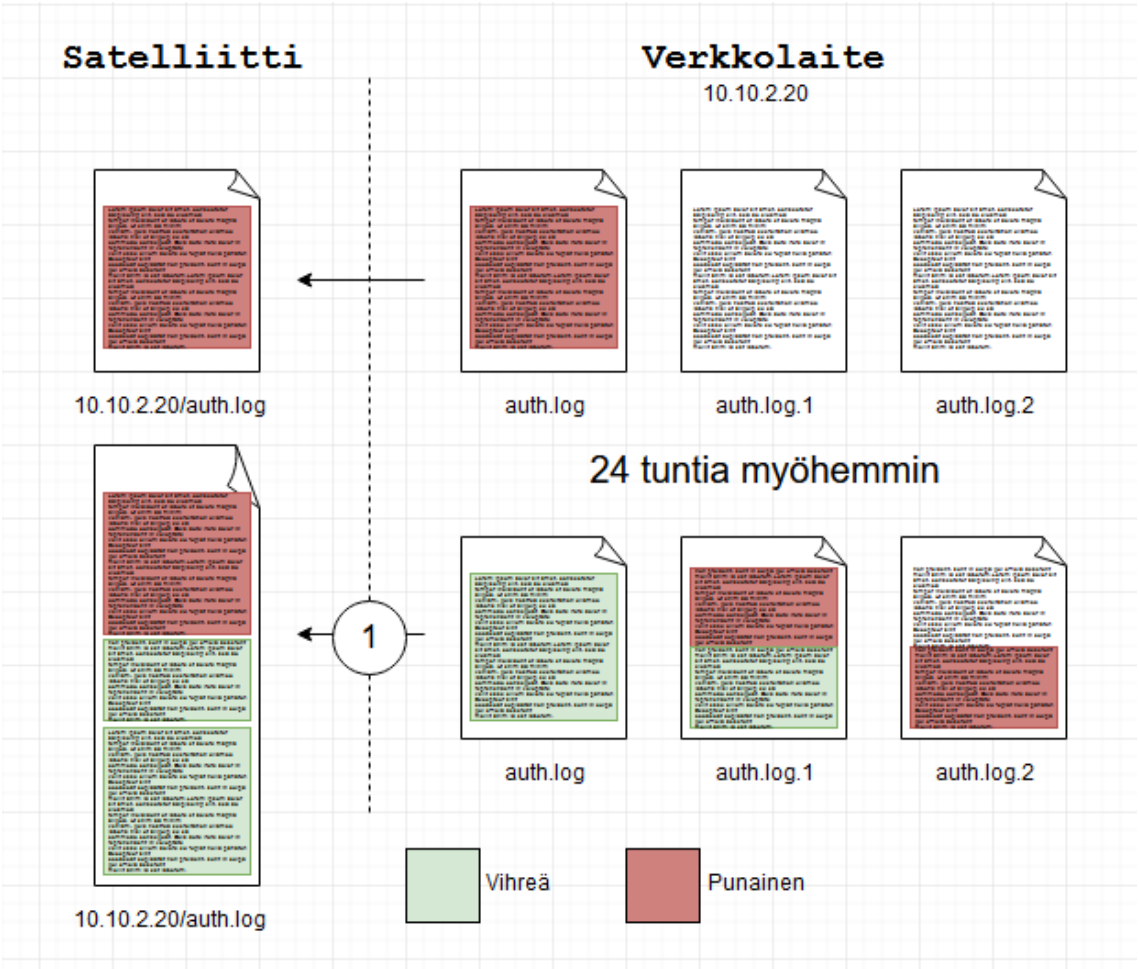
Lokienhakualgoritmin on ensin tarkistettava, haetaanko lokitiedostoa ensimmäistä kertaa. Lokitiedostot ovat uniikkeja IP-osoitteen ja nimen yhdisteenä. Lokitiedoston ensimmäisellä hakukerralla kopioidaan se suoraan verkkolaitteelta (kuva 7, numero 1) Satelliitin pysyvään lokitiedostopolkuun (kuva 7, numero 3). Jos lokitiedosto on haettu jo aikaisemmin, tallennetaan lokitiedosto väliaikaishakemistoon (kuva 7, numero 2).



Kuva 7. Havainnollistava kuva satelliitin ja verkkolaitteen tiedostorakenteista

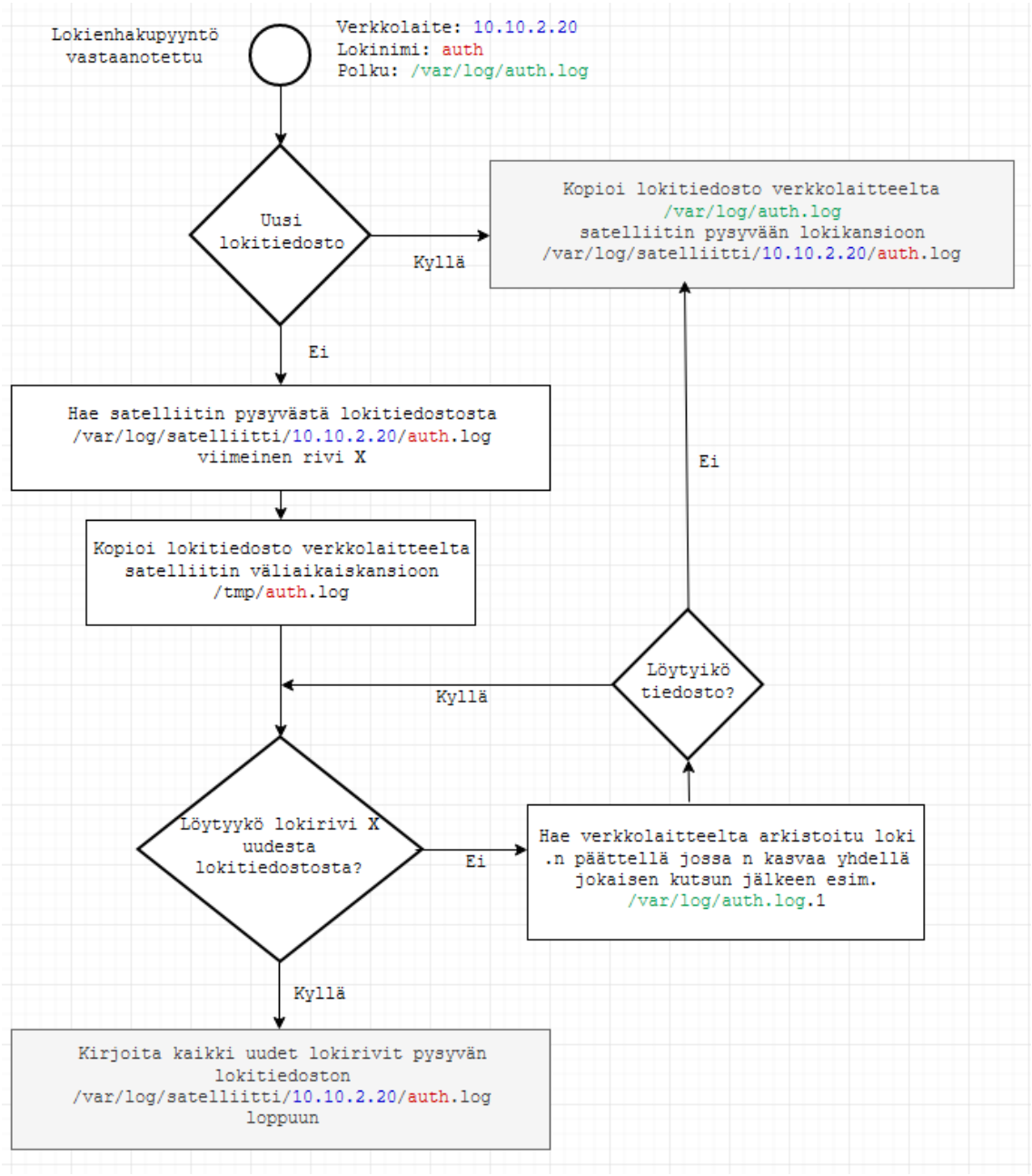
Uudesta väliaikaisesta tiedostosta etsitään vain uudet lokirivit viime kutsukerrasta. Uudet rivit löytyvät yksinkertaisesti niin, että haetaan vanhasta Satelliitin lokitiedostosta viimeinen rivi ja verrataan riviä uuden tiedoston jokaiseen riviin alimmasta rivistä alkaen. Jos alin rivi uudessa lokitiedostossa on sama kuin vanhassa voidaan päätellä, että uusia lokeja ei ole kertynyt. Jos taas alin rivi ei ole sama, niin se tarkoittaa sitä, että se on uusi

loki. Uusia lokeja lisätään listaan niin kauan, kunnes vanha loki löytyy uudesta lokitiedostosta. Kun vanha loki löytyy, kirjoitetaan kaikki uudet lokit sisältävä lista pysyvän lokitiedoston loppuun. Voi kuitenkin käydä niin, että vanhaa lokia ei löydy uudesta lokitiedostosta ollenkaan, jos lokitiedosto on kierrätetty. Tämä tarkoittaa sitä, että kaikki uudessa lokitiedostossa olevat lokit ovat uusia. Se tarkoittaa myös sitä, että kierrätetyssä ”.1”-loppuisessa lokitiedostossa on tuntematon määrä uusia lokeja. Kuvassa 8 on kuvattu kahta lokienhakupyyntöä ja niiden hakemia lokeja. Ensimmäinen pyyntö kirjoittaa koko lokitiedoston Satelliitille. Toisessa kutsussa punaiset rivit ovat aikaisemmassa kutsussa kirjoitetut lokit ja vihreät ovat uusia lokeja. Pysyvään lokitiedostoon kirjoitetaan kaikki vihreät, eli uudet lokit oikeassa kronologisessa järjestyksessä. Numeroitu pallo 1 kuvaa algoritmia, joka etsii uudet lokirivit.



Kuva 8. Kaksi lokienhakupyyntöä ja niiden hakemat lokit värikoodattuina

Algoritmi hakee kierrätetyt uudet lokit rekursiivisesti niin, että jos se ei löydä ensimmäisestä verkkolaitteen lokista vanhaa lokia, niin se hakee seuraavan kierrätetyn lokitiedoston ja etsii sitä siitä. Tämä rekursio loppuu joko siihen, että loki löytyy tai että seuraavaa kierrätettyä lokitiedostoa ei löytynyt. Jos loki löytyy, palauttaa rekursiivinen funktio summatun listan kaikista uusista lokiriveistä, jotka kirjoitetaan pysyvään lokitiedostoon. Jos taas vanhaa lokia ei koskaan löydetty niin kopioidaan uusin väliaikainen lokitiedosto pysyväksi lokitiedostoksi ja vastataan HTTP-kutsuun virheellä, joka kehottaa lokienhakukutsujen tihentämistä. Rekursiivinen algoritmi on visualisoitu vuokaaviona kuvassa 9.



Kuva 9. Lokienhakualgoritmi visualisoitu vuokaaviona.

Ajan myötä pysyvien lokitiedostojen koot kasvavat. Ja jotta ne eivät kasvaisi ongelmallisen suureksi, poistetaan ne kokonaan lokienhakupyynnön yhteydessä, kun ne ovat kasvaneet kooltaan yli 50 megatavuisiksi. Poistetun lokin tilalle luodaan uusi tyhjä lokitiedosto, johon kirjoitetaan sen lokienhakupyynnön uudet lokit. Lokit lähetetään lähes välittömästi lokienhallintajärjestelmään tiedostoon kirjoittamisen jälkeen, joten on erittäin epätodennäköistä, että lokin poiston yhteydessä aikaisempia lokeja ei olisi ehditty lähettämään edelleen. Käytännössä lokin voisi poistaa joka kerta ennen lokiin kirjoittamista,

mutta riskien minimoimiseksi päädyin poistamaan sen vain tietyn tiedostokokorajan jälkeen.

5.3 Lokien rikastus ja edelleen lähetys lokienhallintajärjestelmään

Rsyslog on avoimen lähdekoodin ohjelmisto lokien prosessointiin ja edelleen lähetykseen. Se voi lukea lokeja useista eri lähteistä, muokata niitä ja lähettää ne edelleen useisiin eri määränpäihin. Rsyslog implementoi alkuperäisen RFC 3164 Syslog -standardin lisäksi useita laajennuksia kuten TCP-protokollan, TLS eli viestien salauksen, ISO 8601 standardin aikaleiman ja RFC 5424 Syslog -standardin [10].

Rsyslog koostuu modulaarisista osista, jotka pitää erikseen aktivoida konfiguraatiossa. Lokitiedoston, eli tekstitiedoston monitorointi lokien lähteenä onnistuu ”imfile”-nimisellä moduulilla. Moduulille annetaan ”PollingInterval”-parametri, joka kertoo tiedostojen tarkastelutiheyden sekunteina. Koodiesimerkki 5 aktivoi ”imfile”-moduulin.

```
module(load="imfile" PollingInterval="10")
```

Koodiesimerkki 5. Rsyslog-konfiguraatio imfile-moduulin aktivointiin

Versiosta 8.25.0 lähtien ”imfile”-moduuli tukee villikortteja hakemistonimissä tiedostonimien lisäksi [11]. Tämä on tärkeä ominaisuus, sillä kansiorakenteemme on Rsyslog-konfiguraation luonnin aikana tuntematon, koska verkkolaitteiden IP-osoitteet eivät ole vielä tiedossa. Koodiesimerkki 6 määrittelee lokitiedostojen monitoroinnin, jossa ”File”-parametri kertoo, mistä tiedostoista Rsyslog hakee lokeja. Tähti tarkoittaa villikorttia, eli etsimme satelliittihakemistosta, minkä vain nimisiä hakemistoja, joilla on minkä vain nimisiä lokitiedostoja. Löydettyjen lokitiedostojen uudet lokit annetaan ”Ruleset”-parametrin määrittelemän sääntöfunktion käsiteltäväksi.

```
input(
  type="imfile"
  File="/var/log/satelliitti/*/*.log"
  Ruleset="edelleenLähetys"
)
```

Koodiesimerkki 6. Rsyslog-konfiguraatio lokitiedostojen lukemiseen

Sääntöfunktion toiminnallisuus määritellään myös Rsyslogin konfiguraatiossa. Koodiesimerkki 7 sisältää sääntöfunktion, jolla on yksi yksinkertainen toiminto, lokien edelleen lähetys lokienhallintajärjestelmään. Lokienhallintajärjestelmän IP-osoite ja portti on määritelty konfiguraatio HTTP-rajapinnan kautta, ja Python on kirjoittanut ne Rsyslogin konfiguraatiotiedostoon. Mielenkiintoinen osa konfiguraatiota on ”Template”-parametri, eli sapluunafunktio. Kaikki lokit ajetaan sapluunafunktion läpi ennen lähetystä lokienhallintajärjestelmään.

```
ruleset(name="edelleenLähetys") {{
  action(
    type="omfwd"
    Target="10.10.1.1"
    Port="540"
    Protocol="udp"
    Template="rikastus"
  )
  stop
}}
```

Koodiesimerkki 7. Rsyslog-konfiguraatio sääntöfunktion määrittelylle

Lokeja ei voida lähettää suoraan lokienhallintajärjestelmään, koska loki on kirjoitettu verkkolaitteella eikä se sisällä tarvittavia tietoja, jotta lokienhallintajärjestelmä pystyisi yhdistämään lokin kyseiseen verkkolaitteeseen. Rsyslogin on luotava uusi lokirakenne, jossa IP-osoite on lokin isäntänimi, tiedostonimi on lokin applikaatiotiedoissa ja alkuperäinen loki on uuden lokin viesti. Lokille annetaan myös uusi ISO 8601 -standardin mukainen aikaleima, koska verkkolaitteiden lokit eivät usein sisällä aikavyöhyketietoja. Aikavyöhyketiedot ovat tärkeitä, koska niiden avulla saadaan lokienhallintajärjestelmässä lokeille kronologinen järjestys. Uusi aikaleima ei kuitenkaan tässä tapauksessa vastaa alkuperäisen lokin aikaa, vaan se tarkoittaa sitä aikaa, kun Rsyslog luki lokin. Kuvasta 9 ilmenee, kuinka lokirakennetta halutaan muuttaa siirtämällä koko vanha loki uuden lokin viestiksi.

```
LOKIRAKENNE VERKKOLAITTEELTA
10.10.2.20  /var/log/auth.log

Oct 19 12:15:01 root CRON: session opened for user root


HALUTTU LOKIRAKENNE

2019-10-20T16:34:45.547Z 10.10.2.20 auth Oct 19 12:15:01 root CRON: session
opened for user root
```

Kuva 10. Esimerkki verkkolaitteelta saadun lokin ja halutun lokin rakenteesta

Sapluunafunktiolla muokataan lokien rakennetta ja sisältöä. Se saa metadata muuttujassa lokitiedoston koko tiedostopolun, josta voimme parsia verkkolaitteen IP-osoitteen ja tiedostonimen ja lisätä ne uuteen lokirakenteeseen. Koodiesimerkin 8 konfiguraatio määrittelee lokien rikastuslogiikan. Se ottaa metadata muuttujasta tiedostopolun ja lukee IP-osoite hakemistonimen ja tiedostonimen regex-komennolla. Lopputulos on loki halutussa muodossa.

```
template(
  name="rikastus"
  type="string"
  string="%TIMESTAMP:::date-rfc3339%
%$!metadata!filename:R,ERE,1,FIELD:([^\\\\/]+)\\\\/[^\\\\/]+.$--end%
%$!metadata!filename:R,ERE,1,FIELD:([^\\\\/]+).log$--end%%msg:::sp-if-no-1st-
sp%%msg:::drop-last-lf%\\n"
)
```

Koodiesimerkki 8. Rsyslog-konfiguraatio sapluunafunktion määrittelylle

5.4 Käyttäjät ja käyttöoikeudet

Satelliitti on implementoitu Linux-käyttöjärjestelmässä, ja se koostuu useasta prosessista, joten on otettava huomioon prosessien ja hakemistojen käyttöoikeudet. Ohjelmia ei yleensä haluta ajaa pääkäyttäjänä, koska sillä on oikeudet kaikkiin järjestelmän tiedostoihin ja prosesseihin ja on näin ollen tietoturvariski. Jos ohjelma ajetaan omalla käyttäjällään, on sillä oikeudet vain sen käyttäjän omistamille tiedostoille ja prosesseille.

Satelliitille luodaan uusi käyttäjä nimeltään satelliitti, jonka tarkoituksena on eristää lokienhaku muusta käyttöjärjestelmästä. Käyttäjä luodaan koodiesimerkki 9 ensimmäisellä

komennolla. Satelliittikäyttäjälle annetaan luku ja kirjoitusoikeudet Rsyslogin konfiguraatiohakemistoon koodiesimerin 9 seuraavalla kahdella komennolla. Koodiesimerkki 9 kolmella viimeisellä komennolla luodaan satelliittilokihakemisto ja asetetaan Satelliittikäyttäjä sen omistajaksi.

```
# 1. Satelliittikäyttäjän luonti
useradd satelliitti

# 2. Rsyslog konfiguraation kirjoitusoikeudet
chown satelliitti:satelliitti /etc/rsyslog.d
chmod ug+rw /etc/rsyslog.d

# 3. Satelliittilokihakemiston luonti ja käyttöoikeudet
mkdir /var/log/satelliitti
chown satelliitti:satelliitti /var/log/satelliitti
chmod ug+rwx /var/log/satelliitti
```

Koodiesimerkki 9. Satelliittikäyttäjän luonti ja hakemistojen käyttöoikeuksien asettaminen

Rsyslog-ohjelman voi käynnistää vain pääkäyttäjä, joka johtaa siihen ongelmaan, että HTTP-rajapintaprosessilla ei ole oikeutta uudelleen käynnistää Rsyslog-prosessia konfiguraation tallennuksen jälkeen. Rsyslog-konfiguraatiosta löytyy kuitenkin ”PrivDropToUser”-parametri, jonka avulla voidaan vaihtaa Rsyslog-prosessin omistajaa käynnistyksen jälkeen. Koodiesimerkki 10 ensimmäinen komento kirjoittaa Rsyslogin pääkonfiguraatioon tämän parametrin. Satelliittikäyttäjä saa tämän konfiguraation ansiosta uudelleen käynnistää Rsyslog-prosessin. Yksinkertaisin tapa uudelleen käynnistää prosessi on tuhota se, mutta se vaatii tietyn Systemd-konfiguraation. Systemd on Linux-ohjelmisto, jonka vastuulla on käynnistää ja ylläpitää käyttöjärjestelmän tärkeitä ohjelmistoja. Koodiesimerkki 10 toinen komento muokkaa Rsyslogin Systemd-konfiguraatiota niin, että se uudelleen käynnistyy tarvittaessa automaattisesti. Koodiesimerkin 10 viimeinen komento uudelleen käynnistää Rsyslog-prosessin muutoksien käyttöönottamista varten.

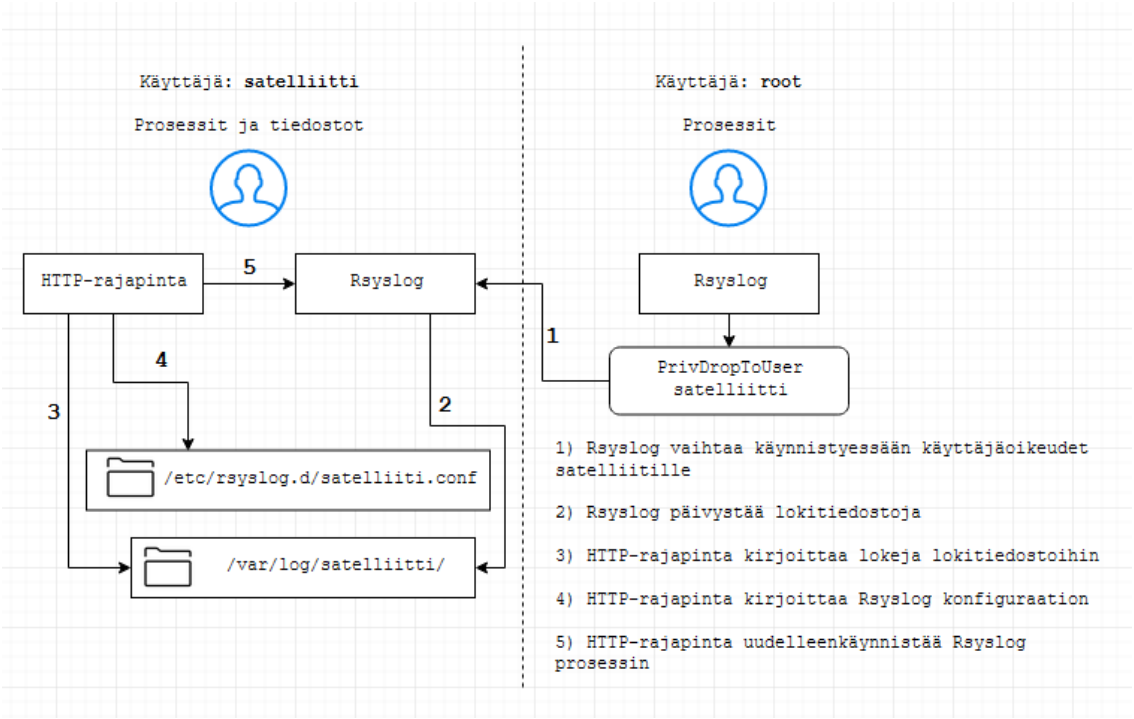
```
# 1. siirrä rsyslog prosessi satelliitti käyttäjälle käynnistyksen jälkeen
echo "$PrivDropToUser satelliitti" >> /etc/rsyslog.conf

# 2. uudelleenkäynnistä rsyslog prosessi jos se sammuu
sed -i 's/Restart=.*/Restart=always/g' /usr/lib/systemd/system/rsyslog.service

# 3. uudelleenkäynnistä rsyslog muutoksien käyttöönottamista varten
service rsyslog restart
```

Koodiesimerkki 10. Rsyslog-ohjelman konfigurointi

Koko Satelliittiohjelmisto on nyt saman käyttäjän alla ja kaikki liialliset käyttöoikeudet on minimoitu. Satelliitin käyttöoikeuksia ja niiden tarkoituksia on visualisoitu kuvassa 10. Kuvan oikealla puolella on pääkäyttäjän luoma Rsyslog-prosessi, joka siirrettään Satelliittikäyttäjälle. Satelliittikäyttäjällä on kaksi prosessia, Rsyslog ja HTTP-rajapinta, jotka ovat vuorovaikutuksessa toistensa ja edellä mainittujen hakemistojen kanssa.

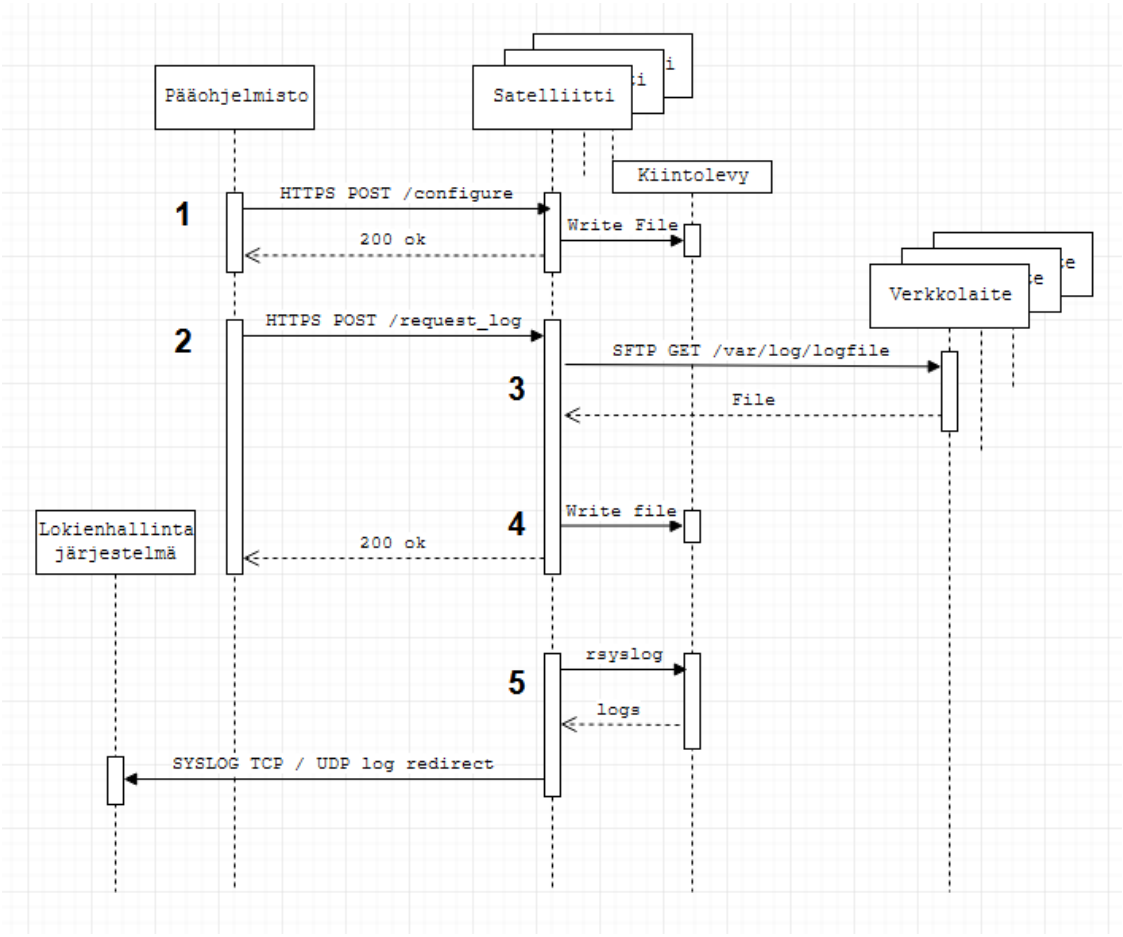


Kuva 11. Satelliitin käyttäjät ja oikeudet

5.5 Työn toiminnallisuuden yhteenveto

Insinöörityön satelliitti ja sitä ympäröivä järjestelmä toimii korkealla tasolla kuvan 11 sekvenssikaavion mukaisesti. Ensin pääjärjestelmän on konfiguroitava Satelliitti, jotta se voi lähettää lokit lokienhallintajärjestelmään (kuva 11, numero 1). Kutsu kirjoittaa konfiguraation Satelliitin kiintolevylle ja käynnistää uudelleen Rsyslog-prosessin. Konfiguraation jälkeen pääohjelmisto voi tehdä lokienhakupyyntökutsuja (kuva 11, numero 2). Satelliitti vastaanottaa lokienhakupyyntökutsulta yhteystiedot verkkolaitteeseen ja hakee halutut lokit verkkolaitteelta (kuva 11, numero 3). Satelliitti etsii lokeista uusimmat rivit ja kirjoittaa

ne paikalliseen lokitiedostoon (kuva 11, numero 4). Rsyslog päivystää paikallisia lokitiedostojen muutoksia, rikastaa lokit metadatalla ja lähettää ne lokienhallintajärjestelmään (kuva 11, numero 5).



Kuva 12. Sekvenssikaavio järjestelmän toiminnasta korkealla tasolla.

6 Jatkokehitys ja työn arviointi

6.1 Jatkokehitys

Jatkokehityksen ensimmäinen vaihe on itse tuotantoversion kehittäminen. Sitä kehittäessä voidaan käyttää insinöörityön prototyyppiä hyväksi. Tuotantoversioon tulee käyttöön välityspalvelimeksi todennäköisesti nginx, joka on konfiguroitava vastaanottamaan HTTPS-liikennettä portilta 443 ja ohjaamaan se satelliitin Python-palvelimelle. HTTPS-

liikenne vaatii satelliitille sertifikaatin, joten on suunniteltava, missä vaiheessa asennusta se sinne annetaan. Satelliitin asennuksen automatisointi on iso osa jatkokehitystä. Insinöörityössä monet ohjelmistot ja konfiguraatiot asennettiin käsin, joten ne on automatisoitava tuotantoversiossa. Asennus tulisi vaatia mahdollisimman vähän manuaalisia askeleita, jotta inhimillisten vikojen määrä minimoitaisiin.

Satelliitin sopimukselliset ja lainsääteiset vaatimukset, kuten GDPR ovat osa jatkokehitystä. Mikäli henkilötietoja sisältävää lokidataa haetaan verkkolaitteilta pitää ne joko anonymisoida tai jättää lähettämättä lokienhallintajärjestelmään.

Verkkolaitteiden kattava yhteensopivuustestaus tulee myös olemaan osa jatkokehitystä. Prototyyppi testattiin vain kahden eri verkkolaitteen kanssa, mutta tuotannossa on mahdollisesti enemmän verkkolaitteita, jonka kanssa satelliitin on toimittava. On mahdollista, että jokin näistä verkkolaitteista ei ole yhteensopiva prototyypin lokienhakulogiikan kanssa. Tässä tapauksessa on sovellettava satelliitin toimintaa joko toimimaan yleisemmin kaikille verkkolaitteille tai sitten jaoteltava satelliitti käsittelemään eräitä verkkolaitteita eri tavalla.

Yksi osa itse satelliittia, jota todennäköisesti tullaan jatkokehittämään, on lokien edelleen lähetys lokienhallintajärjestelmään. Prototyypissä ei ole mitään kunnollista vikatilanteen käsittelyä, jos esimerkiksi lokienhallintajärjestelmän yhteystiedot ovat väärät. Satelliitin konfiguraatiota kirjoittaessa olisi hyvä tehdä jokin automatisoitu testi, joka varmistaa, että annetut yhteystiedot ovat toimivat.

Jatkokehitykselle sopisi myös yrittää korjata epätodennäköinen mutta mahdollinen lokien katoaminen. On mahdollista, mutta epätodennäköistä, että Rsyslog ei ole ehtinyt lukemaan ja lähettämään poistettuja lokeja, jos lokitiedosto poistetaan liian nopeasti uusien lokien lisäämisen jälkeen. Tämän tapahtuman todennäköisyyttä voisi laskea entisestään esimerkiksi rajoittamalla uniikin lokitiedoston pyyntötiheyttä. Tällä hetkellä Rsyslog ja HTTP-rajapinta eivät kommunikoi toistensa kanssa ollenkaan, joka voi johtaa ongelmiin esimerkiksi liian kuormitetussa ympäristössä.

6.2 Työn arviointi

Tämä insinöörityö toteuttaa kaikki kolme kriittistä vaatimusta lukemalla lokeja verkkolaitteesta lisäämällä lokiin dataa ja lähettämällä lokit lokienhallintajärjestelmään.

Työssä on implementoitu HTTP-rajapinta, jonka avulla Satelliitti on konfiguroitavissa ja joka vastaanottaa lokienhakupyyntöjä. Rajapintaa on testattu lähettämällä konfiguraatioita ja lokienhakukutsuja cURL-komennolla lokaalisti Satelliitilta.

Satelliitin levytila ei kasva äärettömästi, koska lokit poistetaan tietyn rajan jälkeen. Lokien poistoa testattiin laskemalla tiedostokokoraja negatiiviseksi, eli lokit poistuivat jokaisen kutsun yhteydessä. Lokien poiston yhteydessä ei jätetty yhtään lokia lähettämättä.

Prototyyppi on manuaalisen testauksen avulla toimittu toimivaksi kahden eri verkkolaitteen kanssa, joten ohjelmisto voidaan todeta tarpeellisen geneeriseksi. Ohjelmisto ei lähettänyt yhtäkään lokia useaan kertaan, eikä se jättänyt yhtäkään lokia lähettämättä. Lokienhallintajärjestelmä vastaanotti lokit muodossa, josta se voi tunnistaa verkkolaitteen.

Työ voidaan todeta onnistuneeksi, sillä se on toteuttanut kaikki vaatimukset tarpeenmukaisesti.

Lähteet

1 Ericsson. Verkkoaineisto. https://www.ericsson.com/en/about-us/company-facts Luettu 30.5.2019.

2 Red Hat. Verkkoaineisto. https://www.redhat.com/en/technologies/linux-platforms/enterprise-linux Luettu 07.09.2019.

3 Karen Kent, Murugiah Souppaya. 2006. Guide to Computer Security Log Management. National Institute of Standards and Technology.

4 GDPR. Verkkoaineisto. https://eugdpr.org/the-regulation/ Luettu 30.10.2019.

5 Logrotate – linux man page. Verkkoaineisto. https://linux.die.net/man/8/logrotate Luettu 10.05.2019.

6 WSGI. Verkkoaineisto. https://www.python.org/dev/peps/pep-3333/ Luettu 21.10.2019.

7 The Syslog Protocol. Verkkoaineisto. https://tools.ietf.org/html/rfc5424 Luettu 10.10.2019.

8 Transmission of Syslog Messages over UDP. Verkkoaineisto. https://tools.ietf.org/html/rfc5426 Luettu 25.10.2019.

9 Logstash. Verkkoaineisto: https://www.elastic.co/products/logstash Luettu 27.10.2019.

10 Rsyslog features. Verkkoaineisto: https://www.rsyslog.com/features/ Luettu 25.10.2019.

11 Rsyslog imfile. Verkkoaineisto. https://www.rsyslog.com/files/temp/doc-indent/configuration/modules/imfile.html#wildcards Luettu 16.05.2019.