Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Mikko Rajala

HTTP-tapahtumadatan muuttaminen RFC5424-muotoon

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

5.5.2021

Tekijä Otsikko Sivumäärä Aika	Mikko Rajala HTTP-tapahtumadatan muuttaminen RFC5424-muotoon 34 sivua 5.5.2021
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Simo Silander Toimitusjohtaja Mikko Kortelainen

Insinöörityön tavoitteena oli kehittää sovellus, joka ottaisi HTTP:llä vastaan lokitapahtumia JSON-muotoisena, muuttaisi ne Syslogin-protokollan RFC5424-standardin mukaiseksi ja lähettää ne eteenpäin. Komponentti olisi tarkoitettu eräänlaiseksi adapteriksi yrityksen lokienhallintajärjestelmän ja lokitapahtumien lähettäjän välillä.

Sovelluksen haluttiin sallivan käyttäjän luoda itselleen kanavia, joita käyttäen lokitapahtumia voitaisiin lähettää. Kanavien avulla käyttäjät voisivat myös tiedustella lähettämiensä lokitapahtumien tiloja. Tämä tulisi toteuttaa antamalla lähetetyille lokitapahtumille tunnus, jota käyttäen lokitapahtuman onnistuneen käsittelyn jälkeen lokitapahtuma asetettaisiin ”acknowledged”-tilaan. Sovellukseen haluttiin toiminnallisuus, jolla näitä ack-tunnuksia hyödyntäen käyttäjä voisi tiedustella lähettämiensä lokitapahtumien tiloja.

Insinöörityön projektina syntyi HTTP Event Capture -sovellus, jonka API-endpointeja kutsumalla voidaan lähettää lokitapahtumia. Sovellus muuttaa JSON-muotoiset lokitapahtumat Syslog-protokollan mukaisiksi ja lähettää ne sitten eteenpäin. Lopuksi lokitapahtumia voidaan hyödyntää käyttämällä Teragrep-sovellusta, jolla voidaan hakea ja analysoida lokitapahtumia Fail-Safe Archive -arkistointitoteutuksesta.

Avainsanat	

Author Title Number of Pages Date	Mikko Rajala Converting HTTP Event Data to RFC5424 34 pages 5 May 2021
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Professional Major	Software Engineering
Instructors	Simo Silander, Senior Lecturer Mikko Kortelainen, General Manager

The goal of the study was to develop a software which receives JSON formatted records with HTTP, reformat them into RFC5424 format and send them forward. The component would be used as an adapter between a company's log management system and the sender of the log events.

Users of the software needed to create channels which they would use to send the logs. With these channels users could also inquire the status of the sent log events. This would be achieved by giving the log events an id, which would be used to set the log event as "acknowledged". The ids would be named as "ack ids". The software needed a functionality which would allow the user to use the ack ids to inquire the status of the log events they have sent.

Within the project, an HTTP Event Capture (HEC) was created. Using the API endpoints of the HEC software users could send in log events in JSON format. HEC converts JSON formatted data to a format which follows the Syslog protocol's RFC5424 standard and sends them forward. This log data can be later utilized using a Teragrep application which can be used to fetch and analyze data from Fail-Safe Archive.

Keywords	

Sisällys

Lyhenteet

HTTP	Hypertext Transfer Protocol eli protokolla, jota selaimet ja palvelimet käyttävät tiedonsiirtoon.
JSON	JavaScript Object Notation eli avoimen standardin yksinkertainen tiedostomuoto tiedonvälitykseen.
HEC	HTTP Event Capture eli insinöörityön projektina tehty sovellus, joka ottaa HTTP:n kautta lokitapahtumia vastaan ja käsittelee niitä.
UUCP	Unix-to-Unix Copy eli joukko verkkoprotokollia ja Unix-työkaluohjelmia, joita käytetään tiedostojen siirtoon ja komentojen suorittamiseen etäjärjestelmässä.
FTP	File Transfer Protocol eli protokolla, jota käytetään tiedon siirtoon kahden tietokoneen välillä.
NTP	Network Protocol eli protokolla, jota käytetään täsmällisen aikatiedon välittämiseen kahden tietokoneen välillä.
TLS	Transport Layer Security eli salausprotokolla, jolla suojataan Internet-sovellusten tietoliikennettä.
API	Application Programming Interface eli rajapinta, jonka mukaan eri ohjelmat voivat tehdä pyyntöjä tai vaihtaa tietoja.
REST	Representational State Transfer eli arkkitehtuurimalli, jolla toteutetaan ohjelmointirajapintoja.

1 Johdanto

Fail-Safe It Solutions Oy on lokienhallintaan erikoistunut yritys. Yrityksellä on tietueiden eli itsenäisten loogisten informaatiokokonaisuuksien keräämiseen tehty tuotekokonaisuus Fail-Safe Record Management. Tähän tuotekokonaisuuteen haluttiin kehittää komponentti, joka ottaisi HTTP:n kautta lokitapahtumia vastaan JSON-muotoisena ja välittäisi ne eteenpäin Record Management -kokonaisuuden käsiteltäväksi ja lopulta Fail-Safe Archive -arkistointitoteutukseen. Tämän komponentin oli lokitapahtumien vastaanottamisen lisäksi myös muunnettava lokitapahtumat Syslog-protokollan mukaiseksi, jotta Record Management -kokonaisuus pystyisi niitä käsittelemään.

Insinöörityöni projektina päätettiin kehittää edellä mainittuun käyttötarkoitukseen sovellus, joka sai nimekseen HTTP Event Capture (HEC). Edellä mainittujen toiminnallisuuksien lisäksi sovellukseen haluttiin käyttäjän todentaminen sekä käyttäjille mahdollisuus luoda kanavia, joita kautta lähettää lokitapahtumia. Näiden toiminnallisuuksien lisäksi ohjelmaan haluttiin kehittää toiminnallisuus, jonka avulla käyttäjät voisivat tiedustella lähettämiensä lokitapahtumien tiloja.

Ennen kuin insinöörityössäni kerron tarkemmin HEC-sovelluksesta, kerron hieman, millaiseen yritykseen ja kokonaisuuteen tämä komponentti tehtiin. Kerron työssäni aluksi Mikko Kortelaisen perustamasta Fail-Safe IT Solutions Oy:sta, kuinka yritys on saanut alkunsa, mitä he tekevät ja tuottavat sekä millaisilla markkinoilla yritys toimii.

Insinöörityössäni tulen kertomaan myös RFC5424-protokollasta eli Syslog-protokollasta, jonka mukaiseksi HEC-sovellus muuttaa käyttäjän lähettämät viestit. Kerron työssäni, mihin tarkoitukseen Syslog-protokolla on kehitetty, minkä näköisiä ovat Syslog-viestit sekä mitä tietoa ne pitävät sisällään.

Mihin lähetetyt lokitapahtumat sitten lopulta päätyvät ja miten niitä voidaan hyödyntää? Tähän kysymykseen vastaan insinöörityössäni, kun kerron Fail-Safen kehittämästä Teragrep-analysointityökalusta. Tuossa luvussa kerron, kuinka Teragrep-sovelluksella voi analysoida ja hyödyntää säilöttyä lokidataa erilaisten hakujen ja raporttien avulla.

Insinöörityöni viimeisessä luvussa kerron HTTP Event Capture -sovelluksesta. Käsittelen tarkemmin, mitkä olivat ohjelman kehittämisen lähtökohdat sekä tavoitteet. Esittelen ohjelman sisältä löytyvät Manager-luokat, joilla hallitaan ohjelman toiminnallisuuksia. Tämän jälkeen kerron ohjelman toiminnallisuudesta siitä näkökulmasta, kun käyttäjä lähettää ohjelmalle lokiviestin, miten viesti otetaan vastaan, miten se muutetaan Syslog-protokollan mukaiseksi sekä miten se lähetetään eteenpäin. Käyn myös läpi toiminnallisuuden, jolla käyttäjä voi tiedustella lokitapahtuman tilaa. Lopuksi näytän sovelluksen käyttöesimerkkejä ja pohdin, mitä mieltä olen lopputuloksesta.

2 Fail-Safe IT Solutions Oy

Mikko Kortelainen perusti Fail-Safe IT Solutions Oy:n vuonna 2010 pilvipalveluliiketoimintaa varten. Fail-Safen varsinainen liiketoiminta käynnistyi vuonna 2015 tietueiden hallinnan saralla finanssialalle suuntautuneen konsultoinnin kautta, jonka rahoittamana Fail-Safe jatkokehitti tietueidenkeräämiseen suunnitellun tuotteen: Fail-Safe Record Managementin, joka mahdollisti suurien datamäärien keräämisen strukturoituina yksittäisten kohteiden tietojen muodostamina kokonaisuuksina eli tietueina [1]. Suurin suomalainen finanssitalo antoi tuotteelle lämpimän vastaanoton, sillä se mahdollisti heidän rakentaa reaaliaikainen tiedonkeruupalvelu, joka on tarkoituksenmukainen ja riippumaton varsinaisesta tietueiden hyödyntämisjärjestelmästä, joka siten mahdollistaa heidän käyttää alustassaan erilaisia analyysi- sekä valvontatuotteita. [2.]

Fail-Safe on myynyt käyttöoikeuden Fail-Safe Record Management -tuotteeseen asiakkailleen. Asiakas kehitti yhtiön johdolla arkistointituotteen, joka tunnetaan nykyisin nimellä Fail-Safe Archive. Tällä tuotteella kyetään tallettamaan kerätyt tietueet regulaatioja lakivelvotteiden mukaisesti sekä toistamaan tietueiden toimitus hyödyntämistuotteisiin. Myöhemmin asiakas myi tämän arkistointituotteen Fail-Safelle. Tämä osto mahdollisti yhtiölle tietueiden välittämiseen ja tallettamiseen soveltuvan kokonaisratkaisun kehittämisen. Edellä mainittujen tuotteiden tueksi Fail-Safe lanseerasi asiakkailleen jatkuvan palvelun, joka mahdollistaa asiakkaalle järjestelmänsä käytön ilman heidän omaa osaamistaan palvelun tuotannossa. [2.]

Fail-Safen liikeidea on tuottaa asiakkailleen tietueidenhallintatuotteita ja niihin liittyvää palvelua. Tällöin asiakas voi säilöä ja tarkastella isoja tietomääriä reaaliaikaisesti täyttämällä samanaikaisesti laki- ja regulaatiovaatimukset erittäin kustannustehokkaasti. [2.]

Markkinoilla ei ole nykyisin reaaliaikaiseen tietueidenhallintaan suunnattuja tuotteita. Kilpailevat tuotteet perustuvat joko vanhoihin Business Intelligence -tuotteisiin, tai sitten ne ovat muuhun käyttötarkoitukseen suunniteltuja tuotteita, kuten esimerkiksi Security Information and Event Management -tuotteita, joita käytetään tietueidenhallintaan. Molempien tuotteiden haasteena ovat korkea hinta, puutteelliset ominaisuudet sekä vähäinen yhteensopivuus muiden tuotteiden kanssa. [2.]

Nykyisiltä asiakkailta saadun palautteen perusteella saatujen lausuntojen pohjalta Fail-Safe on kuitenkin havainnut, että sekä Business Intelligence- että Security Information and Event Management -tuotekokonaisuudet ovat asiakkaiden mielestä vaikeita ja kalliita eivätkä erityisen hyvin integroidu asiakkaan muihin järjestelmiin eivätkä täytä itsessään tietosuojavaatimuksia. [2.]

3 Syslog-protokolla

Tietotekniikassa generoituu monen näköisiä lokiviestejä erilaisista laitteista ja lähteistä. Lokiviestien ylläpitoon tarvitaan standardi, jotta niitä voidaan kerätä eri laitteista ja järjestelmistä yhteen keskitettyyn palveluun. Tähän tarkoitukseen on luotu Syslog-prokolla.

Syslog on alunperin 1980-luvulla Eric Allmanin kehittämä standardi viestien lokittamiseen. Se mahdollistaa erottelun järjestelmien välillä, jotka luovat, säilövät sekä analysoivat lokiviestejä. Syslogissa jokainen viesti on merkitty facility-koodilla, joka kertoo viestin luoneen laitteen tai sovelluksen tyypin. Jokaiselle viestille on myös asetettu severity-taso, jolla ilmoitetaan lokitapahtuman vakavuusaste. Syslogia käytetään järjestelmienhallintaan, turvallisuuden ylläpitämiseen sekä ylipäätään tiedottamisessa, analysoinnissa ja virheenjäljittämisessä käytettävien viestien välittämiseen. Syslogin monikäyttöisyys mahdollistaa erilaisten järjestelmien lokittamisen keskitettyyn säilytyspaikkaan. [3.]

3.1 Severity- ja Facility-koodit

Syslog-viestejä on todella paljon erilaisia, joten niitä pitää jotenkin kategorisoida. Kaksi pääkohtaa, jotka Syslog-viesteistä halutaan samantien selville, ovat viestin vakavuus sekä viestin lähde.

Severity-koodilla ilmoitetaan Syslog-viestin vakavuustaso asteikolla 0–7 (taulukko 1), jossa 0 tarkoittaa vakavinta mahdollista vakavuutta eli hätätilaa. Vähiten vakavalla tasolla 7 olevia viestejä käytetään viestejä tuottavan sovelluksen virheenjäljitykseen.

Taulukko 1. Severity-taulukko

Numeerinen arvo	**Vakavuusaste**	**Kuvaus**
0	Hätätila	Järjestelmä on käyttökelvoton.
1	Hälytys	Vaatii välittömiä toimia.
2	Kriittinen	Kriittinen, hankala tila.
3	Virhe	Virhetila.
4	Varoitus	Varoitustila.
5	Huomautus	Normaali, mutta merkittävä tila.
6	Informatiivinen	Informatiiviset viestit.
7	Virheenjäljitys	Virheenjäljittämiseen käytettävät viestit

Vakavuustasojen tarkoitukset voivat vaihdella järjestelmäkohtaisesti poislukien tasot 0 ja 7. Esimerkiksi jos järjestelmän tarkoitus on käsitellä tilitapahtumia ja päivittää asiakkaan tilin tietoja, ja järjestelmän toiminnan viimeisessä vaiheessa tapahtuu virhe, tästä tulee lokittaa tasolla 1 (Hälytys). Toisaalta jos tapahtuu virhe, kun koitetaan näyttää asiakkaalle hänen oma postinumeronsa, lokitukseen voidaan asettaa taso 3 (Virhe) tai jopa taso 4 (Varoitus). Käytännössä tason 0 hätätilaviestejä ei normaalisti näe, koska viestin

tuottavan järjestelmän ollessa niin pahasti rikki, se ei todennäköisesti kykene lähettämään viestiä. Toisesta ääripäästä virheenjäljittämiseen käytettäviä tason 7 viestejä ei yleensä haluta nähdä normaalissa lokituksessa, koska niitä olisi liian paljon, ja ne ovat harvoin tärkeitä järjestelmän toimintakyvyn kannalta. Tavallisesti tuotantojärjestelmät lokitus on asetettu tasolle 5 tai 6. Joissain tapauksissa viestejä lähettävä järjestelmä saattaa pitää tallessa paikallisen kopion vähemmän vakavista viesteistä, mutta ei lähetä niitä keskitetylle palvelimelle.

Facility-koodit ovat Severity-koodien tapaan numeerisia arvoja, jotka on listattu taulukossa 2. Koodi määrittelee laitteiston tyypin, joka lokittaa viestejä. Viestejä voidaan käsitellä eri tavoilla laitteistotyyppien perusteella

Taulukko 2. Facility-taulukko

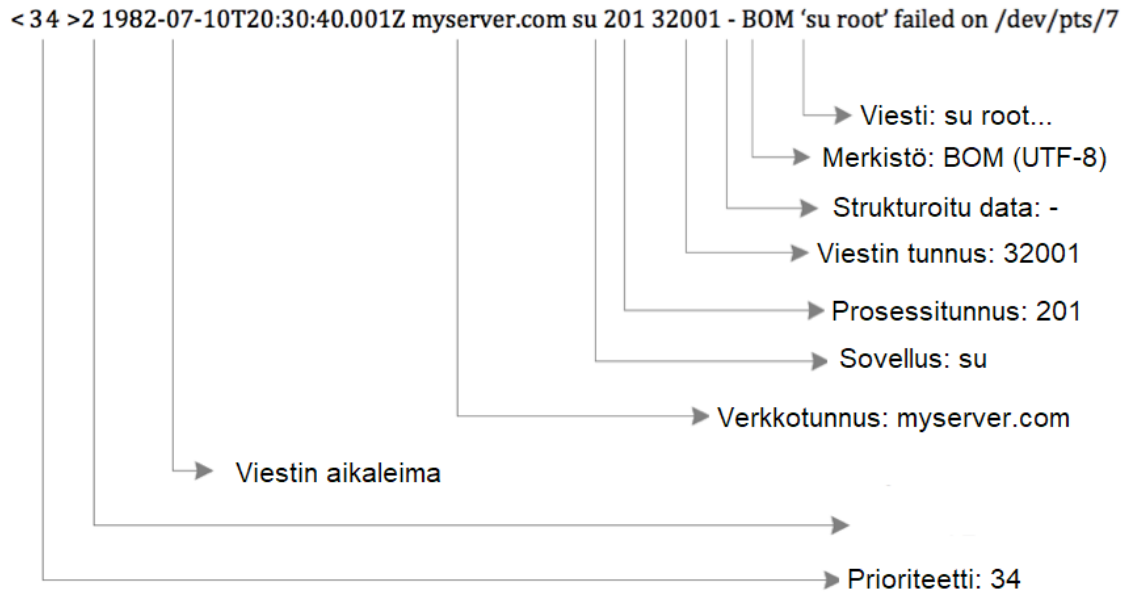
Numeerinen arvo	**Laitteiston nimi**
0	Kerneli
1	Käyttäjätaso
2	Sähköposti
3	Järjestelmän taustaprosessit
4	Turvallisuus- ja todentamisviestit
5	Syslogin sisäiset viestit
6	Rivintulostaja alijärjestelmä
7	Verkoston uutiset alijärjestelmä
8	UUCP-alijärjestelmä
9	Kellotaustaprosessi
10	Turvallisuus- ja todentamisviestit
11	FTP-taustaprosessi
12	NTP-alijärjestelmä
13	Log audit
14	Log alert

15	Aikataulutustaustaprosessi
16-23	local0 – local7

Taulukosta huomataan, että monet koodeista viittaavat vanhoihin Unix-järjestelmiin. Esimerkiksi UUCP:tä ei oikeastaan käytetä enää. UUCP oli epäsynkroninen tapa kopioida tiedostoja automaattisesti palvelinten välillä, jotka pystyivät vaihtamaan dataa keskenään ainoastaan analogisen puhelinverkkoyhteyden välityksellä. Nykyään suurin osa laitteista käyttää yhtä "local"-koodeista Syslog-viesteissään. Esimerkiksi oletusarvoisesti Ciscon ASA-palomuurit käyttävät facility-koodia 20 (local4), kun taas suurin osa Cisco-yhtiön reitittimistä käyttävät facility-koodia 23 (local7). Näitä koodeja käytetään, jotta syslog-palvelimet voivat kätevästi luokitella viestejä eri kategorioihin. Useimpien käyttäjien ei tarvitse muuttaa niitä, koska moderneissa syslog-toteutuksissa facility-koodi on vain yksi monista mahdollisista avainarvoista, joita käytetään viestien etsimiseen tietokannasta. [4.]

3.2 Syslog-viestit

RFC 5424 on protokolla, joka määrittelee Syslogin standardin ja lokiviestien formaatin. Syslog-viestit koostuvat otsikkotiedoista, strukturoidusta datasta sekä viestiosasta. Jokaisen Syslog-viestin alussa on viestin prioriteetti. Prioriteetti merkitään luvulla, joka määräytyy kertomalla facility-koodin numeerinen arvo kahdeksalla ja lisäämällä siihen severity-koodin numeerinen arvo, ja se on ympäröity kulmasulkeilla. Esimerkiksi "<134>" viittaisi ”local0”-laitteiston vakavuustason 6 viestiin [5, s. 10]. Prioriteetin lisäksi Syslog-messagen otsikkotiedoista löytyy versio, aikaleima, verkkotunnus, sovellus, prosessin id sekä viestin id. Otsikkotietojen jälkeen Syslog-viestistä (kuva 1) voi löytyä datalohkoja, joissa on informaatiota "avain=arvo"-formaatissa. Strukturoidun datan jälkeen tulee Syslog-viestin varsinainen viestiosa, jossa on itse lokitettu viesti. [6.]



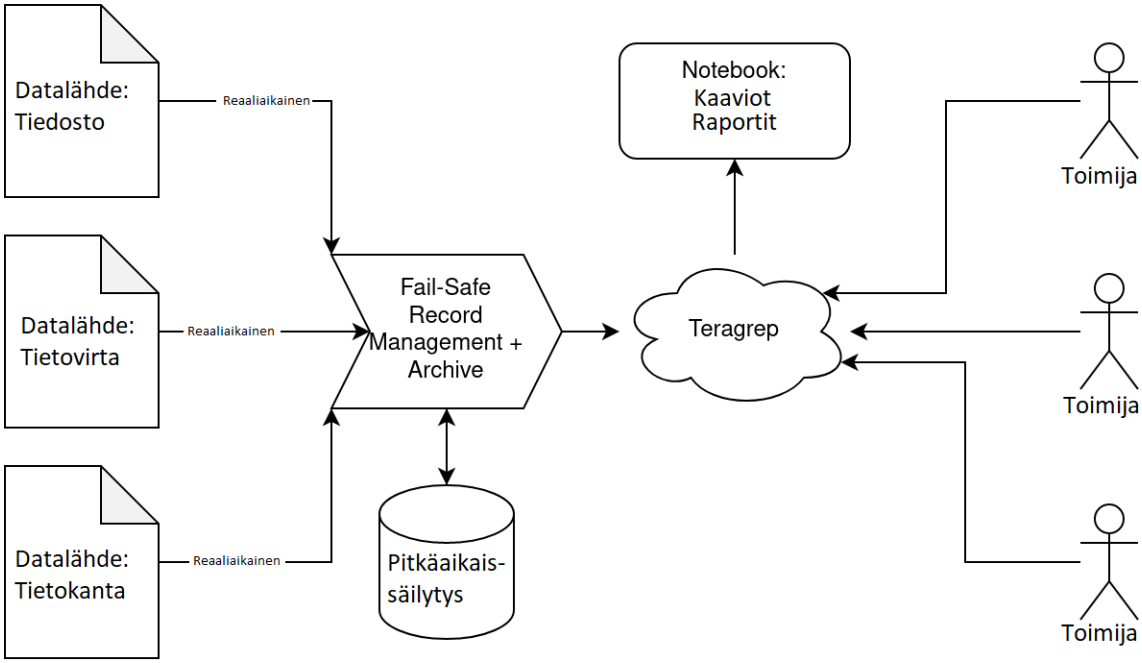
Kuva 1. Syslog-viestiesimerkki.

4 Teragrep

Tapahtumia lokittava taho yleensä haluaa hyödyntää lokitapahtumien tiedot jotenkin. Lokitapahtumia oikein käsitellessä voi saada monenlaista hyödyllistä tietoa, joka voi monesti jopa osoittautua rahallisestikin arvokkaaksi. Lokitapahtumien tutkimiseen ja analysointiin on monia työkaluja. Yksi näistä työkaluista on Fail-Safe IT Solutionsin kehittämä Teragrep-sovellus, josta kerron tässä luvussa.

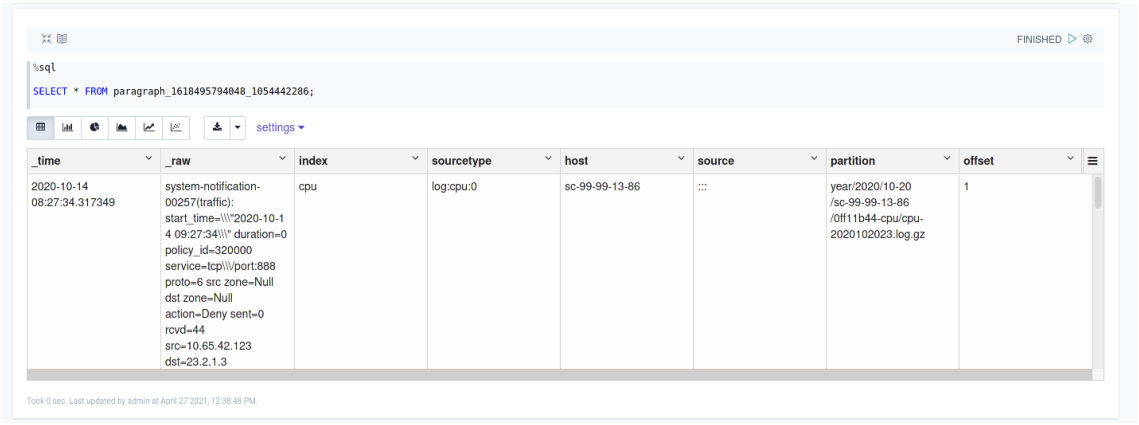
Teragrep on data-analytiikkaan käytettävä sovellus, joka on rakennettu avoimen lähdekoodin ohjelmien Apache Sparkin ja Apache Zeppelinin päälle. Sovellus mahdollistaa niin data-analyytikoiden kuin tavanomaisten käyttäjien tutkailla, analysoida ja raportoida lokidataa. Lokidata syötetään reaaliajassa Teragrepin lähteisiin kuten Fail-Safe Archiveen, johon Teragrep mahdollistaa joustavat haut data-analytiikkaa varten. Prosessointi Fail-Safe Archivesta suorittaa Apache Spark ja sen visualisoi Teragrepin käyttäjärajapinta, joka perustuu Apache Zeppeliniin. Apache Sparkin integraatio mahdollistaa datalle korkean skaalautuvuuden ja toimintavarmuuden. Notebook-toiminnot sisältävät logiikan, jonka sovellus suorittaa tarjotakseen erilaisia taulukoita ja raportteja datasta. Sovelluk-

sen käyttäjät toimivat notebookien kanssa editorin kautta, joka mahdollistaa monen käyttäjän vuorovaikutteisen käytön, tai vaihtoehtoisesti tarkastelevat etukäteen datasta koottuja taulukoita ja raportteja. Kuvassa 2 on ylätason kuvaus siitä, millaisessa ympäristössä Teragrep toimii. [7.]



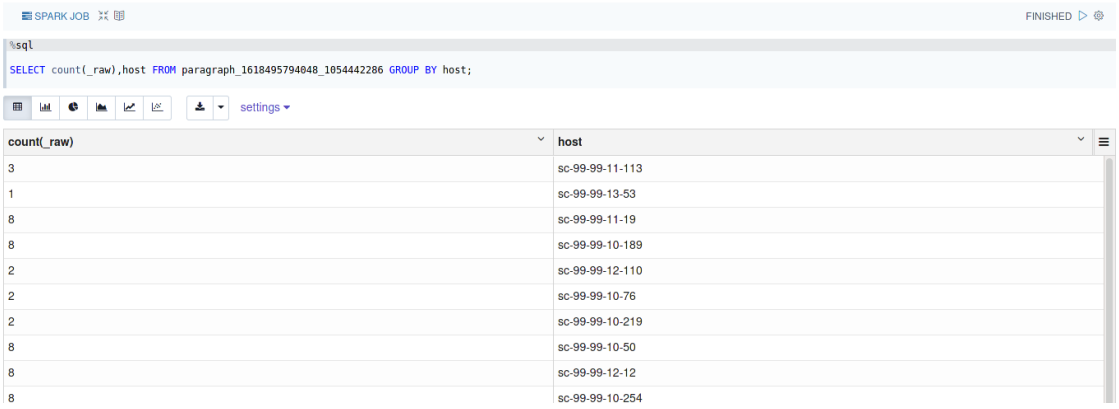
Kuva 2. Teragrepin käyttöympäristö.

Apache Spark, jonka päälle Teragrepin toiminnot on rakennettu, on suurien datamäärien prosessointiin kehitetty ohjelmisto, joka mahdollistaa kokonaisten klustereiden välisen toiminnallisuuden korkealla toimintavarmuudella. Se mahdollistaa joustavat haut erilaisista tietolähteistä käyttäen SQL-hakuja. [8.] Teragrep tarjoaa käyttäjälle suoran yhteyden Fail-Safe Archiveen Java Virtual Machine -kirjastona, joten PySpark (Python) sekä Scala-ohjelmointikieli voivat keskustella suoraan tietovirran kanssa. Kuvassa 3 näytetään esimerkki siitä, miten haettua dataa voidaan tutkailla. Tässä näkyy, kuinka ohjelmisto on automaattisesti jäsennellyt datakokonaisuuden osiin, jotta siitä on helppo nähdä halutut asiat.



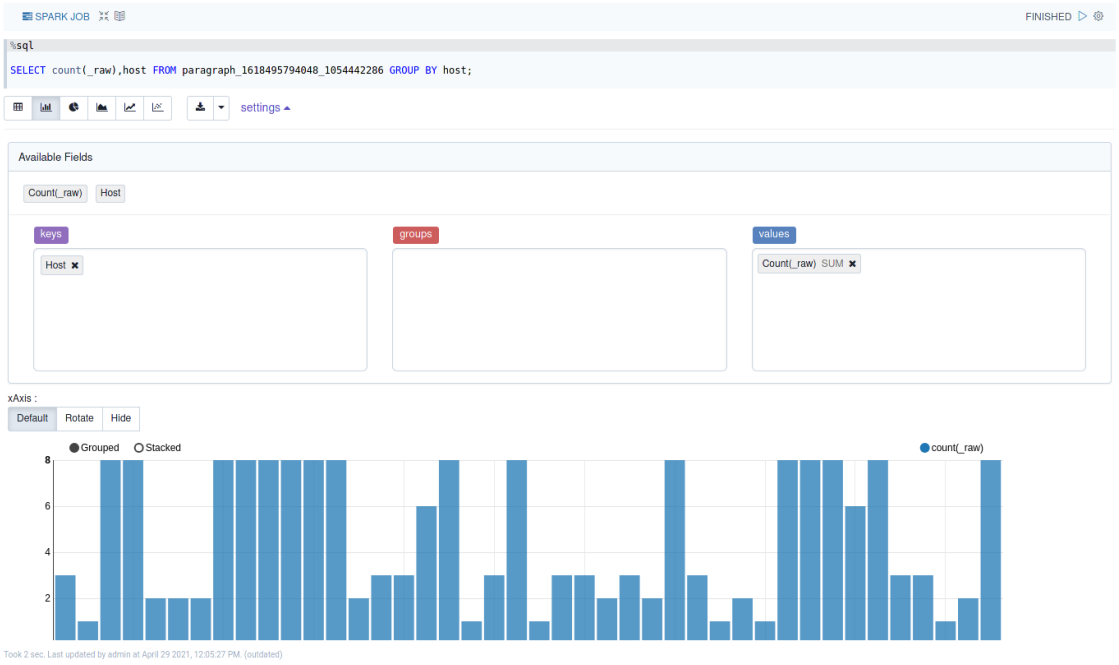
Kuva 3. Yksinkertainen haku Teragrepilla. Yksi datakokonaisuus avattu auki ja jäsennelty osiin.

Teragrepin käyttöliittymä on rakennettu Apache Zeppelinin päälle. Apache Zeppelin on työkalu, jota käytetään datan analysointiin sekä visualisointiin. Apache Zeppelin tarjoaa joustavuutta backendeille, joista dataa haetaan, sillä Apache Zeppelinillä on backend-tuki yli kahdellekymmenelle eri ohjelmointikielelle. [9.] Kuvassa 4 näkyy yksinkertainen ote Teragrepin käyttöliittymässä. Siinä on tiedusteltu SQL-haulla datamääriä ja datakoosteessa on listattu ne palvelinkohtaisesti.



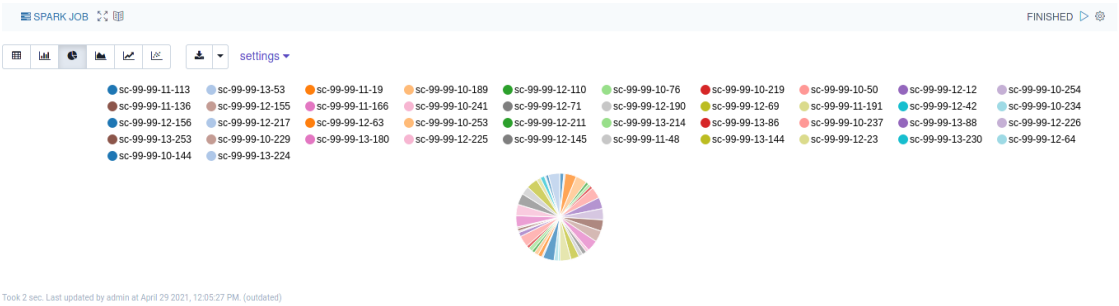
Kuva 4. Datakooste SQL:llä tehdystä hausta. Data on listattu palvelinkohtaisesti.

Kuvassa 5 on siirrytty edellisen kuvan näkymästä seuraavalle välilehdelle, joka tarjoaa haetusta datasta luodun pylväskaavion.



Kuva 5. Teragrepissa datakoosteesta luotu pylväskaavio.

Kuvassa 6 on toisenlainen esimerkki, jossa on käytössä edelleen sama datakooste, kuin edellisissä kuvissa. Tällä kertaa datakoosteesta on luotu ympyräkaavio. Käyttöliittymässä erilaisia koosteita ja kaavioita datakoosteesta saa helposti siirtymällä vain välilehdeltä toiselle.



Kuva 6. Teragrepissa datakoosteesta luotu ympyräkaavio.

5 HTTP Event Capture -sovellus

Fail-Safen tavanomaisessa toteutuksessa asiakkaan palvelimelle asennetaan agentti, joka lähettää asiakkaan palvelimelta Syslog-protokollan mukaiset lokitapahtumat Fail-Safe Record Management -kokonaisuuden CFE-06-komponentille, joka käsittelee lokitapahtumat ja jonka kautta ne lopulta päätyvät Fail-Safe Archive -arkistointitoteutuksen säilytykseen.

Eräällä Fail-Safen asiakkaalla oli tästä poiketen tarve lähettää lokitapahtumia HTTP:llä JSON-muotoisena, joten Fail-Safe Record Management -tuotekokonaisuuteen oli kehitettävä komponentti, joka ottaisi JSON-muotoisen datan vastaan sekä muuttaisi asiakkaan lähetettämän datan sellaiseen muotoon, jossa Fail-Safe Record Managementin muut komponentit voivat sen käsitellä ja lähettää sen arkistointiin.

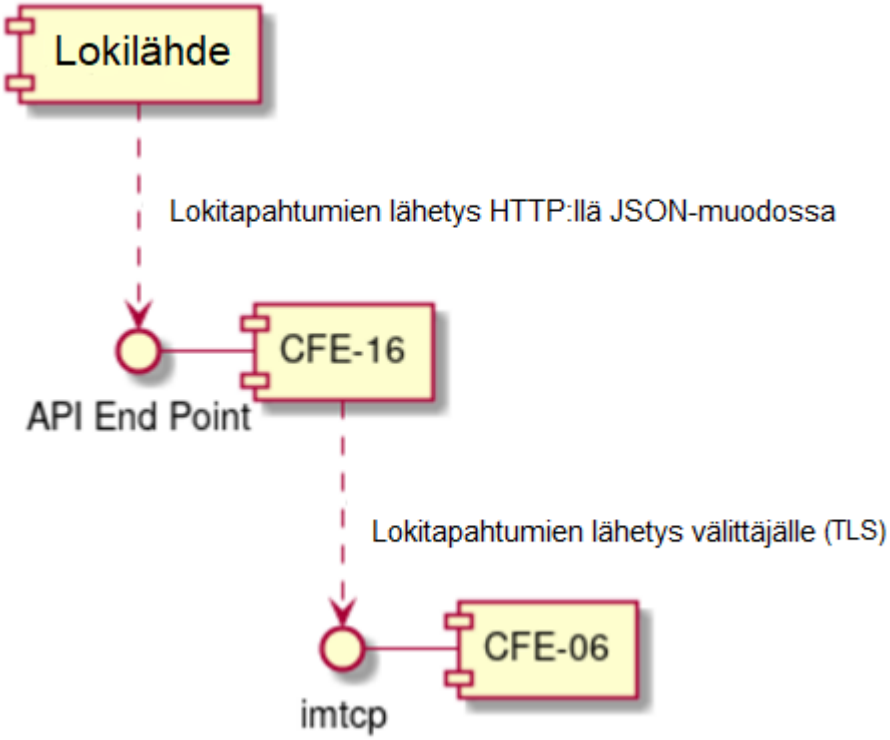
Projekti sai nimekseen HTTP Event Capture (HEC), johon viitataan tekstin aikana myös nimellä CFE-16.

5.1 Suunnitelma – lähtökohdat ja tavoitteet

Projektin tavoitteena oli kehittää ohjelma, joka ottaa vastaan lokitapahtumia HTTP-kutsujen muodossa, muuttaa ne Syslogin RFC5424-formaattiin ja lähettää uudelleenformatoidun lokitapahtuman eteenpäin TLS:n eli tietoliikennettä suojaavan protokollan kautta (kuva 7). Ohjelmaan haluttiin toiminnallisuus, jolla tunnistetaan lokitapahtumien lähettäjä, joten ohjelman tulisi käyttää tokeneita, joilla käyttäjä tunnistettaisiin. Näille käyttäjille tulisi myös luoda prosessointikanavia toiminnan tehostamiseksi.

Käyttäjille haluttiin tarjota mahdollisuus varmistaa, että lähetetty lokitapahtuma on käsitelty loppuun asti. Jokaiselle vastaanotetulle kutsulle, jossa on määritelty erikseen käytettävä kanava, tulisi määrittää "acknowledged"-tunnus (ack). Tämä tunnus palautetaan käyttäjälle lokitapahtuman lähettämisen yhteydessä. Tällä tunnuksella voisi sitten tiedustella, onko lokitapahtuma jo käsitelty ja lähetetty eteenpäin vai onko käsittely vielä kesken. Käyttäjien kanavien ja ack-tunnusten luontia haluttiin kuitenkin rajoittaa ylikuormi-

tuksen estämiseksi, joten ohjelman käyttöönotossa pitäisi voida konfiguroida mahdollisten kanavien enimmäismäärä, ack-tunnusten enimmäisarvo sekä ack-tunnusten enimmäisikä. Komponenttiin tulisi myös lisätä health check endpoint, jolla voi tiedustella, onko ohjelma toiminnassa. [10.]

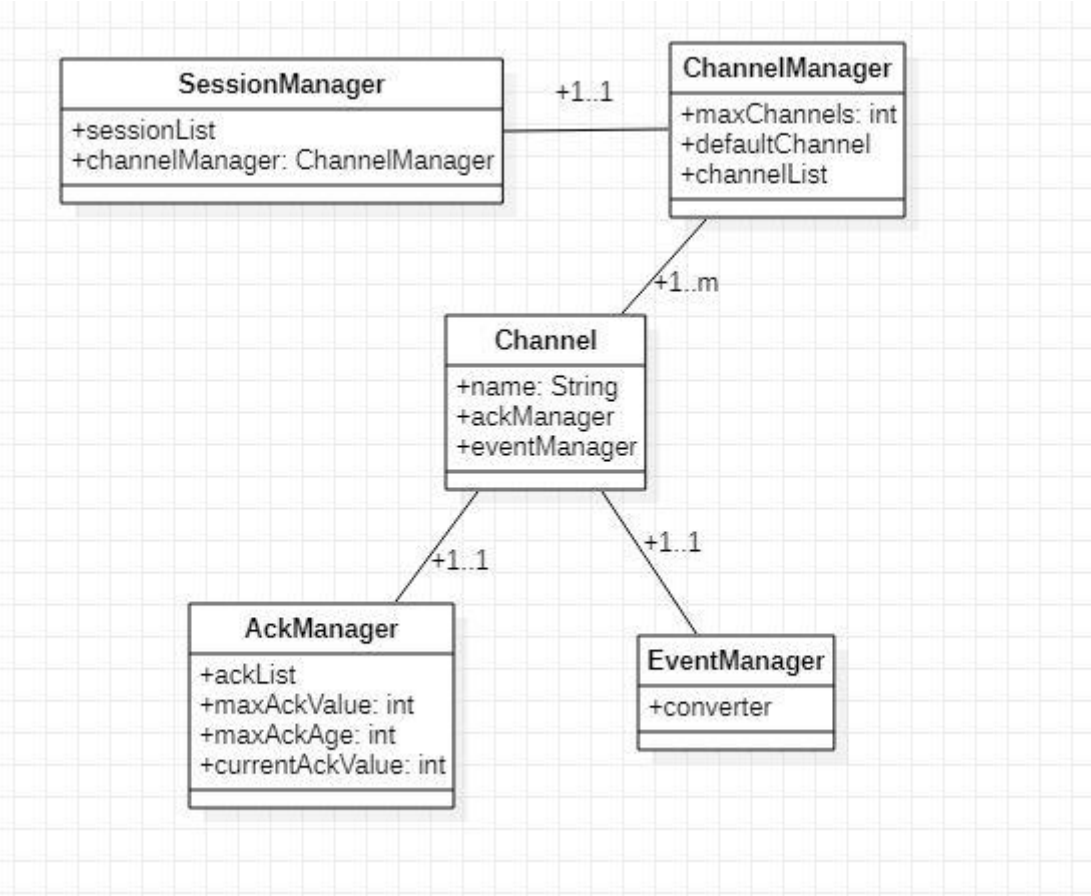


Kuva 7. Ylätason kuva projektin suunnitelmasta.

5.2 Komponentit

HEC-sovelluksessa on paljon erilaisia komponentteja, joiden käyttöä ja ymmärtämistä helpottamaan haluttiin luoda manager-luokka jokaiselle merkittävälle komponentille. Esimerkiksi käyttäjien valtuuttamiseen käytettyjä tokeneita hallitaan TokenManagerilla. Näitä manager-luokkia ohjelmasta löytyy viisi kappaletta: TokenManager, SessionManager, ChannelManager, EventManager sekä AckManager. Näistä luokista kaikki ovat sidoksissa toisiinsa, paitsi TokenManager, joka toimii erikseen muista.

SessionManager-luokka käsittelee sessioita, joita on aina yksi yhtä tunnistukseen käytettävää tokenia kohden. Yhdellä sessiolla voi olla monia kanavia, joten jokainen sessio tarvitsee kanavia hallitsevan ChannelManager-luokan instanssin. Kanavien kautta lähetetään lokitapahtumia ja hallitaan tapahtumien ack-arvoja, joten jokainen kanava tarvitsee lokitapahtumia käsittelevän EventManager-luokan instanssin sekä lokitapahtumien ack-arvoja käsittelevän AckManager-luokan instanssin. Kuvassa 8 on näytetty manageriluokkien väliset suhteet.



Kuva 8. Managerien väliset suhteet.

Seuraavissa luvuissa esittelen manageriluokat yksitellen ja esittelen keskeiset toiminnallisuudet koodiesimerkkien avulla.

5.2.1 TokenManager-luokka

Ohjelman käyttäjät tulee pystyä erottelemaan ja tunnistamaan. Tätä varten jokaisella käyttäjällä, joka kutsuu ohjelmaa, on oma tunnistautumiseen käytettävä token, jota käytetään lokitapahtumia lähettäessä. Tämä token tulee antaa lokitapahtumaa lähettäessä ylätunnisteessa ”authorization”-avainnimikkeellä. TokenManager-luokan tehtävänä on käsitellä tokeneita. Luokassa on isTokenMissing-metodi (esimerkkikoodi 1), joka kertoo, löytyykö sisääntulevasta kutsusta ylätunnisteesta avain "authorization". Jos ylätunnisteesta sellaista ei löydy, tai se löytyy, mutta sille ei ole annettu arvoa, metodi palauttaa arvon true. Muussa tapauksessa metodi palauttaa arvon false. [11.]

```
public boolean tokenIsMissing(HttpServletRequest request) {

    String authHeader = request.getHeader("Authorization");
    return (authHeader == null || authHeader.isEmpty());
}
```

Esimerkkikoodi 1. TokenManager-luokan metodi tarkastaa, onko kutsun ylätunnisteessa annettu autentikaatioon käytettävä token.

5.2.2 SessionManager-luokka

Ohjelmassa halutaan muistaa käyttäjien autentikaatioon käytetyt tokenit. Tämän vuoksi jokaista käyttäjää kohden ohjelmassa luodaan sessio. Toisin sanoen siis jokaista annettua tokenia kohden luodaan Session-olio. Ne sisältävät tokenin, jota käytetään kutsujan tunnistamiseen, sekä ChannelManager-olion, joka hallitsee session kanavia. Näitä olioita hallitsee SessionManager-luokka. Tämä luokka sisältää listan olemassaolevista sessioista. Tätä listaa käytetään tarkastamaan, onko annetulla tokenilla jo luotu sessio, ja jos ei ole, luokka luo sellaisen. Metodiin nimeltä getSession (esimerkkikoodi 2) annetaan parametrina token merkkijonona. Metodi palauttaa null-arvon, jos luokassa oleva sessiolista on tyhjä tai jos annettua tokenista ei löydy sessiolistasta. Jos tokenilla löytyy jo sessio, palautetaan kutsujalle tokenilla löydetty sessio Session-luokan oliona.

```
public Session getSession(String authenticationToken) {
    Session session = null;
    if (sessionList.isEmpty()) {
        return session;
```

```
    } else {
        for (Session s : sessionList) {
            if (s.getAuthenticationToken().equals(authenticationToken)) {
                session = s;
            }
        }
    }
    return session;
}
```

Esimerkkikoodi 2. SessionManager-luokan metodi getSession.

Luokasta löytyy myös createSession-metodi (esimerkkikoodi 3), jolla luodaan uusia sessioita. Tälle metodille annetaan parametrina tunnistautumiseen käytettävä token merkkijonoparametrina. Metodissa luodaan aluksi uusi ChannelManager-olio. Tämän jälkeen metodi luo uuden Session-luokan olion, jolle annetaan parametreina sekä juuri luotu ChannelManager-olio sekä metodille annettu token-merkkijono. Lopuksi metodi lisää luodun session SessionManager-luokan sessiolistaan. [11.]

```
public Session createSession(String authenticationToken) {
    Session session = null;
    channelManager = new ChannelManager(this.props);
    session = new Session(channelManager, authenticationToken);
    sessionList.add(session);
    return session;
}
```

Esimerkkikoodi 3. SessionManager-luokan createSession-metodi

5.2.3 ChannelManager-luokka

Ohjelmassa jokaisella sessiolla on yksi tai useampi kanava, joita hallitaan Channel- sekä ChannelManager-luokilla. Channel-luokka sisältää tiedon kanavan nimestä, AckManager-luokan olion sekä EventManager-luokan olion. ChannelManager-luokka sisältää toiminnallisuuden kanavien luontiin ja niiden etsimiseen sessiosta. Luokka sisältää myös listan sessioon luoduista kanavista sekä tiedon ohjelmaan käyttäjän konfiguroiman kanavien enimmäismäärän. Luokassa on metodi nimeltä isChannelInSession (esimerkki-

koodi 4), joka tarkastaa, löytyykö annettu kanava sessiosta. Metodille annetaan parametrina tarkastettavan kanavan nimi merkkijonona. Metodi palauttaa boolean-arvon true, jos annettu kanava löytyy ChannelManager-luokan kanavalistasta. Muissa tapauksissa, eli jos kanavaa ei löydy listasta tai luokan kanavalista on null-arvoinen, metodi palauttaa boolean-arvon false.

```
public boolean isChannelInSession(String channelName) {
    if(channels == null) {
        return false;
    } else {
        for(Channel c : channels) {
            if (c.getName().equals(channelName)) {
                return true;
            }
        }
    }
    return false;
}
```

Esimerkkikoodi 4. ChannelManager-luokan isChannelInSession-metodi.

ChannelManager-luokan getOrCreateChannel-metodi ottaa vastaan kanavan nimen merkkijonoparametrina. Tässä metodissa ensin tarkastetaan kutsumalla edellämainittua metodia, löytyykö annettu kanava jo session kanavamanagerista. Jos kanava löytyy, kutsutaan seuraavaksi luokan getChannel-metodia, joka palauttaa Channel-luokan olion, jonka sitten getOrCreateChannel-metodi antaa eteenpäin. Jos kanavaa ei löydy kanavalistasta, luodaan sellainen createChannel-metodilla (esimerkkikoodi 5). Tälle metodille niin ikään annetaan parametrina kanavan nimi. Metodissa ensin tarkastetaan, että kanavalistassa on vielä tilaa. Jos lista on saavuttanut suurimman sallitun kanavamäärän, palautetaan käyttäjälle virheviesti. Jos tilaa löytyy, luodaan uusi Channel-olio, jolle annetaan arvoina annettu kanavan nimi sekä uusi AckManager-olio. Tämän jälkeen luodulle kanavalle vielä asetetaan uusi EventManager-luokan olio sekä lisätään kanava ChannelManager-luokan kanavalistaan.

```
public Channel createChannel(String channelName) {
    Channel channel;
    if(channels.size() < maxChannels) {
        channel = new Channel(channelName, new AckManager(props));
        channel.setEventManager(new EventManager(channel.getAckManager(),
props));
        channels.add(channel);
```

```
    }else {
        throw new ServerIsBusyException();
    }
    return channel;
}
```

Esimerkkikoodi 5. ChannelManager-luokan createChannel-metodi.

Channel-olioiden lisäksi ChannelManager-luokka käsittelee myös DefaultChannel-olioita. Näitä käytetään silloin, kun lokitapahtumien lähettäjä ei ole määritellyt kutsussaan kanavaa. DefaultChannel on yksinkertaisempi versio Channel-oliosta, ja se pitää sisällään kovakoodattuna nimensä "defaultchannel" sekä instanssin EventManager-luokasta. Kun ChannelManager-luokan instanssia kutsutaan ensimmäisen kerran, luodaan DefaultChannel-olio, jota käytetään aina seuraavilla kerroilla, kun DefaultChannelia tarvitaan. [11.]

5.2.4 EventManager-luokka

EventManager on luokka, joka käsittelee käyttäjän lähettämiä lokitapahtumia. Luokassa on datan käsittelyn hoitavat metodit convertData sekä convertDataWithDefaultChannel. Molemmat metodit suorittavat suurin piirtein samat toiminnot, mutta convertDataWithDefaultChannel-metodia kutsutaan, kun käyttäjä ei ole määrittänyt käytettävää kanavaa. Tämä metodi siis ei käsittele kanavia eikä Ack-arvoja. EventManager on luokka, jonka sisällä tapahtuu paljon ohjelman oleellisesta toiminnasta, joten tämän luokan toiminnasta kerrotaan tarkemmin luvussa 5.3.2, jossa tarkastelemme ohjelman viestin muuttamisominaisuutta.

5.2.5 AckManager-luokka

Käyttäjät saattavat haluta kysellä lokitapahtuman lähetyksen tilaa. Tämän vuoksi käyttäjän lähettäessä lokitapahtumia käyttäen kanavaa jokaiselle tapahtuman lähetykselle annetaan ack-id. Tätä id:tä voidaan myöhemmin käyttää lähetyksen tilan tiedustelemiseen. Tämä tehdään kysymällä ohjelmalta, onko tietyn ack id:n "acknowledged"-arvo true vai false. Ack-arvot vanhentuvat oletuksena 20 sekunnin kuluttua niiden luomisen jälkeen.

Tämän vanhentumisajan voi konfiguroida ohjelmassa erikseen. Ack-arvojen käsittelyyn ja hallintaan käytetään ohjelmassa Ack- ja AckManager-luokkia.

Ack-luokka sisältää id:n, luokan instanssin luonnin ajankohdan sekä acknowledged-arvon. Luokassa on myös ylikirjoitettu Java-luokan oletusmetodi "equals" (esimerkkikoodi 6). Tämä tehdään sen takia, että halutaan pitää saman ack-id:n omaavat Ack-oliot samanarvoisina.Tämä helpottaa Ackien tarkastelua AckManager-luokassa.

```
@Override
public boolean equals(Object o) {

    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
        return false;
    }
    Ack ack = (Ack) o;
    return id == ack.id;
}
```

Esimerkkikoodi 6. Ack-luokan equals-metodi.

Ack-olioita hallitsee AckManager-luokka, jossa on lista kanavassa luoduista Ack-olioista. Lokien lähetystapahtumalle asettaessa ack-id:tä halutaan tieto siitä, mikä on kanavan AckManager-instanssissa seuraava vapaana oleva ack-id:n arvo. Tämän toiminnallisuuden hoitaa AckManager-luokan metodi getCurrentAckValue (esimerkkikoodi 7). Tässä metodissa ensimmäisenä tarkastetaan, onko luokassa sijaitseva lista Ack-arvoista täynnä. Jos lista on täynnä, palautetaan käyttäjälle virheviesti. Tämän jälkeen metodissa vertaillaan ympäristömuuttujaan asetettua ack-arvoa niin kauan, kunnes vapaana oleva ack-arvo löytyy. Tämän jälkeen arvo tallennetaan muuttujaan, jonka metodi lopulta palauttaa. Ennen palauttamista luokan Ack-listaan lisätään uusi Ack-olio ja vertailuun käytettävää Ack-arvoa kasvatetaan yhdellä, jotta seuraavalla metodin kutsukerralla vapaa Ack-arvo löytyy mahdollisimman nopeasti. Jos vertailuun käytettävä Ack-arvo ylittää konfiguroidun suurimman mahdollisen Ack-arvon, palautetaan vertailuun käytettävä arvo nollaan.

```
public synchronized int getCurrentAckValue() {
    if(!acksAvailable()) {
        throw new ServerIsBusyException();
    }

    ackToCompare.setId(currentAckValue);

    while(ackList.contains(ackToCompare)) {
        currentAckValue++;
        ackToCompare.setId(currentAckValue);
        if(currentAckValue > maxAckValue) {
            currentAckValue = 0;
        }
    }

    int valueToReturn = currentAckValue;
    ackList.add(new Ack(currentAckValue, true, false));
    currentAckValue++;
    if(currentAckValue > maxAckValue) {
        currentAckValue = 0;
    }
    return valueToReturn;
}
```

Esimerkkikoodi 7. AckManager-luokan getCurrentAckValue-metodi.

Edellä mainittua Ack-olioiden vertailuun käytettävää ympäristömuuttujaa tarvitaan myös, kun tarkastetaan, onko tietyn id:n omaavan Ack-olion "acknowledged"-tila true vai false. AckManager-luokan isAcknowledged-metodia (esimerkkikoodi 8) kutsuttaessa annetaan parametrina ack-id, jonka tilaa tiedustellaan. Tämän id:n arvo annetaan vertailuun käytettävälle ympäristömuuttujalle, jonka jälkeen tarkastetaan luokan Ack-listasta, löytyykö kysyttyä Ack-oliota. Jos kysyttyä Ack-oliota ei löydy listasta, palauttaa metodi arvon false. Jos taas Ack-olio löytyy listasta, palautetaan löydetyn Ack-olion "acknowledged"-tilan arvo.

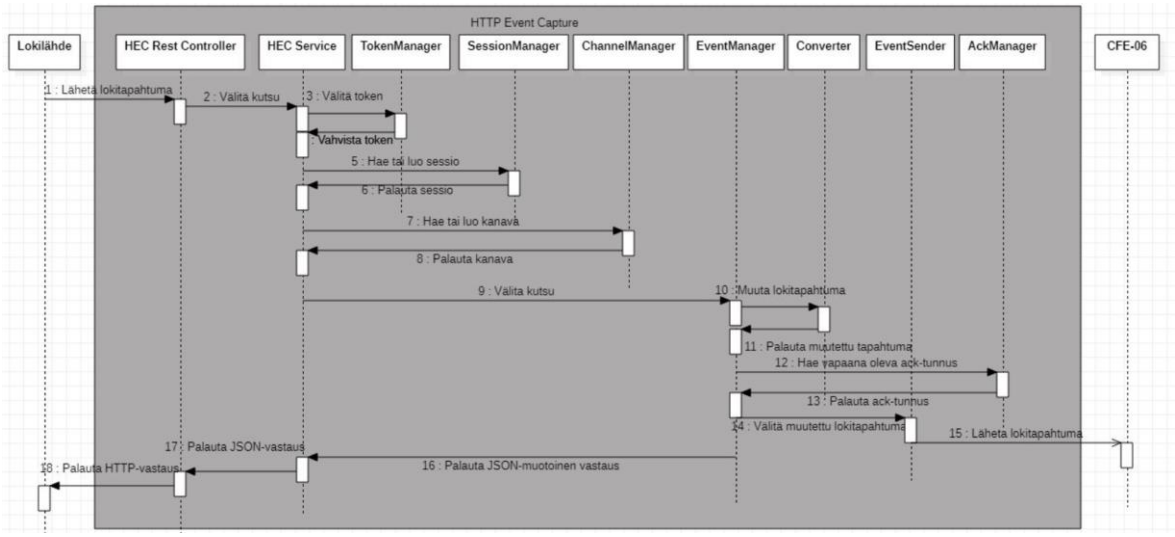
```
public synchronized boolean isAckAcknowledged(int ackId) {
    ackToCompare.setId(ackId);
    if(ackList.contains(ackToCompare)) {
        return ackList.get(ackList.indexOf(ackToCompare)).isAcknowledged();
    } else {
        return false;
    }
}
```

Esimerkkikoodi 8. AckManager-luokan isAcknowledged-metodi.

AckManager-luokassa pyörii taustalla Evictor-luokan instanssi. Evictor-luokan tehtävänä on tietyin väliajoin käydä AckManager-luokan Ack-listaa läpi ja poistaa listasta vanhentuneet Ack-oliot. [11.]

5.3 Toiminnallisuus

Nyt kun ohjelman tärkeimmät pääkomponentit ovat tuttuja, on aika tutustua, mitä tapahtuu, kun käyttäjä lähettää sovellukselle lokitapahtumia sekä miten ohjelma käsittelee, muuttaa ja lähettää lokitapahtumia eteenpäin edellisissä kuvattujen komponenttien avulla. Kuvassa 9 on kuvattu visuaalisesti, mitä ohjelman sisällä tapahtuu, kun sille lähetetään lokitapahtumia. Kuvassa nähdään, miten ohjelma käsittelee käyttäjän lähettämää kutsua, missä vaiheessa ohjelma lähettää lokitapahtuman eteenpäin ja mitä käyttäjälle palautetaan.



Kuva 9. Sekvenssikaavio, jossa kuvataan ohjelman toiminta, kun sille lähetetään lokitapahtumia.

Seuraavissa luvuissa kerron ohjelman toiminnasta ja esittelen keskeiset toiminnallisuudet koodiesimerkkien avulla.

5.3.1 Lokitapahtuman vastaanottaminen

HEC-sovellusta käytettäessä käyttäjälle, joka haluaa lähettää lokitapahtumia, pitää luoda API endpoint, johon tapahtumia voidaan lähettää. Tätä varten ohjelma sisältää REST-kontrollerin, jossa on metodeja, jotka ottavat vastaan REST-kutsuja. Kutsuja lähettää HTTP-lokitapahtuman API endpointiin services/collector (kuva 9:1). Kontrollerissa metodi sendEvents käsittelee kutsujan antamia lokitapahtumia lähetettäväksi. Samalla sendEvents-metodi tallentaa statistiikkaa vastaanotetuista lokitapahtumista.

Kontrolleri lähettää lokitapahtuman eteenpäin servicelle (kuva 9:2), jossa HTTP-kutsusta ensimmäisenä tarkistetaan, onko kutsussa annettu tunnistautumiseen käytettävä token (kuva 9:3). Jos token puuttuu, palautetaan kutsujalle virheviesti, jossa kerrotaan, että token vaaditaan, mutta jos taas token löytyy, ohjelma jatkaa suorittamista (kuva 9:4). Seuraavaksi service tarkistaa, onko annetulla tokenilla olemassa jo sessio. Jos sessiota ei ole vielä, sellainen luodaan tokenille. Jos taas sessio löytyy jo, käytetään tätä (kuva 9:5–6). Jokaisella tokenilla on käytössään sessio, jossa on sisällä ChannelManager-luokan instanssi, joka hallinnoi tokenin käyttämiä kanavia. Service tarkastaa tämän jälkeen, onko kutsuja määritellyt kanavan kutsussaan. Jos kanava on annettu, tarkastetaan, onko kanava luotu jo aiemmin vai onko annettu kanava uusi (kuva 9:7–8). Kanavan ollessa luotu, käytetään tämän kanavan EventManager-luokan instanssia muuttamaan lokitapahtuma, mutta jos annettua kanavaa ei ole valmiina, sellainen luodaan, ja sille luodaan uusi EventManager-luokan instanssi ja käytetään tätä (kuva 9:9). Siinä tapauksessa, jos kanavaa ei ole määritelty ollenkaan, käytetään DefaultChannel-luokan instanssia ja sen EventManageria. Tämä toiminnallisuus on esitelty esimerkkikoodissa 9. [11.]

```
if(channel != null) {
    sessionChannel = session.getChannelManager().getOrCreateChannel(channel);
    ackNode      =      sessionChannel.getEventManager().convertData(authToken,
eventInJson, channel, headerInfo);
}else {
    //channel is not given. use a default channel
    DefaultChannel defaultChannel = session.getChannelManager().getDefaultChan-
nel(props);
    ackNode    =    defaultChannel.getEventManager().convertDataWithDefaultChan-
nel(authToken, eventInJson, defaultChannel.getName(), headerInfo);
}
```

Esimerkkikoodi 9. Tapahtumamanagerin hakeminen kanavasta ja lokitapahtuman muuttaminen servicessä. Vaihtoehtoisesti muutetaan tapahtuma ilman kanavaa.

5.3.2 Viestin muuttaminen

Tapahtumamanageri ottaa annetun kutsun vastaan käyttäen joko metodia convertData tai convertDataWithDefaultChannel riippuen siitä, kutsuuko EventManageria Channel- vai DefaultChannel-olio. Lokitapahtumat annetaan metodiin string-parametrina, joka on JSON-formaatissa. Tämä string-parametri annetaan JsonStreamParser-oliolle, joka alkaa parsimaan string-parametria yksi lokitapahtuma kerrallaan.

EventManager-luokassa on esikäsittelymetodi (esimerkkikoodi 10), joka tarkastaa, että tapahtuma on kunnollisessa JSON-formaatissa. Tämä metodi tarkastaa myös, että annetussa tapahtumassa on kenttä, jolle on annettu nimi "event", ja että "event"-kenttä ei ole tyhjä. EventManager käsittelee myös tapahtumassa annetun tapahtuma-ajan. Tapahtuma-aika tulee antaa epoch-time-muodossa. Jos tapahtuma-aikaa ei ole annettu kutsussa, luodaan aikaleima nykyhetkestä. Jos tapahtuma-aika on annettu merkkijonona, eikä numeraalisena arvona, käsitellään tapahtuma kuin aikaa ei olisi annettu. Esikäsittelymetodi palauttaa HttpEventData-olion, joka sisältää merkkijonona tapahtuman sisällön, aikaleiman, ja tiedon siitä, onko aikaleima parsittu tapahtumasta vai onko se generoitu nykyhetkestä. Esikäsittelymetodin palauttamaan HttpEventData-olioon määritetään metadatan tiedot, jotka löydetään annetusta kutsusta. Tämä tarkoittaa siis, että tallennetaan tunnistautumiseen käytettävä token sekä kanava HttpEventData-olion muuttujiin.

```
if (jsonObject.get("event") != null) {
    eventData.setEvent(jsonObject.get("event").toString());
} else {
    throw new EventFieldMissingException();
}
if (eventData.getEvent().matches("\"\"") || eventData.getEvent() == null) {
    throw new EventFieldBlankException();
}
```

Esimerkkikoodi 10. EventManager-luokan esikäsittelymetodin toimintaa. JSON-objektista etsitään ”event”-kenttä. Jos sellaista ei löydy, annetaan EventFieldMissingException-virhe. Jos kenttä löytyy, mutta se on tyhjä, annetaan EventFieldBlankException-virhe.

Tämän jälkeen tapahtuman data lähetetään Converter-luokan käsiteltäväksi (kuva 9:10). Tämä luokka muuttaa annetun datan Syslog-viestin muotoiseksi. Converter-luokka luo

strukturoidun dataelementin, johon annetaan tapahtuman metadatan tiedot strukturoituina dataparametreina. strukturoituihin dataparametreihin annetaan tiedoksi autentikaatiotoken, kanava, ack-tunnus, aikaleima, ja tieto siitä, onko aikaleima otettu lokitapahtumasta itsestään, vai onko se generoitu nykyhetkestä (esimerkkikoodi 11). Tämän jälkeen Converter-luokka luo SyslogMessage-olion, jolle annetaan tiedoksi aikaleima, luotu strukturoitu Data Element sekä lokitapahtuman sisältö, joka saadaan HttpEventData-oliosta.

```
private void setStructuredDataParams(HttpEventData eventData) {

    metadataSDE = new SDElement("CFE-16-metadata@48577");

    if(eventData.getAuthenticationToken() != null) {
        metadataSDE.addSDParam("authentication_token", eventData.getAuthenti-
cationToken());
    }
    if(eventData.getChannel() != null) {
        metadataSDE.addSDParam("channel", eventData.getChannel());
    }
    if(eventData.getAckID() != null) {
        metadataSDE.addSDParam("ack_id", String.valueOf(event-
Data.getAckID()));
    }
    if(eventData.getTimeSource() != null) {
        metadataSDE.addSDParam("time_source", eventData.getTimeSource());
    }
    if(eventData.isTimeParsed()) {
        metadataSDE.addSDParam("time_parsed", "true");
        metadataSDE.addSDParam("time", eventData.getTime());
    }
}
```

Esimerkkikoodi 11. Converter-luokassa setStructuredParams-metodi asettaa HttpEventData-olioon tallennetut tiedot strukturoidun dataelementin sisälle strukturoituihin dataparametreihin. Tätä strukturoitua dataelementiä käytetään Syslog-viestin luomiseen.

Converter-luokka palauttaa tämän SyslogMessage-olion EventManagerille (kuva 9:11), joka tallentaa tämän, ja jatkaa loppujen kutsussa olleiden lokitapahtumien käsittelyä niin kauan kuin niitä riittää. Sen jälkeen, kun kaikki kutsussa olleet lokitapahtumat on käsitelty, haetaan lokitapahtumille ack-tunnus (kuva 9:12–13). Tämän jälkeen siirrytään käsiteltyjen lokitapahtumien lähettämiseen (kuva 9:14). Samalla kun ohjelma lähettää lokitapahtumia, palautetaan kutsujalle tieto siitä, että tapahtuma on käsitelty onnistuneesti (kuva 9:16–18). Jos kanava oli annettu kutsussa, palautetaan kutsujalle myös ack-tunnus. [11.]

5.3.3 Lokitapahtuman lähettäminen

EventSender-luokalle lähetetään lista SyslogMessage-olioita. Jos kutsussa on määritelty kanava, annetaan EventSender-luokkaa kutsuttaessa parametreina myös kyseisen session Ack-manageri. EventSender-luokassa luodaan SysLogMessageSender-olio, jota käytetään tapahtumien lähettämiseen. SyslogMessageSender-oliolle asetetaan SyslogServerHostname sekä SyslogMessageServerPort, jotka sisältävät tiedot siitä, minne palvelimelle ja mihin porttiin lokitapahtumat lähetetään. Nämä tiedot konfiguroidaan application.properties-tiedostossa. SyslogMessageSender-oliolle asetetaan myös viestin formaatti, joka on tässä ohjelmassa aina RFC5424. Tämän jälkeen lokitapahtumat lähetetään SyslogMessage-listasta yksitellen eteenpäin (kuva 9:15). Jos kanava on käytössä, lokitapahtumien lähettämisen jälkeen session AckManager-luokan instanssi asettaa lähetyksen tilaksi "acknowledged" (esimerkkikoodi 12). [11.]

```
public void sendEvents(List<SyslogMessage> convertedMessages, AckManager ack-
Manager, int ackId) {

    messageSender.setSyslogServerHostname(props.getSyslogHost());
    messageSender.setSyslogServerPort(props.getSyslogPort());
    messageSender.setMessageFormat(MessageFormat.RFC_5424);
    messageSender.setSsl(false);

    for(SyslogMessage syslogMessage : convertedMessages) {

        /*
         * After the event is sent successfully,
         * the event is acknowledged by the AckManager
         */
        try {
            messageSender.sendMessage(syslogMessage);
            ackManager.acknowledge(ackId);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Esimerkkikoodi 12. EventSender-luokan sendEvents-metodi.

5.3.4 Lähetetyn lokitapahtuman tilan tiedusteleminen

Kuten luvussa 5 todettiin, kutsujalla on mahdollisuus tiedustella lähetettyjen lokitapahtumien tilaa ack-tunnusten avulla. Tiedustelua varten käyttäjä kutsuu API endpointia services/collector/ack. Kutsussa annetaan JSON-objektin sisällä ne ack-tunnukset, joita vastaavien tapahtumien tilaa käyttäjä haluaa tiedustella. Tämän lisäksi käyttäjän tulee antaa kutsussa tunnistautumiseen käytettävä token sekä kanava, joita käyttäen lokitapahtumat on lähetetty.

Kuten lokitapahtumia lähettäessä REST-kontrolleri ottaa käyttäjän kutsun vastaan ja lähettää sen service-kerrokselle. Service tarkastaa, onko kutsussa annettu autentikaatiotoken sekä kanava. Tässä vaiheessa tarkistetaan myös, että annetulla tokenilla löytyy käynnissä oleva sessio. Sessiosta tarkistetaan myös, että sieltä löytyy kutsujan antama kanava. Jos joku näistä tarkastuksista ei onnistu, palautetaan käyttäjälle virheviesti, missä kerrotaan, mikä kutsussa on vikana. Siinä tapauksessa, että kutsussa on kaikki kunnossa, service-kerroksessa kutsutaan session kanavamanageria, josta haetaan haluttu kanava. Service-kerroksessa tapahtuva koodi on esitelty esimerkkikoodissa 13.

```
if(channel != null) {
    if(!tokenManager.tokenIsMissing(request)) {
        Session session = null;
        String authToken = "";
        String authHeader = request.getHeader("Authorization");

        if(tokenManager.isTokenInBasic(authHeader)) {
            authToken = tokenManager.getTokenFromBasic(authHeader);
        }else {
            authToken = authHeader;
        }

        session = sessionManager.getSession(authToken);

        if(session != null) {
            if(session.getChannelManager().isChannelInSession(channel)) {

                responseNode = objectMapper.createObjectNode();
                requestedAckStatuses = (ObjectNode) session.getChannel-
Manager().getChannel(channel).getAckManager().getRequestedAckStatuses(re-
questedAcksInJson);
                responseNode.put("acks", requestedAckStatuses);
            }else {
                throw new ChannelNotFoundException("Provided channel was not
found in this session. Channel: " + channel);
            }
        }else {
            throw new SessionNotFoundException("Session with provided authen-
tication token was not found: Authentication token: " + authToken);
```

```
        }
    }else {
        throw new AuthenticationTokenMissingException("Authentication token
must be provided");
    }
} else {
    throw new ChannelNotProvidedException("Channel must be provided when re-
questing ack statuses");
}
```

Esimerkkikoodi 13. Käyttäjän kutsun käsittelyä service-kerroksessa. Koodissa tarkastetaan kutsujan antama token ja kanava. Käyttäjän kutsu lähetetään kanavan EventManager-luokan instanssille käsittelyyn. Esimerkkikoodissa näkyy myös, millaisia virheitä voi tapahtua, jos käyttäjän kutsu on puutteellinen.

Käyttäjän kutsussa antaman kanavan AckManager-luokan instanssista kutsutaan metodia getRequestedAckStatuses, jolle annetaan JsonNode-luokkaparametrina ack-tunnukset, joiden tilaa käyttäjä on tiedustellut. Metodi hakee AckManagerista listan, johon on tallennettu käytetyt ack-tunnukset ja niiden tilan. AckManager palauttaa service-kerrokselle JsonNode-luokkaan kuuluvan olion, johon on tallennettu tiedustelluiden ackien tunnukset ja tilat (eimerkkikoodi 14). Service palauttaa tämän JsonNode-olion REST-kontrollerille, josta se lähetetään käyttäjälle.

```
for(int i = 0; i < requestedAckIds.length; i++) {
    ackToCompare.setId(requestedAckIds[i]);
    if(ackList.contains(ackToCompare)) {
        ack = ackList.get(ackList.indexOf(ackToCompare));

        ackStatuses.put(
                Integer.toString(ack.getId()),
                ack.isAcknowledged());
        deleteAckFromList(ack);
    }else {
        ackStatuses.put(
                Integer.toString(ackToCompare.getId()),
                false);
    }
}
```

Esimerkkikoodi 14. AckManager-luokan toiminnallisuutta. For-silmukassa haetaan jokaisen halutun Ack-olion tila ja tallennetaan se ackStatuses-listaan.

Lokitapahtumien tilan kutsuminen tarjoaa käyttäjälle suhteellisen rajoitetun määrän tietoa. Ackin tilana kerrotaan "true" silloin, jos ack-tunnukselle määritetty tapahtuma on lähetetty onnistuneesti ja jos tilan tiedustelu on tehty tietyn aikavälin puitteissa. Ackin tila

siis voi vanhentua tietyn ajan päästä, jolloin tämän ackin tunnuksen arvo vapautetaan takaisin käyttöön. Ackien vanhentumisaika voidaan määrittää erikseen application.properties-tiedostossa. Ackin tilana kerrotaan olevan "false", jos ack-tunnus on ehtinyt vanhentua, tai jos haetulla ack-tunnuksella ei ole ollutkaan mitään arvoa, tai vaikka tiedusteltu ack-tunnus ylittäisi määritellyn ack-tunnuksien maksimiarvon. [11.]

5.4 Lopputulos ja jälkipohdinnat

HEC-sovellus saatiin valmiiksi, ja se on käytössä sekä testi- että tuotantoympäristöissä. Tässä luvussa näytän esimerkkejä erilaisista sovelluksen käyttötapauksista sekä lopuksi pohdin projektia jälkeenpäin.

5.4.1 Käyttöesimerkit

Kun sovellus ottaa lokitapahtumia vastaan, halutaan tunnistaa tapahtumien lähettäjä. Käyttäjältä vaaditaan myös, että hänellä todella on joku tapahtuma lähetettävänä. Näistä syistä pakollisia asioita käyttäjälle on määrittää kutsun otsikkotiedoissa Authorization-niminen kenttä, sekä itse viestin JSON-osassa kenttä, jonka nimi on "event". Tämä event-niminen kenttä ei saa myöskään olla pelkkä tyhjä kenttä. Jos käyttäjä haluaa tiedustella lähetetyn tapahtuman tilaa lähettämisen jälkeen, on hänen myös määriteltävä kanava. Tällöin ohjelman palautusviestissä annetaan ack-tunnus, jolla tilaa voi tiedustella.

Lokitapahtumien lähettäminen – onnistunut kutsu

Kuvassa 10 näkyy tavanomainen ohjelmaan lähetetty kutsu. Tässä käyttäjä kutsuu API endpointia "services/collector", määrittää kanavaksi "CHANNEL_11111" ja antaa tunnistamiseen käytettävän tokenin "AUTH_TOKEN_11111". Kutsun varsinaisessa JSON-osassa käyttäjä on antanut kaksi avain-arvo-paria, joista tärkeämpi on nimeltään "event". Tästä kentästä sovellus ottaa lokitapahtuman varsinaisen viestiosan. Vastauksena onnistuneessa kutsussa sovellus antaa käyttäjälle "Success"-tekstin sekä koodin 0, joka myös viittaa onnistuneeseen kutsuun. Tässä tapauksessa, kun käyttäjä oli määrittänyt

kanavan, palauttaa sovellus myös ack-tunnuksen. Jos kanavaa ei ole määritelty, palauttaa sovellus ainoastaan onnistuneeseen kutsuun viittaavan tekstin ja koodin. [12.]

JSON-kutsu:

```
curl -k "http://localhost:8080/services/collector?channel=CHANNEL_11111" \
    -H "Authorization: AUTH_TOKEN_11111" \
    -d '{"sourcetype": "mysourcetype", "event": "Hello, world!"}'
```

JSON-vastaus:

```
{"text":"Success","code":0,"ackID":0}
```

Kuva 10. Esimerkki ohjelmaan lähetetystä onnistuneesta kutsusta.

Lokitapahtumien lähettäminen ilman autentikaatiotokenia

Kuvassa 11 on esimerkki käyttäjän tekemästä virheestä, jossa koitetaan lähettää lokitapahtumaa ilman, että ollaan määritelty käyttäjän tunnistamiseen käytettävää tokenia. Vastauksena sovellus antaa tekstin, jossa kerrotaan, että token on pakollinen, sekä tämän tyyppiseen virheeseen viittaavan koodin 2. Koska sovellukselle voidaan lähettää monia lokitapahtumia kerralla, kerrotaan, minkä tapahtuman kohdalla virhe tapahtuu. Tässä tapauksessa kuitenkin, koska tunnistautumiseen käytettävä token on löydyttävä kutsun otsikkotiedoista, virheellisen tapahtuman kerrotaan olevan ensimmäinen lähetetty tapahtuma eli viitataan indeksiin 0. [12.]

JSON-kutsu:

```
curl -k "http://localhost:8080/services/collector?channel= CHANNEL_11111" \
    -d '{"sourcetype": "mysourcetype", "event": "Hello, world!"}'
```

JSON-vastaus:

```
{"text":"Token is required","code":2,"invalid-event-number":0}
```

Kuva 11. Käyttöesimerkki, jossa yritetään lähettää lokitapahtumia määrittelemättä tunnistautumiseen käytettävää tokenia.

Lokitapahtuman lähettäminen ilman event-kenttää

Kuvassa 12 näytetään, mitä tapahtuu, kun käyttäjä yrittää lähettää lokitapahtuman, mutta kutsun JSON-osassa ei ole annettu kenttää nimeltä "event". Tässä tapauksessa sovellus palauttaa tekstinä tiedon siitä, että event-kenttä on pakollinen sekä koodin 12, joka viittaa tämän tyyppiseen virheeseen. Palautetussa viestissä kerrotaan myös, mitä tapahtumaa käsitellessä virhe tapahtui. [12.]

JSON-kutsu:

```
curl -k "http://localhost:8080/services/collector" \
    -H "Authorization: AUTH_TOKEN_11111" \
    -d '{"sourcetype": "mysourcetype"}'
```

JSON-vastaus:

```
{"text":"Event field is required","code":12,"invalid-event-number":0}
```

Kuva 12. Käyttöesimerkki, jossa yritetään lähettää lokitapahtuma ilman ”event”-nimistä kenttää.

Lokitapahtuman lähettäminen – tyhjä event-kenttä

Seuraava esimerkki (kuva 13) on samankaltainen kuin edellinen tapaus, mutta tällä kertaa "event"-niminen kenttä sisältää ainoastaan tyhjän arvon. Tässä tapauksessa palautetaan käyttäjälle tekstinä tieto siitä, että event-kenttä ei saa olla tyhjä sekä koodin 13, joka viittaa tämän tyyppiseen virheeseen. Palautetussa viestissä kerrotaan myös, mitä tapahtumaa käsitellessä virhe tapahtui. [12.]

JSON-kutsu:

```
curl -k "http://localhost:8080/services/collector?channel=CHANNEL_11111" \
    -H "Authorization: AUTH_TOKEN_11111" \
    -d '{"sourcetype": "mysourcetype", "event": ""}'
```

JSON-vastaus:

```
{"text":"Event field cannot be blank","code":13,"invalid-event-number":0}
```

Kuva 13. Käyttöesimerkki, jossa yritetään lähettää lokitapahtuma, mutta event-kentän arvo on tyhjä.

Lähetetyille lokitapahtumille annetaan ack-tunnus ainoastaan, jos tapahtumaa lähetettäessä on määritelty kanava, jonka kautta lokitapahtuma lähetetään. Tästä syystä lokitapahtuman tilaa tiedustellessa kanavan määrittäminen kutsussa on pakollista. Luonnollisesti pakollista on myös määritellä, minkä ack-tunnuksen omaavien tapahtumien tilaa halutaan tiedustella. Lokitapahtumien tilaa tiedustellessa onnistuneesti sovellus palauttaa jokaista tiedusteltua ack-tunnusta kohden joko true- tai false-arvon. Ack-tunnuksen arvon ollessa true tarkoittaa se, että lokitapahtuma on käsitelty onnistuneesti. Ack-tunnuksen arvon ollessa false tarkoittaa se, että lokitapahtuman tila on tuntematon. Tämä voi tarkoittaa sitä, että tiedustellun ack-tunnuksen omaavaa lokitapahtumaa käsitellessä on tapahtunut virhe, sellaista ei ole ikinä annettu tai ack-tunnus on vanhentunut. [12.]

Lokitapahtuman tilan tiedusteleminen – onnistunut kutsu

Kuvassa 14 on esimerkki onnistuneesta lokitapahtumien tilan tiedustelusta. Käyttäjä kutsuu API endpointia "services/collector/ack" ja määrittää sekä kanavan "CHANNEL_11111" että tunnistautumiseen käytettävän tokenin "AUTH_TOKEN_11111". Kutsun JSON-viestiosassa käyttäjä määrittelee ack-tunnukset, joiden tilaa halutaan tiedustella. Vastauksena sovellus antaa ack-tunnusten tilat joko arvona true tai false. Tässä tapauksessa ack-tunnukset 1 ja 3 on onnistuneesti käsitelty, ja ack-tunnuksen tila 4 on tuntematon. [12.]

JSON-kutsu:

```
curl -k "http://localhost:8080/services/collector/ack?channel=CHANNEL_11111" \
  -H "Authorization: AUTH_TOKEN_11111" \
  -d '{"acks": [1,3,4]}'
```

JSON-vastaus:

```
{"acks":{"1":true,"3":true,"4":false}}
```

Kuva 14. Käyttöesimerkki lokitapahtumien tilan onnistuneesta tiedustelemisesta.

Kuvassa 15 on esimerkki siitä, kun käyttäjä yrittää tiedustella lokitapahtumien tilaa, mutta ei ole määritellyt kanavaa. Vastauksena sovellus antaa tiedon siitä, että kanava puuttuu kutsusta sekä tämän tyyppiseen virheeseen viittaavan koodin 10. Ohjelmassa on virheviesteille oma malli, johon kuuluu aina tieto siitä, minkä lokitapahtuman kohdalla virhe on

tapahtunut. Koska lokitapahtumien tilaa tiedustellessa ei lähetetä tapahtumia, näissä tapauksissa palautetaan aina arvo 0. [12.]

JSON-kutsu:

```
curl -k "http://localhost:8080/services/collector/ack" \
    -H "Authorization: AUTH_TOKEN_11111" \
    -d '{"acks": [1,3,4]}'
```

JSON-vastaus:

```
{"text":"Data channel is missing","code":10,"invalid-event-number":0}
```

Kuva 15. Käyttöesimerkki, jossa tiedustellaan lokitapahtumien tilaa, mutta kanavaa ei ole määritelty.

Lokitapahtuman tilan tiedustelu – ei sessiota

Lokitapahtumien tilaa tiedustelevaa kutsua vastaanottaessa oletetaan luonnollisesti, että annetulla tokenilla on aikaisemmin lähetetty lokitapahtumia, ja sille on ohjelmassa luotu sessio. Kuvassa 16 näkyy esimerkki tilanteesta, jossa tiedustellaan lokitapahtumien tilaa, mutta annetulla tokenilla ei löydy sessiota. Tässä tapauksessa palautetaan käyttäjälle tekstinä tieto siitä, että annettu token on viallinen sekä tämän tyyppiseen virheeseen viittaava koodi 4. [12.]

JSON-kutsu:

```
curl -k "http://localhost:8080/services/collector/ack?channel=CHANNEL_11111" \
    -H "Authorization: AUTH_TOKEN_22222" \
    -d '{"acks": [1,3,4]}'
```

JSON-vastaus:

```
{"text":"Invalid token","code":4,"invalid-event-number":0}
```

Kuva 16. Käyttöesimerkki, jossa tiedustellaan lokitapahtumien tilaa, mutta ohjelmassa ei ole annetulle tokenille luotua sessiota.

Lokitapahtuman tilan tiedustelu – ei kanavaa sessiossa

Sovellus olettaa luonnollisesti myös, että jos annetulla tokenilla löytyy sessio, tulee sessiosta löytyä myös kanava, josta käyttäjä tiedustelee lokitapahtumien tilaa. Kuvassa 17 on esimerkki tilanteesta, jossa käyttäjän antamalla tokenilla löytyy sessio, mutta sessiosta ei löydy käyttäjän määrittelemää kanavaa. Tässä tapauksessa palautetaan käyttäjälle tekstinä tieto siitä, että annettu kanava on virheellinen sekä tämän tyyppiseen virheeseen viittaava koodi 11. [12.]

JSON-kutsu:

```
curl -k "http://localhost:8080/services/collector/ack?channel=CHANNEL_22222" \
    -H "Authorization: AUTH_TOKEN_11111" \
    -d '{"acks": [1,3,4]}'
```

JSON-vastaus:

```
{"text":"Invalid data channel","code":11,"invalid-event-number":0}
```

Kuva 17. Käyttöesimerkki, jossa tiedustellaan lähetettyjen lokitapahtumien tilaa, mutta annetun tokenin sessiosta ei löydy määriteltyä kanavaa.

5.4.2 Jälkipohdinnat

Kokonaisuutena olen tyytyväinen HEC-sovellukseen, sillä siihen saatiin kaikki oleellinen toiminnallisuus toteutettua, sekä ohjelma saatiin optimoitua käsittelemään lokitapahtumia käyttötarkoitukseen sopivalla tehokkuudella. Projektin alkaessa lokitapahtumien käsittely ja Syslog olivat vielä itselleni melko uusia asioita, joten näitä asioita piti aluksi tutkia ja opiskella. Mikko Kortelaisen ja hänen vuosikausien kokemuksen avulla projekti kuitenkin eteni sujuvasti ja siitä lopulta kehittyi erittäin kiinnostava ja opettavainen kokemus.

Isoimpia haasteita ohjelmaa kehittäessä oli kanavarakenteen ja ack-tunnusten ja -arvojen toteuttaminen. Nämä haasteet voitettiin lopulta, kun saatiin visio selväksi siitä, miten ohjelman komponentit ovat liitoksissa toisiinsa. Tästä kerroin luvussa 5.2. Pieni haaste oli myös ohjelma päätoiminnallisuuden eli viestien muuttamisen toteuttamisessa. Kun

lopulta selvisi, millaisena data tulee ohjelmaan sisään ja millaisena se tulee ulos, niin asia helpottui. Tässä kohtaa piti myös kunnolla tutustua Syslog-protokollaan eli RFC5424-standardiin, jonka mukaisena viestit tuli lähettää. Syslog-viestien luomista helpotti myös siihen käytettävä Java-kirjasto.

Kuten aiemmin on mainittu, sovellus on nyt käytössä testiympäristössä sekä tuotantokäytössä. Ensimmäisen version valmistuttua jatkokehitettävää kuitenkin vielä jäi. Eräs mahdollinen tuleva toiminnallisuus on statistiikan luominen ohjelman ajon aikana. Tällä voitaisiin nähdä esimerkiksi, kuinka paljon sovellukseen lähetetään lokitapahtumia tietyn ajan sisällä sekä kuinka paljon eri käyttäjät lähettävät kutsuja.

6 Yhteenveto

Insinöörityöni projektina kehitettiin lokienhallintaan erikoistuneelle yritykselle Fail-Safe IT Solutions Oy:lle sovellus, joka ottaa lokitapahtumia vastaan HTTP:llä JSON-muotoisena. Sovellus sai nimekseen HTTP Event Collector (HEC). Sovellus tehtiin osaksi Fail-Safen Record Management -kokonaisuutta. Kerroin insinöörityössäni yrityksestä, johon projekti tehtiin sekä hieman heidän historiastaan ja toiminnastaan.

Insinöörityössäni kerroin myös Syslog-protokollasta ja sen RFC5424-standardista, jonka mukaisiksi HEC-sovellus muuttaa käyttäjän lähettämät JSON-muotoiset lokitapahtumat. Työssäni kerroin, mihin tarkoitukseen Syslog-protokolla on kehitetty sekä mitä informaatiota Syslog-viestit pitävät sisällään.

HEC-sovelluksen välittämät lokitapahtumat päätyvät lopulta Fail-Safen Archive-arkistointitoteutuksen säilöntään. Insinöörityössäni kerroin Teragrep -sovelluksesta, jota käyttäen käyttäjät voivat hakea ja analysoida lähettämiään lokitapahtumia.

Lopuksi kerroin insinöörityöni projektina kehitetystä HEC-sovelluksesta. Kerroin projektin suunnittelusta ja lähtökohdista. Esittelin ohjelman toimintaa käymällä läpi ohjelman erilaiset managereiksi kutsutut komponentit hyödyntäen otteita koodista. Kerroin ohjelman toiminnasta niin ikään hyödyntäen otteita koodista sekä lopuksi esittelin sovelluksen käyttöesimerkkejä ja arvioin projektin lopputulosta.

Lähteet

1 Tietue. 2006. Verkkoaineisto. Wikipedia <https://fi.wikipedia.org/wiki/Tietue>. Päivitetty 30.7.2019. Luettu 30.3.2021.

2 Governance. 2021. Yrityksen sisäinen dokumentti. Fail-Safe IT Solutions Oy.

3 Syslog. 2005. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/Syslog>. Päivitetty 19.2.2021. Luettu 12.4.2021.

4 What Is Syslog and How Does It Work. 2020. Verkkoaineisto <https://www.auvik.com/franklyit/blog/what-is-syslog/>. Luettu 12.4.2021.

5 The Syslog Protocol. 2009. Verkkoaineisto. <https://tools.ietf.org/html/rfc5424>. Luettu 12.4.2021.

6 Syslog Tutorial: How It Works, Examples, Best Practices, and More. 2017. Verkkoaineisto. <https://stackify.com/syslog-101/>. Luettu 12.4.2021.

7 Teragrep Overview. 2021. Yrityksen sisäinen dokumentti. Fail-Safe IT Solutions Oy.

8 Apache Spark. 2012. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/Apache_Spark>. Päivitetty 23.4.2021. Luettu 30.4.2021.

9 Apache Zeppelin – Documentation. Verkkoaineisto <https://zeppelin.apache.org/docs/0.9.0/>. Luettu 30.4.2021.

10 FSLMS-CFE-16. 2019. Yrityksen sisäinen dokumentti. Fail-Safe IT Solutions Oy.

11 CFE-16 GIT. Repositorio. GitHub. <https://github.com/teragrep/cfe_16>.

12 FSLMS-CFE-16 Functionality. 2019. Yrityksen sisäinen dokumentti. Fail-Safe IT Solutions Oy.