Opinnäytetyö (YAMK)

Tradenomi (ylempi AMK), ohjelmistotekniikka ja ICT

2022

Antti Ryökkynen

TAPAHTUMAPOHJAISEN TIEDONVÄLITYSOHJELMISTON MODERNISOINTI

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Opinnäytetyö (YAMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tradenomi (ylempi AMK), ohjelmistotekniikka ja ICT

2022 | 63 sivua

Antti Ryökkynen

Tapahtumapohjaisen tiedonvälitysohjelmiston modernisointi

Opinnäytetyössä käydään läpi kaikki vaiheet mitä tarvitsimme liiketoimintakriittisen tapahtumanvälittäjäkomponentin vaihtoon tapahtumapohjaisessa arkkitehtuurissa.

Työssä kuvataan myös vaiheet, joihin ryhdyttiin toteutusta tehdessä. Kartoitimme välttämättömimmät integraatiot ja toteutimme ne valitulla komponentilla opinnäytetyön toteutuksen yhteydessä. Lisäksi käsitellään kohdattuja ongelmia ja miten ne ovat ratkaistu.

Työn tuloksena tuli tiedonvälityskomponentin onnistunut vaihto modernimpaan komponenttiin, mikä sijoitettiin hybridi ympäristöön. Toisena työn tuloksena tuli paljon uutta kyvykkyyttä toteutuksen ympäriltä. Toteutuksesta tehtiin pilviriippumaton. Tarpeidemme mukaan pystymme käyttämään sitä konesalissa ja pilvessä. Kolmantena työn tuloksena toteutettiin kaksi työkalua millä hallinnoidaan toteutusta. Lisäksi tämä vie myös meitä lähemmäksi uutta arkkitehtuuriratkaisua mihin suuntaan olemme alkaneet yrityksessä siirtyä.

Asiasanat:

Kafka, Kafka Connect, Flume, tapahtumapohjainen arkkitehtuuri, Vertica, BigQuery

Master’s Thesis | Abstract

Turku University of Applied Sciences

Master’s Degree Programme in, Software Engineering and ICT

2022 | 63 pages

Antti Ryökkynen

Modernization of an event-based data ingest product

The present study goes through all the steps that are needed to replace a business-critical event-based data ingest component in an event-based architecture.

The thesis also describes the steps taken when carrying out the implementation. The most essential integrations were identified and implemented with the selected component during the implementation of the study. In addition, the thesis discusses the problems encountered and how they were solved.

As the primary result of the study, the data ingest component was successfully replaced with a more modern component, which was placed in a hybrid environment. The Secondary result of the work was that know-how around the implementation grew significantly. The implementation is made cloud independent and can thus be run both in the datacenter and in the cloud. Thirdly a few new tools for the implementation were created as part of the study. In addition, this brings the company closer to the new architecture solution that it has already started to move towards.

Keywords:

Kafka, Kafka-connect, Flume, event-driven architecture, Vertica, BigQuery

Sisältö

Käytetyt lyhenteet tai sanasto

Lyhenne	Lyhenteen selitys (Lähdeviite)
Apache Nifi	Dataflow orkestrointityökalu
Apache Beam	Yhtenäinen ohjelmointimalli
Apache Flume	Tiedonvälityskomponentti
Avro	Datan serialisointi/deserialisointi ohjelmistokehys
BSON	Binary Javascript Object Notation, binääriviestimuoto datalle
CEP	Complex Event Processing, monimutkainen tapahtumien prosessointi
CI/CD	Continuous integration (CI) / Continuous deployment (CD), osa ketterää ohjelmistokehitystä ja nopeaa iterointisykliä
Docker	Käyttöjärjestelmän virtualisointi alusta, sovellukset voidaan paketoida omiin konteinereihin.
DEA	Downstream Event activity, tapahtuman käsittelyä loppupäässä dataputkea
Deserialisointi	Tietorakenteen purkaminen määritellystä muodosta
ESP	Event Stream Processing, tapahtumavirran käsittely
ETL	EXTRACT, TRANSFORM, LOAD. Yleinen menettelytapa tietojen kopioimiseksi, muokkaamiseksi ja lataamiseksi yhdestä tai useammasta lähteestä kohdejärjestelmään.
Google BigQuery	Googlen tarjoama tietokanta SaaS ratkaisuna

HDFS	Hadoop Distributed File Storage, hajautettu tiedostojärjestelmä
JDBC	Java Database Connectivity, tietokantayhteyksien sovellusohjelmointirajapinta. Määrittelee miten sovellus ottaa yhteyttä tietokantaan.
JSON	JavaScript Object Notation, kevyt viestimuoto tiedolle
Kafka	Viestien jonotus sovellus / message broker
Kafka Connect	Kafkan laajennus
KIP	Kafka Improvement Proposal (KIP)
Rootless	Ei-pääkäyttäjä oikeuksin ajettava sovellus
SEP	Simple event processing, yksinkertaista datavirran tapahtuman käsittelyä
Serialisointi	Tietorakenteen tallentaminen määriteltyyn muotoon
SOAP	Viestiprotokolla XML rakenteelliselle datalle
UUID	Universally unique identifier, käytetään yleisesti uniikin tunnisteen luomiseen tapahtumalle tai tiedolle.
Vertica	Kolumnäärinen analytiikka tietokanta

1 Johdanto

Opinnäytetyön tarkoituksena on vaihtaa vanhentunut tiedonvälityskomponentti jonotusjärjestelmän ja muutaman tallennuskohteen välillä. Komponentti menee vaihtoon useista eri syistä mutta painavimpana syynä on komponentin aktiivisen kehityksen lopettaminen. Komponentin vaihdoksella saamme lisää kyvykkyyttä hallita alati kasvavia datatarpeita, missä useat kohdejärjestelmät tarvitsevat muiden järjestelmien tuottamaa dataa.

Toteutus tehtiin pienellä kehitystiimillä. Toteutusta tekevään tiimiin kuuluivat tuoteomistaja ja kolme ohjelmistokehittäjää, yksi kehittäjä siirtyi pois toteutuksen loppuvaiheilla mikä laski käytössä olevia resurssejamme. Keskimäärin yksi kehittäjä pystyi keskittymään toteutukseen noin 50 % panostuksella. Oma roolini tiimissä on ohjelmistokehittäjä ja Scrum Master. Koordinoin työt kehitystiimin kesken toteutuksen tuoteomistajalta saatujen prioriteettien mukaan. Osallistuin käytännön toteuttamiseen, ohjelmointiin, suunnitteluun, testaamiseen ja määrittelyyn. Suunnittelutyö piti sisällään tuotteen valinnan sekä sovellusarkkitehtuurin rakentamisen.

Sovellusarkkitehtuurin ja tuotevalinnan osalta oli tärkeää toteuttaa ja valita kestävät ratkaisut mitkä tukevat meitä tulevaisuuden haasteissa sekä sopivat data-arkkitehtuuriimme.

Opinnäytetyötä valmistellessa tutustuin aiheesta kirjoitettuihin opinnäytetöihin tapahtumapohjaisesta arkkitehtuurista sekä Kafkan ympärille tehdyistä opinnäytetöistä. Yksi opinnäytetyö mikä kattoi molemmat, oli Sami Saariahon opinnäytetyö vuodelta 2019 otsikolla ”Palveluarkkitehtuurin suunnittelu ja toteutus startup-yritykselle”. Opinnäytetyöstä sai hyviä ideoita, miten toteutusta pystyi lähteä toteuttamaan.

Opinnäytetyössä käydään läpi komponentin vaihdossa tehdyt vaiheet. Vaiheisiin kuuluivat korvaavien komponenttien kartoitus, erilaisten komponenttien ja teknologiavalintojen konseptitodistus ja syvempi tutustuminen. Teimme myös pienen suorituskykytestauksen valitulle komponentille.

Suorituskykytestauksen tulokset on kerrottu luvussa 5. Koska opinnäytetyön toteutuksella oli suhteellisen kiireellinen aikataulu, valintoja tehdessä laitoimme huomattavan painoarvon toteutuksen realistiselle toteutukselle annetussa aikataulussa suhteutettuna käytettävissä oleviin resursseihin. Aikataulu projektille oli hyvin kiireellinen ja lyhyt, myös käytössä olleet resurssimme eivät olleet kauhean laajat.

Luvussa 5 kuvataan uuden toteutuksen monitorointia. Seuraavaksi käydään läpi kohtaamiamme ongelmia, sekä sitä miten nämä ratkaistiin. Lopuksi käydään opinnäytetyön tulokset ja jatkokehitysideat läpi.

2 Muutoksen tarve

2.1 Ongelman kuvaaminen

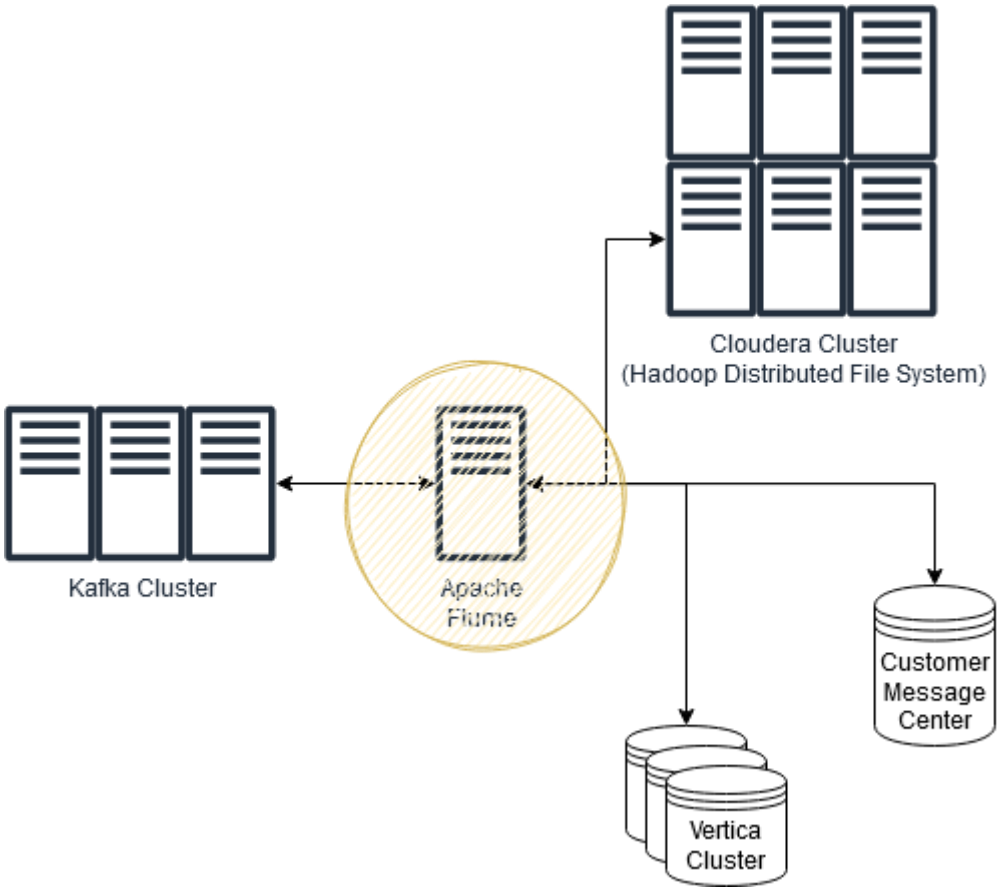
Käytössämme tiedonvälityskomponenttina oli Clouderan alustalla käytettävä Apache Flume. Tuoreimmassa Clouderan tuotteessa Apache Flume on korvattu toisella tuotteella (Cloudera, Updated CDH Components, 2022) Cloudera Flow Management työkalulla mikä on kaupallistettu versio avoimen lähdekoodin Apache Nifistä. Kun huomasimme tämän riippuvuuden päivitykselle, lähdimme pohtimaan, tarvitsemmeko Clouderaa ja sen tuotteita jatkossa käytössämme. Tuotteen ominaisuuksista käytimme vain muutamaa. Käytetyimmät komponentit ympäristössä olivat Hadoopin tiedostojärjestelmä sekä Flume. Ensimmäinen ongelmamme oli, että mikäli halusimme jatkaa Clouderan käyttöä, piti Apache Flume korvata toisella ratkaisulla. Tässä vaiheessa istuimme alas ja pohdimme erilaisia ratkaisuja. Keskusteluiden tuloksena päätimme luopua Clouderasta. Olimme toimittajariippuvaisia teknologiavalinnoistamme, toteutamme Apache Flumen korvaajan paremmin. Jatkossa pystymme itse paremmin määrittelemään mitä tarvitsemme ja mitä pystymme arkkitehtuurimme raameissa käyttämään ilman että toimittajalta tulee ilmoitus, että tuotteen tuki loppuu ja tilalle tarjoamme toista tuotetta.

Pohdimme myös minkälainen prosessi olisi korvata Flumen toiminta Nifillä. Suurimmaksi ongelmaksi muodostui Flumen puolella räätälöity ratkaisumme tapahtumien deserialisoimisessa tapahtumia haluttuun muotoon tietokantaa varten. Teimme pikaisen konseptitodistamisen Nifillä mikä olisi siirtänyt dataa Kafkalta suoraan tietokantaan. Tekovaiheessa tietotyypit mitkä eivät vaatineet räätälöintiä toimivat suorilta. Ensimmäinen haaste tuli heti alussa, miten hallinnoida Nifillä suuria määriä Kafkan jonoja. Tarkemmin tämä ongelma ja haaste kuvataan tuotevertailujen kohdalla kappaleessa 4.

Tämä oli tilanne missä ensimmäisen kerran tulimme yhteisymmärrykseen, että valmista ratkaisua ei löydy vaan meidän pitää joka tapauksessa

tapahtumienkäsittelyn logiikka kirjoittaa valitulle tuotteelle. Tämä avasi oven laajemmalle joukolle valittavia komponentteja.

2.2 Nykytilan kuvaus ylätasolla



Kuva 1. Ylätason kuva nykytilanteesta.

Kuvasta 1. on tarkoitus poistaa Apache Flume ja korvata se uudella tuotteella, tämän poistolla on laajempia vaikutuksia. Emme enää tule hyödyntämään datajärvenämme toiminutta Hadoopin tiedostojärjestelmää. Opinnäytetyössä tehtävä toteutus korvaa datansiirron uuteen datajärveen. Ajamme nykytoteutusta tuotannossa Flumea 6–8 palvelimen klusterissa missä on laskentakapasiteettia todella yläkanttiin. Opinnäytetyön tuotoksella saamme myös tehtyä kustannussäästöjä. Yhdessä klusterissa pyörivässä palvelimessa on 40 ydintä ja 386 gigatavua keskusmuistia. Kyseisiä palvelimia meillä on 10

kappaletta, kaikkia ei ole valjastettu Flumen käyttöön. Ajamme uutta toteutusta kuudella instanssilla, jokaiselle instanssille on allokoitu kahdeksan ydintä sekä 32 gigatavua keskusmuistia.

Uudessa ratkaisussa palvelimet ovat huomattavasti pienempiä, olemme kuitenkin uudessa ratkaisussa varautuneet siihen, että palvelinten määrityksiä pitää jossain vaiheessa kasvattaa. Uusi toteutus tehdään käyttäen konttiteknologiaa mikä helpottaa toteutuksen paketointia ja toteutusta voidaan ajaa käyttöjärjestelmä riippumattomasti. Keskimäärin yhden vanhan palvelimen laskentakapasiteetilla voidaan ajaa kokonaan uutta toteutusta.

Tapahtumia meillä kulkee nykytoteutuksen läpi 200–250 miljoonaa päivässä. Tapahtumien määrä riippuu tarjolla olevista tuotteista. Erilaisia tapahtumia tuottavia integraatioita meillä on noin 150 kappaletta. Integraatioiden aktiivisia jonoja mihin tapahtumia tulee, on noin 400 kappaletta. Tapahtumat voivat tulla yksittäin, tai sitten tapahtumat voivat tulla erinä. Erien koko on 2–20000 tapahtumaa, keskimäärin kuitenkin tapahtumien erälähetykset pyörivät alle tuhannen kappaleen erissä. Suosimme pieniä eriä tapahtumille nopeamman käsittelyn vuoksi. Tapahtumia tiedonvälityskomponentin kautta kulkee noin 350–400 miljoonaa päivässä. Osa tapahtumista menee datajärveen ja liiketoimintatietokantaan. Osa tapahtumista menee ainoastaan datajärveen ja osa tapahtumista tallennetaan ainoastaan liiketoimintatietokantaan. Tallennamme tapahtumat sinne missä niitä tarvitaan.

Komponentit mitä hyödynnetään nykytilanteesta

Opinnäytetyö koskeen tarkkaan rajattua komponenttia mikä hoitaa meidän arkkitehtuurissamme tapahtumien välittäjän roolia. Työn laajuus on rajattu vaihdettavaan tapahtumien välittäjäkomponenttiin ja siitä tuleviin muutoksiin. Tässä yhteydessä ympäristöstä on tarkoitus alas ajaa myös nykyinen datajärvi, mitä olemme käyttäneet pitkäaikaisen datan sijoitukseen sekä satunnaisiin historianostoihin tarpeiden mukaan. Uusi datajärvemme tulee sijaitsemaan pilvessä ja datajärvenämme tulee toimimaan Googlen BigQuery. Kaikki muu

hyödynnetään jo olevassa olevasta arkkitehtuurista, sekä valmiina olevista palveluista.

2.2.1 Apache Kafka

Apache Kafka on avoimen lähdekoodin jonotusjärjestelmä. Tuote alkaa olla yleisesti ottaen olla ”de facto” tapa toteuttaa tapahtumapohjaisia arkkitehtuuriratkaisuja (Kai Waehner, 2022) ja tapahtumien reaaliaikaisien striimien toteutuksessa. Tuotetta saa hyvin monella tapaa käyttöön, halusit sitten itse tehdä asennuksen, ostaa palveluna pilvestä tai konesalista, jokaisella isolla pilvitoimittajalla on toteutus Kafkasta saatavilla pilven kautta. Tuote on julkaistu ensimmäisen kerran vuonna 2011, LinkedInin toimesta ja tuote alkaa olla hyvin kehittynyt ja valmis ison skaalan käyttöön ilman pelkoa lastentaudeista.

Kafka toteuttaa ohjelmistokehityksen viestimallia ”publish-subscribe”, missä viestien lähettäjiä kutsutaan julkaisijoiksi. Viestejä ei merkata tietyille vastaanottajille tai kohteille suoraan. Viestien lukijoita kutsutaan tilaajiksi, jotka tilaavat halutuista jonoista sinne saapuvat uudet viestit. Julkaisijat eivät koskaan tiedä kuinka monta tilaajaa heidän viesteillään on, tai onko heidän viesteillään ollenkaan tilaajia. Tilaajat voivat ilmoittaa Kafkalle kiinnostuksensa yhteen tai useampaan jonoon. Tilaajat eivät myöskään ole tietoisia mitä julkaisijoita Kafkalla on.

Kafkan ekosysteemi on erittäin laaja ja valmiina on rakennettuna hyvinkin paljon eri käyttötarpeisiin olevia komponentteja.

Kafka Core – Pitää sisällään jonotusjärjestelmän ja perustoiminnallisuuden.

Kafka Streams – Pitää sisällään rajapinnan minkä kautta dataa voi käsitellä reaaliaikaisesti eri jonoista. Dataa voi muokata, prosessoida, aggregoida ja tulokset voidaan ohjata uuteen jonoon muiden tilaajien käyttöön.

Kafka Connect – Tällä moduulilla pystyy Kafkan liittämään laajaan kirjoon eri järjestelmiä, joko julkaisijana tai tilaajana. Mikäli laajasta valikoimasta

yhdistämiskomponentteja ei löydy mitä tarvitaan, voidaan sellainen tehdä itse juuri siihen käyttötarkoitukseen, kun tarvitaan. Kafka Connect tarjoaa myös tuen erilaisille datan muokkaus ominaisuuksille.

Kafka REST Proxy – Tarjoaa Kafkalle REST rajapinnan minkä kautta esimerkiksi vanhemman järjestelmän liittäminen Kafkalle helpottuu, voi toimia julkaisijana tai tilaajana.

Schema Registry – Datan validointiin, serialisointiin ja deserialisointiin tarkoitettu palvelu mikä pitää yllä viestijonojen skeemoja mitä vasten tulevat viestit voidaan tarkistaa eheyden kannalta.

Kafka Mirrormaker – Tällä ohjelmalla voidaan replikoida Kafkan datat toiseen klusteriin, esimerkiksi varmuuskopioita varten tai datan saamiseksi toiseen geolokaatioon viiveiden lyhentämiseksi.

Kafkalle voidaan tallentaa dataa usealla eri tavalla. Tässä opinnäytetyön toteutuksessa kaikki data on serialisoitua. Datan serialisointi ja deserialisointi tarkoittaa datan muuttamista binääri merkkijonoiksi, sekä datan muuttamista takaisin merkkijonoiksi. Datan serialisointi auttaa tiedonvaihdossa järjestelmien, ohjelmointikielien ja käsittelykehysten välillä. Pystyt purkamaan serialisoidun datan millä ohjelmointikielellä se tuntuu luonnolliselta, tai jos joku järjestelmä on riippuvainen jonkun tietyn ohjelmointikielen käytöstä, pystyt purkamaan sen siellä.

Tässä opinnäytetyössä käytämme Apache Avroa (Apache Avro, 2022) serialisointi/deserialisointi kirjastonamme. Se on avoimen lähdekoodin projekti ja hyvin tuettu Kafkan ekosysteemissä. Kaikki Avro viestimme ovat myös Snappy pakattuja. Snappy (Snappy - A fast compressor/decompressor, 2022) on Googlen avoimen lähdekoodin projekti nopeaan datan pakkaamiseen ja purkuun. Se ei kokeile olla mahdollisimman tehokas tai yhteensopiva minkään muun pakkauskirjaston kanssa, vaan pääasiallisena tarkoituksena on olla todella nopea ja tarjota kohtalainen pakkaustehokkuus.

2.2.2 Vertica

Vertica on Hewlet-Packardin valmistama ja nykyään Microfocuksen omistama ja ylläpitämä (Microfocus, Products. 2022.) analytiikka tietokanta missä data on kolumnäärisessä muodossa. Tätä tietokantaa käytämme liiketoimintatietokantana ja se sisältää meidän liiketoiminnallemme tärkeän tiedon. Raakadatan laskeutumisalusta on tässä opinnäytetyössä myös liiketoimintatietokannan sisällä. Laskeutumisalustalta data jatkojalostetaan eteenpäin käytettäväksi. Tietokanta tarjoaa suoraan hyvää tukea koneoppimiselle ja sisäänrakennettuna on iso kasa analytiikkatyökaluja.

Verticaa pohdittiin opinnäytetyössä yhdessä mahdollisena väylänä tuoda data Kafkalta tietokantaan, mutta hylkäsimme sen räätälöidyn jäsentäjän kirjoituksen yhteydessä saamamme huonon kokemuksen perusteella. Tästä kerrotaan tarkemmin Apache Flume kappaleessa. Vertica olisi mahdollista liittää lukemaan suoraan datat Kafkalta sen sisäänrakennetun Kafka liitännäisen ansiosta (Vertica, Integrating with Apache Kafka, 2022). Tietokannalla on kyvykkyys kirjoittaa Kafkalle tietoja, mikäli tarve sen vaatii.

2.3 Komponentit mitkä vaihdetaan nykytilasta

Opinnäytetyön tuloksena nykyisestä arkkitehtuurista poistetaan tai vaihdetaan useampi komponentti. Opinnäytetyö keskittyy Flumen ja Hadoopin osuuteen. Ympäristöstä poistetaan nopean tiedon käytölle tarkoitettu kolumnäärinen tietokanta, Apache Kudu, mikä integroituu Hadoop ekosysteemin kanssa. Tämän komponentin korvaus toteutetaan osana ympäristön normaalia kehityssykliä eikä tehdä opinnäytetyönpuitteissa.

2.3.1 Apache Flume

Apache Flume toteuttaa lähdekohde tekniikkaa (Apache Flume, Dataflow model, 2019) ja tuo mukanaan kourallisen erilaisia valmiita liitännäisiä sen

integroimiseksi eri järjestelmiin riippuen halutusta datasuunnasta. Meidän käytössämme tämä oli ainoastaan viestin välittäjän roolissa Kafkan jonoista liiketoimintatietokantaan ja asiakasviestintäjärjestelmään mutta tuotteen voi liittää monipuolisesti tuomaan dataa eri järjestelmiin tai eri järjestelmistä pois.

Tietojen saamiseksi liiketoimintatietokantaan ja asiakasviestintäjärjestelmään jouduimme tekemään itse omat täysin räätälöidyt ratkaisut. Nämä ratkaisut joudutaan tekemään myös uudelle valitulle tuotteelle ja molemmat tulevat olemaan hyvin samankaltaisia toteutuksia logiikaltaan, vaikka työväline vaihtuu.

Mikäli tietue ei ole Kafkan jonossa liiketoimintatietokannan vaatimassa muodossa, joudumme serialisoimaan tietueen tarvittavaan muotoon. Yleisesti tämä jouduttiin tekemään kompleksiselle tapahtumalle mikä sisältää useampia datataulukoita tai sisäkkäisiä datataulukoita. Alkuun olimme rakentaneet ominaisuuden tietokannan päähän omalla räätälöidyllä jäsentäjällä. Tietokannan päässä tuli ongelmia suorituskyvyn kanssa sekä jäsentäjästä löytyi muistivuoto, mitä emme onnistuneet korjaamaan. Tämän seurauksena siirsimme jäsennyksen Flumen päähän ja ongelmat ratkesivat. Tietokantaan ei tullut enää muistivuotoja ja resurssien käyttö pysyi aisoissa.

Flumesta tahdomme eroon koska sitä ei enää aktiivisesti kehitä kukaan eikä tuotteen mukana tule esimerkiksi valmista liitännäistä pilvisiirtymää varten. Jos haluamme viedä datat pilveen tietokantaan tai objektisäilöön on meidän keksittävä pyörä uudestaan ja tehtävä liitännäinen itse. Viimeisin päivitys tuotteeseen on tullut tammikuussa 2019. Tuote on myös merkitty poistuvan tuotteesta millä sitä tällä hetkellä ajamme ja poistuu seuraavassa päivityksessä. Alusta millä hallinnoimme Flumea on Cloudera Data platform. Tästä pääsemme ensimmäiseen ongelmaamme, ettemme voi päivittää Clouderan alustaa, kun Flume poistuu päivityksen yhteydessä. Pikaisen kartoituksen jälkeen huomasimme, ettemme käytä tuotteesta millä hallinnoimme Flumea, kuin lähinnä datajärven hallintaan sekä muita pieniä palveluita. Nämä ovat helposti siirrettävissä muualle. Vaikka opinnäytetyö mahdollistaa, että voimme ajaa koko Clouderan alustan alas, keskittyy työ tapahtuman välittäjä komponentin ja sen riippuvuuksien korvaamiseen.

Flumea ei myöskään enää suositella käytettäväksi, sen on korvannut joukko modernimpia työkaluja esimerkiksi Apache Nifi, Apache Druid, Apache Flink, Logstash, Papertrail. Kaikilla voi tehdä suurin piirtein saman minkä Flumella, hieman käyttötapauksesta riippuen.

2.3.2 Apache Hadoop

Apache Hadoop -projekti kehittää avoimen lähdekoodin ohjelmistoja luotettavaa, skaalautuvaa ja hajautettua tietojenkäsittelyä varten.

Apache Hadoop ohjelmistokirjasto on kehys, joka mahdollistaa suurten tietojoukkojen hajautetun käsittelyn tietokoneklustereiden kesken käyttämällä yksinkertaisia ohjelmointimalleja. Se on suunniteltu skaalautumaan yksittäisistä palvelimista tuhansiin koneisiin, joista jokainen tarjoaa paikallista laskentaa ja tallennusta. Sen sijaan, että luottaisi laitteistoon korkean käytettävyyden takaamiseksi, kirjasto itsessään on suunniteltu havaitsemaan ja käsittelemään sovelluskerroksen vikoja, joten se tarjoaa erittäin saatavilla olevan palvelun useiden tietokoneiden päällä, joista jokainen voi olla altis häiriöille.

Meidän käytössämme käytimme Hadoopista lähinnä ydinkomponenttia, Hadoop Distributed File Systemiä (HDFS). HDFS on tiedostojärjestelmä Hadoopin ekosysteemissä, mikä on suunniteltu pyörimään peruslaitteistolla.

Hadoopin etuihin on lueteltu sen erittäin hyvä vikasietoisuus sekä tarjoaa suuren suorituskyvyn pääsyn sovellustietoihin ja sopii sovelluksiin, joissa on suuria tietojoukkoja.

Käytössämme on vain muutamia sovelluksia mitkä käyttävät hyväkseen Hadoopin tarjoamaa hajautettua laskentaa. Sovellus mikä nostaa meille historiallista dataa, mikäli huomaamme liiketoimintatietokannassa tarpeen käsitellä vanhaa dataa uudestaan.

Toinen käyttötarkoitus Hadoopille meillä on pitää sitä datajärvenä, säilytämme siellä kaiken historiallisen datamme mikä virtaa Kafkan kautta. Tämä putki

tullaan tämän projektin myötä lopettamaan ja ohjaamme historiadatamme Kafkalta eteenpäin BigQueryyn.

Samalla tiedot vanhasta datajärvestä tullaan siirtämään pilveen, sen laajuus kuitenkin ulottuu opinnäytetyön ulkopuolelle ja etenee erillisenä tehtävänään toteutuksen ohessa.

3 Tapahtumapohjainen arkkitehtuuri

Tapahtumapohjainen arkkitehtuuri koostuu tyypillisesti tapahtumien julkaisijoista, tapahtumien tilaajista ja tapahtumakanavista (Amazon, What is an Event-Driven Architecture? 2022). Julkaisijan tehtävänä on havaita, kerätä ja siirtää tapahtumia. Tapahtuman julkaisija ei tunne tapahtuman tilaajaa, se ei edes tiedä, onko tilaaja olemassa, ja jos tilaaja on olemassa, se ei tiedä, miten tapahtumaa käytetään tai käsitellään edelleen. Tilaajan tehtävänä on reagoida heti, kun tapahtuma tuodaan saataville.

Tapahtumapohjaisen arkkitehtuurin avainominaisuuksiin kuuluu seuraavia ominaisuuksia, monilähetyskommunikaatio. Monilähetyskommunikaatiossa julkaisijat tai osallistuvat järjestelmät voivat lähettää tapahtumia useille järjestelmille, jotka ovat tilanneet tapahtumavirran käyttöönsä. Tämä ei ole kuitenkaan sanomaviestintää, missä yksi lähettäjä voi lähettää tietoja vain yhdelle vastaanottajalle. Ominaisuuksiin kuuluu myös reaaliaikainen lähetys missä julkaisijat julkaisevat tapahtumia tilaajille sellaisina kuin ne tapahtuvat reaaliajassa, tämä on arkkitehtuuriominaisuuden käsittely- tai lähetystapa.

Yleensä tapahtuma rakentuu kahdesta osasta, otsaketiedosta ja tapahtuman sisällöstä. Tapahtumat voidaan myös lähettää ilman otsaketietoa, silloin yleensä tiedetään lähettäjästä jo automaattisesti jotain, esimerkiksi mikä järjestelmä lähettää tai mitä asiaa tieto koskee.

Hyviä käytäntöjä on laittaa tapahtumaan mukaan tapahtuman tyyppi, aikaleima, uniikkitunniste esimerkiksi UUID. Nämä kaikki helpottaa tapahtuman koneellista käsittelyä ja parantaa tiedon jäljitettävyyttä, mikäli uniikki tunniste pidetään samana koko tapahtumaputken ajan.

Tapahtumavirran kerrokset:

- Event Generator: Tapahtuma generaattori / lähdejärjestelmä. Tämä lähettää datan.
- Event channel: Tapahtuman kanava, tämä ottaa vastaan datan.

- Event Processing Engine: Tapahtuman käsittely moottori, tämä käsittelee vastaanotetun tapahtuman ja toimii halutulla tavalla, esimerkiksi liputtaa mahdollisen epäonnistuneen maksun tai siirtää saapuneen tiedon eteenpäin datavarastoon.
- Downstream event activity: Tapahtumamoottorin jälkeinen jatkokäsittely. Esimerkiksi datalakesta siirretään tapahtumat tekoälyn käsittelyyn, liputtaa monitorointinäkymään epäonnistuneet maksut, ilmoittaa loppukäyttäjille, että on tullut X määrä virheellisiä tai epäonnistuneita maksuja ja vaatii loppukäyttäjän puolelta toimenpiteitä.

Tapahtumia voidaan käsitellä usealla eri tavalla tapahtumapohjaisessa arkkitehtuurimallissa.

Kolmea eri tyyliä on yleisesti käytössä tapahtumapohjaisessa arkkitehtuurissa, yksinkertainen tyyli käsitellä tapahtumia, käsitellä reaaliajassa datavirran tapahtumia sekä kompleksinen tapahtuma. Näitä käytetään hyvin usein tapahtumapohjaisessa arkkitehtuurityylissä.

Yksinkertainen tapahtumien käsittely (SEP), tapahtumat liittyvät suoraan tiettyihin mitattaviin tilan muutoksiin. Yksinkertaisella tapahtuman käsittelyllä huomataan merkittävä tapahtuma, tämä käynnistää alavirran toiminnot (DEA). Esimerkiksi: Onko tämä maksu onnistunut; jos maksu epäonnistui: kerro käyttäjälle maksun epäonnistumisesta ja ilmoita epäonnistuneen tapahtuman ja maksun tiedot taustajärjestelmille.

Tapahtumanvirran käsittelyssä (ESP) on kahdenlaisia tapahtumia, tavallisia ja merkittäviä. Tavalliset tapahtumat seulotaan, jos ne sisältävät tietoja, jotka voivat tehdä tapahtumasta merkittävän tämä käsitellään eri tavalla. Tätä tapahtuman käsittelytapaa käytetään yleisesti reaaliaikaisessa virtauksessa ja tämä mahdollistaa oikea-aikaisen päätöksenteon. Esimerkiksi: Millä todennäköisyydellä tämä maksu on petollinen, jos arvo ylittyy, liputa maksu ja peruuta maksu.

Monimutkaiset tapahtumat (CEP, complex event processing). Tapahtumavirran käsittelyn tapaan monimutkainen tapahtuman käsittely. Tämä mahdollistaa

yksinkertaisten ja tavallisten tapahtumien mallintamisen, joka johtaa monimutkaisen tapahtumaan. Monimutkaisissa tapahtumakäsittelyllä odotetaan ensin, että kaikki määritellyt kriteerit täyttyvät, ennen kuin se luo tapahtumasanoman jatkotoimenpiteitä varten. Monimutkaisella tapahtumankäsittelyllä kootaan, käsitellään ja analysoidaan valtavia tietovirtoja, jotta saadaan reaaliaikaisia näkemyksiä tapahtumista niiden tapahtuessa.

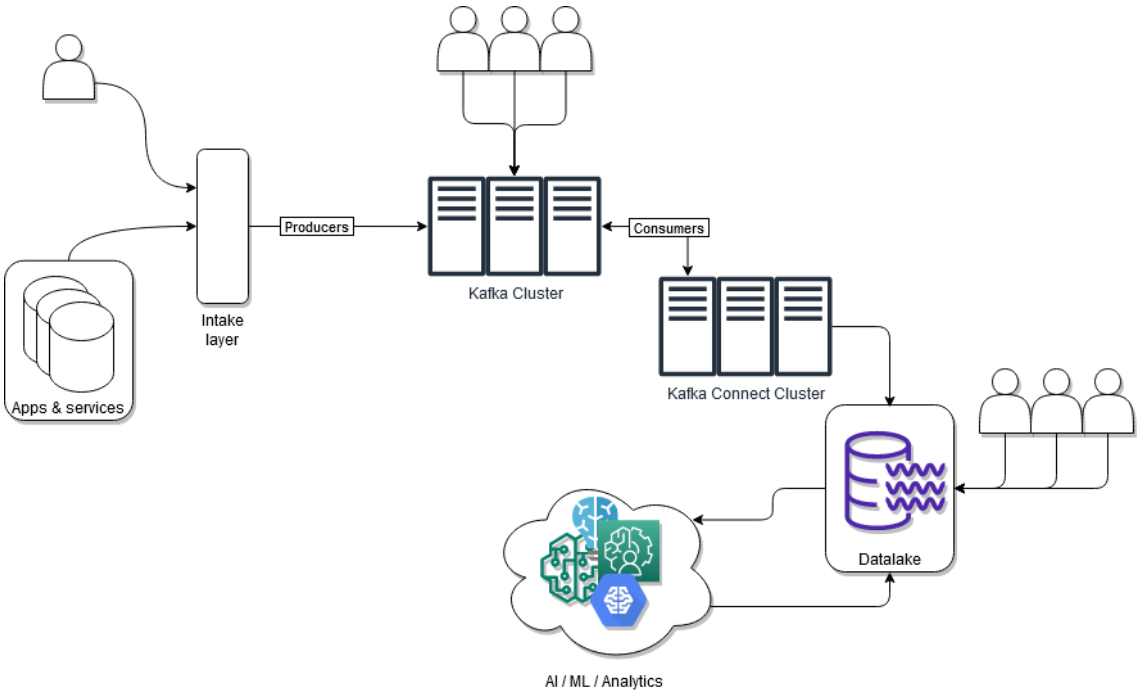
Tätä käytetään yleisesti liiketoiminnan poikkeavuuksien, uhkien ja mahdollisuuksien havaitsemiseen ja niihin vastaamiseen.

Tapahtumapohjainen arkkitehtuuri tuo myös omia hyvä erikoispiirteitään mukana, kuten asynkroniset ominaisuudet.

Kirjassa Fundamentals of Software Architecture: An Engineering Approach Richards M. & Ford N. (2020, 196) kuvaa asynkroniset ominaisuudet seuraavasti tapahtumapohjaisessa arkkitehtuurissa:

“The event-driven architecture style offers a unique characteristic over other architecture styles in that it relies solely on asynchronous communication for both fire-and-forget processing (no response required) as well as request/reply processing (response required from the event consumer). Asynchronous communication can be a powerful technique for increasing the overall responsiveness of a system.”

Kuvassa 2. on kuvattuna esimerkki arkkitehtuuri Kafkan ekosysteemin ympärille rakennettuna. Tapahtumia voi tulla sisään palveluista ja sovelluksista sekä käyttäjiltä. Käyttäjät voivat lukea tapahtumia suoraan Kafkalta tai odottaa että Kafka Connect saa vietyä tiedot datajärveen. Datajärvestä voidaan viedä tiedot eteenpäin analyysijärjestelmiin ja koneoppimisen puolelle minkä jälkeen tapahtumista rikastetut tiedot palautetaan datajärveen loppukäyttäjien käyttöön.



Kuva 2. Esimerkki tapahtumapohjaisesta arkkitehtuurista Kafkan ekosysteemin ympärille rakennettuna.

3.1 Miten tapahtumapohjaista arkkitehtuuria hyödynnetään

Tapahtumapohjaista arkkitehtuuria hyödynnetään tällä hetkellä muutamalla eri tavalla. Ohessa kerrottuna pari tapaa miten meillä tapahtumapohjainen arkkitehtuuri näkyy jokapäiväisessä työssämme.

Haemme tietoja kumppanimme jonotusjärjestelmästä minkä perusteella reititämme tietoja eri kohteisiin. Tietyn tyyppiset tiedot reititetään siten että tiedot ohjataan tapahtuman yksilöivän tunnisteen avulla pilvisäilytykseen jatkokäsittelyä varten. Keräämme eräänlaista rekisteriä tapahtumista ja tapahtumalle tulevista päivityksistä. Aina kun tapahtuman tiedot päivittyvät kumppanimme jonotusjärjestelmään, otamme tämän tiedon talteen pilveen. Samasta tapahtumasta lähtee sanoma myös varsinaiseen liiketoimintatietokantaan mutta siellä tiedot pysyvät tallessa vain lyhyen ajan.

Samassa integraatiossa tehdään reaaliaikaista tietovirran keveää ETL prosessia. Kun tietyn tyyppinen viesti tulee jonoon, käsittelemme sitä jo

etukäteen siten, että varsinaisen ETL prosessin on helppo jatkaa tästä eteenpäin. Reaaliaikaiseen tietovirran ETL prosessissa lisäämme avaimia, muokkaamme kompleksista tietoa helpommin käsiteltävään muotoon ja siirrämme sen eteenpäin omaan jonotusjärjestelmäämme mistä tiedot valuvat varsinaisen ETL prosessin käsittelyyn ja jatkojalostukseen. Tämä käsittely tapahtuu heti kun viesti saapuu kumppanimme jonotusjärjestelmään. Normaali ETL prosessimme toimii ajastetuilla eräajoilla ja voi pyörähtää useammankin tunnin viiveellä.

Toinen esimerkillinen tapa miten käytämme tapahtumapohjaista arkkitehtuuria hyväksi nykyisessä toteutuksessa. Tuotteen suunnitteluvaiheessa ja kehittelyvaiheessa unohtui tärkeä ominaisuus tuotteen tuottamasta raportointi datasta. Tuote ei lähetä meille kaikkia tarvittavia tietoja raporteista vaan, joudumme tutkimaan tapahtumia sitä mukaan, kun niitä saapuu. Tätä paikataan sitten tapahtumapohjaisen arkkitehtuurin tuomilla ominaisuuksilla, kun tietyntyyppisestä tapahtumasta tulee tietoa, me tarkistamme ja puramme tiedon reaaliajassa ja lisäämme tarvittavat tiedot viestiin. Näistä tiedoista muodostuu kerran kuukaudessa ajamamme raportointi rikastetuilla tiedoilla. Jälkeenpäin tätä ominaisuutta on hyvinkin vaikea lisätä valmiiseen ohjelmistoon, väliaikaisratkaisuna teimme ominaisuuden käyttäen hyväksi tapahtumapohjaista arkkitehtuuria. Tästä saimme myös hyvää oppia tuotteiden kehittelyvaiheeseen sekä miten voimme paikata mahdollisia tuotteiden puutteita myös loppupäässä.

3.2 Miksi tapahtumapohjainen arkkitehtuurityyli on valittu tähän projektiin

Haluamme laajentaa reaaliaikaisen tiedon hyödyntämistä ja jalostusta suoraan lennosta, kun tietoa virtaa jonotusjärjestelmäämme. Tämä lisää tiedon hyödyntämisen nopeutta ja on linjassa yrityksemme strategian kanssa missä painotetaan hyvinkin paljon tiedolla johtamista.

Tämä arkkitehtuurityyli tukee myös tulevaisuutemme arkkitehtuurivisioita missä reaaliaikaisen tiedon rooli vain kasvaa. Tulevaisuudessa hyödynnämme tietoa entistä enemmän. Tarkoituksemme on myös laajentaa tiedon hyödyntämistä

mahdollisimman paljon siihen suuntaan, että käyttäjät pystyvät itse hakemaan tarvittavat tiedot. Tämä tarkoittaa, että tiedon mikä on käyttäjille saatavilla pitää olla automaattisesti jo hieman rikastettua. Rikastamme dataa esimerkiksi tarvittavilla avaimilla, mitä pystymme liittämään tietoon tapahtumapohjaisen arkkitehtuurin myötä hyvinkin helposti ja vähällä vaivalla.

Tämän pohjustamiseksi olemmekin jo muutaman kerran yrityksessä järjestäneet Data Akatemian, missä koulutamme ihmisiä käyttämään saatavilla olevia tietoja entistä laajemmin ja kannustaneet tiedon itsenäiseen hakuun.

4 Valintojen kartoitus

4.1 Valitut komponentit ja miksi

Pohdimme pitkään mitkä ovat meidän vaatimuksemme tapahtumien välittäjä komponentille. Tärkeimpänä kriteerinä pidimme, että data ei saa muuttua, ettei datan luku ETL prosessin läpi muutu, vaan voimme vaihtaa uuden komponentin suoraan.

Toisena kriteerinä valitsimme, että tuotteen pitää olla helposti skaalautuva, esimerkiksi tuotetta on pystyttävä komentamaan skaalautumaan tai tuote osaa itsenäisesti skaalautua kasvavaan kuormaan.

Kolmantena kriteerinä pidimme modulaarisuutta ja muokattavuutta hyvin tärkeänä, että voimme liittää sen tulevaisuudessa tarvitsemiimme kohteisiin ja järjestelmiin ilman isompia ongelmia ja pystymme tarvittaessa laajentamaan toimintaa esimerkiksi koodaamalla itse liitännäisen.

Neljäntenä tärkeänä kriteerinä oli tuotteen hallittavuus. Tuotteella pitää pystyä hallitsemaan kaikkia liitännäisiä yksittäin tai massana riippuen tarpeesta. Hallittavuus sai tulla rajapinnan, käyttöliittymän tai muun aihion kautta, kunhan pystyimme hallitsemaan tuotetta helposti ja tehokkaasti.

Viidentenä valintakriteerinä tuotteelle tiimin arvio pystytäänkö toteutus hoitamaan aikataulussa vai ei.

Koetimme myös miettiä pilviriippumatonta vaihtoehtoa, että emme lukitse itseämme tiettyyn pilvitoimittajaan. Tämän vaihtoehdon painoarvo oli hyvin pienessä osassa valinnassa. Lopulta testattavien komponenttien listalle ei päätynyt yhtään pilvinatiivia komponenttia, vaikka yhden niistä sai pilvestä natiivina palveluna. Kyseisen komponentin tapauksessa pilvinatiivius merkitsi ainoastaan, että skaalautuminen on automaattista pilvessä käyttäjän määrittelemien parametriarvojen mukaisesti.

4.2 Valintakriteerit

Tuotteen valintakriteerit ovat jaoteltu eri kokonaisuuksiin ja niitä käydään läpi seuraavissa alikappaleissa. Kaikkien tuotteiden piti osata lukea Kafkalta vähällä vaivalla ja pystyä liittymään eri järjestelmiin ongelmitta.

4.2.1 Datan pitää pysyä muuttumattomana

Yksi isoimmista valintakriteereistä oli, että datan pitää tulla samassa muodossa liiketoimintatietokannan laskeutumisalueelle, kun vanhalla tuotteella. Kaikkien komponenttien osalta tämä tarkoittaa, että jouduimme tekemään erikoisratkaisuja eikä valmista ratkaisua löytynyt hyllyltä. Kafkan kautta kulkee todella paljon erilaista dataa, data voi olla:

- Jäsentämätöntä
- XML muodossa
- Tekstimuotoisia raportteja
- JSON muodossa
- Sensitiivistä

Datassa on myös monia erikoisuuksia, esimerkiksi yksi viesti saattaa olla useamman kymmentä megatavua ja tuotteen pitää pystyä käsittelemään nämä ongelmitta.

Datan muuttumattomuus on myös edellytys, että liiketoiminta raportointimme toimii ilman muutoksia eikä hajoa tämän toteutuksen myötä.

Datan raportointi käyttöä varten laskeutumisalueelta louhitaan ETL työkalullamme raakadata jalostettuun muotoon mikä mahdollistaa datan käytön raportoinnissa.

4.2.2 Skaalautuvuus

Tuotteen pitää olla skaalautuva. Emme tarkemmin määritellyt miten tuotteen pitää olla skaalautuva, mutta meidän on pystyttävä skaalaamaan suorituskykyä ylös tai alaspäin tarvittaessa. Esimerkiksi lisäämällä uusia mikropalvelu instansseja tai uusia servereitä.

4.2.3 Modulaarisuus / muokattavuus

Erityisen tärkeää meille oli, että pystymme tavalla tai toisella muokkaamaan tuotetta omiin käyttötarpeisiin, oli se sitten integraatiot täysin omiin järjestelmiin tai datamuunnokset suoraan tuotteesta.

Myös modulaarisuus tuotteessa oli todella kohtuullisella painoarvolla, että voimme tehdä oman moduulimme täsmällisesti yhtä käyttötarkoitusta varten ja käyttää sitä muualla.

4.2.4 Suorituskyky

Komponentille emme laittaneet mitattavia suorituskykymetriikoita muita kuin että sen on suoriuduttava samasta työstä vähintään yhtä hyvin kuin vanha korvattava komponentti. Valitulle tuotteelle tehtiin kuitenkin suppeat suorituskykytestit.

Meillä on noin 150 erillistä dataintegraatiota mitkä lähettävät tietoja noin 400 jonoon Kafkalla. Joihinkin jonoihin saattaa tulla dataa kerran viikossa tai kuukaudessa, osaan tulee useampi tuhatta viestiä minuutissa.

4.2.5 Hallittavuus

Yksi valintakriteereitämme oli myös tuotteen hyvä hallittavuus joko graafisen käyttöliittymän, rajapinnan- tai komentorivin kautta. Tarvitsemme hyvän hallittavuuden tuotteeseen, sillä ei ollut väliä mikä hallinnan väylä on. Nykyistä

tuotetta hallitaan huonolla tavalla. Kun konfiguraatiomuutos tulee, joudumme käynnistämään koko palvelun uudestaan eikä vain yhtä viestin käsittelijää mitä muutos koskee. Tämä tarkoittaa useamman minuutin datakatkoa jonotusjärjestelmältä kohdejärjestelmiin. Uudelta tuotteelta halusimme hienojakoista hallintaa per jono. Kun jonoon tulee muutos, tarvitsee vain sen osuus tuotteesta käynnistää uudestaan. Parhaassa tapauksessa tuote tunnistaa konfiguraation muuttuneena ja lataa itse tarvittavat komponentit uudestaan automaattisesti ilman, että ihmisen täytyy toimia välissä.

4.2.6 Aikataulu

Tuotteita arvioidessa, tiimi arvioi myös pystytäänkö haluttu toteutus toteuttamaan annetuissa aikataulussa ja resursseissa. Tällä oli huomattava painoarvo tuotetta valittaessa.

4.3 Valintavaiheeseen päätyneet tuotteet

Pitkän ja laajan kartoituksen jälkeen lähdimme karsimaan tuotteita niillä perusteilla mitä olimme saaneet arkkitehdeiltä, mitkä ovat meille tuttuja ja mistä meillä mahdollisesti löytyy kompetenssia hoitaa asiat kunnolla. Tuotteita mitä tutkimme korvaajiksi, oli useita, mutta varsinaiseen syvälliseen selvitysvaiheeseen päätimme rajata testattavien tuotteiden määrän kolmeen.

Seuraavat kolme tuotetta valittiin mukaan koska niillä oli kartoituksessa eniten potentiaalia hoitaa tarvittavat tehtävät ilman että toteutuksesta tulisi liian monimutkainen. Lisäksi tulevaisuuden arkkitehtuurivalinnat ohjasivat näiden kolmen tarkempaan tarkasteluun.

Käyn lyhyesti läpi jokaisen tuotteen kohdalta valintakriteerien täyttymisen.

4.3.1 Apache Beam / Dataflow

Apache Beam (Apache Beam, An advanced unified programming model, 2022) on avoimen lähdekoodin yhtenäinen ohjelmointimalli, joka määrittää ja suorittaa tietojenkäsittelyputkia, mukaan lukien ETL-, erä- ja stream-käsittely. Meidän fokusointimme on stream-käsittelyssä. Tuote on myös käytössä muualla talossa ja sieltä saimmekin hyviä vastauksia, miten sitä käytetään. Toinen tiimi ei kuitenkaan käyttänyt tuotetta stream käsittelyllä millä me olisimme ajatelleet tuotetta käytettävän. Saimme hyvää informaatiota tuotteesta ja tuotteen ominaisuuksista.

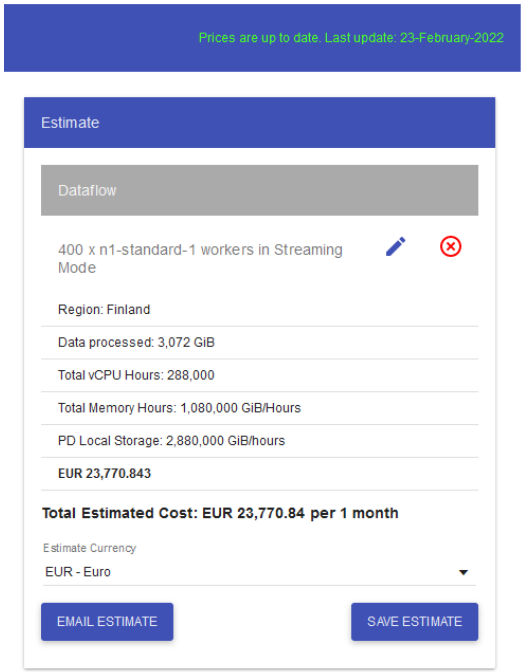
Tuotteesta on myös saatavilla pilvinatiivi versio Googlen pilvestä nimellä Google Dataflow (Google Dataflow, 2022), mikä on sama tuote integroituna Googlen pilveen ja hallintaan. Tämä olisi täysin Googlen ylläpitämä ja ainoa mitä vastuuta käyttäjällä on tehdä Beamin raameja noudattaen omat työskentely yksiköt, loput tulee Googlelta palveluna.

Apache Beam ja Dataflow tuotteena toteuttaa myös lähdekohde tekniikkaa mikä on meille hyvinkin tuttu jo Flumen puolelta.

- Datan muuttumattomuus, toteutettavissa. Tuotetta voidaan ohjata koodin puolelta ja näin taata datan muuttumattomuus.
- Skaalautuvuus, tuote skaalautuu työskentelijä yksiköillä sekä raudalla. Pilvessä työskentelijäyksiköt voidaan helposti määritellä minkä kokoisia ja kuinka monta kappaletta.
- Modulaarisuus löytyy, tuote on koodipohjainen, voit tehdä Beam / Dataflow moduuleista niin kompleksisia tai yksinkertaisia kuin haluat.
- Hallittavuus, Pilven puolella ei ollut meille kätevää tapaa hallinnoida tai orkestroida työskentelijäyksikköjä, sieltä näkyi ainoastaan vain tilannekatsaus (Google Dataflow, Using the monitoring UI, 2022). Positiivisena puolena mainittakoon että, tuote integroituu täysin Googlen metriikkavalvontaan mistä saisimme tarvittaessa käyttämäämme monitorointijärjestelmään metriikat.

- Aikataulu, tiimi ei kokenut, että toteutuksen olisi voinut toteuttaa annetuilla resursseilla ja aikataulussa. Syinä lähinnä tuotteen tuntemattomuus sekä merkit siitä, että toteutuksesta saattaa tulla liian monimutkainen käyttötarpeeseen verrattuna.

Tuote tukee integroitumista BigQueryyn ja JDBC:n välityksellä liiketoimintakantaan ilman erillisiä asennuksia. Tämän tuotteen osalta teimme hyvin suppean koeajon ja testaamisen ja emme nähneet kovinkaan helpoksi tehdä tehokasta tuotetta mikä lukisi jonoista kaiken geneerisesti vastaan ja validoisi datan skeemarekisteriä vasten, ja tämän jälkeen lähettää tiedot kohdejärjestelmille siinä muodossa kuin tarvitsemme. Tuotteesta jäi positiivinen kuva, jos dataa pitää louhia ja jalostaa, pelkkään datanvälitykseen emme suositelleet tätä tuotetta jatkoon. Kysymysmerkiksi myös kasvoi hinta pilvessä, jos meillä on noin 400 erilaista dataputkea, tarkoittaa se vähintään 400 pientä työskentely yksikköä. Googlen puolella käyttäen Dataflowia tämä tarkoittaa, että työskentelijät ovat pieniä ja väliaikaisia virtuaalikoneita. Käytännössä tämä olisi kuitenkin meidän ympäristössämme tarkoittanut 400 virtuaalikonetta mitkä pyörivät jatkuvasti odottaen viestejä meidän dataputkistamme.



Kuva 3. Googlen hintalaskuri Dataflowista

Tämä pyörittäminen olisi kohtuuttoman kallista (Kuva 3.) verrattuna siihen mikä meidän tarpeemme on.

Ehkä tälle olisi löytynyt joku ratkaisu, esimerkiksi rakentamalla logiikka ja pitämällä työskentelijöitä päällä vain silloin kuin tarvitaan, mutta meidän ideologissamme datan pitää välittyä eteenpäin jonotusjärjestelmältä heti kun se sinne saapuu.

4.3.2 Apache Nifi

Apache Nifi (Apache Nifi, An easy to use, powerful, and reliable system to process and distribute data, 2022) on avoimen lähdekoodin ohjelmisto, mikä on suunniteltu automatisoimaan tietovirrat ohjelmistojärjestelmien välillä. Tällä tuotteella meillä on toteutettuna useampi ratkaisu, minkä takia tuote nousi Flumen korvaavien kartoitettavien tuotteiden listalle. Myös Cloudera suosittelee Flumen korvaamista Cloudera Dataflowilla (Cloudera DataFlow, 2022) mikä on kaupallistettu tuote Apache Nifistä.

- Datan muuttumattomuus, Nifille on mahdollista koodata omia datankäsittely prosessoreja, jos valmiilla työkaluilla ei olisi saatu haluttua tulosta aikaiseksi.
- Skaalautuvuus, Nifi skaalautuu sille annetun raudan mukaan horisontaalisesti, myös käytettyjä resursseja per prosessori on mahdollista skaalata ylöspäin. Siihen voidaan liittää useampi palvelin tai siirtää data käsiteltäväksi erillisille Nifi instansseille. Nifiin voidaan liittää myös useita klustereita.
- Modulaarisuus, Nifi on hyvinkin modulaarinen. Suoraan laatikosta Nifin mukana tulee lähemmäs 200 erilaista komponenttia mitä voi käyttää. Omien tekeminen ei myöskään ole ison työn takana vaan ne voi koodata Javalla.
- Hallittavuuden Nifi tarjoaa graafisen käyttöliittymän kautta tai rajapinnan kautta. Tuotteella pystyisimme hallitsemaan ja orkestroimaan kaikki tarvittavat käyttötapaukset.

- Aikataulun osalta tiimi koki, että toteutuksen pystyy tekemään annetuilla resursseilla ja aikataulussa.

Nifi on tuotteena tiimillemme hyvin tuttu, tämä oli varmasti eniten vaakakupissa painava asia, kun nostimme tuotteen tarkempaan tarkasteluun. Lähdimme tutkimaan soveltuisiko se meidän käyttötapaukseemme datanvälittäjänä jonotusjärjestelmästä kohdejärjestelmiin. Tuotteella on suora tuki BigQuerylle ja JDBC:n yli tietokannoille.

Koska tunsimme tuotteen, pikaisten kokeilujen ja pähkäilyjen jälkeen päätimme, että tämä tuote ei ole soveltuva tähän käyttötarkoitukseen, vaikka kaikki näennäisesti onnistuisikin. Isoimpina syinä että meidän pitäisi jollain tavalla hallita Nifin kautta nuo kaikki 400 lähdettä. Emme saaneet pähkäiltyä sellaista dataputkea, että olisimme vain antaneen listan jonoja ja kertoneet tuotteelle lue tuolta, hae skeema täältä, validoi data ja laita tänne ilman, että olisi jouduttu tekemään paljonkin erillistä käsittelyä.

4.3.3 Kafka Connect

Kafka Connect on olennainen osa Kafkan ekosysteemiä ja on tarkoitettu integraatioiden tekemiseen helposti. Tuote on tarkoitettu datan siirtoon Kafkan jonoihin tai jonoista kohdejärjestelmiin.

Tuote toteuttaa lähdekohde tekniikkaa ja Confluentin julkaisema avoimen lähdekoodin versio tuo mukanaan noin 120 kappaletta (Confluent, Confluent Connector Portfolio, 2022) liittimiä millä voit joko tuoda Kafkalle dataa tai viedä dataa Kafkalta tuettuihin järjestelmiin.

- Datan muuttumattomuus onnistuu myös Kafka Connectilla, tähän on mahdollista tehdä omia liittimiä mihin voi kirjoittaa oman logiikan ja varmistaa ettei data muutu. Tarjolla myös valmiina datanmuuntimia.
- Skaalautuvuus kuuluu myös Kafka Connectin arsenaaliin, tuotetta voidaan skaalata joko työläisten määrällä tai jos tehdään hajautettu

asennus, voidaan skaalausta tehdä lisäksi klusterin instanssien määrällä.

- Modulaarisuus on myös yksi Kafka Connectin peruspilareista. Voit tehdä haluamasi liittimen helposti ja nopeasti, valmis runko tarjotaan suoraan liittimien ohjelmointiin.
- Hallittavuus Kafka Connectissa tapahtuu REST rajapinnan kautta, tehokas ja simppeli hallittavuus. Tuotteeseen on saatavilla myös monia työkaluja mitkä toteuttavat Kafka Connectin hallintarajapinnan.
- Aikataulun puolesta tiimiltä tuli myös positiivista valoa ja koettiin että tiimi pystyi suoriutumaan toteutuksesta aikataulun ja resurssoinnin puitteissa.

Yllätyimme miten nopeasti Kafka Connectilla tuli tuloksia testausvaiheessa, muutamassa päivässä meillä oli jo data virtaamassa eteenpäin BigQueryyn sekä Vertica tietokantaan. Tässä vaiheessa toki huomasimme jo, että data ei mene siinä muodossa missä haluamme ja vaati tutkintaa. Myöskään emme osanneet konfiguroida tuotetta vielä tehokkaasti.

Tuotteella oli myös suoraa laatikosta muutamia rajoituksia BigQueryn (Confluent, Kafka Connect Limitations, 2022) suuntaan mitä keskustelimme ja pohdimme, tuleeko tästä ongelmaa vai ei. Rajoitukset kuitenkin näyttivät sellaisilta, etteivät haittaa meidän käyttöämme nyt tai tulevaisuudessa. Rajoitukset koskivat myös ainoastaan tuotteen pilviversiota. Pilviversio ei ainakaan alkuun ollut toteutuksen laajuudessa mukana, mutta jos tulevaisuudessa mietimme pilviversiota Kafka Connectista, eivät rajoitukset ole esteenä. Pilviversion rajoituksina oli yksittäisten viestin muunnokset mitkä koskivat aikaleimojen reititystä ja muutamaa muuntimien käyttöä.

4.4 Valittu tuote

Kartoitus, tutkinta ja testausvaiheen jälkeen tuotteeksi toteutukselle valittiin Kafka Connect.

Päätöstä puolsivat nykyiseen jonotusjärjestelmä ekosysteemiin integroituminen sekä tiimin kompetenssi Kafkan ympärillä. Olemme panostaneet Kafkan kehitykseen paljon ja jatkossa vielä enemmän.

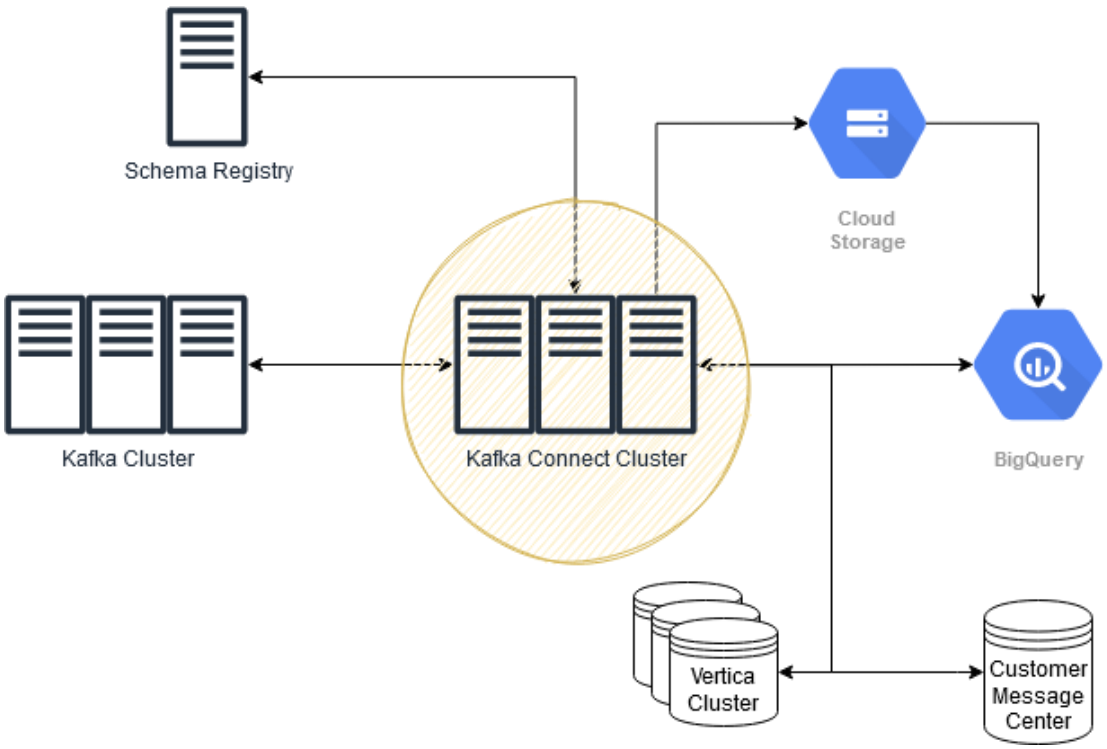
Toisena puoltavana syynä oli tuotteen nopea käyttöönotto suoraan laatikosta vedettävältä tuotteelta. Saimme muutamassa päivässä tiedot luettua jonoista, deserialisoitua ja lähetettyä tiedot kohdekantoihin, vaikkakaan kaikki tiedot eivät ollut siinä muodossa kuin piti.

Kolmantena puoltavana puolena pidimme sitä, että mikäli tulevaisuudessa haluamme, tuote on mahdollista hankkia palveluna. Tällöin meille jää vain käyttö ja käyttöoikeuksien hallinta.

Kafka Connectin mukana tulee myös hyvin paljon erilaisia moduuleja mitä tarjotaan liitännäisten lisäksi. Näillä voi vaikuttaa viestin sisältöön ja tehdä esimerkiksi datamuunnoksia (Confluent, Single Message Transforms for Confluent Platform, 2022). Mikäli nämä valmiin moduulit eivät riitä niitä voi tehdä myös itse ohjelmoimalla. Näitä muunnoksia voi soveltaa kaikkiin viesteihin ennen kuin ne päätyvät Kafkalle tai lähtevät Kafkalta kohdejärjestelmään.

5 Toteutus

5.1 Integraatiot toteutusvaiheessa



Kuva 4. Toteutusvaiheen integraatiot ja haluttu visio.

Toteutusvaiheeseen kartoitimme seuraavat integraatiot, meidän pitää pystyä integroitumaan Kafka Connectilta liiketoiminta tietokantaan, Googlen BigQueryyn sekä asiakasviestintäjärjestelmään.

Toteutus toteutettiin mikropalveluina, paketoimme Kafka Connectin konteineriksi ja automatisoimme testiin ja tuotantoon viennin eri ympäristöihin.

5.2 Integraatio Kafkasta Verticaan

Integraation toteutukseen käytimme valmista Kafka Connectin mukana tullutta JDBC liitännäistä, suoraan hyllyltä saimme viestin luettua ja deserialisoitua. Viesti ei kuitenkaan ollut täysin siinä muodossa missä halusimme.

Tätä varten jouduimme kehittelemään täysin oman viestin muuntajan.

Tutkiessa minkälainen ratkaisu meillä oli Flumella jo toteutettuna. Huomasimme lähdekoodista että, MongoDB:n käyttämä Binary JSON (Bson, BSON (Binary JSON) Serialization, 2022) on hyvin lähellä sitä viestiformaattia mitä me nyt kaipaamme viesteillemme. Tarkemmin vielä tutkien huomasimme, että MongoDB oli tehnyt oman liitännäisen Kafka Connectiin ja sitä myöten oman serialisointi/deserialisointi kirjaston Avrosta viesteistä BSON muotoon.

Tämän käyttöönotto oli erittäin suoraviivaista, kirjaston lisäämisen jälkeen koodin puolella tehtiin kaikille datatyypeille (MongoDB, BSON Types, 2022) haluttu käsittely ja homma toimi suurimmilta osin.

Pieniä datan muunnoksia kuitenkin havaittiin mitä koimme, että ei vaikuta liiketoimintakantamme raportointiin tai datan latauksiin. Kaikki datan muunnokset tulivat Flumen puolelta ja emme halunneet toteuttaa näitä enää uuteen toteutukseen.

```
{ "double_field": 1.0,  "doc" : { "field_1" : "placeholder" } , "timezone": "Europe/Helsinki" }
1c1
< { "double_field": 1,  "doc" : {"field_1":"placeholder"} , "timezone": "Europe\/Helsinki" }
---
> { "double_field": 1.0,  "doc" : { "field_1" : "placeholder" } , "timezone": "Europe/Helsinki" }
```

Kuva 5. Flumen ja Kafka Connectin väliset datamuunnokset.

Ensimmäisellä rivillä on alkuperäinen viesti, ohessa mihin muotoon viesti kääntyy Flumen ja Kafka Connectin käsittelyssä. Punaisella värillä on merkattu Flume ja vihreällä Kafka Connect.

Datamuunnos 1

Mikäli liukuluku päättyy tasalukuun (Kuva 5.), esimerkiksi 1.0, Flume tallentaa sen tietokantaan lukuna 1. Kafka Connectin osalta kerromme, että kyseessä on liukuluku ja 1.0.

Mikäli skeemamäärittelyssä kenttä on laitettu oikein, ei tällä ole mitään merkitystä tietokannan osalta vaan taulu on tietokantaan luotu oikeilla tietotyypeillä.

Datamuunnos 2

Array tyyppisissä kentissä (Kuva 5.) on ylimääräisiä välilyöntejä. Tämän datamuunnoksen juurisyy on käyttämämme kirjaston JSON jäsentäjässä. Flume käyttää eri kirjastoa tiedon jäsentämiseen, mikä oli myös vanhentunut. Teimme päätöksen, että emme halua ottaa uutta kirjastoa ylläpidettäväksi koska muutos on vain kosmeettinen eikä vaikuta tiedon käytettävyyteen. Tämä olisi ollut mahdollista toteuttaa käyttämällä samaa JSON jäsentäjässä mikä vanhassa toteutuksessa oli käytössä. Emme kuitenkaan halunnut lähteä riippuvuuksien päivitysrumbaan ja kirjaston vaihtoon kosmeettisen vian takia.

Datamuunnos 3

Emme erikseen tee ylimääräistä koodinvaihtomerkkiä kenoviivalle (Kuva 5.), Flume tekee tämän JSON jäsentäjän kautta.

Tarvitsimme vielä räätälöidyn viestinmuuntajan lisäksi yhden viestityypin muunnoksen. Kaikkien viestiemme mukana tuleva aikaleimakenttä ei tullut oikeassa muodossa tietokantaan vaan jouduimme käyttämään valmista viestinmuokkaajaa, joka muokkasi aikaleimat haluamaamme muotoon.

Mitään muita eroja emme testeissä havainneet, kun vertailimme Flumen tuottamaa dataa ja Kafka Connectin tuottamaa dataa. Meillä on tähän työhön itse valmistettu ohjelmisto mikä hoitaa datanvertailun massana ja ilmoittaa käyttäjälle eroavaisuudet jatkotarkastelua sekä toimenpiteitä varten.

Pientä datan kahdennusta oli havaittavissa jonoissa mihin tulee muutama viesti päivässä. Tämän juurisyy on vielä avoinna, mutta latauksemme ottavat vain uniikit viestit, joten tämä ei ole iso ongelma. Leimaamme jokaiseen viestiimme oman uniikin tunnisteen, kun vastaanotamme viestin Kafkan jonoon.

Integraatio tietokantaan toi myös mukanaan halutun muutoksen missä indikoimme tietokannalle, että meinaamme tehdä erälatauksen kantaan sisään ja tietokanta itse pystyy päättämään tehokkaimman tavan tuoda se sisään. Flumella tämä oli pakotettu aina käyttämään kopiointi funktiota mikä ei kaikissa tilanteissa ole se tehokkain tapa tuoda tietoja tietokantaan.

Huomasimme eron Kafka Connectin ja Flumen dataerän käsittelyssä. Kafka Connect käsittelee erää vain kuinka iso se voi maksimissaan olla. Flume odottaa, kunnes erä on täynnä tai konfiguraatiossa määritelty aika on kulunut ja tämän jälkeen lataa tiedot tietokantaan.

5.3 Integraatio Kafkasta BigQueryyn

Kafkalta Googlen BigQueryyn integroidessa käytimme Confluentin mukana tullutta liitännäistä. Liitännäinen lukee tiedot Kafka jonosta ja tallentaa tiedot meidän määrittelyjen mukaan BigQueryyn.

Liitännäinen piti huolen, että taulut BigQueryn puolella olivat kunnossa. Se pystyi luomaan taulut, sekä muokkaamaan taulut, jos esimerkiksi tuli uusia kenttiä mukaan.

Tapahtumat menivät BigQueryyn Streaming Insert rajapinta kutsun kautta. Rajapinnalla oli omat rajoituksensa. Emme pystyneet käyttämään tätä kaikille datalähteille vaan piti löytää ratkaisu datalähteille mitkä eivät uppoa tuon rajapinnan kautta.

Streaming Insert rajapinnan kautta isoimpina rajoituksina tuli viestin koko. Rajapinnan puolella raja on laitettu 10 megatavuun per kutsu tai viesti ja meillä on muutama lähde mistä tulee todella isoa viestiä. (Google BigQuery, Quotas and limits, 2022)

Liitännäinen tuki myös tietojen kierrätystä Google Cloud Storagen kautta. Tämän otimme käyttöön niiden jonojen osalta, mistä tuli viestejä, jotka eivät uponneet Streaming Insert rajapinnan kautta. Tarvittavat jonot kartoitettiin ja konfiguroitiin Kafka Connectin puolella käyttämään Cloud Storagea. Kafka Connectin liitännäinen hoitaa latauksen Google Cloud Storageen ja sieltä eteenpäin BigQueryyn. Tämä toiminnallisuus oli jo valmiina rakennettu liitännäiseen, meille jäi tämän osalta vain kartoitus ja määrittely mitkä tiedot kierrätetään Cloud Storagen kautta.

Aikaisemmin historianostot on tehty suoraan datajärvestä. Tulevaisuudessa historianostot tehdään BigQueryn puolelta. Kun ensimmäinen käyttötapaus tulee, rakennamme silloin työkalun millä dataa voidaan tuoda liiketoimintatietokannan käyttöön. Tätä ei rakennettu toteutusvaiheessa. Perustelimme sen sillä, että historialatauksia tulee suhteessa hyvin vähän. Liiketoimintatietokanta säilyttää kuitenkin laskeutumisalueella raakadatan muutaman kuukauden ajan ja ongelmat latauksissa yleensä huomataan siihen mennessä ja voidaan vain ladata uudestaan raakadatan laskeutumisalueelta. Päätöstä puolsivat myös, harvemmin historiadata on kriittistä tai kiireellistä saada sisään. Voimme rauhassa toteuttaa työkalun millä nämä tiedot saadaan liiketoimintatietokannan käyttöön.

5.4 Integraatio Kafkasta asiakasviestintä sovellukseen

Käytimme Flumea viestien välittämiseen asiakasviestinnän sovellukseen ja tämä toiminnallisuus piti myös toteuttaa Kafka Connectilla. Luemme asiakasviestien jonosta viestit mistä pitää asiakkaille lähettää välittömästi viestiä. Lähetämme viestin eteenpäin asiakasviestintä sovelluksen rajapintaan. Jonoon voi tietyt omat järjestelmämme laittaa viestin mikä sitten lähtee asiakkaalle eteenpäin.

Tähän emme löytäneet valmista liitännäistä vaan päätimme rakentaa itse. Netistä kuitenkin löytyi hyvin paljon valmiita liitännäisiä mitkä pystyvät jutella erilaisten HTTP rajapintojen kanssa. Niistä kaikista puuttui aina jotain pientä mitä tarvitsimme, Totesimme että tämä homma on helpompi tehdä täysin räätälöidyllä liitännäisellä.

Liitännäisen piti toteuttaa samat toiminnallisuudet kuin Flumelle tehty ominaisuus mikä oli lukea Kafka jonosta ja lähettää SOAP rajapintaan viestit eteenpäin sekä tallentaa kuittaukset lähetetyistä viesteistä.

Oman liitännäisen koodaaminen oli helppoa. Muutamassa päivässä saimme jo ensimmäisen version pihalle millä onnistuimme lähettämään paikalliseen jäljittelyrajapintaan tarvittavat tiedot. Muutaman päivän iteroinnin jälkeen meillä

oli valmis tuote millä pystyimme korvaamaan Flumen toiminnallisuuden millä viestimme asiakkaille.

Oman liitännäisen ohjelmoimisen helppous tuli tässä esille ja pystyimme varmistumaan kyvykkyydestä ohjelmoida liitännäinen itse, jos valmista liitännäistä mitä tarvitsemme ei ole. Pystymme sen kohtuullisella vaivalla ohjelmoida itse. Mitä enemmän joudumme toteuttamaan räätälöityjä liitännäisiä, sitä helpompaa se meille tulevaisuudessa on. Tämän tekeminen toi tiimiimme kyvykkyyttä mitä ei ennen meiltä löytynyt.

5.5 Suorituskykytestit

Suorituskykytestit pyrimme pitämään mahdollisimman neutraalina, Kafka Connect ja Flume käyttivät samaa testitietokanta klusteria sekä samaa viestijonoa ja viestijoukkoa. Samoja käyttäjätunnuksia tietokantaan, kaikki on pyritty pitämään identtisenä paitsi palveluiden tarvitsemat palvelimet ja asetukset. Molemmilla oli käytössään sama määrä virtuaalisuorittimia, mutta Flumella oli käytössään muistia huomattavasti enemmän.

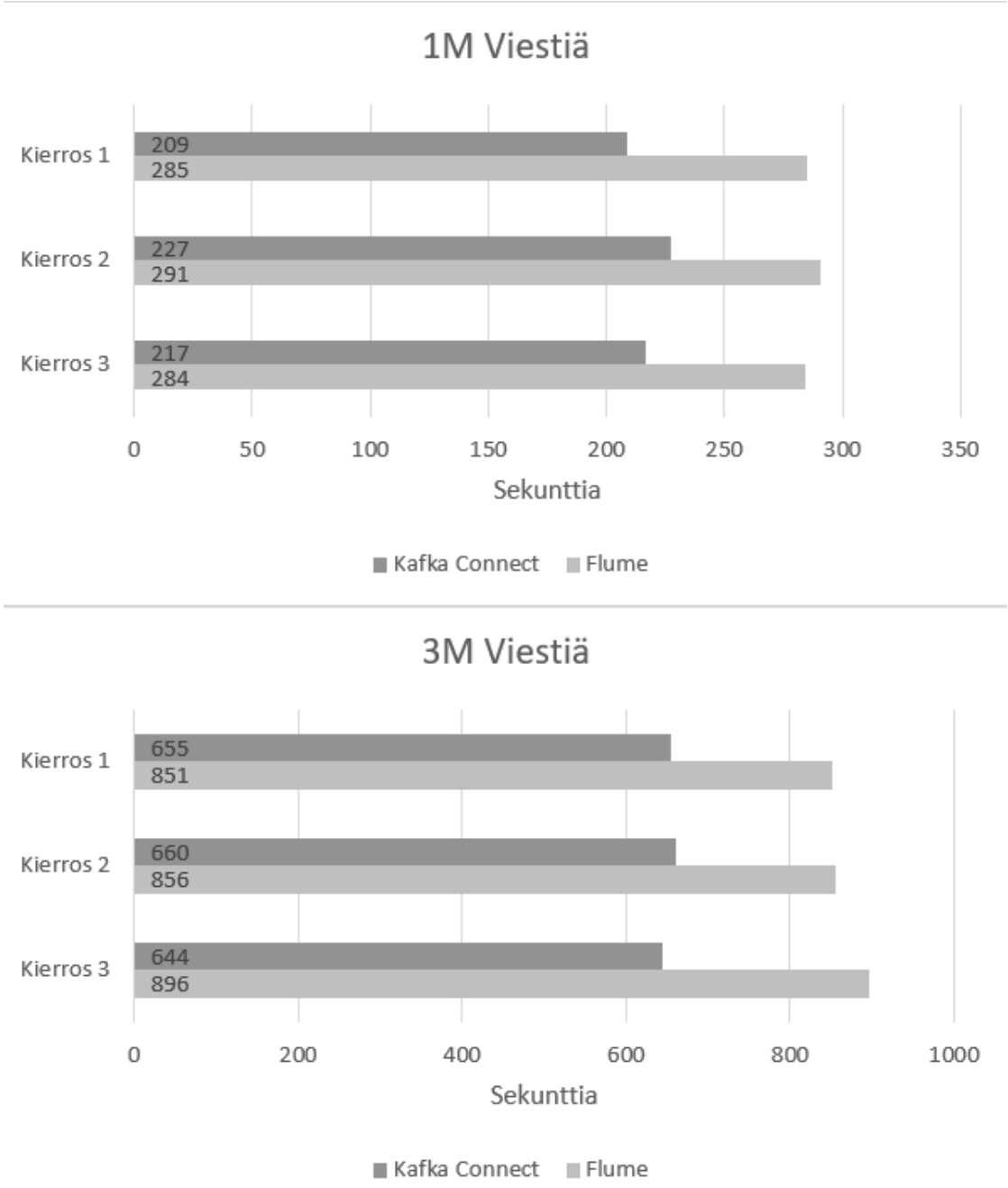
Myös tietokannan puolella molemmat käyttivät identtistä taulua määrityksiltään samassa skeemassa. Data vietiin Kafkan jonoon generoimalla 1M viestiä kerrallaan ja sen jälkeen käynnistämällä joko Flume tai Kafka Connect. Nopeus mitattiin tietokannan sisällä kentästä mihin muodostetaan automaattinen aikaleima, kun data tulee sisään. Ensimmäisen ja viimeisen viestin välinen aika on tulos. Tämä toistettiin kolme kertaa.

Tässä testissä tulee hieman ylimääräistä tietoliikennettä, kun neutraalisuuden nimissä molemmat tuotteet laitettiin pilveen, molemmat tuotteet joutuivat lukemaan jonon konesalista, tuomaan datan pilveen ja lähettämään tiedot eteenpäin tietokantaan mikä sijaitsi konesalissa.

Viestijonoksi mitä testasimme, valikoitui viesti missä oli käytetty useampaa räätälöityä viestinmuunnosta. Viestit generoitiin Kafkan jonoon käyttäen avoimen lähdekoodin K6 (K6, Open source load testing tool and SaaS for

engineering teams, 2022) nimistä ohjelmistoa, mikä on tarkoitettu erinäköisten kuormien generoimiseen lähteisiin. Tätä testiä varten kirjoitimme oman testausmoduulin mikä lähetti tarvittavat tiedot Kafkan jonoon vastaanottorajapintamme kautta.

Tuotteeseen voi tehdä myös synteettiset skenaariot esimerkiksi jäljittelemään miten kuormitus tuotannossa toimii.



Kuva 6. Flume ja Kafka Connect suorituskykytestit.

Suppeasta suorituskykytestistä (Kuva 6.) voimme päätellä, että Kafka Connect on noin 30 % nopeampi ilman kunnollisia optimointeja. Emme kuitenkaan uskonut, että tuote suoraan olisi tuonut näin suurta suorituskykyä, vaan uskomme vahvasti, että siirtyminen Java 8 → Java 11 versioon toi isomman muutoksen suorituskykyyn kuin tuotteen vaihto. Flumea emme kuitenkaan voineet testata Java 11 versiolla. Hallintaohjelma minkä kautta Flumea hallinnoidaan ja orkestroidaan, ei tue Java 11 versiota.

Olimme vain tyytyväisiä, että suorituskyky ei ollut huonompi kuin vanhalla toteutuksella, se oli isoin kysymysmerkki. Vaikka mitään erillisiä suorituskykyyn liittyviä vaatimuksia meillä ei ollut sen lisäksi että pitää pystyä samaan kuin vanha toteutus.

5.6 Monitorointi ja hälytykset

Palvelulle toteutettiin monitorointi sen omia työkaluja käyttäen. Toteutusta sekä palvelimia monitoroidaan riippuen ympäristöstä joko pilven omilla monitorointi työkaluilla tai Prometheuksen liitännäisellä, node exportterilla mikä tuo metriikkaa saataville palvelimilta. Itse palvelun monitorointi toteutettiin tuotteen sisäänrakennetulla Java Management Extensions moduulilla. Metriikka haetaan Prometheuksen avulla ja visualisoidaan tämän jälkeen Grafanaan.

Grafanaan rakentelimme meidän käyttöömme hyödyllisen kojelaudan (Kuva 7.) mistä näemme vilaukselta kaikki tarvittavat tiedot.

Kuva 7. Grafana kojelauta Kafka Connectin monitorointiin.

Kuvasta näkyy palveluiden tilanne, työskentelijäyksiköiden määrät sekä niiden tehtävät, kuinka monta on epäonnistunut ja kuinka monta on pysäytetty. Samoista metriikoista näemme myös helposti muistinkäytön sekä kuinka kauan Kafkalle kuittauksissa kestää keskimäärin.

Näiden tietojen pohjalta loimme myös hälytyksiä mistä tulee tarvittaville henkilöille automaattisesti tiedot, mikäli joku on pielessä.

5.7 Tietoturva

Toteutukselle tehtiin auditointi ulkoisen kumppanimme toimesta, auditoinnista ei löytynyt mitään kriittistä tai hälyttävää mikä olisi estänyt käyttöönoton. Korjauksia toteutettiin auditoinnin pohjalta. Auditoinnin tulokset ja korjaustoimenpiteet sisältävät muun muassa seuraavaa:

- Kovensimme palomuuriasetuksia ja poistimme ylimääräisiä palomuurisääntöjä
- Muokkasimme tiedosto-oikeuksia, ainoastaan käyttäjällä millä ajamme mikropalvelua, on lupa lukea tiedostoja mikropalvelun ulkopuolelta. Tarkoitus on integroitua ulkoiseen salaisuuksien hallintajärjestelmään, kuten HashiCorpin Vaulttiin (Vault Manage Secrets & Protect Sensitive

Data, 2022) tai Googlen Secret Manager (Google Secret Manager, 2022) palveluun.

- Lisäsimme rajapinnan autentikoinnin taakse (Confluent HTTP Basic authentication, 2022), tuote ei tällä hetkellä tue rooliperusteista oikeuksien hallintaa.
- Auditoinnissa myös huomattiin Docker palvelun konfiguraatiossa korjattavaa, palvelua ajettiin pääkäyttäjä oikeuksilla. Vaikka itse toteutusta konteinerin sisällä ei ajeta pääkäyttäjä oikeuksin niin palvelimella Docker on ajossa pääkäyttäjäoikeuksin. Tämä muutos pienentää hyökkäyspinta-alaa. Siirrämme palvelun toimimaan normaalikäyttäjän oikeuksin pääkäyttäjältä (Docker Run the Docker daemon as a non-root user (Rootless mode), 2022)

Auditoinnissa käytiin läpi pilviympäristömme, tietoturvan asetukset ja kovennukset pilviympäristössä, virtuaalikoneiden asetukset sekä toteutuksen tietoturvallisuus.

Toteutukselle tehtiin myös ohjelmiston koostumuksen analyysi missä kartoitettiin toteutuksen haavoittuvuudet ja lisenssit. Tämä automatisoitiin aina kun tulee muutos tai päivitys toteutukselle niin, automaattisesti muutoksen yhteydessä toteutukselle tehdään uusi kartoitus. Ajastimme myös viikoittaisen kartoituksen, mikäli emme päivitä tuotetta viikkoon, se tarkastetaan joka tapauksessa haavoittuvuuksien ja lisenssien osalta. Kartoitus tehdään Atlassianin työkalulla nimeltä Whitesource (Whitesource, Software Composition Analysis, 2022), saamme tästä automaattisen raportin sekä hälytyksen, jos toteutuksesta löytyy kriittinen haavoittuvuus mikä vaatii välitöntä toimenpidettä.

5.8 Työkalut

Toteutusta tehdessä huomasimme, ettemme löytäneet Kafka Connectin hallintaan valmista työkalua mikä hoitaisi kaikki tarpeemme. Kokeilimme useita

eri työkaluja mutta mikään kokeilemamme työkalu ei pystynyt hallitsemaan liitännäisiä massana mikä oli tärkein kaipaamamme ominaisuus.

Työkalun vaatimuksina työkalulla oli, että pystyi käsittelemään konfiguraatiot massana mutta tarvittaessa porautumaan yksittäiseen liitännäiseen asti joko graafisen tai komentopohjaisen käyttöliittymän kautta. Klusteria pitää pystyä myös hallinnoimaan työkalun kautta.

Lenses.io:n Kafka Connect UI (Web tool for Kafka Connect, 2022) oli ensimmäisiä mitä kokeilimme, se toimi muutamalla liitännäisellä hyvin, mutta kun skaalattiin ylöspäin testattavia lähteitä, kyseinen työkalu hyytyi täysin. Sekä siitä puuttui muutamia meille tärkeitä ominaisuuksia, kuten klusterin hallinta ja liitännäisten käsittely massana.

Tämän jälkeen kokeilimme Kowl-ui:ta (Kowl - A Web UI for Apache Kafka, 2022) missä oli myös samoja puutteita. Tällä työkalulla suurikaan määrä liitännäisiä ei ollut ongelma. Isoimpana ongelmana oli, että liitännäisiä ei voinut hallita massana, liitännäisten tilat, sovelluslokit ja virheet kyllä näkyivät työkalun kautta. Tämä oli ainoa työkalu mikä tuki useampaa Kafka-Connect klusteria samanaikaisesti.

Kokeilimme myös Xeotekin Kadeckkia (Xeotek KaDeck - The Apache Kafka Desktop Client & Web UI, 2022), tämän työkalun osalta päädyimme samoihin tuloksiin kuin edellisillä. Ei täyttänyt kaikkia kriteereitämme.

Alkuun ajoimme käsin tehtyjä pieniä shell-skriptejä hallinnoimaan klusteria, mutta mitä pitempään niitä käytettiin sitä vakuuttuneempia meistä tuli että, tähän tarvitaan täysin oma työkalu millä voisi hoitaa kaikki käyttötarpeet.

Erilaisten työkalujen kokeiluun oli käytetty jo sen verran aikaa, että päätimme tehdä oman työkalun millä hallita klusteria. Confluent (Confluent Kafka Connect REST Interface, 2022) tarjoaa hyvät REST rajapinnat Kafka Connectin hallitsemiseen mitkä helpottivat tarvittavien työkalujen tekoa. Ohjelmat kirjoitettiin pythonilla. Toteutetut työkalut pystyivät hoitamaan meille tärkeät tehtävät mitkä kokeilluista ohjelmistoista puuttuivat. Näiden tuloksena tuotimme

2 erilaista työkalua, jotka tekevät hyvin erilaista hommaa, vaikka käyttävät samaa rajapintaa.

Työkalu #1 - Konfiguraatiogeneraattori

Konfiguraatiogeneraattori hoitaa uusien liitännäisten konfiguraation generoinnin automaattisesti annetuilla parametreillä. Voimme myös päivittää kaikki liitännäiset tämän kautta, jos esimerkiksi päivityksessä tulee uusia parametreja mitä tarvitsemme, tai jotain pitää yhteisesti muuttaa kaikissa. Tällä työkalulla hallinnoimme Kafka Connect liitännäisten konfiguraatiota eri lähteisiin. Esimerkiksi BigQuery liitännäisiä varten työkalu luo automaattisesti tarvittavat Google Cloud Storage ämpärit datalähteille mitä emme voi pistää suoraan BigQuery Streaming inserttien kautta rajoituksien vuoksi. Esimerkiksi tapahtumat missä viestin koko on isompi kuin 10 megatavua. Tätä rajoitusta ei voi kiertää tai nostaa, se on BigQuery streaming inserttien ominaisuus.

Työkalu #2 - Hallintatyökalu

Tällä työkalulla hallinnoimme Kafka Connect klusteria. Kyseisellä työkalulla voimme hoitaa massana koko klusteria, esimerkiksi sammuttaa tai käynnistää uudestaan. Tarpeen mukaan voimme myös hallinnoida yhtä liitännäistä, miten sen parhaaksi näemme. Esimerkiksi jos yksi liitännäinen oireilee, pääsemme työkalulla kiinni suoraan sen liitännäisen virhetietoihin kiinni. Työkalulla näemme suoraan missä on ongelma minkä jälkeen korjaustoimenpiteet voivat alkaa. Myös liitännäisten poistot onnistuvat massana tai sitten yksittäin.

5.9 Automatisointi, jatkuva integraatio ja jatkuva käyttöönotto (CI/CD)

Toteutusta lähdettiin automatisoimaan Ansiblella (Ansible, Ansible is Simple IT Automation, 2022). Teimme pelikirjat ja erilliset kohdelistat, miten mihinkin ympäristöön toteutuksen voi ottaa käyttöön yhdellä komennolla. Valitsimme Ansiblen tuotteeksi koska joudumme ainakin alkuvaiheessa tukemaan hybridi ympäristöä, pilveä ja konesalia. Tulevaisuudessa saatamme siirtää palvelun esimerkiksi Googlen Cloud Buildin (Google Cloud Build, 2022) päälle tai muun

pilvinatiivin CI/CD työkalun päälle. Toteutuksen rakentaminen konteinereihin rakennettiin Jenkinsin (Jenkins, Build great things at any scale, 2022) päälle ja käytämme kahta erillistä konttisäilöä, Googlen Container Registryä ja Dockerin omaa Container Registryä (Docker Registry, 2022), tulevaisuudessa nämä ovat tarkoitus konsolidoida GitLabin tarjoaman Container Registryn (Gitlab Container Registry, 2022) päälle. Gitlab työkalua ei vielä otettu käyttöön yrityksessä kaikilla osastoilla vaan käyttöönotto on kevään aikana. Samalla on tarkoitus siirtää kaikki Jenkinsin päällä pyörivät konteinereiden rakennukset ja käyttöönotot tuotteen piiriin.

6 Ongelmat toteutuksen aikana

Toteutus ei ollut täysin ongelmaton mutta saimme ratkaistua havaitut ongelmat kuitenkin jo kehitysvaiheessa, mitkä olisivat estäneet käyttöönoton. Yksi ongelma kuitenkin jäi vielä tarkempaan selvitykseen.

6.1 Avro skeema viestin mukana

Koska joudumme ajamaan Flumea ja Kafka Connectia rinnakkain, kulkee meillä jonoissa viestin mukana sisäänrakennettu skeema, Kafka Connect ei tue tätä oletuksena vaan käyttää skeemarekisteriä mihin viitataan viestissä olevilla taikabiteillä. Koska meillä ei tätä ollut, ei viestin deserialisointu onnistunut suoraan tuotteella. Skeemarekisteri on meillä käytössä jo vastaanottorajapinnassa, kun validoimme tietoja sisään Kafkalle ja on luonnollista laajentaa sen toimivuutta myös viestin välittäjään.

Kafka Connectiin kuitenkin pystyi tehdä omia räätälöityjä avainarvo muuntimia. Nämä hallitsevat miten Kafka Connect serialisoi ja deserialisoi viestejä Kafka Jonosta. Löysimme työmme rungoksi vanhan avoimen lähdekoodin projektin. Projektia ei oltu muutamaan vuoteen päivitelty. Teimme tarvittavat muutokset ja päivitimme sen toimimaan uusien Kafka versioiden kanssa. Oletuksena löydetty projekti kuitenkin käytti tiedostopohjaisia skeemoja mistä olemme halunneet eroon jo Flumen kanssa. Päivitimme muuntimen hakemaan suoraan skeemarekisteristä tarvittavan skeeman mikä vastaa Kafka jonoa, tämän jälkeen Kafka Connect käytti skeemaa validointiin sekä tietojen tallentamiseen tietokantaan, BigQueryyn sekä asiakasviestintä järjestelmään.

Tämä on varsinaisesti vain yliheittovaiheen ongelma. Kun Flumesta päästään eroon pääsemme eroon myös tästä ongelmasta pysyvästi. Tämä vähentää myös käytetyn tilan määrää, kun skeemaa ei tarvitse kuljettaa viestissä mukana enää Kafkalla.

6.2 Kafka – Timeout ongelmat

Seuraava ongelma ilmeni, kun otimme isomman määrän liittimiä käyttöön. Huomasimme että Kafka alkoi oireilemaan. Kafkan suorituskyky ja toiminta heikkeni aina kun toimme ison määrän liittimiä käyttöön. Tämä johtui siitä, kun toimme uusia viestien tilaajaryhmiä kerralla ison määrän. Tämä korjattiin Kafkan palvelimen parametrimuutoksella, että oletuksena ryhmä ei ole käynnissä 3 sekunnin päässä, vaan Kafka odottaa pari minuuttia ennen kuin alkaa käsittelemään tätä ryhmää aktiivisena. Tämä vähensi osioiden uudelleen balansointia ryhmien kesken. Tässä on myös iso ero, miten tämä toimi verrattuna Flumeen. Flumella meillä oli aina käytössä yksi ryhmä, nyt jokainen jono on oma ryhmänsä paremman hallittavuuden takia.

Tässä vaiheessa Kafka:n KIP (Kafka Improvement Proposals, 2022) osio tuli hyvin tutuksi, meitä koskettava ongelma kulki nimellä: Kafka, KIP-134: Delay initial consumer group rebalance. KIP:istä löytyvät kaikki Kafkaa koskevat isot keskusteltavat parannusehdotukset sekä prosessikuvauksen, kuinka Kafkan avoimen lähdekoodin projektissa muutoksia tehdään.

Siellä oli kuvattu suoraan meidän ongelmamme ja miksi muutos on tehty sekä mitä meidän pitää tehdä että saamme Kafka Connectin tilaajaryhmät toimimaan tehokkaasti ilman häiriöitä ja ongelmia. Muutostiketillä ongelma on kuvattu seuraavasti (KIP-134: Delay initial consumer group rebalance, 2019):” When a new consumer group is created the current behaviour will result it at least two rebalances for groups with more than a single member. These rebalances can become expensive depending on what needs to be initialized and torn down. For example, during a rebalance of stateful streams processing application it will typically need to initialize any local state, often by replaying from a changelog. When partitions are revoked state needs to be persisted, which can involve flushing to disk, snapshotting, and writing to a changelog.

Consumers often don’t all start up at the same time resulting in many rebalances. As each rebalance has to go through the initialization and close

phases, it can become a lengthy process to get an application into a steady running state.”

Lisäsimme parametrin ”group.initial.rebalance.delay.ms=120000” Kafkan palvelimen konfiguraatioon ja tämä ongelma saatiin pois päiväjärjestyksestä, oletuksena oleva 3 sekuntia ei riittänyt meidän tarpeeseemme. Jos toimme ryhmät rauhallisesti ja pienissä erissä, ongelmaa ei myöskään esiintynyt.

6.3 Kafka – “Stop the world” ongelmat

Ensimmäisen ison ongelman jälkeen huomasimme jälleen, että Kafka oireilee muutoksiemme jälkeen, nyt vain hieman eri tavalla. Koko Kafka uudelleen balansoi jatkuvasti itseään eikä ota viestejä sisään jonoihin sillä välin kuin uudelleen balansointi on käynnissä. Arvelimme tämän johtuvan, kun toimme Kafkalle noin 800 uutta jonoryhmää. Tutkinnan jälkeen löysimme vahvistuksen ongelmalle. Olimme törmänneet niin sanottuun ”Stop the world” ongelmaan Kafkan kanssa missä oma toimintamme pysäyttää koko Kafkan maailman.

Löysimmekin pian Kafkan kehittäjien kirjoituksen aiheesta, Confluent: Incremental Cooperative Rebalancing in Apache Kafka: Why Stop the World When You Can Change It? (Confluent. 2019.)

Kirjoituksessa käytiin läpi täysin meidän ongelmamme ja mitä kehittäjät ovat tehneet sen korjaamiseksi. Tästä syntyi KIP-429: Kafka Consumer Incremental Rebalance Protocol sekä KIP-415 Incremental Cooperative Rebalancing in Kafka Connect, muutostiketeillä on käyty yksityiskohtaisesti ongelmat ja muutokset läpi.

Meidän tapauksessamme tämä tarkoitti vain protokollan vaihtoa tilaajaryhmillemme. Lisäsimme työkaluumme mikä generoi meille konfiguraatiot seuraavan uuden parametrin.
”**consumer.override.partition.assigment.strategy=org.apache.kafka.clients.consumer.CooperativeStickyAssignor**”, tämä parametri mahdollistaa sen, mikäli yksittäinen jono esimerkiksi verkkolatenssin tai muun ongelman vuoksi

tippuu Kafkan listoilta. Ainoastaan kyseinen jono uudelleen balansoi itsensä eikä kaikki Kafkan jonot. Kafka ei myöskään jumiudu enää, kun uudelleen balansointia tapahtuu. Kafka versio mikä meillä oli myös käytössä, vaikutti tähän huomattavan paljon, emme ajaneet uusinta Kafka versiota. Tämä asetus olisi ollut oletus, mikäli Kafka versiomme olisi ollut 3.0 tai uudempi. Päivitimme ongelman korjauksen jälkeen Kafkan versioon 3.1 ja nyt kaikki uudet ja vanhat Kafkan päälle rakennetut jonojen tilaajat ovat oletuksena yhteistyö protokolla asetuksilla, innokkaan protokollan sijaan.

6.4 Kafka – Vertica arrayt

Datan suhteen oli ongelmia toteutuksen vaiheessa. Vertica on kolumnäärinen tietokanta ja haluaa että kaikki taulut olisivat mahdollisimman litteitä. Jos toisimme kaikki tiedot tietokannan haluamassa muodossa, niin tämä vaatisi todella isoa työtä ETL puolelta latausten muutoksissa. Oletuksena Vertica omat JSON jäsentäjät tekee datasta hyvinkin litteää, toteutuksessa datan jäsennys ja serialisointi Vertican haluamaan muotoon on kuitenkin rakennettu Kafka Connectin päälle. Tästä pääsemme ongelmaamme, että kaikki merkkijonojoukko tiedot pitää tulla tietokantaan yhtenä merkkijonona yhteen kenttään.

Valittu tuote ei tätä suoraan laatikosta otettuna tue, vaan jouduimme tekemään oman datanmuokkaajamme hoitamaan datan haluttuun muotoon.

Tähän työhön kartoitimme, että MongoDB:n BSON dokumenttityyppi on juuri haluttua tyyppiä mitä tarvitsemme. MongoDB on tehnyt myös oman avoimen lähdekoodin liitännäisensä Kafka Connectiin. Sieltä oli saatavilla valmis deserialisointi ja serialisointi kirjasto mikä auttoi meitä keksimättä pyörää uudestaan. Valmiin kirjaston ansiosta pääsimme nopeasti kehittämään vaadittuja ominaisuuksia.

Uudet versiot Verticasta tukevat monimutkaisia datatyyppejä. Tämä saattaa olla tulevaisuudessa ratkaisu, jos haluamme purkaa räätälöidyt ratkaisumme. Vertican monimutkaiset datatyypit pystyvät ottamaan datan vastaan siinä

muodossa kuin se on meillä ennen tietokantaan kirjoitusta. Tämä kuitenkin vaatii huomattavia muutoksia meidän toteutuksemme logiikkaan, tietokantataulujen tyyppimäärittelyihin sekä ETL latauksiin mikä kasvattaa muutoksen kokoa ja vaativuutta. Vertica kuvailee (Vertica, Complex types, 2022.) tuetut monimutkaiset tyyppinsä seuraavasti:” Complex types such as structures (also known as rows) and arrays are composed of primitive types and sometimes other complex types. Complex types can be used in the following ways:

- Arrays and rows (in any combination) can be used as column data types in both native and external tables.
- Sets of primitive element types can be used as column data types in native and external tables.
- Maps can be used only in external tables using Parquet or ORC data.
- Arrays and rows, but not combinations of them, can be created as literals, for example to use in query expressions.” (Vertica, Complex types, 2022)

BigQueryssä datahan on tässä toteutuksessa jo jaoteltu monimutkaisiin kenttiin. Tämän muotoisen datan muoto tulee enemmin tai myöhemmin käyttöön, mikä tukisi myös, että tämä siirtymä toteutetaan Vertican osalta ennemmin tai myöhemmin. Monimutkaisten kenttien käyttö BigQueryn puolella on hyvin triviaalia, niistä voi hakea haluttuja tietoja standardi SQL kyselyillä.

Vertican puolella kuitenkin kyselyt ovat hieman monimutkaisia mutta eivät mahdottomia opetella tai toteuttaa ETL latauksiin.

6.5 Kafka – BigQuery datasettien poistaminen

Toteutusta tehdessä huomasimme, jos poistamme BigQueryn puolelta datasetin, ei Kafka Connect automaattisesti osaa enää tallentaa kyseiseen datasettiin tietoja, mikäli datasetti luodaan samalla nimellä takaisin. Lokitiedosta kun katselimme niin Kafka Connectin BigQuery liitännäinen koitti lähetellä tietoja väärille maantieteellisille alueille Googlen BigQueryn puolella. Meillä

datasettien käyttö on rajoitettu vain Suomen sijaintiin. Pystyimme toistamaan toiminnan mutta emme korjaamaan sitä Kafka Connectin puolella. Liitännäisessä ei ole tapaa kertoa täsmällisesti, että datasetti sijaitsee BigQueryssä tietyssä sijainnissa. Tämä ongelma ei vaikuta meidän käyttöömme, testausvaiheessa oli vain helpompi siivota koko datasetti pois, kun koittaa muistaa mitä asetuksia olimme säätäneet datasetistä. Toiminnallisuudesta on hyvä olla tietoinen. Mikäli tulee tarvetta poistaa datasetti, ennen liitännäisen korjaamista, muistaa vaan luoda sen eri nimellä kuin vanha ja ongelmaa ei ole.

Googlen dokumentointi kertoo (Google BigQuery, Error Messages, 2022.), että datasettien metadata saattaa olla useamman tunnin vielä saatavilla. Ajattelimme alkuun, että ongelma johtuisi siitä. Odottelimme alkuun pari päivää, jos pystyisimme tallentamaan sen jälkeen datasettiin tietoja. Odottamisella ei ollut mitään vaikutusta ja lataukset näyttivät vieläkin sovelluslokien mukaan menevän väärälle alueelle. Jostain syystä liitännäinen koitti tallentaa datoja Googlen oletus alueelle mikä sijaitsee Amerikassa.

6.6 Kafka – BigQuery Liian isot viestit

Toteutusvaiheessa huomasimme myös, että kaikki viestit eivät menneet BigQueryyn, vaan Kafka Connectin liitännäinen oireili muutaman jonon kohdalla. Tämän selvitys pikaisesti johti siihen, että huomasimme laittavamme liian isoja viestejä BigQueryyn Streaming Inserts rajapinnan kautta missä on rajoituksia viestien koolle.

Kartoitimme kaikki lähteet mistä voi tulla lähellä rajoituksia olevia viestejä ja teimme niille poikkeukset konfiguraatiogenerointi ohjelmaamme. Konfiguraatiogeneraattorin kautta pystymme helposti hallitsemaan mitkä jonot kiertävät Google Cloud Storagen kautta ja siirtyvät normaaleina latauksina BigQueryyn.

Kartoituksesta selvisi noin pari kymmentä jonoa mitä kierrätämme Google Cloud Storagen kautta. Jonot näyttivät olevan pitkälti jonoja mihin tulee

harvakseltaan dataa eikä viestimäärät ole kauhean isoja, ainoastaan viestien koko ylittää rajapinnan rajoitukset.

Mitään erikoisratkaisuja konfiguraatiogeneraattorin muutosten lisäksi ei tarvinnut tehdä, Kafka Connectin liitännäinen tuki Google Cloud Storageen tallentamista valmiiksi ja, vaati vain muutaman konfiguraatioparametrin mikä kertoi toteutukselle, että nämä jonot kiertävät Google Cloud Storagen kautta.

Seuraavat parametrit vaadittiin datan kierrättämiseksi Google Cloud Storagen kautta (Confluent, Google BigQuery Sink Connector Configuration Properties, 2022)

- EnableBatchLoad
    - **Beta Feature** The sublist of topics to be batch loaded through GCS.
- gcsBucketName
    - The name of the bucket where Google Cloud Storage (GCS) blobs are located. These blobs are used to batch-load to BigQuery. This is applicable only if enableBatchLoad is configured
- batchLoadIntervalSec
    - The interval, in seconds, in which to attempt to run GCS to BigQuery load jobs. Only relevant if enableBatchLoad is configured.

Kafka Connectin voi myös määritellä luomaan omat ämpärit Google Cloud Storageen. Me kuitenkin hoidimme sen konfiguraattorigeneraattorin puolella, sillä olisimme joutuneet antamaan muuten palvelutilille liian laajat oikeudet Google Cloud Storageemme mitä emme halunneet. Koska lataus Google Cloud Storagen kautta on vielä beeta vaiheessa, voi olla, että tulevaisuudessa tuota Google Cloud Storagea ja sen sijaintia voi paremmin hallinnoida. Tämä on meidän ratkaisumme, kunnes pystymme määrittelemään vain yhden sijainnin, minne Kafka Connect liitännäinen pystyy generoimaan tarvittavat alikansiot.

6.7 Viestien kahdentuminen

Toteutuksen aikana meillä oli ongelmaa viestien kahdentumisen kanssa. Osa viesteistä kahdentui ilman hyvää syytä kohdetauluun tietokannan puolella. Huomasimme Kafka 3.1 julkaisussa oli korjattuna yksi bugi mikä näytti liittyvän meidän ongelmaamme.

Tiketti mikä liittyi meidän ongelmiimme Kafkan tiketöinti järjestelmästä:

- KAFKA-13472: Connect can lose track of last committed offsets for topic partitions after partial consumer revocation (Kafka, KAFKA-13472, 2021)

Tämän seurauksena päivitimme oman Kafka versiomme versiosta 2.6 uusimpaan 3.1.0 versioon ja tutkimme tilanteen päivityksen jälkeen. Päivityksen jälkeen viestien kahdentuminen poistui lähes kokonaan. Nyt kahdentumiset ovat poistuneet lähes kokonaan. Ainoastaan jonot mihin tulee harvoin uutta dataa näyttää silloin tällöin tuplautuvan sekä sovelluksen lokeissa näkyy olevan aikaviivekatkaisu virhettä. Tämä pieni virhe ei estä meitä etenemästä toteutuksen kanssa, mutta on hyvä korjata. Tästä jäi korjaustiketti työnkoordinointi sovellukseemme ja priorisoimme tämän työlistalle, kun priorisointi ja resurssimme sen sallivat.

7 Tulokset ja pohdinta

Tuloksena tästä opinnäytetyöstä meillä on käytössä uusi datanvälittäjäkomponentti mikä on kustannustehokkaampi pienempien käyttökustannusten ja avointen lähdekoodin ratkaisujen puolesta. Toteutusta on myös helppo hallita isolla skaalalla tai pureutua yksittäiseen liitännäiseen. Pystymme skaalautumaan usealla tavalla, jos johonkin jonoon tulee liian iso määrä viestejä mitä eivät oletusasetukset pysty hoitamaan mallikkaasti. Pystymme lisätä saumattomasti lisää työskentelijäyksiköitä. Jos näillä ei vielä pystytä vastaamaan kasvaviin datamääriin niin pystymme helposti lisäämään uusia konteinereita klusteriin, tai jos huomaamme että kontit käyvät tyhjällä pystymme yhtä helposti niitä poistamaan ja skaalaamaan alaspäin.

Toteutuksesta tehtiin pilviriippumaton. Teimme toteutuksesta hybridimallisen, ratkaisun. Voimme laittaa sovelluksen pyörimään pilveen tai konesaliin. Ensimmäisen vaiheen toteutus on sijoiteltuna konesaliin sekä pilveen. Mikäli datajärvi tai tietokanta jostain syystä vaihtaa toiseen teknologiaan voimme kytkeä vain uuden kohteen mihin laitamme jonoista viestit kulkemaan.

7.1 Tulokset ja toteutuksen onnistuminen

Tuloksina opinnäytetyöstä tuli myös erittäin paljon uutta kyvykkyyttä mitä meillä ei ennen tätä ollut. Pystymme entistä paremmin vastaamaan tulevaisuuden tarpeisiin.

Projektia toteuttaessa työnohjaus sovellukseemme tuli yhteensä 76 tehtäviä, joista valmistui 66 kappaletta. Jäljelle jääneet tehtävät ovat lähinnä parannusasioita mitä teemme, kun aika ja resurssointi on sopiva. Saimme toteutettua kaikki halutut ominaisuudet mitä suunnittelimme ja opimme todella paljon lisää Kafkan ekosysteemistä.

Toteutus onnistui annetuissa aikatauluraameissa ja käytössämme olevilla resursseilla.

7.2 Jatkojalostus ja tulevaisuuden pohdinta

Ohessa lyhyitä mietteitä miten pystymme opinnäytetyön jälkeen rakentamaan uusia kyvykkyyksiä teknisessä mielessä sekä käytössä kun aika, priorisointi ja resurssointi sen sallivat toimintaympäristössämme.

Kafka Connectin voisi siirtämään toimimaan Kuberneteksen päälle mikä tuo mukanaan kyvykkyyden skaalata tuotteita automaattisesti tarpeen mukaan. Tähän tarkoitukseen suunnittelemme käyttävämme Googlen pilvestä saatavaa Google Kubernetes Engineä (GKE), tämä poistaisi tarpeen skaalata uusia konteinereita käsin.

Yhtenä nostona otan myös Flumen ominaisuuksien purkamisen nykyisestä ympäristöstä. Nykyinen ympäristö oli pitkälti rakennettu Flumen ehdoilla, nyt kun tuote ei ole enää käytössä pystymme aloittamaan vanhojen ominaisuuksien purkamisen pois. Ensimmäisenä ja isoimpana on skeeman kuljettaminen pois kaikista Avro serialisoiduissa viesteissä.

Pystymme myös alkamaan käyttää laajemmin Kafkan ekosysteemiä, kun saamme jokaiseen Avro serialisoituun viestiin viittauksen skeemarekisteriin tapahtuman skeemasta. Tämä mahdollistaa meidän käyttävän esimerkiksi KSQL ohjelmistoa, mikä on tarkoitettu reaaliaikaisen datan reaaliaikaiseen prosessointiin. Tämä tuo meille kyvykkyyden tuottaa tapahtumavirrasta SQL pohjaista ETL työtä reaaliajassa, tämän tyyppistä kyvykkyyttä tarvitsemme tulevaisuuden arkkitehtuurinäkymissä.

Myös yhtenä ideana olen pohtinut automatisoinnin lisäämistä. Mikäli alamme laajentamaan Kafkan ekosysteemin käyttöä, siinä vaiheessa tulee infrastruktuurin automatisointi entistä isommalle käytölle. Ehdotuksena tälle ongelmalle on infrastruktuurin automatisointi esimerkiksi Terraformin avulla. Tämä pystyttäisi kaikki tarvittavat infrastruktuurit ja palvelut mitä tarvitsemme tulevaisuudessa ja voisimme myös mahdollisesti luoda kevyt versioita ympäristöstämme tämän avulla.

Opinnäytetyö tuo myös kyvykkyyttä tarjota tuottamaamme tapahtumapohjaista tietoa laajemmalle joukolle talossa. Tiedolla johtaminen on nousemassa yhdeksi isoksi asiaksi yrityksessämme ja opinnäytetyön toteutus tukee tätä hyvin paljon. Ennen emme pystyneet toimittamaan dataa kauhean ketterästi ilman raskasta integraatiota, toteutuksen ansiosta pystymme toimittamaan tapahtumia entistä ketterämmin. Suoraan hyllyltä tuotteessa on tuet suurimpiin tietokantoihin ja alustoihin. Mikäli nämäkään eivät auta, pystymme ongelmitta toteuttamaan oman räätälöidyn toteutuksen mikä vie tapahtumat kohteeseen.

8 Yhteenveto

Tämän opinnäytetyön tarkoituksena oli toteuttaa tapahtumapohjaisen välittäjäkomponentin vaihdos modernimpaan ja tuoda uusia kyvykkyyksiä tapahtumien välittämiseen ja Kafkan ekosysteemin ympärille.

Ongelman kuvauksessa käydään läpi perustelut minkä takia haluamme vaihtaa komponentin toiseen. Pääasiallisena syynä on komponentin kehityksen lopetus sekä ympäristön alasajo missä pyöritämme kyseistä komponenttia.

Työssä kartoitamme vaihtoehtoja millä tuotteella pystymme korvaamaan tuotteen, testauslistalle päätyy kolme komponenttia, Apache Beam, Apache Nifi sekä Kafka Connect. Työssä käydään läpi valinta perustelut Kafka Connectin valinnalle komponentiksi millä muutos toteutettiin. Painavimpana syynä oli toteutuksen tiukka aikataulu ja toteuttavan tiimin mielipide siitä pystymmekö toteuttamaan vaihdoksen annetussa aikataulussa ja annetuilla resursseilla.

Seuraavana vaiheena opinnäytetyössä oli kartoittaa tarvittavat välttämättömät integraatiot, minkä perusteella pystyimme rajaamaan, mitä kaikkea joudumme tekemään valitulla komponentilla. Tärkeitä integraatioita toteutettiin kolme kappaletta, integrointi liiketoimintatietokantaan, integrointi uuteen datajärveen sekä integrointi asiakasviestintäsovellukseen mihin jouduimme toteuttamaan räätälöidyn tapahtuman välittäjän Kafka Connectin päälle.

Toteutukselle tehtiin tietoturva-auditointi ulkoisen kumppanimme puolesta, mistä seurasi muutama korjattava kohta. Auditoinnista ei löytynyt yhtään kriittistä tai korkean luokan haavoittuvuutta mikä olisi estänyt käyttöönoton.

Työn lopputuloksena saatiin vaihdettua komponentti uuteen ja modernimpaan komponenttiin mitä kehitetään jatkuvasti. Toisina tuloksina tuli myös muutama työkaluja millä pystymme hallinnoimaan toteutusta haluamallamme tavalla sekä päivittämään konfiguraatioita tarvitsemallamme tavalla.

Pystymme myös jatkossa uusien kyvykkyyksien myötä tarjoamaan paremmat tapahtumapalvelut sidosryhmille.

Lähteet

Amazon, What is an Event-Driven Architecture?, 2022. Viitattu 29.3.2022 https://aws.amazon.com/event-driven-architecture/

Ansible, Ansible is Simple IT Automation, 2022. Viitattu 11.1.2022. https://www.ansible.com/

Apache Avro™ is a data serialization system, 2022, Viitattu 13.2.2022. https://avro.apache.org/

Apache Beam, An advanced unified programming model, 2021. Viitattu 22.12.2021. https://beam.apache.org/

Apache Flume, Dataflow model, 2019, Viitattu 9.3.2022, https://flume.apache.org/releases/content/1.9.0/FlumeUserGuide.html#data-flow-model

Apache Nifi, An easy to use, powerful, and reliable system to process and distribute data, 2022. Viitattu 26.2.2022 https://nifi.apache.org/

Bson, BSON (Binary JSON) Serialization, 2022, https://bsonspec.org/

Cloudera DataFlow, 2022. Viitattu 26.2.2022 https://www.cloudera.com/products/cdf.html

Cloudera - Updated CDH Components, 2022. Viitattu 26.2.2022 https://docs.cloudera.com/cdp-private-cloud-upgrade/latest/release-guide/topics/cdpdc-rt-updated-cdh-components.html

Confluent, Google BigQuery Sink Connector Configuration Properties, 2022. Viitattu 1.3.2022. https://docs.confluent.io/kafka-connect-bigquery/current/kafka_connect_bigquery_config.html

Confluent, Confluent Connector Portfolio, 2022. Viitattu 1.3.2022. https://www.confluent.io/product/connectors/

Confluent, Incremental Cooperative Rebalancing in Apache Kafka: Why Stop the World When You Can Change It?, 2019. Viitattu 2.3.2022. https://www.confluent.io/blog/incremental-cooperative-rebalancing-in-kafka/

Confluent, Kafka Connect Limitations, 2022. Viitattu 1.3.2022. https://docs.confluent.io/cloud/current/connectors/limits.html#google-bigquery-sink-connector

Confluent Kafka Connect REST Interface, 2022. Viitattu 28.2.2022. https://docs.confluent.io/platform/current/connect/references/restapi.html

Confluent HTTP Basic authentication, 2022. Viitattu 25.2.2022. https://docs.confluent.io/platform/current/security/basic-auth.html#basic-auth-kconnect

Confluent, Single Message Transforms for Confluent Platform, 2022. Viitattu 1.3.2022. https://docs.confluent.io/platform/current/connect/transforms/overview.html

Docker Run the Docker daemon as a non-root user (Rootless mode), 2022. Viitattu 25.2.2022. https://docs.docker.com/engine/security/rootless/

Docker Registry, 2022. Viitattu 11.1.2022. https://docs.docker.com/registry/

Richards M.; Ford N., 2020. Fundamentals of Software Architecture

Gitlab Container Registry, 2022. Viitattu 11.1.2022. https://docs.gitlab.com/ee/user/packages/container_registry/

Google BigQuery, Quotas and limits, 2022, Viitattu 22.2.2022. https://cloud.google.com/bigquery/quotas#streaming_inserts

Google BigQuery, Error Messages, 2022. Viitattu 9.3.2022. https://cloud.google.com/bigquery/docs/error-messages#notFound

Google Cloud Build, 2022. Viitattu 11.1.2022. https://cloud.google.com/build

Google Dataflow, 2022. Viitattu 11.1.2022. https://cloud.google.com/dataflow

Google Dataflow, Using the monitoring UI, 2022. Viitattu 11.1.2022. https://cloud.google.com/dataflow/docs/guides/using-monitoring-intf

Google Secret Manager, 2022. Viitattu 11.1.2022. https://cloud.google.com/secret-manager

Jenkins, Build great things at any scale, 2022. Viitattu 11.1.2022. https://www.jenkins.io/

Lenses - Web tool for Kafka Connect, 2022. Viitattu 2.3.2022. https://github.com/lensesio/kafka-connect-ui

Microfocus, Products, 2022, Viitattu 9.3.2022. https://www.microfocus.com/en-us/products

MongoDB, BSON Types, 2022. Viitattu 3.3.2022. https://docs.mongodb.com/manual/reference/bson-types/

K6, Open source load testing tool and SaaS for engineering teams, 2022. Viitattu 3.3.2022. https://k6.io/

Kafka, Kafka Improvement Proposals, 2022. Viitattu 24.04.2022

https://cwiki.apache.org/confluence/display/kafka/kafka+improvement+proposals

Kafka, KIP-134: Delay initial consumer group rebalance, 2019. Viitattu 2.3.2022. https://cwiki.apache.org/confluence/display/KAFKA/KIP-134%3A+Delay+initial+consumer+group+rebalance

Kafka, KIP-415: Incremental Cooperative Rebalancing in Kafka Connect, 2019, Viitattu 2.3.2022. https://cwiki.apache.org/confluence/display/KAFKA/KIP-415%3A+Incremental+Cooperative+Rebalancing+in+Kafka+Connect

Kafka, KAFKA-13472, Connect can lose track of last committed offsets for topic partitions after partial consumer revocation, 2021, Viitattu 2.3.2022. https://issues.apache.org/jira/browse/KAFKA-13472

Kafka, KIP-429: Kafka Consumer Incremental Rebalance Protocol, 2021. Viitattu 2.3.2022. https://cwiki.apache.org/confluence/display/KAFKA/KIP-429%3A+Kafka+Consumer+Incremental+Rebalance+Protocol

Kai Waeher, Kafka API is the De Facto Standard API for Event Streaming like Amazon S3 for Object Storage, 2021. Viitattu 28.12.2021. https://www.kai-waehner.de/blog/2021/05/09/kafka-api-de-facto-standard-event-streaming-like-amazon-s3-object-storage/

Kowl - A Web UI for Apache Kafka, 2022. Viitattu 2.3.2022. https://cloudhut.dev/

Saariaho Sami, Palveluarkkitehtuurin suunnittelu ja toteutus startup-yritykselle 2019, viitattu 12.3.2022. https://urn.fi/URN:NBN:fi:amk-2019112021744

Snappy - A fast compressor/decompressor, 2022. Viitattu 11.1.2022. https://github.com/google/snappy

Vault Manage Secrets & Protect Sensitive Data, 2022. Viitattu 4.3.2022. https://www.vaultproject.io/

Vertica, Integrating with Apache Kafka, 2022. Viitattu 4.3.2022. https://www.vertica.com/docs/11.0.x/HTML/Content/Authoring/KafkaIntegrationGuide/KafkaIntegrationGuide.htm

Vertica, Complex types, 2022, Viitattu 3.3.2022 https://www.vertica.com/docs/11.0.x/HTML/Content/Authoring/SQLReferenceManual/DataTypes/ExternalTypes.htm

Whitesource, Software Composition Analysis, 2022. Viitattu 12.2.2022. https://www.whitesourcesoftware.com/

Xeotek KaDeck - The Apache Kafka Desktop Client & Web UI, 2022. Viitattu 2.3.2022. https://www.xeotek.com/apache-kafka-monitoring-management/