Opinnäytetyö (AMK)

Tieto- ja viestintätekniikka

2025

Otso Saarinen

Reaaliaikainen potilastiedon kerääminen ja analysointi Apache Kafka -alustan avulla

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Tieto- ja viestintätekniikka

2025 | 40 sivua

Otso Saarinen

Reaaliaikainen potilastiedon kerääminen ja analysointi Apache Kafka -alustan avulla

Apache Kafka on alusta, joka on tarkoitettu suurten tietomäärien ja tietovirtojen prosesointiin. Tässä opinnäytetyössä tarkasteltiin Apache Kafkan arkkitehtuuria, tapahtumavirta-alustojen toimintaperiaatteita sekä reaaliaikaisen analytiikan hyötyjä ja haasteita nyky-yhteiskunnassa. Näiden tietojen pohjalta luotiin prototyyppi, joka mahdollistaa reaaliaikaisen potilastiedon keräämisen ja analysoinnin Apache Kafka -alustan avulla.

Prototyyppi toteutettiin simuloimalla potilastietoa Python-ohjelmointikielellä, lähettämällä potilastieto Apache Kafkaan ja tallentamalla potilastieto InfluxDB-tietokantaan. Potilastietojen visualisointi toteutettiin Grafana-visualisointisovelluksella, joka mahdollisti reaaliaikaisen analyysin. Järjestelmän suorituskykyä ja skaalautuvuutta arvioitiin simuloimalla potilastietoa erilaisilla taajuuksilla.

Tulokset osoittivat, että Apache Kafka kykenee käsittelemään suuria tietomääriä tehokkaasti ja vastaavanlaista järjestelmää voitaisiin hyödyntää terveydenhuollossa. Jatkokehityksessä tulisi esimerkiksi huomioida GDPR:n mukaiset tietoturvavaatimukset, jotta prototyypin käyttöönotto terveydenhuollossa olisi mahdollista.

Avainsanat:
Apache Kafka, potilastieto, reaaliaikaisuus, analytiikka, visualisointi

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Information & Communications Technology

2025 | 40 pages

Otso Saarinen

Real-time patient data collection and analysis using the Apache Kafka platform

Apache Kafka is a platform designed for processing large volumes of data and data streams. This thesis examined the architecture of Apache Kafka, the principles of event streaming platforms, and the benefits and challenges of real-time analytics in modern society. Additionally, the aim was to develop a prototype that enables real-time collection and analysis of patient data using the Apache Kafka platform.

The prototype was implemented by simulating patient data with the Python programming language, sending the data to Apache Kafka, and storing it in an InfluxDB database. The visualization of patient data was carried out using the Grafana visualization tool, enabling real-time analysis. The system’s performance and scalability were evaluated by simulating patient data at different frequencies.

The results demonstrated that Apache Kafka can efficiently process large amounts of data, and a similar system could be utilized in healthcare. Future development should consider, for example, GDPR-compliant security requirements to enable the adoption of the prototype in healthcare applications.

Keywords:
Apache Kafka, patient data, real-time processing, analytics, visualization

Sisältö

Kuvat

Sanasto

API	Application programming interface. Rajapinta, joka mahdollistaa sovellusten keskinäisen kommunikoinnin. (Amazon Web Services 2025a).
Java	Ohjelmointikieli. (Oracle 2025b).
JDK	Java Development Kit. Sisältää eri työkaluja Java-ohjelmien rakentamiseen. (Oracle 2025a).
pip	Pythonin paketinhallintajärjestelmä. (PyPI 2025).
Python	Ohjelmointikieli. (The Python Software Foundation 2025).
Scala	Ohjelmointikieli. (Scala 2025)
SQL	Kyselykieli, jonka avulla relaatiotietokannoista voidaan etsiä haluttuja tietoja tai tehdä niihin muutoksia. (Amazon Web Services 2025d).
TCP	Transmission Control Protocol. Tietoliikenneprotokolla, joka määrittelee säännöt tehokkaalle ja turvalliselle tiedonsiirrolle. (NordVPN 2022).
WSL	Windows Subsystem for Linux. Mahdollistaa Linuxin käytön Windows-käyttöjärjestelmässä ilman virtuaalikonetta tai kaksoiskäynnistystä. (Microsoft 2023).
Ympäristömuuttuja	Ennalta määrättyjä avain-arvo pareja, joita tietokone käyttää hyödyksi prosesseissaan. (Linux.fi-wiki 2025).

1 Johdanto

Reaaliaikainen tiedon analysointi on noussut keskeiseksi työkaluksi nykyaikaisissa järjestelmissä, sillä se mahdollistaa nopean ja tarkan reagoinnin muuttuviin olosuhteisiin. (Splunk 2023). Tämän opinnäytetyön päätarkoituksena oli tutustua Apache Kafka -tapahtumavirta-alustaan sekä reaaliaikaiseen tiedon analysointiin analytiikka-järjestelmien avulla.

Apache Kafka on avoimen lähdekoodin tapahtumavirta-alusta, joka mahdollistaa suurten tietomäärien nopean käsittelyn ja prosessoinnin. Apache Kafkaa käytetään laajasti eri toimialoilla, joissa reaaliaikainen tiedon käsittely on tärkeää. Monet suuret yrityksen, kuten Microsoft, AirBnB ja Netflix, käyttävät Apache Kafkaa tarjotakseen asiakkailleen reaaliaikaisia kokemuksia. (IBM 2025.)

Tässä työssä kehitettiin prototyyppi reaaliaikaisen potilastiedon keräämiseksi ja visualisoimiseksi hyödyntäen Apache Kafka -tapahtumavirta-alustaa. Prototyypin tavoitteena oli demonstroida Apache Kafkan kykyä käsitellä tietoa reaaliajassa ja yhdistää se tehokkaasti muihin järjestelmiin, kuten InfluxDB:hen ja Grafanaan. Prototyypissä InfluxDB toimi tietokantana, johon Apache Kafkan keräämä tieto tallennettiin, ja Grafana puolestaan visualisoi tämän tiedon mahdollisti sen tehokkaan analysoinnin.

2 Apache Kafka ja tapahtumavirta-alustat

2.1 Apache Kafka

Apache Kafka on alun perin 2011 LinkedInin käyttöön suunniteltu avoimen lähdekoodin hajautettu tapahtumavirta-alusta (engl. distributed event streaming platform), jota käytetään reaaliaikaiseen tiedon käsittelyyn, palvelimien väliseen integrointiin sekä suurten tietomäärien varastointiin ja prosessointiin. (Confluent 2025b.) Tapahtumavirta-alustat toimivat keräämällä tietoa reaaliajassa erilaisista tapahtumien lähteistä, kuten potilasseurantalaitteista, sensoreista, tietokannoista sekä erilaisista järjestelmistä ja pilvipalveluista. Kerätyt tiedon tallennetaan myöhempää käyttöä varten ja niitä voidaan muokata tarvittaessa ennen eteenpäin lähettämistä. Tiedot kulkevat tapahtumavirta-alustan sisällä jatkuvan virtauksen mukana, jotta oikea tieto on saatavilla oikeassa paikassa, kun sitä tarvitaan. (Apache Kafka 2025.)

Apache Kafkan toiminta koostuu kolmesta pääpiirteestä, jotka ovat: julkaisija-tilaaja-mallinen (engl. publish-subscribe pattern) tietovirtojen hallinta, tiedon tallentaminen luotettavasti ja kestävästi halutuksi ajaksi sekä reaaliaikainen tietovirtojen prosessointi. Kafka yhdistää viestinnän, varastoinnin ja tietovirtojen prosessoinnin, mikä mahdollistaa sekä vanhan että uuden tiedon reaaliaikaisen analysoinnin. Julkaisija-tilaaja-mallin lisäksi Kafka tukee myös jonotusmallia saadakseen molempien mallien parhaat hyödyt käyttöönsä. Perinteisessä jonotusmallissa vain yksi käyttäjä voi lukea viestejä kerrallaan, kun taas julkaisija-tilaaja-malli mahdollistaa monen käyttäjän viestien samanaikaisen lukemisen. Julkaisija-tilaaja-mallin heikkous on kuitenkin se, että se ei sovellu viestien jakamiseen eri prosessien kesken. Tämä on ratkaistu Kafkassa käyttämällä ositettua lokia. Loki on järjestettu lista eri tietueita, jotka jaetaan pienempiin osiin, partitioihin (engl. partition). Näin samaan Kafka-aiheeseen (engl. topic) voi liittyä useita käyttäjiä ja lukea vain heille tarkoitettuja partitioita sen sijaan, että kaikkien tarvitsisi käsitellä koko aihetta. (Amazon Web Services 2025c.)

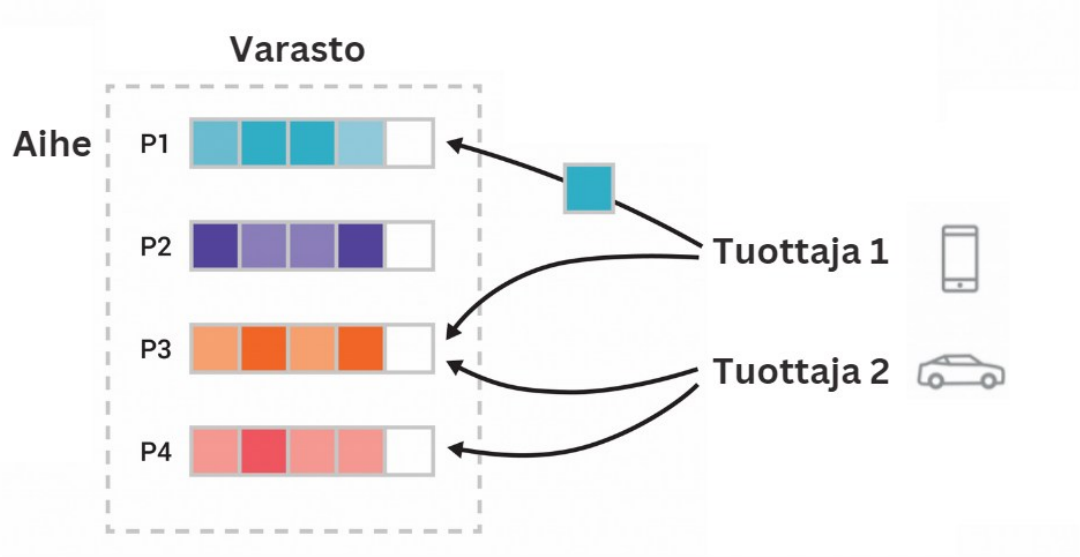
2.2 Kafka-arkkitehtuuri

Kafka toimii hajautettuna järjestelmänä ja koostuu palvelimista (engl. server) sekä asiakaskirjastoista (engl. client), jotka kommunikoivat keskenään käyttäen TCP-tietoliikenneprotokollaa. TCP-tietoliikenneprotokolla varmistaa, että laitteiden välinen tiedonsiirto tapahtuu verkossa oikein ja turvallisesti. Kafka toimii klusterina, mikä tarkoittaa sen koostumista yhdestä tai useammasta palvelimesta. (Apache Kafka 2025.) Osa Kafka-klusterin palvelimista toimii broker-palvelimina. Broker-palvelimien tehtävänä on vastaanottaa Kafkaan saapuvia viestejä ja välittää niitä edelleen. (GeeksforGeeks 2024.) Muut Kafka-klusterin palvelimet huolehtivat tiedon sisään- ja uloskuljettamisesta, jotta klusteri pysyy jatkuvasti yhteydessä muihin järjestelmiin, kuten tietokantoihin. Asiakaskirjastot mahdollistavat käyttäjän omien sovellusten ja palveluiden kehittämisen sekä integroinnin Kafka-järjestelmään. Kafka sisältää valmiiksi asiakaskirjastoja Java- ja Scala-kielille sekä näiden lisäksi Kafka-yhteisö tarjoaa asiakaskirjastoja esimerkiksi Python-kielelle. (Apache Kafka 2025.)

Asiakaskirjastoja on Kafkassa kahden tyyppisiä: tuottajat (engl. producer) sekä kuluttajat (engl. consumer). Tuottajat ovat ohjelmia, jotka lähettävät tietoa Kafkaan, ja kuluttajat ovat ohjelmia, jotka lukevat tietoa Kafkasta. Aiheet ovat loogisia kanavia, joihin tuottajat lähettävät tietoa ja joista kuluttajat lukevat tietoa. Aiheet jaetaan vielä pienempiin osiin joita kutsutaan partitioiksi. (BasuMallick 2022.) Yksittäisen partition sisällä olevat tapahtumat säilötään niiden saapumisjärjestyksessä ja jokaiselle tapahtumalle annetaan oma uniikki tunnus, jota kutsutaan offsetiksi. Jotta Kafka toimisi mahdollisimman vikasietoisesti, jokaisesta partitiosta luodaan kopio toisille broker-palvelimille. Tämän ansiosta, vaikka yksittäinen broker-palvelin lakkaisi toimimasta, tiedot ovat edelleen tallella ja saataville toisilla palvelimilla. (Verma 2025.) Jokainen aihe pystyy vastaanottamaan tietoa useilta tuottajilta samanaikaisesti sekä useat eri kuluttajat voivat lukea tietoa samasta aiheesta. Tuottajat ja kuluttajat toimivat täysin toisistaan riippumattomasti, mikä mahdollistaa korkea skaalautuvuuden, sillä niiden ei tarvitse odottaa toisiaan. Aiheiden sisältämät

tapahtumat säilytetään myös niiden lukemisen jälkeen, ja ne poistetaan vasta, kun käyttäjä niin haluaa. (Apache Kafka 2025.)

Kuvassa 1 visualisoidaan tapahtumien saapuminen Kafka-aiheeseen. Kuvassa tuottajat 1 & 2 lähettävät tapahtumia aiheeseen, ja nämä tapahtumat jaetaan aiheen sisällä eri partitioihin P1-P4. Tapahtumat ohjataan oikeaan partitioon niiden sisältämän avaimen mukaan. Eri avaimia kuvataan eri väreillä kuvassa, ja saman avaimen omaavat tapahtumat ohjataan aina samaan partitioon. Molemmat tuottajat voivat lähettää viestejä myös samaan partitioon, mikäli näin on määritelty.



Kuva 1. Tapahtumien saapuminen Kafka-aiheeseen.

2.3 API-arkkitehtuuri

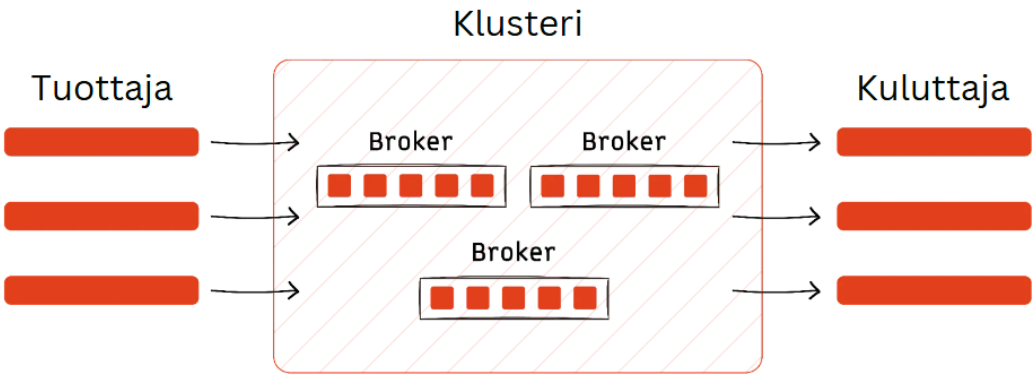
Ohjelmointirajapinnalla (API) tarkoitetaan mekaniikkaa, joka mahdollistaa sovellusten keskinäisen kommunikoinnin. Kafka-arkkitehtuuri koostuu viidestä keskeisestä API:sta, jotka mahdollistavat asiakaskirjastojen sekä klusterien hallinnan. Nämä ovat: Admin API, Producer API, Consumer API, Kafka Streams API ja Kafka Connect API. (Confluent 2025a).

Admin API mahdollistaa klusterien, aiheiden sekä broker-palvelimien hallinnan. Admin API:n avulla kehittäjät voivat luoda, poistaa ja muokata aiheita sekä broker-palvelimia. Producer API:n avulla voidaan lähettää viestejä Kafka-aiheisiin. Se tarjoaa myös monia konfigurointimahdollisuuksia, kuten tapahtumien pakkausasetusten hallinnan. Consumer API:a tarvitaan, kun halutaan lukea viestejä Kafka-aiheista. Sen avulla voidaan myös määrittää, mistä tapahtumasta alkaen viestejä luetaan ja montako tapahtumaa halutaan käsitellä kerralla. Streams API mahdollistaa tapahtumavirtaprosessoinnin Kafkaan tallennetulla datalla. Se lukee tietoa yhdestä tai useammasta aiheesta, käsittelee sen ja kirjoittaa tulokset yhteen tai useampaan uuteen aiheeseen, muuntaen näin sisääntulevan tietovirran ulosmeneväksi tietovirraksi. Connect API:n avulla Kafkaan voidaan tuoda ja viedä tietoa ulkoisista järjestelmistä, kuten tietokannoista. Kafka-yhteisö tarjoaa valmiita liittimiä (engl. connector) ilman, että käyttäjien tarvitsee rakentaa omia liittimiä alusta asti. (Confluent 2025a.)

2.4 Tapahtumavirta-alustat

Tapahtumavirta-alustojen keskiössä on itse tapahtuma, joka kuvaa mitä vain järjestelmässä tai ohjelmassa tapahtunutta asiaa. Tapahtuma voi olla esimerkiksi käyttäjän näppäimistön painallus tai ohjelman avautuminen. Kun tapahtumia syntyy nopeasti, saadaan jatkuva virta tapahtumia, eli tapahtumavirta. Tapahtuneet tapahtumat tallennetaan ja siirretään eri sijanteihin tallennettavaksi, prosessoitavaksi tai reaaliaikaisesti analysoitavaksi. Tapahtumavirtaus eroaa perinteisestä eräkäsittelystä (engl. batch processing) siten, että virran mukana saapuneet käsitellaan välittömästi sen sijaan, että tietoa käsiteltäisiin suurissa erissä kerralla. Jokainen tapahtuma sisältää myös avaimen, jolla kysyinen tapahtuma voidaan tunnistaa mahdollisesti jopa miljardien tapahtumien joukosta. Tämä avain voi olla esimerkiksi aikaleima, joka kertoo, milloin tapahtuma tapahtui, itse tapahtuman arvo tai muuta tapahtumaan liittyvää metadataa. (China & Goodwin 2024.)

Tapahtumalähtöisen arkkitehtuurin kolme pääkomponenttia ovat tapahtumien tuottajat, tapahtumien reitittäjät ja tapahtumien kuluttajat. Tuottaja lähettää tapahtuneen tapahtuman reitittimelle, joka käsittelee sen ja lähettää sen eteenpäin kuluttajille. Kuvassa 2 on visualisoitu tapahtumalähtöisen arkkitehtuurin peruspilarit. Se koostuu tuottajista, kuluttajista ja tapahtumien reitittäjästä. Reitittäjä voi olla esimerkiksi klusteri, joka koostuu pienemmistä broker-palvelimista. Näiden broker-palvelimien tehtävänä on suodattaa sinne saapuneita tapahtumia ja välittää ne oikeille kuluttajille. (Amazon Web Services 2025b.)



Kuva 2. Tapahtumalähtöinen arkkitehtuuri.

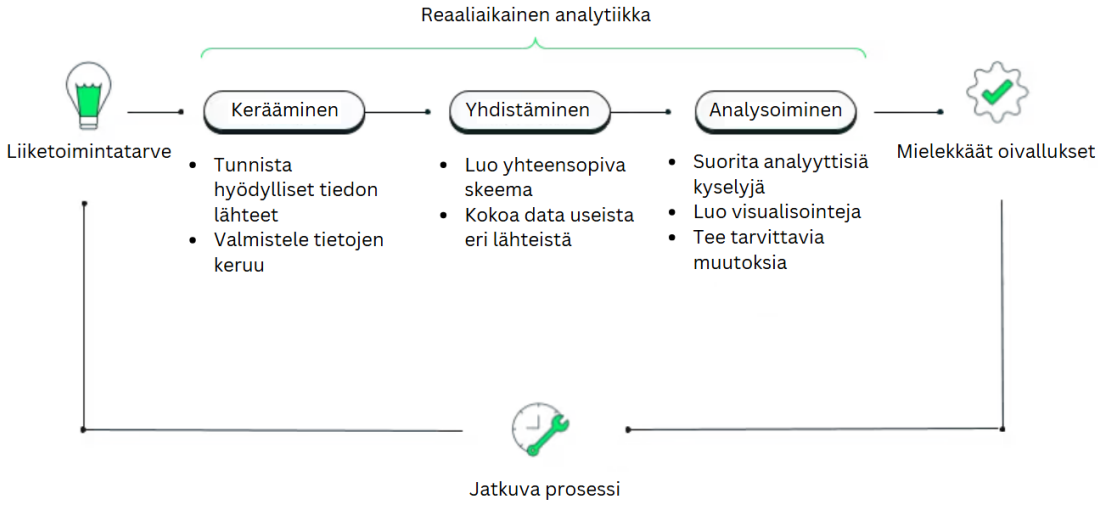
3 Reaaliaikainen tiedon analysointi analytiikkajärjestelmien avulla

3.1 Reaaliaikainen analytiikka

Reaaliaikaisuus tarkoittaa tietokoneen tai järjestelmän vastausajan nopeutta, jonka käyttäjä kokee viiveettömänä tai välittömänä. Reaaliaikaisella tiedon analysoinnilla viitataan tiedon analysointiin välittömästi sen saapuessa järjestelmään. Tämä liittyy usein läheisesti tapahtumavirta-arkkitehtuuriin, jossa tiedon käsittely tapahtuu jatkuvana virtana ja analyysi suoritetaan heti tiedon saavuttua. (Brush 2022.) Reaaliaikaisen tiedon analysoinnin etu perinteiseen eräanalyysiin (engl. batch analysis) on juuri se, että tieto voidaan analysoida heti sen saavuttua järjestelmään. Eräanalyysissä taas odotetaan, että kaikki analysoitava tieto on saapunut ennen käsittelyn aloittamista sekä suuren tietomäärän käsittely kerralla vie enemmän aikaa. Reaaliaikainen analytiikka mahdollistaa nopean reagoinnin sekä tarkempien päätösten tekemisen analyysistä saadun tiedon perusteella verrattuna eräanalyysiin. (Marr 2025.) Esimerkkejä reaaliaikaisen analytiikan käytöstä ovat esimerkiksi asiakkaan tarpeita vastaavan tiedon antaminen heidän reaaliaikaisesti analysoidun käytöksensä perusteella tai reaaliaikainen petollisten maksutapahtumien estäminen. (MongoDB 2025).

Reaaliaikainen analytiikka koostuu kahdesta keskeisestä osasta: reaaliaikaisuudesta ja analytiikasta. Reaaliaikaisuus viittaa tiedon välittömään käsittelyyn sen keräämisen jälkeen, kun taas analytiikka tarkoittaa algoritmien ja matemaattisten mallien käyttämistä mielekkäiden oivallusten ja ratkaisujen saamiseksi. Kuvassa 3 on vuokaavio reaaliaikaisen analytiikkaprosessin eri vaiheista, jotka ovat kerääminen, yhdistäminen ja analysointi. Ensimmäisessä vaiheessa etsitään ja tunnistetaan hyödylliset tiedonlähteet ja valmistellaan tarvittavat työkalut tiedon keräämistä varten. Toisessa vaiheessa yhdistetään useita eri tiedon lähteitä yhdeksi kokonaisuudeksi. Viimeisessä vaiheessa

suoritetaan kerätylle ja muokatulle tiedolle analyyttisiä kyselyjä, visualisoidaan käsiteltyä tietoa ja tehdään tarvittavia muutoksia. (MongoDB 2025.)



Kuva 3. Reaaliaikainen analytiikkaprosessi.

3.2 Reaaliaikaisen analytiikan hyödyt ja haasteet

Reaaliaikaisen analytiikan hyödyt liittyvät usein siihen, että päätöksiä voidaan tehdä mahdollisimman nopeasti ja ajankohtaiseen tietoon perustuen. Esimerkiksi yritysmaailmassa tai terveydenhuollossa päätökset on tehtävä viipymättä, jolloin reaaliaikainen analytiikka toimii hyödyllisenä työkaluna päätöksenteossa. Reaaliaikaista tietoa käytetään esimerkiksi ohjelmien ja ohjelmistojen automatisoinnissa, jolloin ne voivat tehdä päätöksiä ja suorittaa toimintoja reaaliaikaisten mittareiden perusteella. Reaaliaikaisen analytiikan on pitkään uskottu olevan kalliimpaa kuin eräanalyysin, mutta sen avulla yritykset voivat vähentää kulujaan pitkällä aikavälillä. Tietoa voidaan käyttää moniin eri tarkoituksiin, kuten yritys- ja tuotantoprosessien optimoimiseen sekä energiankulutuksen vähentämiseen. Reaaliaikainen analytiikka tuo myös etua yritysten välisessä kilpailussa, sillä se mahdollistaa erottuvien ominaisuuksien kehittämisen sekä tiheämmät palautesilmukat asiakastarpeiden perusteella. (Tinybird 2023.)

Reaaliaikaisen analytiikan käyttöön liittyy myös haasteita, kuten se, että sitä käyttävien ohjelmien on pystyttävä tarjoamaan korkea saatavuus sekä matalat

vastausajat. Saatavuudella tarkoitetaan järjestelmän tai ohjelman kykenee toimimaan ilman merkittäviä katkoksia tai häiriöitä. Lisäksi niiden tulee kyetä käsittelemään suuria tietomääriä, jopa teratavujen verran, ilman suorityskyvyn heikkenemistä. (Sisense 2025). Nämä suuret vaatimuksen reaaliaikaisen tiedon käsittelylle tekevät siihen perustuvien ohjelmien rakentamisesta haastavaa. Korkeat vaatimukset tiedon prosessointinopeuden suhteen vaikeuttavat oikeanlaisen arkkitehtuurin luomista. Monimutkaisen arkkitehtuurin lisäksi, reaaliaikaisen analytiikan käyttöönotto kohtaa myös haasteita yritysten työntekijöiden suunnalta. Monet voivat olla haluttomia ottamaan käyttöön uusia työkaluja, mikä voi hidastaa reaaliaikaisen analytiikan käyttöönottoa. Työntekijöille on tärkeää kommunikoida selkeästi, miksi reaaliaikaisen analytiikan työkaluja otetaan käyttöön, sekä tarjota tarpeellinen koulutus analytiikan käyttöön. (Brush 2022.)

4 Prototyypin suunnittelu ja toteutus

Prototyypillä tarkoitetaan aikaista versiota tai mallia tuotteesta, jonka pohjalta valmista tuotetta lähdetään kehittämään. (Kirvan 2023). Tässä opinnäytetyössä luotava prototyyppi muodostuu neljästä keskeisestä osasta: potilastiedon simuloinnista Pythonilla, Apache Kafkan käyttöönotosta ja konfiguroinnista, potilastiedon tallentamisesta InfluxDB-tietokantaan sekä potilastiedon visualisoinnista Grafanalla.

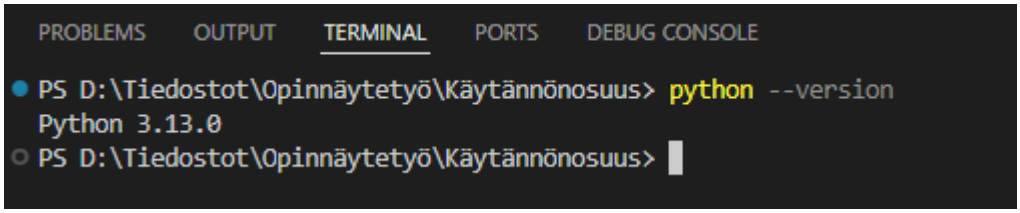
Prototyypin tavoitteena on luoda järjestelmä, jossa simuloitujen potilasseurantalaitteiden keräämä tieto visualisoidaan reaaliajassa Apache Kafkaa, InfluxDB:tä ja Grafanaa käyttäen. Prototyypissä Python-koodi toimii Kafka-arkkitehtuurin tuottajana, joka luo tietyin väliajoin potilastietoa ja lähettää sen Kafka-aiheelle. Toinen Python-koodi taas toimii kuluttajana, joka lukee tiedot Kafka-aiheesta ja siirtää ne InfluxDB-tietokantaan pitkäaikaista säilyttämistä varten. InfluxDB ja Grafana on integroitu keskenään, joten tiedot kulkevat saumattomasti tietokannasta Grafanalle. Grafanassa tiedot visualisoidaan helposti luettavaksi ja analysoitavaksi erilaisten hallintapaneeleiden (engl. dashboard) avulla.

Kuvassa 4 on esitetty vuokaavio suunnitellusta prototyypistä. Ensin Python-koodi luo simuloitua potilastietoa, joka lähetetään Apache Kafkalle. Tiedot luetaan tämän jälkeen Apache Kafkasta toisen Python-koodin avulla, joka lähettää tiedot myös eteenpäin InfluxDB:lle. InfluxDB on integroitu Grafanaan, joten tieto kulkee näiden välillä helposti Grafanan hallintapaneeleille visualisointia varten.

Kuva 4. Prototyypin suunnitelma.

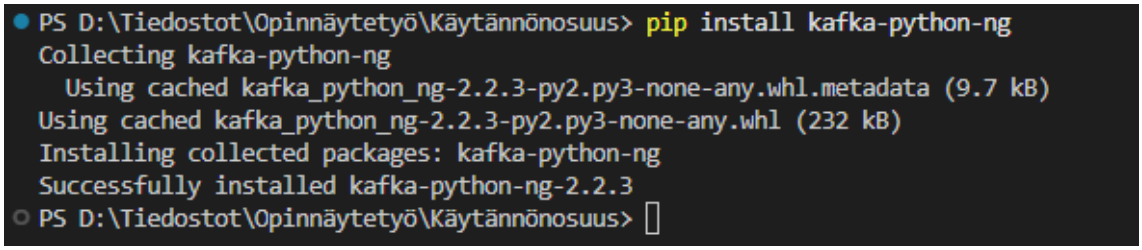
4.1 Potilastiedon simulointi Pythonilla

Ennen Python-koodin kirjoittamista tarkistetaan, että käytettävältä tietokoneelta löytyy Python-ohjelmointikieli asennettuna. Kuvassa 5 tarkistetaan asennetun Python-ohjelmointikielen versio käyttämällä ”python –version”-komentoa. Komento palauttaa versionumeron, jos ohjelmointikieli löytyy oikein asennettuna.



Kuva 5. Python-ohjelmointikielin version tarkistaminen.

Python-koodin kirjoittamista varten tarvitaan ”kafka-python-ng” -kirjasto, joka sisältää tarvittavat funktiot ja luokat Kafkan kanssa työskentelemiseen. Kuvassa 6 kirjasto asennetaan Pythonin paketinhallintajärjestelmä (pip) kautta ”pip install”-komennon avulla.



Kuva 6. ”kafka-python-ng”-kirjaston asentamisprosessi.

Kuvassa 7 on esitetty viimeistelty producer.py-tiedosto, joka lähettää potilastietoa Kafka-aiheeseen. Tiedoston alussa ladataan tarvittavat moduulit: time, json ja random, sekä KafkaProducer-luokka Kafkan kanssa työskentelemistä varten. Kafka-palvelin on oletuksena käynnissä osoitteessa localhost:9092, joten KafkaProducer-luokka yhdistetään samaan osoitteeseen.

Seuraavaksi on määritelty generoi_data-funktio, joka vastaa simuloidun potilastiedon tuottamisesta. Funktio palauttaa kahden eri potilaan simuloitua potilastietoa, johon sisältyy potilaan ja laitteen yksilöintitunnukset, potilaan syke, happisaturaatio sekä aikaleima. Potilastiedot luodaan random-moduulia käyttäen, jolloin tiedot vaihtuvat funktion eri kutsunta kerroilla.

Try-except-rakenne kutsuu 3 s:n välein KafkaProducer-luokan send-metodia, joka lähettää simuloidun tiedon Kafka-aiheen ”potilastiedot-events” alle. Ennen lähettämistä tiedot pakataan vielä JSON-muotoon, jotta Kafka voi vastaanottaa ne. Try-except-rakenne jatkaa toimintaansa loputtomasti, mutta jos käyttäjä päättää itse lopettaa koodin suorittamisen niin flush- ja close-metodit varmistavat, että kaikki tapahtumat on lähetetty onnistuneesti ennen koodin sulkemista.

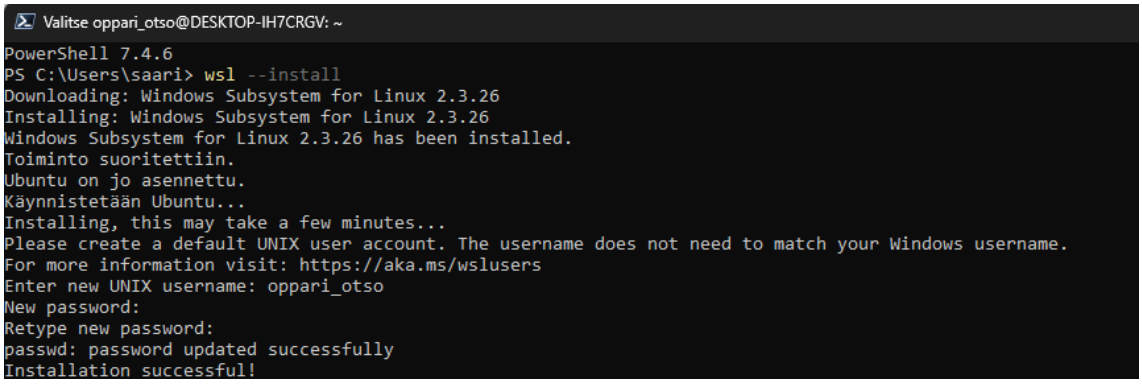
producer.py > ...

```python
import time
import json
import random
from kafka import KafkaProducer

producer = KafkaProducer(bootstrap_servers="localhost:9092")

if producer.bootstrap_connected() == True:
    print("Yhdistetty Kafkaan onnistuneesti")

def generoi_data():
    return {
        "potilas_1": {
            "potilas_id": 1,
            "laite_id": 1,
            "syke": random.randint(50, 100),
            "kehon_lampotila": round(random.uniform(36.0, 37.5), 2),
            "happisaturaatio": round(random.uniform(90.0, 100.0), 2),
            "aikaleima": time.time(),
        },
        "potilas_2": {
            "potilas_id": 2,
            "laite_id": 2,
            "syke": random.randint(65, 120),
            "kehon_lampotila": round(random.uniform(37.2, 38.5), 2),
            "happisaturaatio": round(random.uniform(75.0, 89), 2),
            "aikaleima": time.time(),
        },
    }

try:
    while True:
        data = generoi_data()
        for potilas, info in data.items():
            producer.send(
                topic="potilastiedot-events",
                value=json.dumps(info).encode("utf-8"),
            )
            print(f"Lähetetään {potilas}: {info}")
        time.sleep(5)

except KeyboardInterrupt:
    print("Keskeytetty käyttäjän toimesta")

finally:
    producer.flush()
    producer.close()
    print("Kafka-tuottaja suljettu")
```

Kuva 7.producer.py-tiedosto luo simuloitua potilastietoa.

4.2 Apache Kafkan käyttöönotto ja konfigurointi

Apache Kafka tarvitsee oikein toimiakseen Linux-ympäristön, joten asennetaan Windows Subsystem for Linux 2 (WSL2) PowerShellin kautta. WSL2 mahdollistaa Linux-ympäristön käyttämisen Windows-laitteella, jolloin Kafkan asentaminen onnistuu ilman erillistä virtuaalikonetta ja samalla voidaan helposti työskennellä rinnakkain Windows-käyttöjärjestelmän työkalujen kanssa. WSL2 on toimiva ratkaisu, sillä sen asentaminen onnistuu yhden komennon avulla, ja Windows-integraatio mahdollistaa helpon Grafanan käyttämisen myöhemmässä vaiheessa. PowerShell asentaa myös oletuksena Ubuntu-jakelun Linuxista. Ubuntu-jakelu on Linux-käyttöjärjestelmä, joka sisältää valmiiksi monia hyödyllisiä paketteja kuvankäsittelyyn, viestintään ja pakettienhallintaan. (Ubuntu Suomi 2025). Asennetaan kuvassa 8 WSL2-ympäristö ”wsl –install”-komentoa käyttämällä sekä luodaan uusi käyttäjä Ubuntu-jakelun sisälle.



Kuva 8. WSL2-ympäristön asentaminen sekä uuden käyttäjän luominen.

Kuvassa 9 asennetaan Java Development Kit (JDK), jonka Apache Kafka tarvitsee toimiakseen. Java Development Kit sisältää työkaluja Java-ohjelmien kehittämiseen, testaamiseen ja monitorointiin. Tarkistetaan asentamisen jälkeen myös asennetun JDK:n versionumero.

```
oppari_otso@DESKTOP-IH7CRGV:/mnt/c/Users/saari$ sudo apt install openjdk-11-jdk
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openjdk-11-jdk is already the newest version (11.0.25+9-1ubuntu1~24.04).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
oppari_otso@DESKTOP-IH7CRGV:/mnt/c/Users/saari$ java -version
openjdk version "11.0.25" 2024-10-15
OpenJDK Runtime Environment (build 11.0.25+9-post-Ubuntu-1ubuntu124.04)
OpenJDK 64-Bit Server VM (build 11.0.25+9-post-Ubuntu-1ubuntu124.04, mixed mode, sharing)
```

Kuva 9. Java Development Kitin asentaminen.

JDK:n asentamisen jälkeen luodaan ”JAVA_HOME”-ympäristömuuttuja avaamalla Ubuntun konfiguraatiotiedosto (Kuva 10), joka sijaitsee kansiossa ”/etc/environment”. Ympäristömuuttujat ovat ennalta määritettyjä avain-arvopareja, joita tietokoneen prosessit tarvitsevat toimiakseen oikein. Kafka on Java-pohjainen ohjelma, joten se edellyttää ”JAVA_HOME”-ympäristömuuttujan määrittämistä toimiakseen oikein.

```
oppari_otso@DESKTOP-IH7CRGV: /mnt/c/Users/saari
  GNU nano 7.2                                    /etc/environment *
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin"
JAVA_HOME="/usr/lib/jvm/java-11-openjdk-amd64"
```

Kuva 10. ”JAVA_HOME”-ympäristömuuttujan luominen.

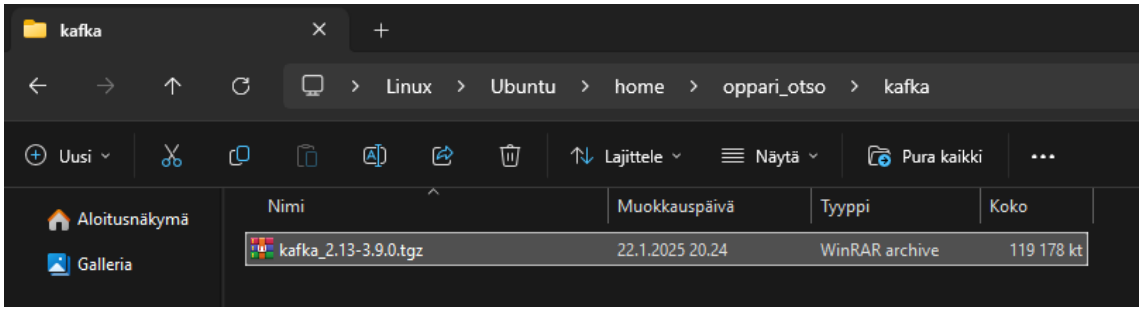
Kuvassa 11 ”JAVA_HOME”-ympäristömuuttuja aktivoidaan joko kirjautumalla Ubuntuun uudelleen tai suorittamalla ”source /etc/environment”-komento. ”echo $JAVA_HOME”-komento palauttaa JDK:n asennuspolun.

```
oppari_otso@DESKTOP-IH7CRGV:/mnt/c/Users/saari$ sudo nano /etc/environment
oppari_otso@DESKTOP-IH7CRGV:/mnt/c/Users/saari$ source /etc/environment
oppari_otso@DESKTOP-IH7CRGV:/mnt/c/Users/saari$ echo $JAVA_HOME
/usr/lib/jvm/java-11-openjdk-amd64
oppari_otso@DESKTOP-IH7CRGV:/mnt/c/Users/saari$
```

Kuva 11. ”JAVA_HOME”-ympäristömuuttujan aktivoiminen.

WSL ja Java on asennettu onnistuneesti, joten siirrytään seuraavaksi itse Apache Kafkan asentamiseen. Vaihdetaan tässä vaiheessa myös käyttämään

WSL:ää komentokehotteen kautta. Kafkan käyttämisen helpottamiseksi, tarvitaan mahdollisuus avata useita välilehtiä, mikä ei ole mahdollista PowerShellissä. Ladataan Kafkan asennuspaketti (kafka_2.13-3.9.0.tgz) Apache Kafkan omilta verkkosivuilta ja siirretään se Windows-käyttöjärjestelmän ladatuista tiedostoista Ubuntun omaan kansioon (Kuva 12).



Kuva 12. Kafkan asennuspaketti Ubuntu-jakelun sisällä.

Kuvassa 13 Kafkan asennuspaketti puretaan käytettäväksi. Asennustiedostot on pakattu paketin koon pienentämiseksi eikä niihin ei päästä käsiksi ennen paketin purkamista. Asennuspaketti sisältää kaikki tarvittavat konfiguraatio- ja kooditiedostot Kafkan asentamista varten.

```
oppari_otso@DESKTOP-IH7CRGV:~/kafka$ tar -xzf kafka_2.13-3.9.0.tgz
oppari_otso@DESKTOP-IH7CRGV:~/kafka$
```

Kuva 13. Kafkan asennuspaketin purkaminen.

Asennuspaketin purkamisen jälkeen siirrytään puretun kansion sisään ja luodaan Kafka-klusterille uniikki tunniste ”KAFKA_CLUSTER_ID”. Alustetaan Kafka myös toimimaan KRaft-tilassa, joka mahdollistaa Kafkan käyttämisen ilman Zookeeperia (Kuva 14). KRaft on vaihtoehtoinen metadatavarasto Kafkan käyttämiseen. Kafkan käyttäminen KRaftilla Zookeeperin sijaan yksinkertaistaa Kafka-arkkitehtuuri, sillä se poistaa tarpeen käyttää erillistä metadatan hallintajärjestelmää.

```
oppari_otso@DESKTOP-IH7CRGV:~/kafka/kafka_2.13-3.9.0$ KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
oppari_otso@DESKTOP-IH7CRGV:~/kafka/kafka_2.13-3.9.0$ bin/kafka-storage.sh format --standalone -t $KAFKA_CLUSTER_ID -c c
onfig/kraft/reconfig-server.properties
Formatting dynamic metadata voter directory /tmp/kraft-combined-logs with metadata.version 3.9-IV0.
oppari_otso@DESKTOP-IH7CRGV:~/kafka/kafka_2.13-3.9.0$
```

Kuva 14. Kafkan konfigurointi KRaft-tilaan.

Kaikki alustavat toimenpiteet on tehty ja Kafka on valmis käynnistettäväksi. Kuvassa 15 käynnistetään Kafka-palvelin ”bin/kafka-server-start.sh config/kraft/reconfig-server.properties” -komennon avulla.

```
oppari_otso@DESKTOP-IH7CRGV:~/kafka/kafka_2.13-3.9.0$ bin/kafka-server-start.sh config/kraft/reconfig-server.properties
```

Kuva 15. Kafka-palvelimen käynnistäminen.

Kafka-palvelin on käynnissä ja seuraavaksi tulee luoda aihe, johon Python-koodin luomat tapahtumat lähetetään. Kuvassa 16 luodaan aihe nimeltä ”potilastieto-events” uuteen komentokehotteen välilehteen.

```
oppari_otso@DESKTOP-IH7CRGV:~/kafka/kafka_2.13-3.9.0$ bin/kafka-topics.sh --create --topic potilastieto-events --bootstr
ap-server localhost:9092
Created topic potilastieto-events.
```

Kuva 16. potilastiedot-events aiheen luominen.

Kuvassa 17 testataan vielä, että aihe on luotu onnistuneesti lähettämälle sinne muutama testitapahtuma.

```
oppari_otso@DESKTOP-IH7CRGV:~/kafka/kafka_2.13-3.9.0$ bin/kafka-console-producer.sh --topic potilastiedot-events --boots
trap-server localhost:9092
>Ensimmäinen tapahtuma!
>Toinen tapahtuma on tässä.
>^Coppari_otso@DESKTOP-IH7CRGV:~/kafka/kafka_2.13-3.9.0$
```

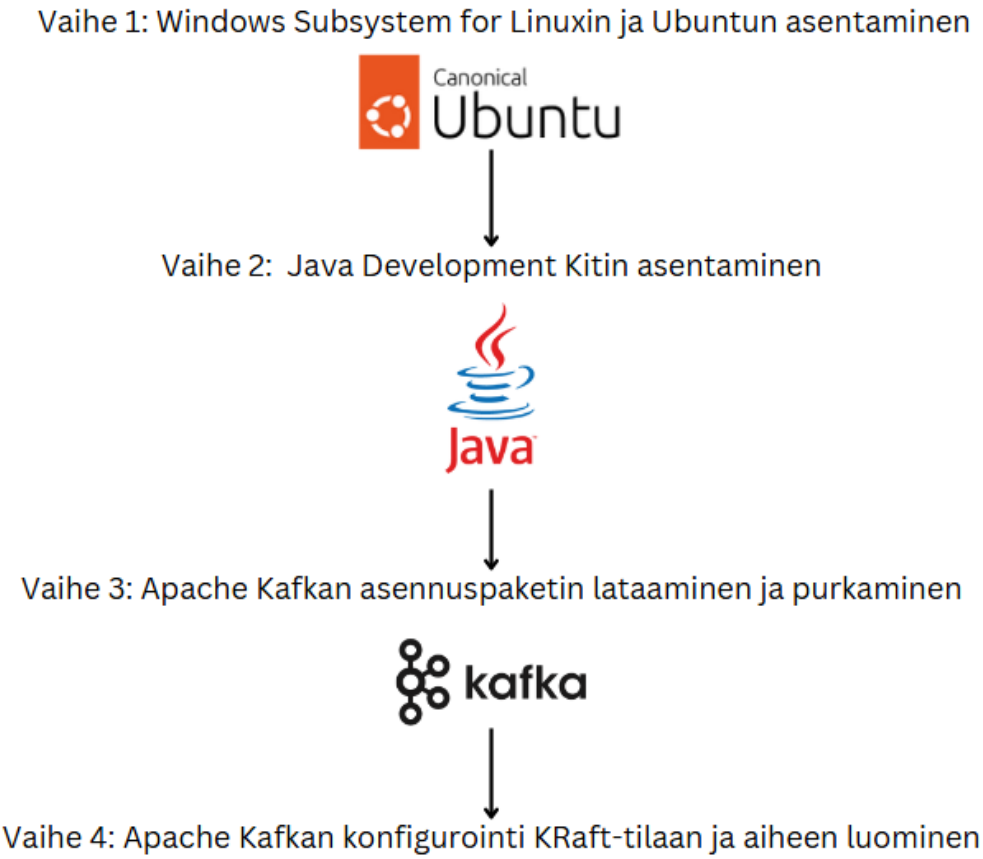
Kuva 17. Kafka-aihe on luotu onnistuneesti, sillä se vastaanottaa sinne lähetettyjä tapahtumia.

Avataan vielä kolmas komentokehotteen välilehti ja luetaan sen kautta Kafka-aiheelle lähetetyt testitapahtumat. Kuvassa 18 nähdään, että tapahtumat ilmestyvät komentokehotteeseen reaaliaikaisesti.

```
oppari_otso@DESKTOP-IH7CRGV:~/kafka/kafka_2.13-3.9.0$ bin/kafka-console-consumer.sh --topic potilastiedot-events --from-
beginning --bootstrap-server localhost:9092
Ensimmäinen tapahtuma!
Toinen tapahtuma on tässä.
```

Kuva 18. Lähetetyt testitapahtumat luetaan onnistuneesti.

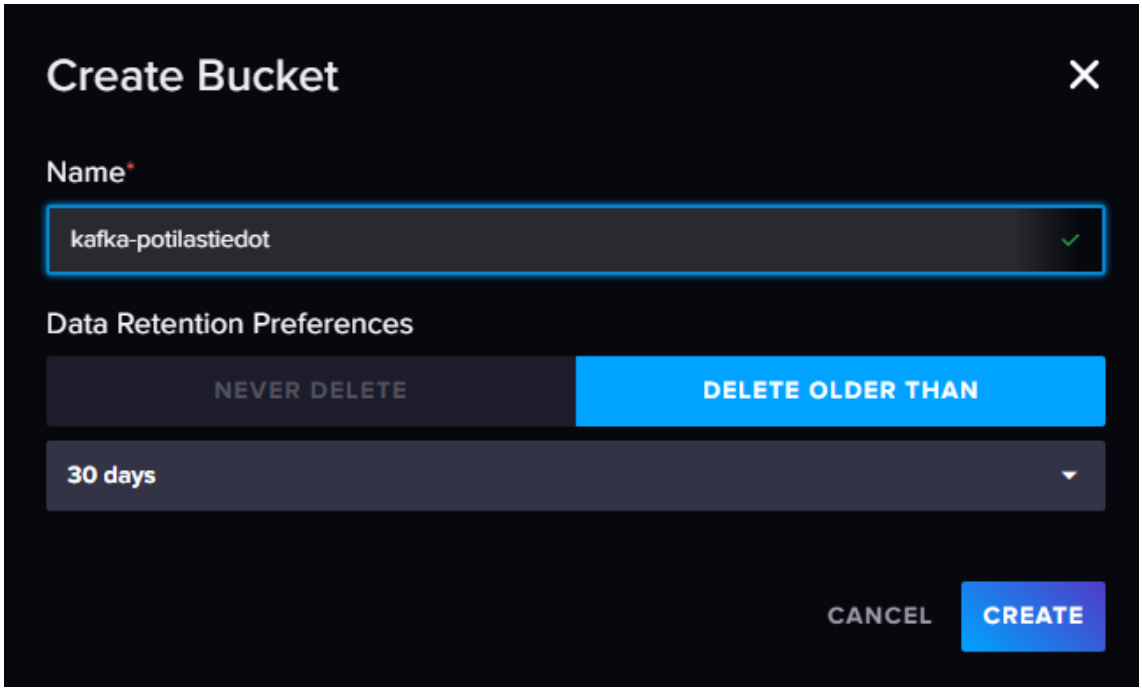
Kafka-arkkitehtuuri on asennettu ja konfiguroitu onnistuneesti, joten se on valmis vastaanottamaan Python-koodin luomaa simuloitua potilastietoa. Kuvassa 19 on visualisoitu Apache Kafkan asennusprosessi, jossa on esitelty tiivisti kaikki sen eri vaiheet.



Kuva 19. Apache Kafkan asennus- ja konfigurointiprosessi.

4.3 Potilastiedon visualisointi Grafanalla

Työn seuraavassa vaiheessa luodaan toinen Python-koodi (consumer.py), jonka tehtävänä on lukea saapuneet tapahtumat Kafka-aiheesta ja siirtää ne InfluxDB-tietokantaan pitkäaikaista tallentamista varten. Jotta potilastietoja voidaan tallentaa InfluxDB-tietokantaan, tulee ensin luoda ”bucket” InfluxDB:n verkkokäyttöliittymän kautta, kuten kuvassa 20. Bucket-termillä tarkoitetaan tietokantakokoelmaa, johon tiedot InfluxDB:tä käyttäessä tallennetaan.



Kuva 20. InfluxDB ”bucketin” luominen.

Valmis consumer.py-tiedosto on esitelty kuvassa 21. Python-koodi seuraa Kafka-aihetta ”potilastiedot-events” hyödyntäen KafkaConsumer-luokkaa, joka saadaan käyttöön ”kafka-python-ng”-kirjaston kautta. Aiheeseen saapuneet tapahtumat luetaan ja tallennetaan InfluxDB-tietokantaan hyödyntämällä InfluxDBClient3-luokkaa.

Kun consumer.py vastaanottaa tapahtuman Kafka-aiheesta, tiedot deserialisoidaan ja muunnetaan Point-objekteiksi. Serialisoinnilla tarkoitetaan tiedon muuttamista muotoon, jossa sitä on helpompi ja kevyempi siirtää järjestelmien välillä. Esimerkkejä serialisoiduista tiedostomuodostoista ovat

esimerkiksi JSON- sekä binäärimuoto. Deserialisointi taas tarkoittaa tiedon muuttamista alkuperäiseen muotoon serialisoidusta muodosta. Point-objekti sisältää sille määritetyt tiedot, kuten ”potilas_id”, ”laite_id”, ”syke”, ”kehon_lampotila”, ”happisaturaatio” ja ”aikaleima”. Tämän lisäksi InfluxDB liittää tallennushetken aikaleiman osaksi Point-objektia.

Tiedostosta löytyy myös token-muuttuja, joka sisältää avaimen InfluxDB-tietokantaan yhteyden muodostamiseksi. Tuotantoympäristössä avain tulisi piilottaa erillisen konfiguraatiotiedoston sisään tietoturvan varmistamiseksi. Koska kyseessä on kuitenkin vain prototyyppi, tietoturvaa ei ole tässä vaiheessa huomioitu.

consumer.py > ...

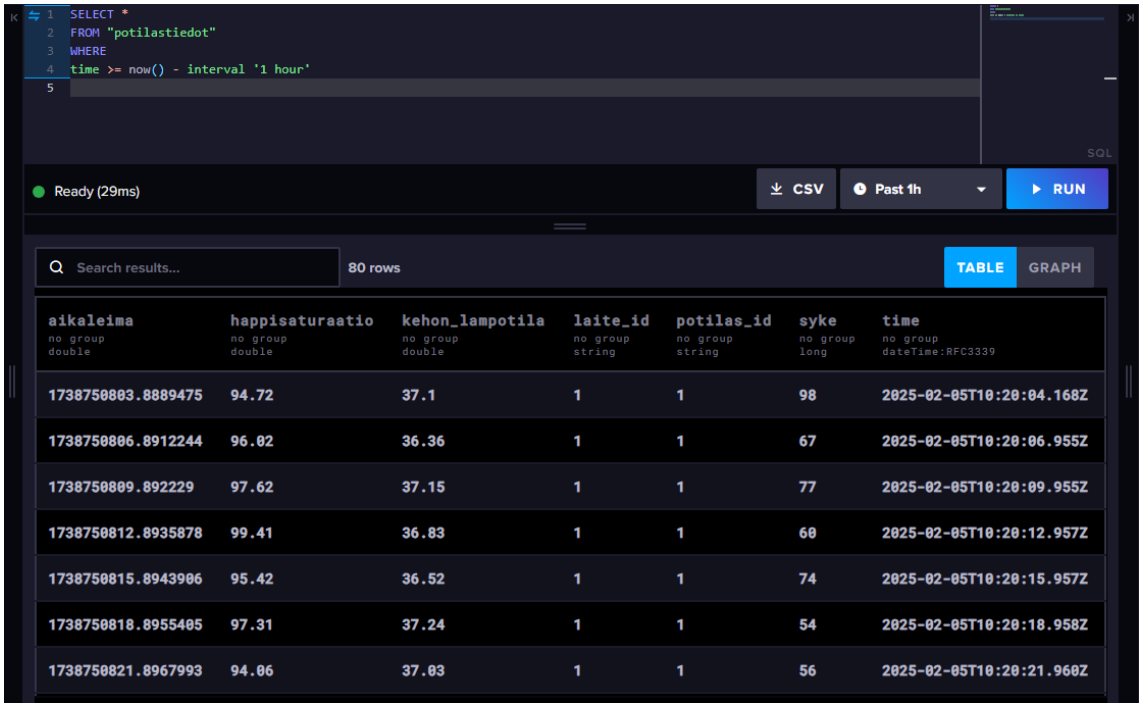
```python
from kafka import KafkaConsumer
from influxdb_client_3 import InfluxDBClient3, Point
import json


def deserialize_message(message):
    return json.loads(message.decode("utf-8"))


token = "zxS8u7r59b5_Xt8hANEwtya0Q-Vv2QallPaHctXh-cH7Kb7V30SsVyX36xJut2iS-oMTcA_9m--91qCIOtJZQQ=="
org = "Opinnäytetyö"
host = "https://eu-central-1-1.aws.cloud2.influxdata.com"
client = InfluxDBClient3(host=host, token=token, org=org)
database = "kafka-potilastiedot"

consumer = KafkaConsumer(
    "potilastiedot-events",
    bootstrap_servers="localhost:9092",
    value_deserializer=deserialize_message,
)

print("Kuunnellaan Kafka-topicia...")

try:
    for message in consumer:
        data = message.value
        print("Vastaanotettu data:", data)

        point = (
            Point("potilastiedot")
            .tag("potilas_id", data["potilas_id"])
            .tag("laite_id", data["laite_id"])
            .field("syke", data["syke"])
            .field("kehon_lampotila", data["kehon_lampotila"])
            .field("happisaturaatio", data["happisaturaatio"])
            .field("aikaleima", data["aikaleima"])
        )
        client.write(database=database, record=point)
        print("Tiedot kirjoitettu InfluxDB tietokantaan.")

except KeyboardInterrupt:
    print("Kafka-lukija keskeytetty")

finally:
    consumer.close()
    print("Kafka-kuluttaja suljettu")

```

Kuva 21. consumer.py-tiedosto lukee viestejä Kafkasta.

Seuraavaksi käynnistetään molemmat producer.py sekä consumer.py. Varmistetaan myös, että Kafka-palvelin on edelleen käynnissä komentokehotteessa. Producer.py -tiedosto lähettää nyt viestejä 5 s:n välein Kafka-aiheeseen. Consumer.py kuuntelee Kafka-aihetta, lukee kaikki sinne saapuneet tapahtumat ja lähettää ne InfluxDB-tietokantaan.

InfluxDB-tietokantaan saapuneita viestejä voidaan tarkastella InfluxDB:een verkkokäyttäliittymän kautta, kuten kuvassa 22. Kaikki tietokantaan saapuneet

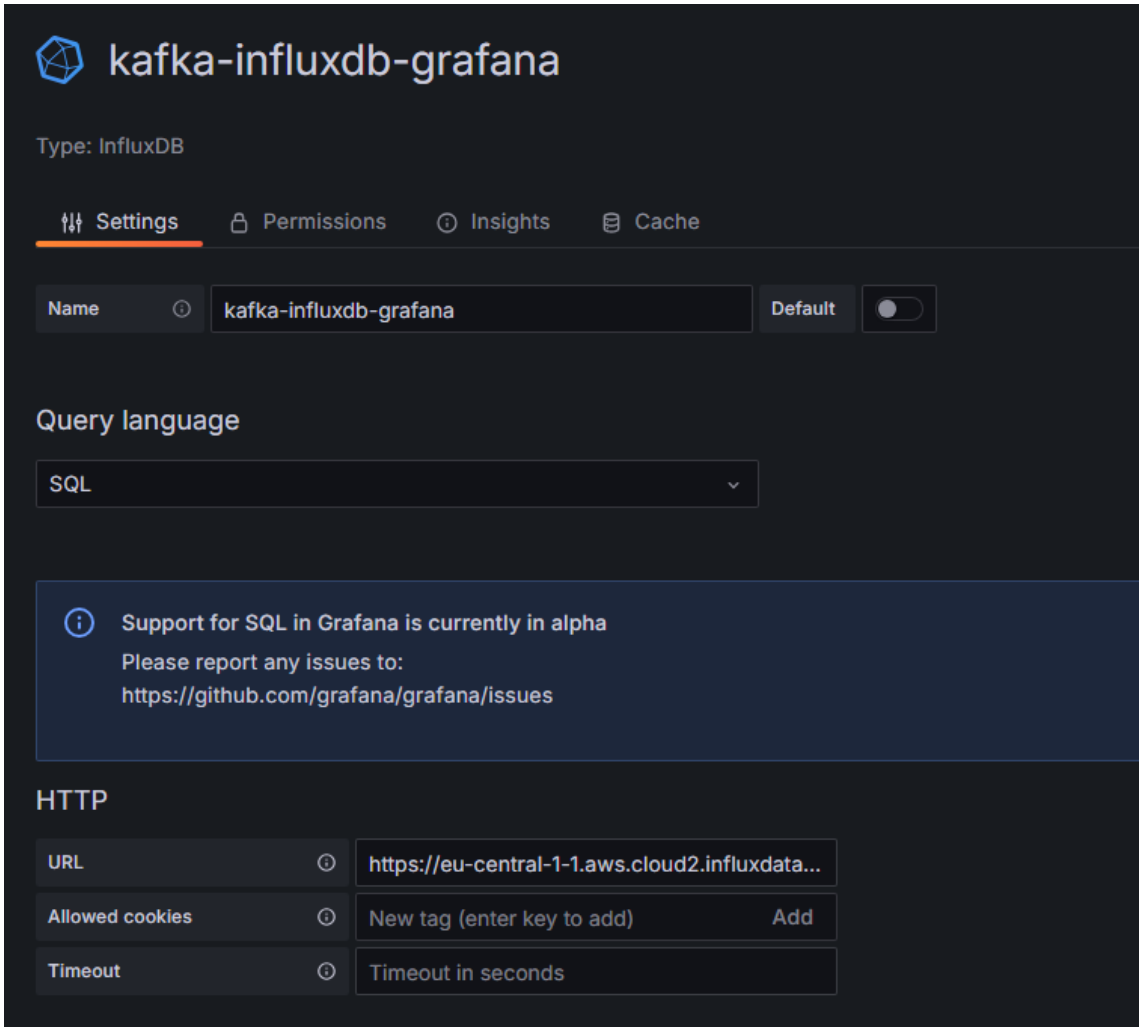
tapahtumat sisältävät aikaleiman, happisaturaation, kehon lämpötilan, laite id:n, potilas id:n, sykkeen sekä Point-objektin oman aikaleiman, kuten consumer.py:ssä määriteltiin.



Kuva 22. InfluxDB-tietokantaan saapuneet tapahtumat voidaan lukea SQL-kielen avulla.

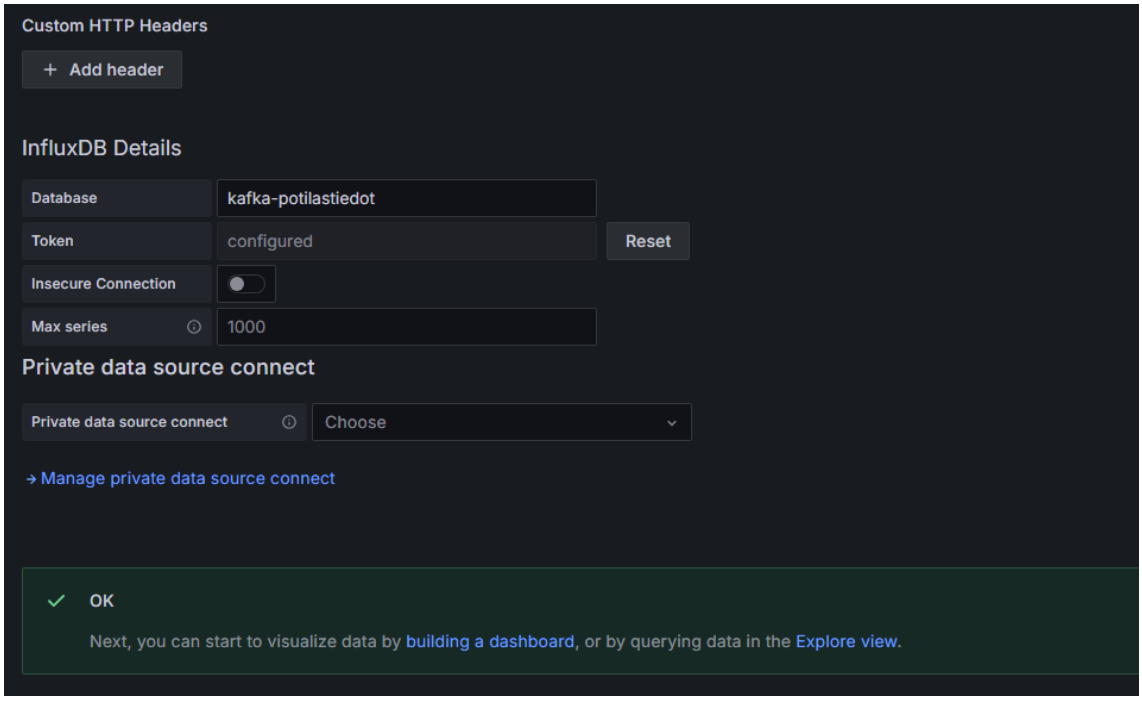
Yhteys Pythonin, Kafkan ja InfluxDB:een välillä toimii nyt oikein, joten siirrytään prototyypin viimeiseen vaiheeseen, jossa InfluxDB-tietokanta yhdistetään Grafanaan.

Grafana tukee valmiiksi InfluxDB:tä tiedonlähteenä, joten yhteyden luominen näiden välillä on vaivatonta. Kuvassa 23 tiedonlähteelle annetaan haluttu nimi, kuten ”kafka-influxdb-grafana”, valitaan haluttu kyselykieli sekä asetetaan InfluxDB-tietokannan url-osoite.



Kuva 23. InfluxDB:een lisääminen tiedonlähteeksi Grafanassa.

Testataan vielä, että yhteys on muodostettu onnistuneesti painamalla ”Save & test” -painiketta, joka on konfigurointisivun alareunassa (Kuva 24).



Kuva 24. Yhteys InfluxDB:een ja Grafanan välillä on muodostettu onnistuneesti.

Grafanassa Dashboards-välilehdeltä voidaan luoda erilaisia hallintapaneeleja sen mukaan, millä tavalla tietoa halutaan visualisoida. Kuvassa 25 näkyy yhteenveto molemmille potilaille luoduista Grafana-hallintapaneeleista. Potilaiden sykkeiden seuraamista varten on valittu ”time series” -paneeli, jonka avulla potilaiden sykkeiden muutoksia valitulla aikavälillä on helppo seurata. Happisaturaatiota seurataan gauge-mittarilla, joka näyttää potilaiden sen hetkisen happisaturaation sekä vaihtaa väriä punaisen, keltaisen ja vihreän välillä potilaan happisaturaation mukaan. Viimeisenä näkyvät potilaiden kehonlämpötilat, joita visualisoidaan ”stat”-paneelin avulla.

Kuva 25. Grafana-hallintapaneeli, joka seuraa potilaiden sykettä, happisaturaatiota sekä kehonlämpötilaa.

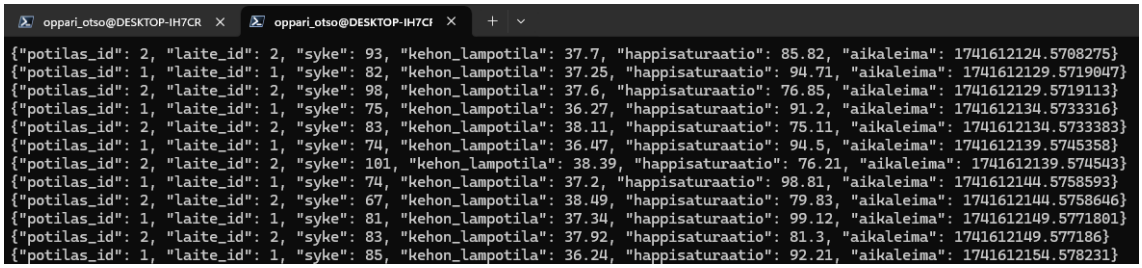
Prototyyppi on nyt valmis ja toimintakuntoinen vastaanottamaan sekä visualisoimaan tietoa reaaliajassa. Grafana-hallintapaneelien tihein päivitysnopeus on kerran 5 s:ssa, joten pientä viivettä voi ilmeentyä ennen kuin tieto on visualisoitu Grafanassa.

5 Tulokset ja arviointi

Prototyypin tuottama potilastieto on luotu Python-koodin avulla, joka lähettää potilastiedon Apache Kafkalle käsiteltäväksi. Toinen Python-koodi lukee potilastiedon Apache Kafkasta ja lähettää sen InfluxDB-tietokannalle. InfluxDB:stä potilastieto kulkee Grafanalle, jossa se visualisoidaan erilaisten hallintapaneelien avulla.

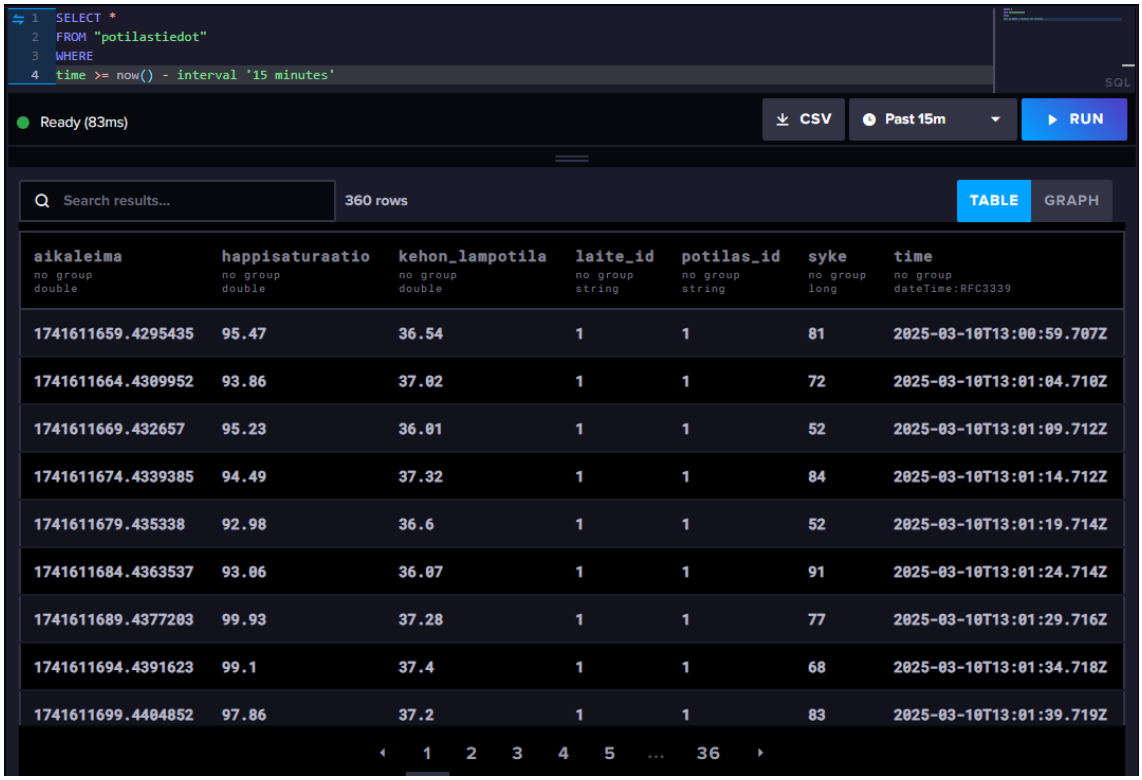
5.1 Prototyypin toiminta

Prototyyppi luo onnistuneesti simuloitua potilastietoa, joka sisältää seuraavat kentät: potilas_id, laite_id, syke, kehon_lampotila, happisaturaatio ja aikaleima. Kuvassa 26 on esitetty Python-koodin luomaa potilastietoa Apache Kafkan sen vastaanotettua.



Kuva 26. Apache Kafka vastaanottaa Python-koodin luoman potilastiedon.

Seuraavaksi toinen Python-koodi lukee Apache Kafkaan saapuneen potilastiedon ja lähettää sen InfluxDB-tietokantaan pitkäaikaista säilyttämistä varten. Kuvassa 27 nähdään, että potilastieto saapuu onnistuneesti InfluxDB-tietokantaan



```
SELECT *
FROM "potilastiedot"
WHERE
time >= now() - interval '15 minutes'
```

Ready (83ms) CSV Past 15m RUN

Search results... 360 rows TABLE GRAPH

aikaleima no group double	happisaturaatio no group double	kehon_lampotila no group double	laite_id no group string	potilas_id no group string	syke no group long	time no group dateTime:RFC3339
1741611659.4295435	95.47	36.54	1	1	81	2025-03-10T13:00:59.707Z
1741611664.4389952	93.86	37.02	1	1	72	2025-03-10T13:01:04.710Z
1741611669.432657	95.23	36.01	1	1	52	2025-03-10T13:01:09.712Z
1741611674.4339385	94.49	37.32	1	1	84	2025-03-10T13:01:14.712Z
1741611679.435338	92.98	36.6	1	1	52	2025-03-10T13:01:19.714Z
1741611684.4363537	93.06	36.07	1	1	91	2025-03-10T13:01:24.714Z
1741611689.4377203	99.93	37.28	1	1	77	2025-03-10T13:01:29.716Z
1741611694.4391623	99.1	37.4	1	1	68	2025-03-10T13:01:34.718Z
1741611699.4404852	97.86	37.2	1	1	83	2025-03-10T13:01:39.719Z

‹ 1 2 3 4 5 ... 36 ›

Kuva 27. Potilastieto tallennetaan InfluxDB-tietokantaan.

InfluxDB siirtää vielä potilastiedon Grafanalle, jolloin Grafana-hallintapaneelit pystyvät visualisoimaan saamansa tiedon. Kuvassa 28 on esitetty potilaan 1 syke viimeisen 5 minuutin aikana.

Kuva 28. Grafana-hallintapaneeli, joka esittää potilaan 1 sykkeen muutokset.

Potilastieto kulkee onnistuneesti kaikkien järjestelmän osien kautta, joten prototyyppi toimii onnistuneesti ja halutulla tavalla. Potilastieto kulkee lähes viiveettä ensimmäiseltä Python-koodilta Grafanalle, jolloin suurin viive ja haaste reaaliaikaisuudelle on Grafana-hallintapaneelien päivitysnopeus.

5.2 Suorituskyky

Prototyypin suorituskyky on korkea, ja se toimii lähes moitettomasti myös suuremmilla potilastiedon lähetystaajuuksilla. Testauksen yhteydessä alkuperäinen 5 s:n välein tapahtuva potilastiedon luominen muutettiin 0,01 s:n välein tapahtuvaksi eli 500-kertaiseksi. Prototyyppi toimi oikein myös tällä nopeudella, eikä Apache Kafkan suorituskyky heikentynyt lainkaan. Potilastiedon lähettäminen InfluxDB-tietokantaan hidastui kuitenkin merkittävästi ja Python-koodi jatkoi potilastiedon lähettämistä tietokantaan vielä muutamien minuuttien ajan, vaikka potilastietoa luotiin vain 10 s:n ajan. 10 s:ssa luotiin noin 2000 tapahtumaa yhteensä ja näiden siirtäminen tietokantaan kesti huomattavasti pidempään kuin tiedon simuloinnissa ja Kafkaan lähettämisessä. Kuvassa 29 esitetään Grafana-hallintapaneelin näkymä tilanteessa, jossa potilastiedon luontitaajuus on erittäin korkea.

Kuva 29. Grafana-hallintapaneelin näkymä korkealla potilastiedon luontitaajuudella.

6 Yhteenveto

Opinnäytetyön tavoitteena oli tutustua Apache Kafka -tapahtumavirta-alustaan sekä reaaliaikaiseen analytiikkaan ja luoda näiden tietojen pohjalta prototyyppi reaaliaikaisen potilastiedon keräämiseksi ja analysoimiseksi. Prototyypin luominen onnistui, ja se pystyy reaaliaikaisesti keräämään ja analysoimaan simuloitua potilastietoa. Simuloitu potilastieto kulkee tehokkaasti järjestelmän eri osien läpi, ja tiedon käsittely tapahtuu sujuvasti lyhyessä ajassa.

Apache Kafka toimi erinomaisesti myös korkeammilla potilastiedon luontitaajuuksilla, mutta Pythonin ja InfluxDB:n välinen yhteys kuitenkin hidastui merkittävästi potilastiedon luontitaajuuden ollessa erittäin korkea.

Tulevaisuudessa prototyyppiä voisi kehittää luomalla esimerkiksi oman Java-pohjaisen kirjaston, joka mahdollistaisi sujuvamman työskentelyn Kafkan kanssa prototyypin omat tarpeet huomioiden. Tällöin ei oltaisi enää riippuvaisia ulkoisista Python-kirjastoista, joita tässä prototyypissä käytettiin. Reaaliaikaisuuden lisäämiseksi voisi myös harkita oman analytiikkajärjestelmän kehittämistä, sillä käytössä oleva Grafana rajoittaa hallintapaneelien päivitysnopeuden 5 sekuntiin. Tällä muutoksella potilastiedot saataisiin visualisoitua muutamia sekunteja nopeammin.

Kehitettyä prototyyppiä voitaisiin käyttää esimerkiksi sairaaloiden potilastietojärjestelmissä, jolloin potilastietojärjestelmien keräämät tiedot olisivat helpommin seurattavissa ja analysoitavissa. Tämä edellyttäisi kuitenkin integraatiota potilastietojärjestelmän ja Apache Kafkan välillä sekä varmistusta siitä, että integraatio noudattaa kaikkia GDPR:n ja tietosuojalainsäädännön vaatimuksia. Mikäli integraatio toteutettaisiin, prototyyppiä voitaisiin edelleen kehittää esimerkiksi siten, että se hälyttäisi hoitajalle, jos potilaan terveysarvot laskevat tai nousevat terveydelle vaaralliselle tasolle.

Potilastietojärjestelmien lisäksi prototyyppi voisi toimia potilasmonitorien tukena, jolloin monien eri potilasmonitorien tiedot olisivat kootusti katseltavissa. Tämä

mahdollistaisi paremman seurannan ja nopeamman reagoinnin potilaiden elintoimintojen muutoksiin.

Lähteet

Amazon Web Services 2025a. What is an API (Application Programming Interface)? Viitattu 18.3.2025.
https://aws.amazon.com/what-is/api/

Amazon Web Services 2025b. What is an Event-Driven Architecture? Viitattu 19.2.2025.
https://aws.amazon.com/event-driven-architecture/

Amazon Web Services 2025c. What is Apache Kafka? Viitattu 6.2.2025.
https://aws.amazon.com/what-is/apache-kafka/

Amazon Web Services 2025d. What is SQL (Structured Query Language)? Viitattu 18.3.2025.
https://aws.amazon.com/what-is/sql/

Apache Kafka 2025. Documentation. Viitattu 3.2.2025.
https://kafka.apache.org/documentation/

BasuMallick, C. 2022. What Is Kafka? Definition, Working, Architecture, and Uses. Viitattu 13.2.2025.
https://www.spiceworks.com/tech/data-management/articles/what-is-kafka/

Brush, K. 2022. real time analytics. Viitattu 22.2.2025.
https://www.techtarget.com/searchcustomerexperience/definition/real-time-analytics

China C. & Goodwin M. 2024. What is event streaming? Viitattu 19.2.2025
https://www.ibm.com/think/topics/event-streaming

Confluent 2025a. Kafka APIs. Viitattu 17.2.2025.
https://docs.confluent.io/kafka/kafka-apis.html

Confluent 2025b. What is Apache Kafka®? Viitattu 23.1.2025.
https://www.confluent.io/what-is-apache-kafka/

GeeksforGeeks 2024. What is Kafka Broker? Viitattu 6.2.2025.
https://www.geeksforgeeks.org/what-is-a-kafka-broker/

IBM 2025. What is Apache Kafka? Viitattu 18.3.2025.
https://www.ibm.com/think/topics/apache-kafka

Kirvan, P. 2023. prototype. Viitattu 24.1.2025.
https://www.techtarget.com/searcherp/definition/prototype

Linux.fi-wiki 2025. Ympäristömuuttuja. Viitattu 18.3.2025.
https://www.linux.fi/wiki/Ymp%C3%A4rist%C3%B6muuttuja

Marr, B. 2025. What Is Real-Time Data Analytics (And Why It's So Important)? Viitattu 22.2.2025.
https://bernardmarr.com/what-is-real-time-data-analytics-and-why-its-so-important/

Microsoft 2023. What is the Windows Subsystem for Linux? Viitattu 18.3.2025.
https://learn.microsoft.com/en-us/windows/wsl/about

MongoDB 2025. Real-Time Analytics Explained. Viitattu 22.2.2025
https://www.mongodb.com/resources/basics/real-time-analytics-examples

NordVPN 2022. TCP IP -mikä se on, mihin sitä tarvitaan ja mitä se tekee? Viitattu 18.3.2025.
https://nordvpn.com/fi/blog/tcp-ip-protokolla/

Oracle 2025a. Java™ Platform Standard Edition 24 Development Kit - JDK™ 24. Viitattu 18.3.2025.
https://www.oracle.com/java/technologies/javase/jdk24-readme-downloads.html

Oracle 2025b. What is Java technology and why do I need it? Viitattu 18.3.2025.
https://www.java.com/en/download/help/whatis_java.html

PyPI 2025. pip. Viitattu 18.3.2025.
https://pypi.org/project/pip/

Scala 2025. TOUR OF SCALA. Viitattu 19.3.2025
https://docs.scala-lang.org/tour/tour-of-scala.html

Sisense 2025. Real Time Analytics. Viitattu 3.3.2025.
https://www.sisense.com/glossary/real-time-analytics/

Splunk 2023. Real-Time Analytics: Definition, Examples & Challenges. Viitattu 18.3.2025.
https://www.splunk.com/en_us/blog/learn/real-time-analytics.html

The Python Software Foundation 2025. Python For Beginners. Viitattu 18.3.2025.
https://www.python.org/about/

Tinybird 2023. Real-Time Analytics: Examples, Use Cases, Tools & FAQs. Viitattu 3.3.2025.
https://www.tinybird.co/blog-posts/real-time-analytics-a-definitive-guide#what-is-real-time-analytics

Ubuntu Suomi 2025. Esittely. Viitattu 18.3.2025
https://ubuntu-fi.org/esittely/

Verma, P. 2025. Mastering Apache Kafka: Powering Modern Data Pipelines. Viitattu 13.2.2025.
https://dev.to/pragativerma18/mastering-apache-kafka-powering-modern-data-pipelines-5ebh