Jens Helin

Tietokantapohjainen järjestelmä PLC-laitteiden reaaliaikaiseen tiedonkeruuseen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Sähkö- ja automaatiotekniikka

Insinöörityö

26.5.2025

Tiivistelmä

Tekijä:	Jens Helin
Otsikko:	Tietokantapohjainen järjestelmä PLC-laitteiden reaaliaikaiseen tiedonkeruuseen
Sivumäärä:	42 sivua + 1 liite
Aika:	26.5.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Sähkö- ja automaatiotekniikka
Ammatillinen pääaine:	Automaatiotekniikka
Ohjaajat:	Lehtori Matti Välikylä

Tässä opinnäytetyössä suunniteltiin ja toteutettiin OPC UA -protokollaan pohjautuva reaaliaikainen datankeruujärjestelmä Metropolia Ammattikorkeakoulun automaatiolaboratorioon. Opinnäytetyön tavoitteena oli mahdollistaa automaatiolaboration laitteista tulevan prosessidatan tehokas ja luotettava tiedonkeruu ja tallennus myöhempää analysointia varten.
Järjestelmä rakentuu kerroksittain. OPC UA -asiakasohjelma toteutettiin Python-kielellä tiedon keräämiseksi palvelimelta. Asiakasohjelman rinnalla tietokantapalvelimena on TimescaleDB-aikasarjatietokanta datan tallennukseen. Järjestelmässä hyödynnetään OPC UA -tilauspohjaista tiedonsiirtoa aina kun mahdollista, ja vaihtoehtoisesti turvaudutaan säännölliseen kyselyyn tilanteissa, joissa tilausmekanismi ei ole käytettävissä.
Tietokantaan suunniteltiin datan skeema, joka tallentaa monentyyppisiä mittausarvoja aikaleimoineen. TimescaleDB:n ominaisuuksien avulla toteutettiin automaattinen tietojen arkistointi sekä tunnittain päivittyvät aggregaattinäkymät.
Ohjelmistojärjestelmää testattiin simuloidulla OPC UA -palvelimella. Testaustulokset osoittivat, että ohjelmistojärjestelmä pystyy keräämään ja tallentamaan dataa luotettavasti muutaman sekunnin välein tapahtuvalla päivitystahdilla ilman havaittavaa viivettä. Opinnäyteyön lopputuloksena on dokumentaatio ohjelmistojärjestelmän rakenteesta, toteutuksesta ja testeistä sekä toimiva PLC-dataa keräävä järjestelmä.

Avainsanat: Tietokanta, PostgreSQL, PLC, OPC UA

Abstract

Author:	Jens Helin
Title:	A Database-Driven System for Real-Time Data Collection from PLC Devices
Number of Pages:	42 pages + 1 appendix
Date:	26 May 2025
Degree:	Bachelor of Engineering
Degree Programme:	Electrical and Automation Engineering
Professional Major:	Automation Engineering
Supervisors:	Matti Välikylä, Senior Lecturer

In this thesis work, a real-time data collection system based on the OPC UA protocol was designed and implemented for the automation laboratory at Metropolia University of Applied Sciences. The goal of the project was to enable efficient and reliable data collection and storage of process data from the laboratory devices for later analysis.

The system is built in layers: an OPC UA client program was implemented in Python to collect data from the server, and TimescaleDB, a time-series database, serves as the backend for data storage. The system utilizes OPC UA subscription-based data transfer whenever possible, and in situations where the subscription mechanism is unavailable, regular polling is used as an alternative.

A schema was designed for the database to store various types of measurement values along with timestamps, and using TimescaleDB's features, automatic data archiving and hourly updating aggregate views were implemented.

The system was tested using a simulated OPC UA server. The test results showed that the solution is capable of collecting and storing data reliably with an update rate of a few seconds, without noticeable delay. The outcome of the thesis work includes documentation of the system's structure, implementation, and testing, as well as a functional data collection system.

Keywords: Database, PostgreSQL, PLC, OPC UA

Sisällys

Lyhenteet ja käsitteet

Aikasarjatietokanta:	Tietokantaohjelma, joka on optimoitu ja tarkoitettu aikasarjadatan tallennukseen ja hakemiseen.
OPC UA:	*Open Platform Communications Unified Architecture*. Teollisuusautomaation koneiden välinen tietoliikenneprotokolla.
PLC:	*Programmable logic controller, eli* ohjelmoitava logiikka.
PostgreSQL:	Avoimen lähdekoodin relaatiotietokanta.
SOA:	*Service-oriented architecture*, palvelukeskeinen arkkitehtuuri. Ohjelmistotekniikan arkkitehtuurityyppi, jossa järjestelmien eri osat toimivat itsenäisinä palveluina.
SQL:	*Structured Query Language*. Standardoitu kyselykieli, jolla tallennetaan ja prosessoidaan tietoa relaatiotietokannassa.
TimescaleDB:	Avoimen lähdekoodin aikasarjatietokanta ja PostgreSQL:n laajennus.

1 Johdanto

Opinnäytetyössä kehitettiin tietoa välittävä OPC UA -asiakasohjelma ja tietokantasovellus, joita voidaan käyttää ja laajentaa käytettäväksi erilaisille ohjelmoitaville logiikoille (PLC) reaaliaikaisen tiedon tallentamiseen. Tavoitteena oli rakentaa skaalautuva ja luotettava ratkaisu, joka mahdollistaa automaatiojärjestelmien keräämän tiedon hyödyntämisen esimerkiksi analytiikassa, valvonnassa ja opetuksessa. Aihe on ajankohtainen, koska teollisuus 4.0:n ja älykkäiden järjestelmien kehittymisen myötä reaaliaikaisen datan kerääminen ja hallinta ovat keskeisiä osia nykyaikaisessa automaatiossa.

Opinnäytetyön lopputulos on tarkoitus ottaa käytettäväksi Metropolia Ammattikorkeakoulun Myyrmäen kampuksen automaatiolaboratorion PLC-tietojen käsittelyyn. Kehittämistyössä hyödynnettiin ajankohtaisia ja luotettavia lähteitä, kuten OPC UA -standardin dokumentaatiota, TimescaleDB-aikasarjatietokannan teknistä aineistoa sekä alan kirjallisuutta ja tieteellisiä julkaisuja teollisen Internetin ja automaatioteknologioiden alalta.

OPC UA -asiakasohjelma toteutettiin Python-ohjelmointikielellä käyttäen opcua-kirjastoa, ja sen tehtävänä on noutaa tietoa jatkuvasti PLC-laitteilta ja tallentaa se suoraan tietokantaan ilman välikerroksia. Tietokantasovellus toteutettiin PostgreSQL:llä hyödyntäen TimescaleDB-laajennusta aikaleimapohjaisen tiedon tehokkaaseen hallintaan. Näin rakennettu kokonaisuus tarjoaa pohjan tulevalle laajennukselle ja soveltamiselle erilaisissa opetuksen ja kehitystyön tarpeissa.

OPC UA (Open Platform Communications Unified Architecture) on standardoitu tiedonsiirtoprotokolla, jonka avulla on mahdollista vaihtaa tietoja laitteiden välillä riippumatta laite- tai ohjelmistovalmistajasta (OPC Foundation n.d.). OPC UA:n avulla voidaan luoda joustavia ja turvallisia tietoliikenneratkaisuja teollisuusympäristöihin. Tämän opinnäytetyön keskeinen osa on kehittää OPC UA -asiakas-

ohjelma, jonka avulla PLC-laitteista kerätään ja siirretään dataa tietokantaan analysoitavaksi.

PLC-tietojen tallennukseen käytetään PostgreSQL-tietokantaa, johon on integroitu TimescaleDB-laajennus. TimescaleDB on optimoitu suurten aikasarjadatamäärien käsittelyyn ja sisältää tehokkaita työkaluja tietojen hallintaan ja analysointiin. Hypertable-rakenteen ansiosta tietokanta skaalautuu hyvin kasvavan PLC-datamäärän mukana ja mahdollistaa nopeat tietokantahaut sekä tehokkaan tallennuksen tietokantaan.

Tässä opinnäytetyössä käsitellään PLC-dataa tallentavan järjestelmän suunnittelua, toteutusta ja suorituskykyä. Opinnäyteyön tavoitteena on rakentaa toimiva järjestelmäkokonaisuus, jossa OPC UA -asiakasohjelma pystyy lukemaan ja tallentamaan reaaliaikaista dataa PLC-laitteista tietokantaan. Lisäksi opinnäytetyössä analysoidaan järjestelmän suorituskykyä ja skaalautuvuutta sekä pohditaan järjestelmän mahdollisia kehityssuuntia tulevaisuudessa.

Opinnäytetyön rakenne on seuraava: luvussa 2 käsitellään opinnäytetyön teknologioita ja teoreettista viitekehystä, mukaan lukien OPC UA -protokollaa ja PLC-laitteiden sekä PostgreSQL TimescaleDB -tietokannan periaatteita. Luvussa 2 käsitellään opinnäytetyön teknologioiden teoreettista taustaa. Luvussa 3 esitellään opinnäytetyön suunnitelma ja perustellaan järjestelmän teknologiavalintoja, kun taas luvussa 4 käsitellään järjestelmän toteutusta. Luvussa 5 käsitellään järjestelmän testausta. Luvussa 6 tiivistetään ja esitetään opinnäytetyön tuloksia ja johtopäätöksiä.

Opinnäytetyöraportin kieliasun muotoilussa, tarkistamisessa, ja kappalerakenteen suunnittelussa on käytetty OpenAI:n ChatGPT:n mallia GPT-4o. Opinnäytetyön tekijä on vastuussa kaikesta opinnäytetyön sisällöstä ja muotoilusta.

2 Teknologiavalintojen teoreettinen tausta

Tässä opinnäytetyön luvussa 2 käsitellään opinnäytetyön järjestelmän keskeisiä teknologioita ja taustatietoja. Teoreettisen taustan aihepiirit alkavat OPC UA -protokollasta yleisellä tasolla, jatkuvat OPC UA -kirjastojen hyödyntämiseen Python-kielessä ja reaaliaikaiseen tiedonkeruuseen PLC-laitteista, ja etenevät tietokantatasolle PostgreSQL- ja TimescaleDB-teknologioihin. Lisäksi tarkastellaan järjestelmän suunnitellun tiedonkeruun ja -tallennuksen integraatioarkkitehtuuria sekä OPC UA -pohjaisten järjestelmien haasteita ja rajoitteita.

2.1 OPC UA -protokollan historia ja käyttökohteet

OPC UA (Open Platform Communications Unified Architecture) on valmistajariippumaton tiedonsiirtostandardi teollisuusautomaation järjestelmien väliseen kommunikointiin. Se on OPC-standardien seuraava sukupolvi, joka kehitettiin ratkaisemaan aiemman OPC Classic -teknologian puutteet, kuten toimivuus eri alustoilla ja puutteellinen tietoturvallisuus (Cisco n.d.). Ensimmäinen versio OPC UA -määrittelystä julkaistiin vuonna 2006, ja sen tavoitteena oli yhdistää aiemmat erilliset OPC-määritykset, kuten Data Access, Alarms & Events, Historical Data, yhdeksi yhtenäiseksi protokollaksi (Cisco n.d.). OPC UA pohjautuu palvelukeskeiseen arkkitehtuuriin (SOA) ja on alusta- ja käyttöjärjestelmäriippumaton toisin kuin OPC Classic, joka perustui Windowsin COM/DCOM-tekniikkaan (Claroty Team82 2023). Tämä tarkoittaa, että OPC UA -palvelimet ja -asiakkaat voivat toimia erilaisilla laitteilla ja käyttöjärjestelmissä, mikä oli merkittävä parannus aiempaan vain Windows-ympäristöön rajautuneeseen OPC Classic -standardiin. OPC UA on nykyään standardoitu IEC 62541 -standardisarjassa ja siitä on tullut tärkeä osa teollisuus 4.0 -ratkaisuja sekä IT- ja OT-järjestelmien integraatiota (Hoppe 2023; Waehner 2022).

OPC UA noudattaa asiakas-palvelin-arkkitehtuuria, jossa OPC UA -palvelin lähettää tietoa ja OPC UA -asiakas lähettää pyyntöjä ja vastaanottaa dataa. OPC UA:n kommunikointi tapahtuu standardoiduilla palvelukutsuilla, kuten Read,

Write ja Browse, joilla asiakas saa pääsyn palvelimen tietoihin. Keskeinen osa OPC UA:n arkkitehtuuria on sen komponenttien osoiteavaruus ja datan tietomalli. Palvelimen hallinnoima tieto esitetään hierarkkisessa osoiteavaruudessa solmuina (nodes), joilla on tyypitetyt ominaisuudet ja keskinäiset suhteet (Cisco n.d.). OPC UA määrittelee muun muassa Objekti, Muuttuja ja Metodi -solmutyypit: Objekti-solmut edustavat usein laitteita tai järjestelmäkomponentteja, Muuttuja-solmut sisältävät arvoja tai ominaisuuksia, ja Metodi-solmut vastaavat palvelimen tarjoamia Objekti-solmujen toiminnallisuuksia joita asiakas voi käyttää pyynnöillä (Cisco n.d.).

Yksi OPC UA:n merkittävimmistä parannuksista edeltäjäänsä on sisäänrakennettu tietoturva-arkkitehtuuri. OPC UA hyödyntää nykyaikaisia tietoturvakäytäntöjä, kuten viestien digitaalista allekirjoitusta ja salausta Transport Layer Security (TLS) -protokollalla sekä X.509-varmenteisiin perustuvaa laite- ja käyttäjäautentikointia (PTC n.d; Cisco n.d.). Jokaisella OPC UA -sovelluksella on oma sovellusvarmenteensa, jota käytetään tunnistautumiseen toiselle osapuolelle. Näin varmistetaan, että vain luotetut laitteet voivat muodostaa yhteyden. Asiakkaan ja palvelimen kommunikointi tapahtuu suojatussa kanavassa, joka luodaan neuvottelemalla sopiva tietoturvasääntö asiakkaan ja palvelimen välillä ja vaihtamalla varmenteet (Cisco n.d.). Suojatun kanavan sisällä OPC UA ylläpitää istuntoa, joka sisältää tietoa käyttäjän todentamisesta ja käyttäjän roolista (Cisco n.d.).

OPC UA:ta käytetään monipuolisesti teollisuuden tiedonsiirrossa. OPC UA:ta käytetään tavallisesti PLC:n liittämiseen valvomo/HMI-järjestelmiin ja tuotantotietojen siirtoon prosessinohjausjärjestelmistä ylemmille IT-tasoille, esimerkiksi toiminnanohjausjärjestelmissä.

OPC UA:n valmistajariippumattomuus mahdollistaa ”horisontaalisen” integraation eri laitevalmistajien järjestelmien välillä ja ”vertikaalisen” integraation tehtaan lattialta pilvipalveluihin asti (Hoppe 2023). Esimerkiksi modernit teollisuus 4.0 -sovellukset hyödyntävät OPC UA:ta yhdistäessään IoT-laitteita ja anturidataa pilvialustoihin analysoitavaksi. Monissa tuotantolaitoksissa OPC UA toimii

informaatiorakenteiden perustana: se välittää reaaliaikaiset anturitiedot, koneiden tilatiedot ja hälytykset eri järjestelmien kesken luotettavasti ja turvallisesti.

OPC UA on käytössä laajasti mm. valmistavassa teollisuudessa, energia- ja sähköverkkojen valvonnassa, rakennusautomaatiossa ja prosessiteollisuudessa, käytännössä siellä, missä tarvitaan standardoitua laiterajapintaa eri laitteiden väliseen tiedonvaihtoon (Waehner 2022). OPC UA -standardin laajan hyväksynnän ansiosta monet automaatioalan laitevalmistajat, kuten Siemens, Beckhoff, Schneider Electric ovat sisällyttäneet OPC UA -palvelintoiminnallisuuden suoraan laitteisiinsa, mikä on edelleen kiihdyttänyt OPC UA:n leviämistä yleisenä standardina teollisessa internetissä.

2.2 Pythonin OPC UA -kirjastot

Python-ohjelmointikielen avulla OPC UA -integraatioita voidaan liittää järjestelmiin joustavasti hyödyntäen valmiita kirjastoja. Suosituimpiin Python-kirjastoihin OPC UA -yhteyksien rakentamiseksi kuuluu avoimen lähdekoodin FreeOpcUa-projektiin perustuva kirjasto, joka tunnetaan nimellä Python-opcua (asennusnimellä ”opcua”). Tästä kirjastosta on olemassa sekä perinteinen synkroninen versio että uudempi asynkroninen versio ”opcua-asyncio”. Nämä kirjastot mahdollistavat OPC UA -asiakkaiden ja -palvelimien kehittämisen Pythonilla.

Python-opcua on kehitetty lähes täydellisenä OPC UA -protokollapinona Python-ohjelmointikielellä. Python-opcua tukee sekä OPC UA -asiakas- että palvelinrooleja (Roulet-Dubonnet 2018). Python-opcua sisältää matalan tason rajapinnan, jolla voidaan käsitellä kaikkia OPC UA -spesifikaation määrittelemiä rakenteita ja viestejä sekä korkean tason luokkia ja funktioita, joiden avulla yksinkertaiset asiakas- ja palvelinohjelmat saa kehitettyä helposti (Roulet-Dubonnet 2018). Esimerkiksi yksinkertaisimmillaan OPC UA -asiakasohjelma voidaan toteuttaa luomalla Client-olio kirjastolla ja kutsumalla sillä connect()-metodia, jonka jälkeen voidaan suorittaa lukemisia ja kirjoituksia palvelimen solmuihin korkean tason OPC UA -metodeilla. Kirjaston korkean tason rajapinta huolehtii mm. viestien paketoinnista OPC UA -binääriprotokollan mukaisiksi ja vastausten

tulkinnasta takaisin Python-olioksi. Tarvittaessa kehittäjä voi myös yhdistää korkean tason metodien käyttöä ja matalan tason viestirakenteiden käsittelyä, mikä antaa joustavuutta edistyneisiin OPC UA:n käyttötapauksiin (Roulet-Dubonnet 2018).

Python-opcua-kirjasto on syntynyt samasta projektista kuin FreeOpcUa C++ -toteutus. Suuri osa koodista generoidaan automaattisesti OPC UA -määrittelyn XML-kuvauksista (Roulet-Dubonnet 2018). Tämä varmistaa, että kirjaston toteutus pysyy spesifikaation mukaisena ja tukee laajasti protokollan ominaisuuksia. Kirjasto on testattu yhteensopivaksi useiden muiden OPC UA -toteutusten kanssa, mikä on tärkeää eri valmistajien järjestelmien yhteistoiminnassa (Roulet-Dubonnet 2018). Python-opcua tukee muun muassa seuraavia keskeisiä toimintoja: yhteyden muodostus Discoveryn kautta tai suoraan palvelimen URL-osoitteella, tietojen lukeminen ja kirjoittaminen (read/write), solmujen selaus (browse), Subscription-mekanismin käyttö mittausarvojen jatkuvaan seurantaan ja OPC UA -metodikutsut palvelimelle.

Esineiden internetin ja asynkronisen ohjelmoinnin yleistyessä Python-opcua-kirjastosta kehitettiin myös Asyncio-pohjainen versio nimeltään Asyncua. Asyncua hyödyntää Pythonin Asyncio-kirjastoa, jossa on asynkroniset vastineet OPC UA -operaatioille (FreeOpcUa 2023). Asynkroninen ohjelmointimalli mahdollistaa useiden samanaikaisten yhteyksien ja operaatioiden hallinnan tehokkaasti yhdessä prosessorin säikeessä hyödyntämällä tapahtumasilmukkaa. Tämä vähentää tarvetta asiakasohjelman monisäikeisyydelle ja asynkronisten tehtävien lukituksille. Tämän vuoksi, koodin rakenne on yksinkertaisempaa ja suorituskyky voi olla parempi IO-intensiivisissä tilanteissa (FreeOpcUa 2023). Käytännössä Asyncua-kirjastossa monet toiminnot, kuten palvelimeen yhdistäminen tai arvon lukeminen, ovat async def -metodeja, joita kutsutaan await-operaatiolla.

Asyncua-kirjasto on taaksepäin yhteensopiva siinä mielessä, että käytettävissä on myös synkroninen rajapintakerros. Asynkronisen sovelluksen kehittäjä voi käyttää asyncua-kirjastoa perinteiseen tapaan ilman await-syntaksia, jolloin kirjasto sisäisesti käynnistää asyncio-silmukan taustalle (FreeOpcUa 2023). Suo-

siteltavaa kuitenkin on hyödyntää asynkronista mallia aina, kun sovellusarkkitehtuuri sen sallii, sillä silloin voidaan luonnollisesti toteuttaa esimerkiksi usean OPC UA -palvelimen samanaikainen kysely tai datan puskurointi tietokantaan odottamatta kutakin operaatiota erikseen loppuun.

Pythonin OPC UA -kirjastot on havaittu hyödyllisiksi prototyyppien luonnissa ja kevyissä integraatioissa. Asyncua-kirjaston korkean tason metodit yksinkertaistavat PLC-laitteen muuttujien lukemista: kehittäjän ei tarvitse itse muodostaa monimutkaisia binääriviestejä, vaan esimerkiksi get_node() ja read_value() -kutsut riittävät datan hakemiseen palvelimelta. Asynkronisuuden ansiosta asiakasohjelma voi samanaikaisesti kerätä dataa ja lähettää sitä tietokantaan tehokkaasti. On kuitenkin huomioitava, että Pythonilla toteutettu puhdas ohjelmistopino ei välttämättä saavuta yhtä suurta suorituskykyä kuin esimerkiksi C++:lla toteutetut OPC UA -kirjastot, erityisesti erittäin raskaan datavirran kuormituksessa. Toisaalta Python-kehityksen etuna on nopea ohjelmistokehitys ja helppo ohjelman muokattavuus. Nämä ominaisuudet ovat arvokkaita kokeilu- ja tutkimusprojekteissa.

2.3 Tiedonkeruu PLC-laitteilta OPC UA:n avulla

Ohjelmoitavilta logiikkaohjaimilta (PLC) kerätään reaaliaikaista dataa usein valvomojärjestelmiä, historian tallennusta ja analytiikkaa varten. OPC UA -protokollalla on tätä varten vakioitu rajapinta: tyypillisessä asetelmassa PLC toimii OPC UA palvelimena altistaen halutut muistialueet tai prosessiarvot OPC UA -osoiteavaruudessa, ja erillinen asiakas muodostaa yhteyden PLC:hen verkon yli ja lukee tietoja. Tiedonkeruu voidaan toteuttaa kahdella tavalla: joko kyselynä (pollaus, asiakas lähettää säännöllisesti Read-pyyntöjä) tai tilaamalla muutokset. OPC UA tukee molempia, mutta yleensä Subscription-mekanismia suositellaan reaaliaikaiseen monitorointiin, koska se vähentää verkon kuormitusta ja viivettä – palvelin puskee uudet arvot automaattisesti, jolloin vältytään tarpeettomilta kyselyiltä silloin kun arvo ei ole muuttunut (Claroty Team82 2023).

OPC UA ei ole kovareaaliaikainen (hard real-time) protokolla, eli sen läpi kulkevan datan viive riippuu monista tekijöistä: palvelimen näytteenottosyklistä, verkon viiveistä (TCP/IP) ja asiakkaan kysely-/tilaustaajuudesta. Tyypillisesti PLC:n OPC UA -palvelimen päivitystiheys voi olla luokkaa kymmeniä millisekunteja. Toisin sanoen OPC UA soveltuu hyvin pehmeisiin reaaliaikaisuuden tarpeisiin, kuten prosessien seurantaan, jossa kymmenien tai satojen millisekuntien viive on hyväksyttävä, mutta kaikkein aikakriittisimmissä ohjaussilmukoissa se ei korvaa erikoistuneita väyliä. On kuitenkin huomattava, että OPC UA -standardi kehittyy: uusi OPC UA Pub/Sub -malli ja siihen liitetty TSN (Time-Sensitive Networking) -teknologia tähtäävät aidosti deterministiseen reaaliaikaisuuteen Ethernet-verkossa (Waehner 2022). Tuore tutkimus on myös osoittanut, että oikein konfiguroituna ja TSN-tekniikalla tuettuna OPC UA kykenee saavuttamaan monien teollisten sovellusten tiukat reaaliaikavaatimukset tavallisella laitteistolla (Kirdan ym. 2023).

Kun yhdeltä PLC:ltä kerätään monia signaaleja samanaikaisesti, on tärkeää huolehtia tiedonkeruun tehokkuudesta. OPC UA -palvelimet, kuten PLC:n sisäinen palvelin, rajoittavat kerralla luettavien tai tilattujen muuttujien määrää. Esimerkiksi palvelin saattaa rajoittaa yhden Read-pyynnön kohteiden lukumäärän tiettyyn enimmäismäärään tai tilauksessa sallittujen Monitored Item -kohteiden, eli tarkkailtujen muuttujien määrän esimerkiksi muutamaan tuhanteen per asiakas (Bleuzé 2024). OPC UA -asiakkaan tilauksissa hyvä käytäntö onkin ryhmitellä luettavat signaalit useaksi erilliseksi Read-pyynnöksi sen sijaan, että yrittää lukea esimerkiksi tuhansia arvoja yhdellä Read-pyynnöllä (Bleuzé 2024). Liian suuren kertapyynnön ongelma on, että jos yksikin hidas kohde viivästyttää vastausta, koko pyyntö voi aikakatkaista – jakamalla vaikkapa 5 000 muuttujan luvun kymmeneen 500 muuttujan pyyntöön, yksittäisen arvon viive vaikuttaa vain kyseiseen ryhmään, eikä lamauta koko lukuoperaatiota (Bleuzé 2024).

Subscription-tilausten osalta palvelimilla on samankaltaisia rajoitteita. Tyypillisesti OPC UA -palvelin saattaa sallia vain tietyn määrän yhtäaikaisia monitored item -kohteita asiakkaan mukaan. Mikäli asiakas yrittää tilata liikaa muuttujia, palvelin palauttaa virheen (Bleuzé 2024). Tähän on kaksi ratkaisua: nostaa pal-

velimen rajoitusta tai jakaa tilaukset usean rinnakkaisen asiakasyhteyden kesken (Bleuzé 2024). Jälkimmäinen tarkoittaa esimerkiksi sitä, että voidaan ajaa kahta OPC UA -asiakasohjelmaa, joista kumpikin tilaa puolet muuttujista samalta palvelimelta. Joissain tapauksissa on raportoitu, että palvelin saattaa laskea vanhentuneidenkin istuntojen tilaamat muuttujat rajoihin mukaan. Tällöin on huolehdittava, että asiakasohjelma sulkee istuntonsa, jotteivät virheellisesti päälle jääneet istunnot varaa palvelimen resursseja (Bleuzé 2024).

Reaaliaikaisessa PLC:n tiedonkeruussa on olennaista, ettei PLC-dataa menetetä. OPC UA:n yli kulkeva tieto liikkuu TCP-protokollan välityksellä, mikä takaa että periaatteessa kaikki lähetetyt viestit saapuvat perille järjestyksessä – toisin sanoen satunnaiset pakettien katoamiset (packet loss) verkossa eivät johda datan häviämiseen ilman, että yhteys asiakkaan ja palvelimen välillä katkeaisi. Yhteyskatkokset tai viiveet voivat aiheuttaa tietynlaista tietohäviötä. Jos asiakas on tilannut palvelimelta muuttujan arvoja ja yhteys palvelimelle katkeaa muutamaksi sekunniksi, tuona aikana tapahtuneet arvomuutokset eivät välttämättä kaikki välity asiakkaalle, kun yhteys palautuu. OPC UA -standardiin on tosin määritelty ominaisuus nimeltä Durable subscription, jonka avulla palvelin voi pitää tilausten tietoja tallessa pidempään katkoksia varten (OPC Foundation 2017). Kaikki OPC UA -toteutukset eivät kuitenkaan tue tätä; tyypillisesti PLC:n sisäiset OPC UA -palvelimet saattavat pudottaa tilauksen aikaeron kasvaessa liian suureksi. Siksi asiakasohjelman on hyvä kyetä havaitsemaan katkokset ja mahdollisesti hakea viimeisin tilatun muuttujan arvo uudelleen yhteyden OPC UA -palvelimelle palattua, jotta tilattujen muuttujien uusimmat arvot saadaan noudettua.

2.4 Relaatiotietokanta PostgreSQL ja aikasarjatietokanta TimescaleDB

PostgreSQL on avoimen lähdekoodin relaatiotietokanta, joka tunnetaan luotettavuudesta ja monipuolisista ominaisuuksista, kuten sitä laajentavista ohjelmista, tuesta useille eri datatyypeille ja kattavasta dokumentaatiosta. Sitä voidaan käyttää yleiskäyttöisenä tietokantana esimerkiksi teollisuusdatankin tallennukseen. Perinteisesti reaaliaikaisen aikasarjadatan tallentaminen relaatiotietokan-

taan on vaikea toteuttaa säilyttämällä hyvä suorituskyky, kun datamäärät kasvavat suuriksi, esimerkiksi miljooniin riveihin päivässä. Tähän tarpeeseen on kehitetty TimescaleDB-aikasarjatietokanta, joka on PostgreSQL:n laajennus (extension), joka sovittaa tietokantaa nimenomaan aikaleimallisen datan käsittelyyn.

TimescaleDB:n keskeinen konsepti on Hypertable, joka on eräänlainen PostgreSQL:n taulu. Hypertable-taulu näkyy käyttäjälle yhtenä isona tauluna, johon OPC UA:n mittausdata kirjoitetaan. Taulun taustalla TimescaleDB pilkkoo datan automaattisesti pienempiin osiin eli lohko-osatauluihin aikajaksojen perusteella (Timescale 2023). Toisin sanoen Hypertable-taulun data on pilkottu ajan mukaan: esimerkiksi jokainen kuukausi tai viikko voidaan tallentaa omaan osatauluunsa. Tämä on läpinäkyvää tietokantojen ulkoiselle sovellukselle. Sovelluksen kehittäjä tekee normaalin SQL-kyselyn Hypertable-tauluun, ja tietokanta itse ohjaa kyselyn vain niihin osatauluihin (chunk) jotka sisältävät kyselyn aikaväliin sopivaa dataa (Timescale 2023). Tämä parantaa tietokantapalvelimen suorituskykyä huomattavasti isoissa tietomassoissa, koska esimerkiksi haettaessa kyselyllä dataa viimeiseltä tunnilta, tietokannan ei tarvitse skannata koko taulua, joka saattaa sisältää dataa vuosien ajalta, vaan se tarkistaa vain viimeisimmät osiot.

Suuri aikasarjadata on usein tarpeen tiivistää pidemmän aikavälin analyyseja varten. Continuous Aggregates eli jatkuvat aggregaattinäkymät ovat TimescaleDB:n tarjoama ominaisuus, joka muistuttaa PostgreSQL:n materiaalisoituja näkymiä (Materialized Views), mutta on optimoitu päivittymään jatkuvasti tietokantapalvelimen taustalla (Timescale 2023).

Continuous aggregate -näkymään määritellään aggregointikysely, ja Timescale huolehtii siitä, että näkymä pysyy ajan tasalla automaattisesti. Uutta dataa lisättäessä vain uusin aikajakso lasketaan uudelleen, eikä koko historian aggregaattia tarvitse laskea tyhjästä (Timescale 2023). Tämä tarkoittaa, että esimerkiksi reaaliaikaisessa kojelaudassa voidaan suorittaa monimutkaisiakin aggregoituja tietokantakyselyitä nopeasti, koska tietokanta on jo esilaskenut nämä arvot jatkuvasti taustalla. Continuous aggregate -näkymille voidaan määritellä päivityssääntö. Tällainen sääntö voi olla esimerkiksi että tunnin aggregaatit päivitetään

5 minuutin välein, jolloin pientä viivettä vastaan saadaan aina lähes tuoreet tunneittaiset koontitiedot käyttöön.

Continuous aggregatessa Timescale hyödyntää PostgreSQL:n materiaalisoitua näkymää (Materialized View) taustalla, mutta laajennettuna niin, että vain muuttunut aikaikkuna lasketaan. Lisäksi Timescale mahdollistaa reaaliaikaisen aggregoinnin: kyselyn suoritushetken kaikkein tuorein data yhdistetään näkymän esilaskettuihin tuloksiin kyselyä suorittaessa (Timescale 2023). Näin käyttäjä saa täydellisen vastauksen kyselylle, jossa yhdistyy historiallinen aggregoitu data ja aivan viime hetken tiedot.

Aikasarjadatan erityispiirre on, että vanhenevaa dataa kertyy valtavasti. Kaikkea raakadataa ei ole mielekästä säilyttää ikuisesti täydellä resoluutiolla. TimescaleDB:n tärkeitä ominaisuuksia ovat mekanismit datan elinkaaren hallintaan. Yksi keskeinen työkalu on tietojen säilytyskäytäntö (data retention policy) eli vanhan datan automaattinen poisto. Voidaan esimerkiksi määrittää, että hypertablesta poistetaan kaikki data, joka on vuotta vanhempaa. Timescale hoitaa tällöin vanhimpien lohkojen pudottamisen taulusta säännöllisesti, mikä hillitsee tietokannan kokoa (Timescale 2024a). Toinen työkalu on datakompressio: TimescaleDB osaa pakata historialliset lohkot tehokkaasti sarake-formaattiin (columnar). Esimerkiksi kun data on yli kolme kuukautta vanhaa, voidaan määrittää lohkon pakkautuvan, jolloin sen käyttämä tila pienenee huomattavasti, kuitenkin niin että kyselyt voivat yhä hakea tietoa hieman hitaammin suoraan pakkauksesta purkamatta kaikkea (Timescale 2023).

Continuous aggregaten ja säilytyspolitiikan yhdistelmä on erityisen hyödyllinen: voidaan säilyttää tarkka data vain rajallisen ajan, mutta säilyttää pitkän aikavälin trendit aggregaattinäkymissä pidempään. Esimerkiksi raakadatana pidetään 2 kuukautta sekuntitasolla, mutta päivätason keskiarvot tallentava jatkuva aggregaatti pidetään 2 vuoden ajalta. Jotta aggregaatit eivät ”katoa” raakadatan poiston myötä, on huolehdittava että aggregaattien päivitysikkuna ei ulotu poistettuun dataan (Timescale 2024b). Käytännössä tämä on helppo toteuttaa: määritellään, että aggregaatti laskee esimerkiksi 7 päivää vanhoihin tietoihin saakka,

ja raakadatasta poistetaan kaikki, mikä on vanhempaa kuin 7 päivää. Näin aggregaatti ei yritä päivittyä enää poistetulle aikavälille, eikä historiatieto katoa aggregaatista (Timescale 2024b).

PostgreSQL ja TimescaleDB toimivat yhdessä skaalautuvana ja aikasarjoille optimoituna ratkaisuna OPC UA:lta kerätyn datan tallennukseen. Hypertable mahdollistaa jatkuvan datavirran kirjoittamisen ja vanhan datan hallitun poistamisen ilman suorituskyvyn rapautumista. Jatkuvat aggregaatit tarjoavat valmiiksi lasketut tunnusluvut ja trendit nopeasti saataville esimerkiksi raportointia varten. Kaikki tämä saavutetaan säilyttäen SQL-kielen tarjoama monipuolisuus: kehittäjä tai analyytikko voi yhä käyttää standardia SQL:ää kyselyihin, ja TimescaleDB hoitaa optimoinnin taustalla. Tämä on merkittävä etu verrattuna moniin erikoistuneisiin aikasarjatietokantoihin, joissa joudutaan opettelemaan uusi kyselykieli tai joissa joustavuus on rajatumpaa. Lisäksi TimescaleDB on yhteensopiva muun PostgreSQL-ekosysteemin kanssa, eli esimerkiksi visualisointityökalut ja ORM-kirjastot toimivat suoraan. Tästä syystä PostgreSQL + TimescaleDB on hyvä valinta järjestelmässä, jossa OPC UA:n kautta tulevaa sensoridataa halutaan varastoida ja analysoida skaalautuvasti tietokantaan.

2.5 Tiedonkeruun ja tallennuksen järjestelmäintegraatio

Laajan IoT- tai automaatiojärjestelmän suunnittelussa pelkkä tiedonkeruuohjelma ja tietokanta eivät toimi tyhjiössä, vaan ne on integroitava kokonaisuudeksi. Järjestelmäintegraation keskeisiä vaatimuksia tässä yhteydessä ovat skaalautuvuus, vankkuus ja eriaikaisuus tietovirran käsittelyssä.

Skaalautuvuudella tarkoitetaan sekä pystysuuntaista skaalautumista, eli yksittäisen järjestelmän suorituskyvyn lisäämistä suuremmille datamäärille, että vaakasuuntaista skaalautumista, eli mahdollisuutta lisätä järjestelmään komponentteja tai solmuja datamäärän kasvaessa. OPC UA -pohjaisessa tiedonkeruussa tiedonsiirron pystysuuntainen skaalautuvuus ilmenee esimerkiksi siinä, kuinka monta mittausarvoa yksi OPC UA -asiakas pystyy lukemaan sekunnissa tai kuinka suuren näytemäärän tietokanta pystyy vastaanottamaan ja tallentamaan

reaaliajassa. Jos yksittäisen asiakkaan teho ei riitä, voidaan skaalata vaakasuunnassa ottamalla käyttöön useampia asiakasprosessoreita tai -sovelluksia: esimerkiksi kymmenen samanaikaista OPC UA -asiakasyhteyttä voi kukin hoitaa osan PLC-laitteista. OPC UA -protokolla tukee monia asiakkaita palvelimille rinnakkain. PLC-palvelin yleensä palvelee useita asiakkaita yhtäaikaisesti, toki palvelimen suorituskyvyn mukaan.

Robustilla tiedonkeruujärjestelmällä tarkoitetaan, että se kestää vikatilanteita menettämättä dataa tai kaatumatta. Tähän päästään ensinnäkin decoupling-periaatteella – erotetaan datan kerääminen ja datan tallennus toisistaan selkeästi. Käytännössä tämä voidaan toteuttaa puskurin tai viestijonon avulla: OPC UA -asiakasprosessi voi lähettää kerätyt mittaukset jonkin välimekanismin kautta tietokantaan sen sijaan, että kirjoittaa suoraan ja synkronisesti tietokantaan. Esimerkiksi arkkitehtuuri, jossa OPC UA -asiakas julkaisee viestejä Apache Kafka -jonoon tai vastaavaan, on nykyaikainen tapa rakentaa asynkroninen tiedonkeruuketju (Waehner 2022). Tällöin tietokantaan kirjoittava osa noutaa dataa jonosta omaan tahtiinsa. Tällainen löyhä kytkentä (loose coupling) mahdollistaa sen, että jos tietokanta on hetkellisesti kuormittunut tai poissa käytöstä, dataa kertyy jonoon odottamaan – eikä OPC UA -asiakkaan tarvitse pysähtyä tai menettää mittauspisteitä (Waehner 2022).

Kaikissa järjestelmissä ei kuitenkaan tarvita raskasta viestinvälitysjärjestelmää; joskus riittää sisäinen puskurointi. Python-asiakasohjelmassa voidaan toteuttaa tuottaja-kuluttaja-malli vaikkapa käyttämällä asynkronista ohjelmointia tai erillistä säiettä: yksi osa ohjelmasta lukee OPC UA -palvelinta jatkuvasti ja työntää tulokset viestijonoon, ja toinen osa lukee jonosta ja kirjoittaa tietokantaan. Tällä tavoin kirjoitus voi hieman jäljessäkin ottaa kiinni ruuhkan purkauduttua. Oleellista robustiudelle on myös virhehallinta: OPC UA -yhteyden katketessa asiakkaan tulee yrittää automaattisesti uudelleen yhteyttä. Samoin tietokantayhteyden ongelmat – esimerkiksi hetkellinen verkko-ongelma tietokantapalvelimelle – tulisi käsitellä niin, että kirjoitukset yritetään myöhemmin uudelleen, sen sijaan että sovellus vain kirjaa virheen ja häviää datan.

Asynkronisuus viittaa siihen, että eri komponentit toimivat yhtäaikaisesti riippumatta toisistaan. OPC UA on itsessään asynkroninen viestiprotokolla: asiakas voi pitää useita pyyntöjä ”lennossa” ja palvelin lähettää dataa subscriptionin puitteissa itsenäisesti. Järjestelmän kokonaissuunnittelussa asynkroninen pipeline-arkkitehtuuri tarkoittaa, ettei datan liikkuminen yhdessä vaiheessa suoraan estetä toista. Esimerkiksi perinteisessä synkronisessa mallissa saatettaisiin tehdä: lue arvo -> tallenna tietokantaan -> lue seuraava arvo. Asynkronisessa mallissa voidaan jatkuvasti lukea arvoja rinnakkain kun aiempia vielä kirjoitetaan tietokantaan. Pythonin async/await helpottaa juuri tämän kaltaisen putken rakentamista: voidaan await-operaatioilla odottaa esimerkiksi tietokantakirjoitusta blokkaamatta koko ohjelmaa – sillä välin muita arvonlukuja tai kirjoituksia voi tapahtua.

Kokonaisuutena voisi hahmottaa seuraavan järjestelmän: Useita PLC:itä kytkettynä teollisuusverkkoon tarjoavat OPC UA -palvelimia. Ajonaikainen gateway-sovellus on sijoitettu teollisuus-PC:lle, joka toimii OPC UA -asiakkaana kaikille näille PLC:ille. Gateway-sovellus kerää tiedot asynkronisesti ja puskee ne eteenpäin yrityksen verkon puolella olevalle tietokantapalvelimelle. Turvallisuussyistä ja skaalautuvuuden vuoksi mittausdata välitetään Kafka-klusterin kautta: edge-PC julkaisee OPC UA:lta saamansa datapaketit Kafkaan, josta ne muutaman sekunnin viiveellä luetaan pilvessä sijaitsevaan TimescaleDB-tietokantaan. Tietokantaan on rakennettu continuous aggregate -näkymät, joita hyödynnetään selainpohjaisessa käyttöliittymässä tuotannon KPI-mittareiden visualisointiin lähes reaaliajassa. Tällaisessa arkkitehtuurissa jokainen kerros on irrotettavissa: OPC UA -datan keruu ja Kafka-välitys käsittelevät viestejä jatkuvana virtana, ja TimescaleDB tallennus käsittelee saapuvat viestit tahtiin, joka ei suoraan vaikuta OPC UA -puolen toimintaan.

Kappaleen yhteenvetona, hyvä järjestelmäintegraatio OPC UA -pohjaisessa tiedonkeruussa nojaa löyhään kytkentään ja asynkronisuuteen. Näin järjestelmä skaalautuu kasvavaan datamäärään lisäämällä resursseja, ja kestää verkko- tai komponenttivikoja ilman kokonaistoiminnan lamaantumista. Tuloksena on jous-

tava pipeline, joka kykenee siirtämään suuren määrän teollisuusdataa luotettavasti talteen ja edelleen käsiteltäväksi.

2.6 Haasteet ja rajoitteet OPC UA -järjestelmissä

Kuten edellä todettiin, OPC UA ei takaa determinististä viivettä perinteisessä asiakas-palvelin-mallissaan. Kommunikointi tapahtuu TCP/IP-verkon yli, mikä tuo mukanaan pienen mutta ei täysin vakion viiveen. Lisäksi OPC UA -viestien käsittely palvelimella saattaa tuoda muutaman kymmenen millisekunnin viiveitä. Useimmille valvontasovelluksille tämä ei ole ongelma, mutta jos vaaditaan kovaa reaaliaikaa, OPC UA yksinään ei riitä. OPC Foundationin ratkaisu tähän on ollut laajentaa protokollaa OPC UA Pub/Sub -mallilla yhdistettynä Time-Sensitive Networking -teknologiaan, jolla pyritään alle millisekuntien ennustettaviin viiveisiin (Waehner 2022). Nämä teknologiat ovat kuitenkin vasta tulossa käytännön laajempaan käyttöön. Nykyisissä asennuksissa on hyväksyttävä, että OPC UA:n kautta tuleva data on parhaimmillaankin muutamien kymmenien millisekuntien tarkkuudella ajantasaista.

Vaikka TCP takaa yksittäisten viestien perille tulon, pidempikestoiset yhteyskatkokset voivat johtaa datan menetykseen OPC UA -järjestelmässä. Esimerkiksi, jos verkkoyhteys katkeaa 10 sekunniksi OPC UA -asiakkaan ja palvelimen välillä, tämän aikana muuttuneita arvoja ei oletusarvoisesti tallennu minnekään ellei palvelimessa ole erikseen ominaisuutta niiden puskurointiin. Kuten aiemmin mainittiin, durable subscription voi tarjota ratkaisun, mutta harva kentällä oleva laite tukee sitä laajasti vielä. Usein käytännössä on hyväksyttävä, että tällaisissa tilanteissa järjestelmä ”missaa” joitakin näytteitä.

Yhteiskatkosten tappio voidaan minimoida asettamalla subscriptionin QueueSize riittävän suureksi: tällöin palvelin jonottaa muuttuvat arvot muistissa odottaen Publish-pyyntöä. Mikäli yhteys palautuu ennen kuin jono täyttyy, asiakas saa kaikki väliin jääneet muutokset kerralla (OPC Foundation 2017). Jos jono täyttyy, palvelin alkaa pudottaa vanhimpia arvoja, mikä johtaa uudempien tietojen säilymiseen mutta vanhempien menetykseen. Tämän vuoksi on tärkeää mitoit-

taa OPC UA -palvelimen puskurit riittäviksi ja/tai käyttää rinnakkaista tallennusmenetelmää kriittisille tiedoille. Joissain sovelluksissa PLC:lle ohjelmoidaan esimerkiksi datan paikallisloggaus, joka myöhemmin luetaan jos reaaliaikainen datavirta katkesi. OPC UA:lla on myös mahdollista selata historiaa (Historical Data Access), jota voisi hyödyntää katvealueiden täyttämiseen, mikäli PLC tallentaa historiadataa.

Vaikka OPC UA itsessään on suunniteltu tietoturvalliseksi, järjestelmän kokonaisturvallisuus riippuu oikeasta konfiguraatiosta ja käyttötavoista. Varmenteiden hallinta on yksi käytännön haaste. Jotta yhteys on suojattu, asiakkaan ja palvelimen pitää vaihtaa ja luottaa toistensa varmenteisiin. Tämä vaatii esimerkiksi sitä, että PLC:hen on asennettu asiakkaan varmenne Trust Store -rekisteriin ja päinvastoin. Isossa tehtaassa, jossa kymmeniä laitteita keskustelee OPC UA:lla, manuaalinen varmenteiden hallinta voi olla työlästä ja altis virheille – jos jokin varmenne unohtuu asentaa, yhteys ei muodostu. Tähän OPC Foundation on kehittänyt Global Discovery Server -mekanismin, mutta sen käyttöönotto tuo lisäkompleksisuutta järjestelmään (Cisco n.d.). Usein kentällä nähdään, että OPC UA -yhteyksiä ajetaan ilman salausta ja autentikointia (Security Mode: None) käyttöönoton helpottamiseksi. Tämä kuitenkin altistaa järjestelmät hyökkäyksille.

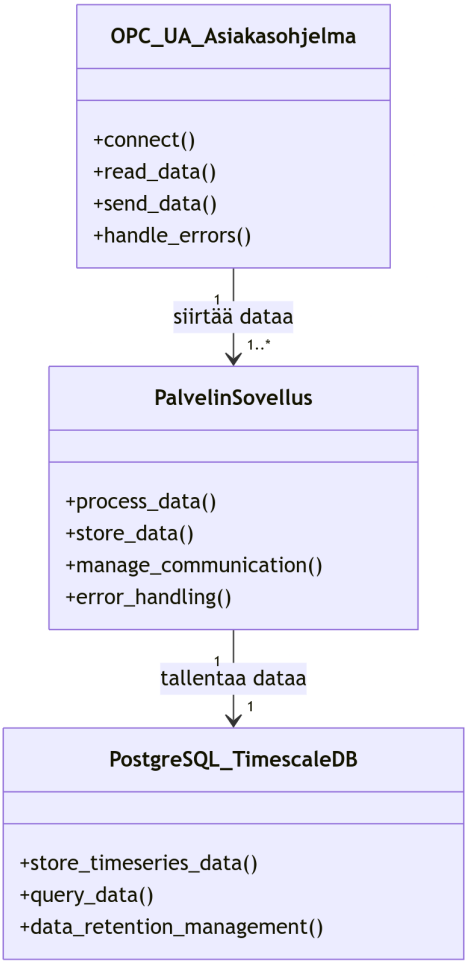
OPC UA -implementaatioista on myös löydetty haavoittuvuuksia vuosien varrella. Esimerkiksi Clarotyn Team82-tietoturvatiimi raportoi vuonna 2023 seitsemästä kriittisestä haavoittuvuudesta useiden valmistajien OPC UA -tuotteissa (Claroty Team82 2023). Nämä haavoittuvuudet vaihtelivat muistivuodoista autentikointiongelmiin ja saattoivat pahimmillaan mahdollistaa hyökkääjän suorittaa etäkoodia OPC UA -palvelimella. Vaikka valmistajat ovat julkaisseet paikkaukset, tapaus korostaa tarvetta pitää OPC UA -palvelimet ajan tasalla ja seurata tietoturvatiedotteita. Myös ns. tyypilliset IT-uhat koskevat OPC UA -ympäristöjä: man-in-the-middle-hyökkäykset, jos salaus ei ole päällä, palvelunestohyökkäykset lähettämällä ylisuuri määrä pyyntöjä, jne. Siksi OPC UA -verkko kannattaa eristää tuotantoverkkoon ja rajoittaa pääsy vain sallittuihin laitteisiin.

OPC UA on standardi, mutta siitä on useita versioita, joista 1.04 on yleinen, ja 1.03 ja 1.02 ovat yhä käytössä vanhemmissa laitteissa. Pääsääntöisesti versiot ovat taaksepäin yhteensopivia, mutta käytännössä on havaittu yhteensopivuusongelmia joidenkin asiakkaiden ja palvelinten välillä, erityisesti jos palvelimen ominaisuusjoukko poikkeaa. Testaus etukäteen on tärkeää monitoimittajaympäristössä. Onneksi OPC Foundationilta voi ladata Compliance Test Tool -työkalun ja sertifiointiohjelman, joka on parantanut tilannetta – sertifioidut OPC UA -laitteet toimivat yleensä ristiin ongelmitta.

3 Järjestelmän suunnittelu ja teknologiavalinnat

3.1 Kokonaisarkkitehtuuri

Tässä opinnäytetyön luvussa 3 käsitellään asiakasohjelmasta ja tietokantasovelluksesta koostuvan järjestelmän suunnittelua ja järjestelmän kehittämiselle valittuja teknologioita. Suunnitellun järjestelmän päätarkoituksena on kerätä reaaliaikaista dataa ohjelmoitavilta logiikoilta, käsitellä se ja tallentaa tehokkaasti relaatiotietokantaan, joka on optimoitu aikasarjadatan käsittelyyn. Järjestelmä rakentuu kolmesta pääkomponentista: OPC UA -asiakasohjelmasta, joka kommunikoi PLC-laitteiden kanssa, palvelimella ajettavasta sovelluksesta, joka vastaa datan esikäsittelystä ja siirrosta tietokantaan sekä PostgreSQL-pohjaisesta tietokannasta, johon on asennettu TimescaleDB-laajennus aikasarjatiedon tallennusta ja hallintaa varten (kuva 1). Kuvassa 1 on havainnollistettu järjestelmän arkkitehtuuri ja tiedon kulku komponenttien välillä.



Kuva 1: Suunniteltu järjestelmän rakenne toiminnoilla

Järjestelmäarkkitehtuuri noudattaa modulaarista rakennetta, jossa kukin komponentti on loogisesti erotettu omaksi kokonaisuudekseen. Asiakasohjelma huolehtii yksinomaan tiedon hakemisesta PLC:iltä OPC UA -rajapinnan yli, kun taas erillinen palvelinsovellusprosessi hoitaa datan puskuroinnin ja tallennuksen tietokantaan. Tällainen modulaarisuus mahdollistaa kokonaisuuden helpon laajentamisen, testaamisen ja ylläpidon. Tiedonkeruu toteutetaan asynkronisesti OPC UA -asiakasohjelman ja PLC-laitteiden välillä, mikä mahdollistaa tehokkaan ja resurssitehokkaan datavirran hallinnan teollisuusympäristössä (Mahmoud ym. 2020: s. 1). Aikasarjadatan tallennuksessa hyödynnetään TimescaleDB-laajennusta, joka mahdollistaa skaalautuvan tietorakenteen ja tehokkaat kyselytoiminnot kasvaville datamäärille (Freedman, Blackwood-Sewell 2025).

3.2 Teknologiavalintojen perustelu

Käytettävät teknologiat valittiin opinnäytetyön vaatimusten ja teoreettisen taustan (pääluku 2) perusteella. OPC UA valittiin tiedonsiirtoprotokollaksi PLC-laitteiden ja tietokeruusovelluksen välille sen valmistajariippumattomuuden ja laajan teollisen hyväksynnän vuoksi. OPC UA mahdollistaa standardoidun tavan hakea tietoja erilaisilta PLC-merkiltä, mikä oli tärkeää, jotta ratkaisu toimisi Metropolian automaatiolaboratorion monimuotoisessa laiteympäristössä. Lisäksi OPC UA:n sisäänrakennetut ominaisuudet, kuten tietoturva ja strukturoitu tietomalli, tukevat luotettavaa ja turvallista tiedonsiirtoa teollisuusympäristössä.

Ohjelmointikieleksi ja toteutusalustaksi valittiin Python sen nopean kehityssyklin ja laajan kirjastoekosysteemin vuoksi. Pythonille on saatavilla kypsä OPC UA -kirjasto (asyncua), joka tukee asynkronista toimintaa ja helpottaa useiden samanaikaisten PLC-yhteyksien hallintaa. Vaihtoehtoisia toteutusvaihtoehtoja olisivat voineet olla esimerkiksi C++ tai C# olemassaolevien OPC UA -kirjastojen kera, mutta Python nähtiin soveliaaksi prototyyppivaiheen toteutukseen helpoutensa ansiosta. Koska tiedonkeruusovellus ajetaan palvelinympäristössä eikä suoraan resurssikriittisellä sulautetulla alustalla, Pythonin suorituskyky on riittävä suunnitellulle näytetaajuudelle (sekuntitaso) ja datamäärille.

Tietokantateknologiaksi valittiin PostgreSQL relaatiotietokanta laajennettuna TimescaleDB-aikasarjalaajennuksella. Valinta perustui TimescaleDB:n kykyyn käsitellä suuria määriä aikaleimattua mittausdataa tehokkaasti sekä mahdollisuuteen tehdä ajallisesti optimoituja kyselyitä ilman, että kehittäjän tarvitsee luopua SQL-kielestä tai PostgreSQL:n tutuista ominaisuuksista. Vaihtoehtona harkittiin myös erillisiä aikasarjatietokantoja kuten InfluxDB, mutta TimescaleDB:n etuna on saumaton integraatio PostgreSQL:ään – olemassa olevat osaaminen ja työkalut ovat hyödynnettävissä suoraan. Lisäksi TimescaleDB tukee monipuolisia lisäominaisuuksia kuten continuous aggregates (jatkuvat aggregaattinäkymät) ja automatisoitu tietojen elinkaaren hallinta, jotka olivat hyödyllisiä tämän opinnäytetyön tarpeisiin. Valittu teknologiayhdistelmä OPC UA, Python ja Postg-

reSQL/TimescaleDB on avoimen lähdekoodin joustava ja skaalautuva ratkaisu PLC:n datankeruun toteuttamiseen.

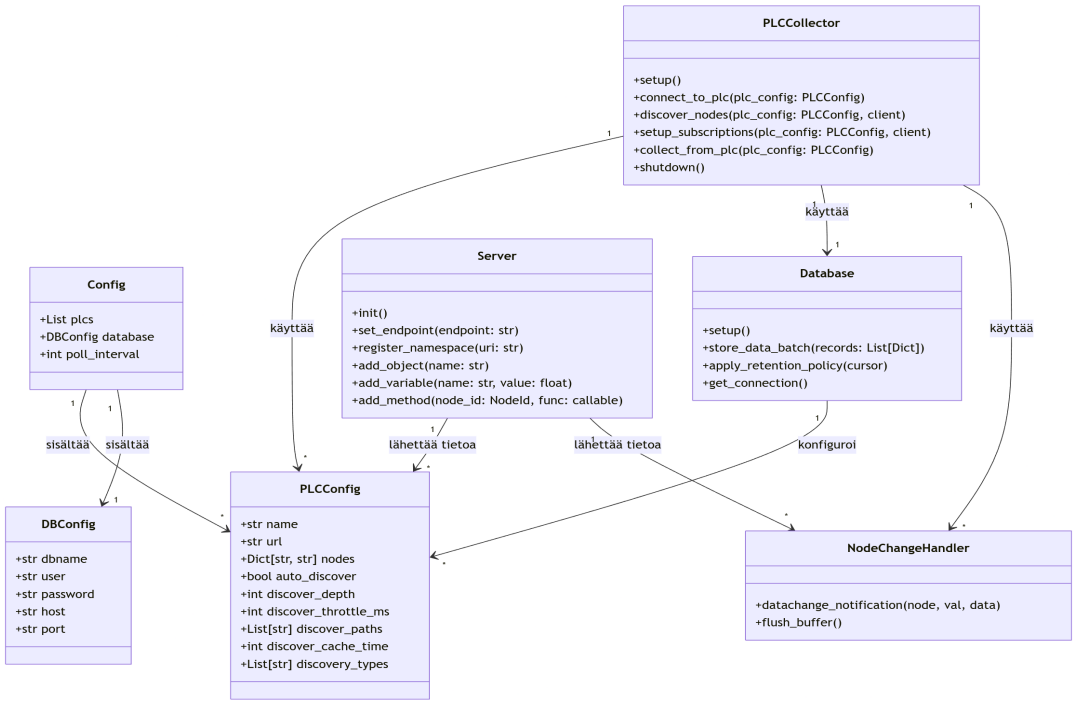
3.3 OPC UA -asiakasohjelman suunnittelu

OPC UA -asiakasohjelman suunnittelussa määriteltiin joukko keskeisiä toiminnallisia vaatimuksia. Näiden vaatimusten tarkoituksena on varmistaa, että järjestelmä täyttää teollisen ympäristön tarpeet ja on laajennettavissa tulevaisuudessa. Tärkeimmät toiminnalliset vaatimukset olivat:

- Järjestelmän tulee tukea useiden PLC-laitteiden samanaikaista tietojenkeruuta rinnakkain. Tämä tarkoittaa, että asiakasohjelma voi olla yhtä aikaa yhteydessä useaan OPC UA -palvelimeen ilman, että yhteydet häiritsevät toisiaan tai data menisi sekaisin.
- Järjestelmän on pystyttävä lukemaan ja tallentamaan mittausdataa vähintään sekuntien tarkkuudella. Datan tulee virrata tietokantaan lähes reaaliajassa, jotta nopeatkin muutokset saadaan talteen analysointia varten.
- Asiakasohjelman on toimittava erilaisten OPC UA -palvelimia tarjoavien PLC-merkkien kanssa. Sen tulee hyödyntää OPC UA -standardia siten, ettei ratkaisu rajoitu vain tiettyyn laitevalmistajaan, vaan toimii yleisesti standardin mukaisesti.
- Yhteyskatkosten tai virhetilanteiden varalta järjestelmän on kyettävä toipumaan automaattisesti. Jos yhteys PLC-laitteeseen katkeaa, asiakasohjelman tulee yrittää uudelleenyhdistämistä taustalla ilman käyttäjän väliintuloa ja jatkaa tiedonkeruuta, kun yhteys on palautunut.
- Järjestelmän muuttujat (kuten seurattavat solmut, näytteenottoväli ja palvelimen osoite) tulee määritellä helposti ulkoisen konfiguraatiotiedoston kautta. Käyttäjän pitää pystyä ottamaan järjestelmä käyttöön eri ympäristöissä muokkaamalla vain asetustiedostoa, eikä sovelluskoodiin tarvitse tehdä muutoksia perusasetusten takia.

Näiden vaatimusten täyttyminen luo pohjan joustavalle käyttöönotolle erilaisissa ympäristöissä ja mahdollistaa myöhemmän jatkokehityksen ilman merkittäviä muutoksia perusrakenteeseen. Vaatimukset huomioitiin suunnitteluvaiheessa määrittelemällä arkkitehtuuri ja komponentit siten, että edellä mainitut ominaisuudet on mahdollista toteuttaa.

OPC UA -asiakasohjelma tuotettiin olio-ohjelmointiperiaatteita noudattaen, mikä edistää koodin selkeyttä, komponenttien uudelleenkäytettävyyttä ja järjestelmän modulaarista laajentamista. Pääkomponenttina sovelluksessa oli PLCCollector-luokka, joka vastaa useiden PLC-yhteyksien hallinnoinnista ja tietovirran orkestroinnista (kuva 2). PLCCollector alustetaan keskeisillä konfigurointitiedoilla ja luo jokaiselle PLC:lle oman asynkronisen käsittelytehtävän. Näin useiden laitteiden käsittely samaan aikaan on toteutettu yhdellä ohjelmalla hyödyntämällä Pythonin await/async -rakenteita rinnakkaisuuden saavuttamiseksi.



Kuva 2: Järjestelmän luokkarakenne, jossa mukana testauksessa käytetty palvelin.

Jokaisen PLC-yhteyden sisällä PLCCollector-luokka käyttää OPC UA -kirjaston tarjoamia Client-olioita yhteyden muodostamiseen. Tietyn PLC:n mittauspisteiden, eli OPC UA -solmujen hallintaa varten on suunniteltu NodeChangeHandler-luokka, joka toimii OPC UA -tilausten (subscription) tapahtumankäsittelijänä. NodeChangeHandler ylläpitää kirjaa kyseisen PLC:n seurattavista solmuista ja niiden viimeksi tunnetuista arvoista. Luokka toteuttaa metodin datachange_noti-

fication, jota OPC UA -kirjasto kutsuu aina, kun jokin tilattu arvo muuttuu. Tämä tapahtumakäsittelijä lisää muuttuneet arvot puskuriin, kunnes kaikki data puskurissa tallennetaan tietokantaan. Muutokset suodatetaan ennen puskurointia metodilla should_record_value, joka tarkistaa onko muutos riittävän merkittävä, kun se esimerkiksi ylittää asetetun %-muutoskynnyksen tai absoluuttisen deadband-arvon jotta se kannattaa tallentaa. Näin vältetään turhien ja vähäisten vaihteluiden kirjaaminen ja säästetään tallennustilaa.

Asiakasohjelman luokkarakenne erottaa myös tietokantatoiminnot omaan Database-luokkaansa. Database-luokka on vastuussa PostgreSQL/TimescaleDB-yhteyden muodostamisesta ja sisältää metodit datan tallentamiseen sekä tietokantarakenteiden alustamiseen. Tietokantatoimintojen eriyttäminen mahdollistaa esimerkiksi toisenlaisen tallennusratkaisun käyttöönoton myöhemmin vaihtamalla vain Database-luokan toteutuksen, vaikuttamatta OPC UA -asiakaslogiikkaan. Vastaavasti konfiguraatiotiedoston lukeminen järjestelmän juurikansiosta on toteutettu omassa moduulissaan (config.py), jossa YAML-muotoinen asetustiedosto luetaan ja muunnetaan dataclass-pohjaisiksi konfiguraatio-olioiksi (Config, PLCConfig, DBConfig) ohjelman käyttöön. Eri osa-alueiden jakaminen luokkiin ja moduuleihin lisää järjestelmän laajennettavuutta: uusia ominaisuuksia, kuten vaikkapa tietojen edelleenlähetys toiseen järjestelmään, voidaan lisätä laajentamalla sopivaa moduulia ilman, että koko järjestelmän rakenne muuttuu.

Ohjelmistojärjestelmä on modulaarinen siten, että uusia PLC-laitteita tai mittauspisteitä voidaan lisätä pelkkiä konfiguraatiotiedostoja muokkaamalla, ilman koodausta. Myös arkkitehtuuri tukee mahdollista jatkokehitystä: esimerkiksi OPC UA -tilauskäsittelijää (NodeChangeHandler) voidaan laajentaa tukemaan monimutkaisempia päätöksentekomekanismeja, kuten hälytyslogiikkaa tai Database-luokkaa voidaan laajentaa toteuttamaan datan rinnakkaistallennus toiseen kohteeseen tietokannan ohella. Koska luokat on pidetty vastuuiltaan rajattuina, muutokset yhdessä komponentissa eivät vaadi suuria muutoksia muualla järjestelmässä. Tämä noudattaa yleisiä ohjelmistosuunnittelun periaatteita ja auttaa pitämään koodin ylläpidettävänä ja testattavana.

3.4 PostgreSQL-tietokantarakenteen suunnittelu

Järjestelmän tietokanta on toteutettu PostgreSQL-tietokantaan TimescaleDB-laajennuksella. Tietokantaan suunniteltiin kaksi päätaulua: plc_data ja plc_nodes. Näillä kahdella taululla erotellaan yksittäisten mittausarvojen tallennus ja laitekohtaisten solmujen säilytys.

Taulu plc_data on suunniteltu yksittäisten OPC UA -solmujen arvojen tallentamiseen riveittäin. Jokainen tietue sisältää vähintään seuraavat tiedot: laitteen tunniste (plc_name), mittauspisteen solmu-ID (node_id), solmun selväkielinen nimi (node_name), datatyyppi (data_type) sekä aikaleima (time). Itse mittausarvo talletetaan yhteen viidestä mahdollisesta sarakkeesta tyypin mukaan: numeeriset arvot sarakkeeseen value_number, totuusarvot sarakkeeseen value_boolean, tekstimuotoiset arvot sarakkeeseen value_string, aikaleimat sarakkeeseen value_timestamp ja rakenteelliset tai kompleksit arvot JSON-muodossa sarakkeeseen value_json. Kunkin rivin sarakkeista siis täyttyy vain yksi tai muutama arvon tyypistä riippuen, muut jäävät NULL-arvoiksi. Tällainen sarakekohtainen normalisoitu rakenne säilyttää arvon alkuperäisen tietotyypin ja mahdollistaa tietokannan suorituskyvyn optimoinnin erityisesti indekseille ja kyselyille. Stonebraker ja Çetintemel (2005) huomauttavat, että vahva tietotyypitys relaatiokannassa auttaa hyödyntämään kyselyoptimoijaa paremmin kuin vapaan rakenteen data, mikä tukee tätä suunnitteluratkaisua (Stonebraker & Çetintemel, 2005).

Alla on esitetty tiivistettynä plc_data-taulun esimerkki skeema:

```
CREATE TABLE plc_data (
     time           TIMESTAMPTZ        NOT NULL DEFAULT NOW(),
     plc_name       TEXT               NOT NULL,
     node_id        TEXT               NOT NULL,
     node_name      TEXT               NOT NULL,
     data_type      TEXT               NOT NULL,
     value_number   DOUBLE PRECISION   NULL,
     value_boolean       BOOLEAN       NULL,
     value_string        TEXT          NULL,
     value_timestamp     TIMESTAMPTZ   NULL,
     value_json          JSONB         NULL
);
```

Esimerkkikoodi 1: Tietokannan plc_data-taulun rakenne.

Aikaleimakenttä time toimii TimescaleDB:n hypertable-rakenteen aikajanan perustana, ja taulu muutetaan aikasarjoihin optimoiduksi hypertable:ksi (ks. alaluku 4.2) hyödyntämällä TimescaleDB-laajennusta. Taulu plc_data sisältää siis jokaisen mittausarvon omalla rivillään yhdessä selitteiden kanssa, mikä hieman lisää tallennustilan tarvetta, mutta toisaalta yksinkertaistaa kyselyitä ja tietojen käsittelyä huomattavasti.

Lisäksi tietokantaan on suunniteltu aputaulu plc_nodes, joka ylläpitää luetteloa tunnetuista PLC-laitteiden mittauspisteistä. Tähän tauluun tallennetaan muun muassa muuttujat plc_name, node_id, node_name, data_type sekä last_seen-aikaleima, joka kertoo, milloin kyseinen solmu viimeksi havaittiin datavirrassa. Plc_nodes-taulu on meta-tietotaulu: sen avulla voidaan nopeasti selvittää, mitä muuttujia kustakin laitteesta on saatavilla ja milloin niistä on viimeksi saatu tietoa. Taulussa on yhdistetty primääriavain (plc_name, node_id) estämässä duplikaattirivejä samalle muuttujalle. Tämä rakenne tukee erityisesti muuttujien automaattista löytämistä (autodiscovery) ja dokumentointia, koska kaikki löydetyt solmut kirjautuvat tähän tauluun automaattisesti.

Suurten datamäärien tehokas käsittely edellyttää huolellisesti suunniteltuja indeksejä ja tietojen elinkaaren hallintaa. plc_data-tauluun luodaan useita indeksejä yleisimpiä kyselytarpeita silmällä pitäen. Koska tyypilliset hakukyselyt kohdistuvat usein tietyn laitteen tai tietyn mittauspisteen tietoihin, tauluun luodaan indeksit ainakin sarakkeille plc_name ja node_id. Lisäksi data_type-sarakkeelle voidaan luoda indeksi, jos halutaan nopeuttaa hakuja tietyntyyppisistä arvoista, esimerkiksi kaikki numeeriset mittaukset tietyltä aikaväliltä. TimescaleDB:n hypertable hyödyntää aikaleimakenttää time automaattisesti osittaiseen indeksöintiin jakamalla datan aikaperusteisiin lohkoihin. Tässä opinnäytetyössä lohkokooksi määriteltiin yksi päivä, eli TimescaleDB jakaa plc_data-taulun datan päiväkohtaisiin paloihin. Tämä tarkoittaa, että kyselyt, jotka rajataan aikavälin perusteella, esimerkiksi viimeisen viikon data, kohdistuvat automaattisesti vain relevantteihin data-osioihin, mikä tehostaa hakua.

Tietojen elinkaaren hallinta (data retention) on tärkeää, jotta tietokannan koko ei kasva hallitsemattomasti ajan myötä. Suunnitellussa järjestelmässä on määritelty sääntö, jonka mukaan yli 90 päivää vanhat mittausdatat poistetaan tai arkistoidaan. TimescaleDB:ssä on tähän valmis toiminto drop_chunks, jolla voidaan poistaa koko tietokantataulun vanhimpia lohkoja tietyin aikavälein. Järjestelmään kehitettiin automaattinen mekanismi, joka kutsuu drop_chunks('plc_data', INTERVAL '90 days') -funktiota päivittäin, poistaen yli 90 päivän ikäiset datapalat tietokannasta. Näin tietokannan koko ei kasva liikaa pitkällä aikavälilläkään. Jos TimescaleDB:n toiminto ei olisi käytettävissä, järjestelmä käyttäisi perinteistä SQL-poistoa asetetun aikaleiman perusteella saman vaikutuksen saavuttamiseksi, eli esimerkiksi 90 päivää vanhempien rivien poistamiseksi.

TimescaleDB:n etuna on myös datan pakkausominaisuus (data compression), joka säästää tilaa tietokannassa pakkaamalla esimerkiksi yleisesti esiintyvät rivit pienempään tilaan. Järjestelmässä plc_data-taululle on määritelty segment-by -pohjainen pakkaus avainkenttien (plc_name, node_id) mukaan, ja TimescaleDB:hen konfiguroitiin pakkaussääntö, joka pakkaa yli seitsemän päivää vanhat lohkot automaattisesti. Tietokannan datan pakkaus pienentää vanhojen tietojen tilankäyttöä merkittävästi. Käytännössä tuorein viikko dataa pidetään pakkaamattomana nopeita reaaliaikaisia hakuja varten, ja vanhempi data pakataan tiiviimpään muotoon käyttäen TimescaleDB:n algoritmeja. Tietokannan taulujen indeksit toimivat edelleen myös pakatussa datassa, joskin pakatun datan haku saattaa vaatia purkamista hakua suorittaessa. Tästä huolimatta datan pakkaaminen on hyödyllistä, koska useimmiten vanhempiin tietoihin kohdistuu harvemmin kyselyitä.

Lopuksi järjestelmään suunniteltiin jatkuvat tietokannan aggregoinnit hyödyntävä materiaalinen näkymä (materialized view) plc_data_hourly. Tämä on TimescaleDB:n continuous aggregate -ominaisuudella toteutettu näkymä, joka laskee tunnin välein esimerkiksi keskiarvon, minimin, maksimin ja havaintomäärän kullekin muuttujalle kyseisen tunnin ajalta. plc_data_hourly päivittyy automaattisesti taustalla kun uutta dataa kertyy, määritellyn säännön mukaisesti. Tämän ansiosta esimerkiksi vuorokausitasoisten trendikäyrien piirtäminen käyttöliittymäs-

sä on kevyttä, kun aggregoinnit on jo valmiiksi laskettu tunnin resoluutiolla tietokantaan. Indeksoimalla plc_data_hourly-näkymän keskeiset kentät, kuten plc_name, node_id, saadaan myös pitkän aikavälin trendikyselyt suoritettua erittäin nopeasti.

Yhteenvetona, tietokantarakenne on suunniteltu siten, että se tukee sekä reaaliaikaista kirjoitusta että tehokasta suurten tietomäärien lukemista tietokannasta. Indeksien ja taulujen avulla mittausarvojen käsittelyn suorituskyky paranee, kun taas TimescaleDB huolehtii aikasarjadatan tehokkaasta säilömisestä ja datan hallinnasta automaattisesti.

4 Asiakasohjelman toteutus

4.1 OPC UA -asiakasohjelman toteutus

Tässä opinnäytetyön luvussa 4 esitellään PLC-dataa keräävän asiakasohjelman toteutusta. OPC UA -asiakasohjelman toteutusvaiheessa keskeinen osa oli luotettavan yhteyden muodostaminen PLC-laitteiden OPC UA -palvelimiin. Toteutuksessa hyödynnettiin asyncua-kirjastoa, joka mahdollistaa asynkronisen OPC UA -kommunikaation Pythonissa. Jokaiselle konfiguraatiossa määritellylle PLC-laitteelle ohjelma käynnistää oman asynkronisen tehtävän (asyncio.create_task), jossa suoritetaan yhteydenmuodostus ja datankeruu kyseiselle laitteelle samanaikaisesti muiden kanssa. Tämä mahdollistaa, että esimerkiksi viiden PLC:n dataa kerätään rinnakkain estämättä toisiaan.

Asiakasohjelman yhteyden muodostus PLC:lle on toteutettu PLCCollector.connect_to_plc-metodissa. Metodi yrittää luoda OPC UA -asiakasohjelman annetulle URL-osoitteelle ja muodostaa yhteyden palvelimeen konfiguroidulla aikakatkaisulla. Mikäli PLC:n OPC UA -palvelin vaatii todennusta, asiakasohjelma asettaa tarvittaessa käyttäjätunnuksen ja salasanan (client.set_user / set_password) ennen yhdistämistä palvelimelle. Samoin tuettiin OPC UA -yhteyden suojausasetuksia: konfiguraatiossa voidaan määritellä security policy ja security mode (Sign tai SignAndEncrypt). Toteutuksessa asiakasohjelmalle voidaan asettaa

haluttu suojaussääntö merkkijonolla (client.set_security_string()), mutta oletuksena (laboratorio-olosuhteissa) käytettiin salaamatonta yhteyttä, jotta sertifikaattien hallintaan ei tarvinnut paneutua. Ennen varsinaista yhdistämistä OPC UA -palvelimelle ohjelma asettaa asiakasohjelmalle myös yksilöivän sovellus-URI-tunnisteen (client.application_uri) tunnistamaan tämän asiakassovelluksen OPC UA -palvelimella. Tämä on OPC UA -protokollan piirre, joka mahdollistaa palvelimen lokien ja istuntojen hallinnan asiakaskohtaisesti.

Palvelimelle yhdistämisessä on huomioitu uudelleenyrittäminen ja aikakatkaisut: connect_to_plc yrittää oletuksena viisi kertaa muodostaa yhteyden ennen luovuttamista. Yritysten välissä on viive, jota kasvatetaan portaittain epäonnistumisten jatkuessa, kuitenkaan ylittämättä asetettua maksimi-intervalia. Tämä eksponentiaalinen backoff-mekanismi ehkäisee palvelimen ylikuormitusta jatkuvalla yritystulvalla, jos esimerkiksi PLC on pois päältä. Mikäli kaikki yritykset epäonnistuvat, tilanne kirjataan virhelokiin ja kyseisen PLC:n osalta odotetaan hetki ennen kuin yritetään alusta uudelleen. Tämä sykli jatkuu koko sovelluksen suoritusajan taustalla, joten jos PLC tulee takaisin verkkoon, ohjelma havaitsee sen seuraavalla yhteysyrityksellä ja jatkaa normaalisti. Kun OPC UA -yhteys on saatu auki, asiakasohjelma kirjaa lokiin yhteyden onnistumisesta ja siirtyy datankeruuseen. Yhteys pidetään avoinna jatkuvana OPC UA -sessiona; Asyncua-kirjasto huolehtii alustan puolella OPC UA -pysähtymisten ja keep-alive -viestien hallinnasta. Mikäli yhteys katkeaa odottamattomasti, se havaitaan joko seuraavassa tietoluvussa (pollaus) tai tilausten yhteydessä (subscription keep-alive) ja käsitellään virhetilanteiden hallinnan mukaisesti.

Tiedonkeruussa hyödynnetään OPC UA -protokollan kahta tapaa lukea dataa: tilaaminen (subscription) ja kysely/pollaus (read). Ensisijaisena menetelmänä käytetään tilaamista, jolloin PLC:n OPC UA -palvelin lähettää automaattisesti ilmoituksen aina, kun jokin seurattu arvo muuttuu. Toteutuksessa PLCCollector-luokka luo onnistuneen yhteyden jälkeen kullekin PLC:lle OPC UA -tilauksen (create_subscription) tietyllä päivitysintervallilla. Samalla luodaan aiemmin mainittu NodeChangeHandler-olio, joka rekisteröidään tilausten tapahtumankäsittelijäksi. Tämän jälkeen ohjelma käy läpi konfiguraatiossa määritellyt OPC UA -

solmu-ID:t kyseiselle PLC:lle ja lisää tilaukseen jokaisen solmun data-change -kuuntelun (subscribe_data_change(node)). Jos tilaus kaikkiin haluttuihin solmuihin onnistuu, siirtyy ohjelma odottamaan muutostapahtumia.

NodeChangeHandler.datachange_notification -metodi kutsutaan jokaisesta saapuvasta arvonmuutosilmoituksesta. Handleri toteuttaa logiikan, jossa se ensin muuntaa OPC UA -palvelimen antaman node-olion tunnisteen merkkijonoformaattiin ja hakee solmun nimen node_names-sanakirjasta. Tämän jälkeen tarkastetaan should_record_value-metodilla, tulisiko kyseinen muutos tallettaa: käytössä on ns. deadband-suodatus sekä suhteelliselle että absoluuttiselle muutokselle. Oletuksena järjestelmä on asetettu tallentamaan arvon, jos muutos on vähintään 1 % edellisestä arvosta, tai vaihtoehtoisesti jos absoluuttinen muutos ylittää asetetun kynnyksen (deadband). Mikäli muutos on merkittävä, handleri lisää uuden tietueen sisäiseen puskuriin. Puskuri on listarakenteinen ja sisältää sanakirjoja, joissa on kentät plc_name, node_id, node_name, value ja timestamp. Aikaleima otetaan OPC UA -ilmoituksesta: käytetään SourceTimestamp-arvoa, joka on PLC:n päässä leimattu ajanhetki, tai jos sitä ei ole, ServerTimestamp-arvoa, eli OPC UA -palvelimen aikaleimaa. Näin varmistetaan, että tietokantaan tallentuva aikaleima vastaa mahdollisimman tarkasti tapahtumahetkeä PLC:llä.

Puskurointi mahdollistaa usean arvon kirjoittamisen tietokantaan kootusti. NodeChangeHandler seuraa puskurin kokoa ja viimeisintä tietokantakirjoitusta, ja toteuttaa metodin flush_buffer, joka kutsutaan joko kun puskuriin on kertynyt tietty määrä tietueita tai tietty aika on kulunut edellisestä tallennuksesta. flush_buffer metodi siirtää puskurin tietueet tietokantaan kutsumalla Database.store_data_batch -metodia ja tyhjentää puskurin onnistuneen tallennuksen jälkeen. Batch-tyyppinen tallennus on huomattavasti tehokkaampaa kuin yksittäisten rivien lisäys, koska tietokantaan tehdään vain yksi INSERT-kysely, joka sisältää kaikki puskurin tiedot. Tämä vähentää verkko- ja transaktio-overheadia ja parantaa suorituskykyä.

Kaikki PLC:t eivät välttämättä tue tilaustoiminnallisuutta, tai tilausten käyttö saattaa olla epäluotettavaa tietyissä olosuhteissa. Tämän vuoksi järjestelmään toteutettiin varmistuksena vaihtoehtoinen lukumekanismi, pollaus. Jos tilauksen luominen epäonnistuu tai heittää poikkeuksen, asiakasohjelma kirjaa varoituksen ja siirtyy käyttämään kyseiselle PLC:lle jatkuvaa kyselysilmukkaa (polling). Pollauksessa ohjelma lukee jokaisella kierroksella peräkkäin kaikki konfiguroidut solmut OPC UA -metodilla read_data_value(). Tulokset kerätään listaksi ja lopuksi tallennetaan tietokantaan yhdellä kutsulla samaan tapaan kuin tilaustenkin kanssa. Tämän jälkeen prosessi odottaa määritellyn poll_interval-ajan ennen seuraavaa kierrosta. Pollauksen etuna on yksinkertaisuus ja varmuus siitä, että arvot saadaan tietyin väliajoin, mutta haittapuolena on se, että myös muuttumattomia arvoja luetaan ja verkkoa sekä PLC:tä kuormitetaan säännöllisesti. Tästä syystä pollaus on käytössä vain varajärjestelmänä tilanteissa, joissa subscription ei ole käytettävissä.

PLC-datankeruun toteutus sisältää myös automaattisen muuttujien löytämisen (autodiscovery) tuen. Jos PLC:n konfiguraatiossa on asetettu auto_discover: true, järjestelmä yrittää aluksi selata (browse) OPC UA -palvelimen osoiterakennetta löytääkseen kaikki kiinnostavat muuttujat. Toteutus käy läpi määritellyt aloituspolut tietyllä syvyydellä ja kerää löydetyt muuttujasolmut talteen. Selainalgoritmi pyrkii rajaamaan haun vain muuttujasolmuihin (NodeClass Variable) ja lukea niiden nimet, tunnisteet ja datatyypit. Mikäli autolöydetyille solmuille ei ole suoraan tukea tilauksissa, ne voidaan lisätä dynaamisesti seurantaan. Tässä projektissa autodiscovery-ominaisuus toteutettiin ensisijaisesti dokumentointitarkoituksiin: se kirjaa kaikki löydetyt muuttujat lokiin ja tallentaa ne plc_nodes-tauluun meta-tietoina. Jatkokehityksessä ominaisuutta voisi laajentaa niin, että ohjelma alkaisi automaattisesti myös kerätä dataa kaikista löydetyistä muuttujista. Nyt kuitenkin varsinainen datankeruu tapahtuu niille solmuille, jotka on eksplisiittisesti määritelty konfiguraatiossa, tai jotka käyttäjä lisää config-tiedostoon autodiscover-lokin perusteella.

4.2 Tietokannan toteutus

Tietokannan toteutusvaihe aloitettiin asentamalla TimescaleDB-laajennus PostgreSQL-tietokantapalvelimeen. Ennen TimescaleDB-ominaisuuksien käyttöä tietokantaan aktivoitiin laajennus komennolla CREATE EXTENSION IF NOT EXISTS timescaledb;. Tämä mahdollistaa TimescaleDB:n omien funktioiden ja optimointien hyödyntämisen tietokantakyselyissä.

Sovelluksen Database-luokka on suunniteltu alustamaan tietokantarakenne automaattisesti ohjelman käynnistyessä, joten erillisiä SQL-skriptejä ei tarvinnut ajaa manuaalisesti. Konfiguraatiossa määriteltiin tietokantapalvelimen osoite, tietokannan nimi ja käyttäjätunnus sekä salasana, jotka sovellus lukee käynnistyessään. Yhteyksien hallintaa varten luotiin connection pool (yhteysallas) psycopg2-kirjaston ThreadedConnectionPool-luokalla, minimi- ja maksimiyhteysmäärällä. Yhteysallas mahdollistaa sen, että useampi rinnakkainen asynkroninen tehtävä voi samanaikaisesti suorittaa tietokantakirjoituksia suorituskyvyn kärsimättä. Yhteysallas huolehtii yhteyksien uudelleenkäytöstä ja turvallisesta jaosta säikeiden kesken.

Tietokantapalvelimen konfiguraatiossa (postgresql.conf) ei tarvittu merkittäviä erikoissäätöjä tavanomaisten asetusten lisäksi. Joitakin optimointeja kuitenkin tehtiin: max_connections arvoa nostettiin hieman, jotta usean yhteysaltaan rinnakkaiskäyttö sallitaan. Lisäksi TimescaleDB:n taustatehtävien määrän varmistettiin kattavan continuous aggregate -näkymän päivityksen ja muut aikataulutetut tehtävät. Testiolosuhteissa resurssit, kuten muisti ja CPU eivät olleet rajoittavia tekijöitä, joten vakioasetuksilla pärjättiin pitkälti.

PostgreSQL:n puolella varmistettiin, että shared_buffers, work_mem ja maintenance_work_mem -arvot olivat riittävät käsittelemään aikasarjakyselyt ja indeksien luomiset tehokkaasti. shared_buffers asetettiin ~25% fyysisestä muistista ja work_mem muutamaan megatavuun per yhteys. Koska TimescaleDB:n continuous aggregate -prosessointi voi vaatia laskentaa, maintenance_work_mem

pidettiin oletusta korkeampana, 256MB:n kohdalla, jottei aggregointinäkymän päivittäminen hidastu muistirajoitteisiin.

Sovelluksen tietokannalle asetettiin myös TimescaleDB:n telemetria-asetus ”off”-tilaan asetuksella timescaledb.telemetry_level=off, koska kyseessä on suljetun verkon asennus eikä haluttu lähettää käyttödataa ulospäin. Tämä ei sinänsä vaikuta tietokannan toimintaan, mutta telemetrian estäminen on hyvä käytäntö tuotantoympäristössä ottaa huomioon.

Toteutettu Database.setup()-metodi huolehtii tarvittavien taulujen ja indeksien luomisesta sovelluksen käynnistyessä ensimmäistä kertaa. Tämä metodi suorittaa joukon SQL-käskyjä käyttäen psycopg2-kirjaston kursoria. Ensiksi luodaan plc_data-taulu, ellei sitä jo ole, määritellyillä sarakkeilla (ks. alaluku 3.4). Tämän jälkeen kutsutaan edellä mainittu create_hypertable muuttamaan taulun rakenteen TimescaleDB:n hallinnoimaksi. Mahdolliset virheet kirjataan varoituksena lokiin, mutta eivät estä sovelluksen toimintaa – jos Timescalea ei olisi, taulu jäisi tavalliseksi PostgreSQL-tauluksi ja toimisi pienillä datamäärillä silloinkin.

Seuraavaksi asetetaan tietokannan sarakkeille indeksit: idx_plc_data_plc_name laitteen nimelle, idx_plc_data_node_id solmu-ID:lle ja idx_plc_data_type datatyypille. Nämä toteutettiin IF NOT EXISTS -ehdolla, jotta samaa indeksiä ei yritetä luoda uudelleen jokaisella käynnistyksellä. Indeksien luonti alussa on kevyt operaatio, koska taulu on tyhjä – jatkossa, jos taulu on jo täynnä dataa, indeksit ovat olemassa ja vain mahdolliset puuttuvat indeksit luotaisiin.

Datan pakkaus otettiin käyttöön plc_data-taululle asettamalla TimescaleDB:n tauluasetuksia: ALTER TABLE plc_data SET (timescaledb.compress, timescaledb.compress_segmentby = 'plc_name, node_id');. Tämä kertoo Timescalelle, että kyseisen hypertablen data voidaan pakata segmentoiden se plc_name ja node_id -kenttien mukaan, eli käytännössä kunkin laitteen kunkin mittauspisteen aikasarja pakataan erikseen. Tällöin esimerkiksi yhden anturin kaikki vanhat arvot voivat tiivistyä erittäin hyvin, koska peräkkäiset arvot ovat usein samantyyppisiä (pienten vaihteluiden aikasarja). Pakkaus ei tapahdu automaatti-

sesti ilman käskyä, joten seuraavaksi lisättiin pakkauskäytäntö: SELECT add_compression_policy ... pakkauskäytäntö plc_data-taululle: komennolla SELECT add_compression_policy('plc_data', INTERVAL '7 days');määritettiin, että yli viikon ikäiset dataosat pakkautuvat automaattisesti taustaprosessina. Tämä optimoi tallennustilan käyttöä pitkällä aikavälillä. Lisäksi Database.setup()` lisää plc_nodes-taulun, jos sellaista ei ole, kirjaamaan ylös tunnetut mittauspisteet. Lopuksi luodaan plc_data_hourly-materiaalinäkymä jatkuvia aggregaatteja varten sekä asetetaan sen automaattinen päivityskäytäntö TimescaleDB:n funktioilla.

Tietojen elinkaaren hallinta kehitettiin myös osittain tietokantatason politiikalla ja osittain sovellustasolla. TimescaleDB:n tietojen elinkaaren hallintaan valittiin drop_chunks-käytäntö. Elinkaaren hallinta on myös mahdollista suorittaa poistamalla vanhat rivit tai lohkot manuaalisesti. Tässä toteutuksessa Database-luokka sisältää metodin _apply_retention_policy, joka yrittää pudottaa yli 90 päivän ikäiset lohkot Timescale-funktiolla (drop_chunks('plc_data', INTERVAL '90 days')) aina vuorokauden välein. Jos TimescaleDB ei ole käytettävissä, koodi käyttää varapoistona tavallista DELETE-kyselyä aikaleiman perusteella. Tämä varmistaa, että tietokanta ei kasva rajatta ja vanhentunut data poistuu. Retention-tarkistus ajoitetaan sovelluksessa siten, että jokaisen tietokantakirjoituksen yhteydessä tarkistetaan viimeisimmästä poistosta kulunut aika; jos yli 24 tuntia, kutsutaan poistometodia taustalla ennen uusien kirjoitusten jatkamista. Näin elinkaaren hallinta tapahtuu dynaamisesti sovelluksen ollessa käynnissä, eikä se vaadi erillisiä ajoitettuja tietokantatoimenpiteitä.

4.3 Asiakasohjelman ja tietokantapalvelimen integraatio

Järjestelmän eri komponenttien integraatiossa keskeistä on saumaton tiedonsiirto OPC UA -asiakasohjelmasta tietokantaan. Toteutus on järjestetty siten, että NodeChangeHandler toimii linkkinä OPC UA:n ja tietokannan välillä. Kun OPC UA -tilaus havaitsee uuden arvon, datachange_notification kerää datan puskuriin ja flush_buffer-metodi huolehtii puskurin tyhjentämisestä tietokantaan

säännöllisesti. Tämä mekanismi erottelee reaaliaikaisen saapuvan datavirran ja tietokantakirjoitukset siten, että piikkimäiset nopeat tapahtumat voidaan tasoittaa ennen tallennusta.

Database.store_data_batch(records)-metodi ottaa vastaan joukon mittaustietueita ja muodostaa niistä yhden SQL-lauseen usean rivin INSERT-toimitukseen. Koodi rakentaa dynaamisesti VALUES-listan kaikille puskurissa oleville tietueille ja liittää parametrien arvot peräkkäin parametrilistaan, jotta kysely voidaan suorittaa yhdellä kutsulla. Ennen rivien lisäämistä suoritetaan myös metatietojen päivitys plc_nodes-tauluun: jokaisen tietueen osalta lisätään tai päivitetään laitteen kyseisen mittauspisteen nimi ja viimeksi nähty aika. Tämä takaa, että plc_nodes pysyy ajan tasalla järjestelmän tuntemista solmuista automaattisesti.

Kun SQL-kysely on muodostettu, se suoritetaan psycopg2-kirjaston kursoriolion avulla. Koska yhteydet on avattu autocommit-tilassa tai koska ne suljetaan lohkon lopussa, jokainen store_data_batch suorittaa transaktion, jossa kaikki puskurin arvot tallentuvat kerralla. Mikäli kysely epäonnistuu, se heittää poikkeuksen, joka napataan flush_buffer-metodissa ja kirjataan virheeksi lokiin. Tällöin kyseiset arvot voivat jäädä puskuriin, mutta virheen toistuessa data voisi potentiaalisesti kadota. Tätä voisi jatkokehittää esimerkiksi siten, että epäonnistuneet tietueet säilytetään ja yritetään uudelleen myöhemmin.

Tiedonsiirrossa on tärkeää huomata, että OPC UA -asiakas toimii täysin asynkronisesti: se ei jää odottamaan tietokantakirjoituksen valmistumista, vaan puskurin tyhjennys tapahtuu taustalla nopeasti ja asiakas jatkaa seuraavien arvojen vastaanottamista. Psycopg2:ta käytetään synkronisesti kussakin asyncio-tehtävässä, mutta koska kyseinen tietokantakutsu on hyvin nopea optimoidun moniarvolisäyksen vuoksi, se ei muodostu pullonkaulaksi. Testeissä havaittiinkin, että tietokantakirjoitus pysyy vaivatta perässä, kun arvoja tulee muutamia kymmeniä sekunnissa. Mikäli datavirta kasvaisi satoihin viivoihin sekunnissa, voitaisiin harkita event-driven lähestymistapaa, jossa tietokantakirjoitukset delegoitaisiin erilliselle työjonolle tai prosessille.

Järjestelmän integroidessa OPC UA -liikennettä ja tietokantaa varmistettiin, että tiedot tallentuvat oikein. Testidatana ajettiin simuloitua dataa käyttämällä FreeOpcUa:n omaa palvelinta, kuten config.yaml:ssa esimerkkinä opc.tcp://localhost:4840/freeopcua/server/. Molemmissa tapauksissa varmistettiin, että tietueet ilmestyvät plc_data-tauluun odotetusti, oikeilla plc_name-, node_id-, value- ja time-kenttien arvoilla. Myös data_type kirjautui oikein. Yhteys OPC UA -datasta aina tietokantaan on siis automatisoitu ja tapahtuu keskeytyksettä sovelluksen pyöriessä.

Järjestelmän konfiguroitavuus toteutettiin keskitetyllä YAML-muotoisella asetustiedostolla. Tämän tiedoston avulla käyttäjä voi määrittää uuden laitteen lisäämisen, solmujen tunnisteet, näytteenottotaajuudet ja muut parametrit ilman ohjelmakoodin muokkausta. Sovellus lukee konfiguraation käynnistyessään load_config-funktiolla, joka muodostaa Config-olion dataluokkien avulla. Näin saadaan tyypitetty oliorakenne ohjelman käyttöön.

Konfiguraatiossa on oma lohkonsa jokaiselle PLC-laitteelle plcs: listassa. Esimerkiksi plc1 voidaan määritellä seuraavasti:

```
plcs:
  - name: plc1
     url: "opc.tcp://localhost:4840/freeopcua/server/"
     auto_discover: true
     discover_depth: 2
     discover_throttle_ms: 100
     discover_paths:
       - "/Objects"
     # ... mahdolliset tietoturva-asetukset ...
     username: ""
     password: ""
     # Tilauksen päivitysnopeus
     subscription_interval: 500   # ms
     # Puskurin asetukset
     buffer_size: 100             # maksimi tietueiden määrä ennen kir-
joitusta
     buffer_flush_interval: 5     # sekuntia puskurin tyhjennyksessä
     # Arvon tallennusrajat
     value_deadband_percent: 1.0  # tallennus, jos muutos yli 1%
     value_deadband_absolute: 0.0 # absoluuttinen deadband
```

Esimerkkikoodi 2: Osa config.yaml konfiguraatiosta.

Yllä oleva esimerkki osoittaa, kuinka monipuolisesti järjestelmän käyttäytymistä voi säätää. subscription_interval määrittää OPC UA -tilauksen julkaisuvälin (publish interval) millisekunteina; tässä 500 ms tarkoittaa että PLC-palvelin lähettää muutostiedot enintään 0,5 s välein. buffer_size ja buffer_flush_interval säätävät puskurointia, kuten luvussa 4.1.2 käsiteltiin. value_deadband_percent ja value_deadband_absolute määräävät, milloin arvon muutos on riittävän suuri tallennettavaksi. Esimerkiksi yllä, jos arvo muuttuu yli 1% edellisestä, se kirjataan.

Konfiguraatiotiedostosta luetaan myös tietokantayhteyden tiedot (dbname, user, password, host, port), jotka asetetaan DBConfig-olioon. Tämän ansiosta mitään yhteysosoitteita tai salasanoja ei ole kovakoodattu ohjelmaan, mikä parantaa tietoturvaa ja ylläpidettävyyttä. Ympäristökohtaiset erot voidaan siis hallita konfiguraatiolla: esimerkiksi testiympäristössä käytetään paikallista tietokantaa ja PLC-simulaattoria, kun taas tuotantoon siirryttäessä config-tiedostoon vaihdetaan oikean palvelimen osoitteet.

Järjestelmän parametrisointi on joustavaa myös laitelisäysten osalta. Uuden PLC:n lisääminen onnistuu kopioimalla config-tiedostoon uusi lohko, antamalla sille ainutkertainen nimimuuttuja name ja oikea osoitemuuttuja url sekä lista halutuista solmuista. Mikäli nodes-listaa ei anneta ja auto_discover on päällä, ohjelma yrittää löytää solmut automaattisesti. Usein on kuitenkin käytännöllistä määritellä eksplisiittisesti seurattavat node-id:t ja niille selkokieliset nimet, jotta dataan tallentuu heti kuvaava nimi.

On syytä huomioida, että tällä hetkellä konfiguraatio luetaan vain käynnistyessä. Jos muutoksia halutaan ohjelman suorituksen aikana, ohjelma täytyy käynnistää uudelleen. Jatkokehityksessä voisi harkita dynaamista asiakasohjelman konfiguraation päivitystä, mutta opinnäytetyön laajuudessa konfiguraatiotiedoston staattinen luenta riitti.

5 Järjestelmän testaus

5.1 Järjestelmän testausmenetelmät ja -ympäristö

Tässä opinnäytetyön luvussa 5 käsitellään kehitetyn PLC:n datankeruuohjelmiston testausta. Ohjelmiston testaus toteutettiin simuloidussa ympäristöissä. Aluksi kehitysvaiheessa hyödynnettiin yksikkötestejä keskeisille komponenttifunktioille, kuten datan puskuroinnille ja tietokantakirjoitukselle. Esimerkiksi should_record_value-metodia testattiin erikseen useilla arvoilla varmistamaan, että deadband-logiikka toimii oikein – odotetusti pieni muutos palautti False ja riittävän suuri muutos True. Samoin store_data_batch-toiminnallisuutta testattiin erillisellä testitietokannalla, johon lisättiin muutama rivi ja varmistettiin, että tulokset tallentuivat oikein sekä että kaksoiskappaleiden hallinta plc_nodes-taulussa toimi.

Testausmenetelmät sisälsivät myös häiriötestausta: pysäytettiin OPC UA -asiakasohjelma ja simuloitu laite, testataksemme kuinka hyvin ohjelma reagoi ja kirjaa tilanteet lokitiedostoon. Jokaisen testiskenaarion jälkeen ohjelman lokitiedosto ja tietokannan sisältö analysoitiin, jotta varmistuttiin tavoitteiden täyttymisestä.

5.2 Suorituskykytestit

Reaaliaikaisen tiedonkeruun nopeutta mitattiin kahdella mittarilla: kuinka tiheästi dataa pystyttiin lukemaan ja tallentamaan sekä kuinka luotettavasti jokainen muutos kirjautui. Testissä asetettiin simuloitu PLC tuottamaan dataa mahdollisimman nopeasti – käytännössä OPC UA -palvelimelle konfiguroitiin muuttuja, joka päivittyy jatkuvasti. Asiakasohjelman tilausten publishing interval asetettiin 100 ms:iin ja deadband poistettiin käytöstä. Tällöin järjestelmä yritti kirjata jopa 10 arvoa sekunnissa tietokantaan.

Toisessa testissä otettiin mukaan useampi PLC-lähde samanaikaisesti. Molempiin syötettiin dataa vastaavaan tahtiin. Asiakasohjelman configiin lisättiin kaksi PLC:tä omilla nimillään, ja ohjelma käynnistettiin. Tällöin PLCCollector loi kaksi asynkronista tehtävää. Havainnot osoittivat, että ohjelma käsitteli molempia lähes yhtä sujuvasti: lokissa vuorottelivat viestit plc1:stä ja plc2:sta, ja tietokannan plc_data-tauluun kertyi dataa kummallekin plc_name:lle suunnilleen yhtä aikaa. Maksimissaan kirjattiin noin 20 riviä sekunnissa, eikä viivettä tietokantaan kirjoittamiseen kertynyt merkittävästi.

6 Opinnäytetyön tulokset ja johtopäätökset

6.1 Opinnäytetyön tulosten arviointi

Tässä opinnäytetyössä tarkasteltiin asiakasohjelman toteuttamista PLC-datan keräämiseen. Opinnäytetyössä käytettiin lähteinä muun muassa standardeja, akateemisia julkaisuja ja teknologioiden dokumentaatiota. Opinnäytetyön kehittämisprosessissa ohjelmoitiin Python-kielellä PLC-dataa keräävä asiakasohjelmasta ja tietokannasta koostuva ohjelmistojärjestelmä, jonka tavoitteena oli kyetä kommunikoimaan usean PLC:n kanssa samanaikaisesti, kerätä dataa PLC:ltä reaaliaikaisesti ja säilyttää dataa tietokannassa tehokkaasti.

Tämän opinnäytetyön kehittämisprosessissa saavutettiin asetetut tavoitteet, joita olivat: OPC UA -asiakasohjelman toteutus ja sen ominaisuus lukea reaaliaikaista dataa useilta PLC-laitteilta rinnakkain tietokantaan analysoitavaksi. Järjestelmäarkkitehtuuri osoittautui toimivaksi: modulaarinen kolmijaottelu (OPC UA -asiakasohjelma, datansiirtosovellus, tietokanta) helpotti sekä ohjelmistokokonaisuuden toteutusta että testausvaihetta.

Opinnäytetyön kehittämisprosessissa havaittiin, että järjestelmä pystyy käsittelemään datavirtaa nopeasti. Tavoite useiden samanaikaisten PLC-yhteyksien

tuesta toteutui: testit osoittivat, että järjestelmä hallitsee rinnakkaisesti monen laitteen yhteyksiä ja dataa pääasiallisesti ilman PLC-yhteyksien konflikteja.

Yhteensopivuutta eri laiteympäristöihin ei saatu testattua täydellisesti. Järjestelmää testattiin simuloimalla OPC UA -palvelinta Pythonilla, mutta ei lopullisessa käyttöympäristössä palvelimella Metropolian Myyrmäen kampuksella. Virheiden ja katkoksien hallinta toimi käytännössä kuten suunniteltu: uudelleen yhdistämisen mekanismit havaitsivat laitekatkokset ja yhdistivät uudelleen automaattisesti, ja järjestelmä jatkoi keruuta normaalisti häiriön jälkeen. Ohjelmistokokonaisuuden testeissä ei havaittu tilannetta, jossa ohjelma olisi jumiutunut tai pysähtynyt yrittämättä toipua. Jokainen virhe joko ratkaistiin tai kirjattiin lokitiedostoon ja ohitettiin niin, että asiakasohjelman pääsilmukka pysyi käynnissä.

Asiakasohjelman konfiguroitavuuden tavoite täyttyi erinomaisesti. Ohjelman käyttöönotossa konfiguraatiotiedostoa muokkaamalla voitiin lisätä uusia laitteita ja mittauspisteitä nopeasti ohjelman uudelleenkäynnistyksellä ilman, että lähdekoodia tarvitsee muuttaa. Tämä osoitti, että suunniteltu asiakasohjelman parametrisointi on onnistunut ja järjestelmä on siirrettävissä eri ympäristöihin helposti. Opinnäytetyön prosessin aikana asiakasohjelmaa kokeiltiin vain simuloidussa ympäristössä, mutta on oletettavaa, että vastaavalla konfiguraation muokkauksella se toimisi myös oikeassa käyttöympäristössä, esimerkiksi Metropolian automaatiolaboratoriossa, kunhan OPC UA -yhteydet ovat saatavilla.

6.2 Johtopäätökset

Opinnäytetyön lopputuloksena kehitettiin PLC-tiedonkeruuseen sopiva ratkaisu, joka hyödyntää moderneja teollisuusstandardeja ja avoimen lähdekoodin tietokantateknologiaa reaaliaikaisen datankeruun tarpeisiin. Järjestelmä saavuttti sille asetetut tavoitteet: se on luotettava, skaalautuva ja riittävän reaaliaikainen Metropolian automaatiolaboratorion käyttötarkoituksiin. Opinnäytetyössä yhdistyivät automaatiotekniikan ja tietotekniikan osa-alueet. OPC UA mahdollisti laitelähеisen datan standardoidun keräämisen ja TimescaleDB mahdollisti datan tehokkaan tallennuksen ja käsittelyn.

Merkittävä havainto OPC UA -asiakasohjelman kehityksessä oli, että suhteellisen pienellä määrällä Python-koodia voitiin hyödyntää valmiita Python-kirjastoja ja tietokantatoiminnallisuuksia näin laajan järjestelmän rakentamisessa. Tämä korostaa avoimien standardien ja työkalujen hyötyä: OPC UA -rajapinnan käyttöönotto säästää aikaa verrattuna valmistajakohtaisten protokollien integrointiin. Ohjelmistojärjestelmän suorituskyky riitti ja muistin sekä prosessorin käyttö pysyi ohjelmaa suorittavalla tietokoneella maltillisena testatuissa skenaarioissa. TimescaleDB:ssä on ominaisuuksia, joita voitiin käyttää automaattiseen datan arkistointiin ja aggregointiin (Timescale 2024a; Timescale 2024b). TimescaleDB:n ominaisuudet toimivat opinnäytetyön ohjelmistojärjestelmässä odotetusti ja vähentävät tietokannan manuaalisen ylläpidon tarvetta.

Tulevaisuudessa opinnäytetyössä toteutettua ohjelmistojärjestelmää voidaan laajentaa ja kehittää edelleen. Yksi jatkokehitysidea on OPC UA -standardiin kuuluvan Pub/Sub-toiminnallisuuden hyödyntäminen: julkaisu-tilaus mallilla voisi vähentää viiveitä ja verkon kuormaa entisestään suurissa mittausjärjestelmissä. Myös turvallisuuden parantaminen on jatkuva tehtävä. Laboratorioympäristössä voitiin toimia paikallisverkossa melko avoimesti, mutta jos järjestelmää laajennetaan saataville paikallisen verkon ulkopuolelle, on otettava käyttöön OPC UA -varmenteet ja vahvempi käyttäjien hallinta. Järjestelmän käyttöönoton yhteydessä tulisi myös tehdä kattavampia testejä ohjelmiston ja sen jatkokehityksen virheiden varalta. Lisäksi reunalaskenta -periaatteita voisi soveltaa tuomalla PLC-datan analyysitoimintoja lähemmäs datan lähdettä: esimerkiksi liittämällä PLC:lle pienen tietokantapuskurin tai suodattimen, joka lähettäisi vain olennaiset tiedot tietokantaan.

Kaiken kaikkiaan opinnäytetyön tulokset osoittavat, että OPC UA -pohjainen datankeruujärjestelmä on toteuttamiskelpoinen ja hyödyllinen ratkaisu opetuksen ja tutkimuksen tueksi. Järjestelmä on myös yleiskäyttöinen: vastaavaa arkkitehtuuria voidaan soveltaa muissakin ympäristöissä, joissa tarvitaan reaaliaikaista tiedonkeruuta ja suurten aikasarjadatamäärien hallintaa, aina teollisista IoT-sovelluksista tutkimusprojekteihin. Opinnäytetyön ohjelmiston kehittämisprojektia on mahdollista jatkaa lisäämällä uusia ominaisuuksia esimerkiksi PLC-datan

monitorointiin liittyen ja ottamalla ohjelmiston käyttöön Metropolian automaatiolaboratoriossa. Käyttöönotossa on harkittava muun muassa palvelimen ja ohjelmiston konfigurointia, fyysisten verkkoyhteyksien luomista PLC:lle ja palveluprosessien (service process) luomista palvelintietokoneelle. Työtä voidaan käyttää hyödyntämään avoimia standardeja automaation ja digitaalisen tiedon hallinnan rajapinnassa.

Lähteet

Bleuzé, S. 2024. OPC-UA and time synchronization: Common errors and how to fix them. Verkkoaineisto. Factry (blogi). <https://www.factry.io/blog/opc-ua-and-time-synchronization-common-errors/>. 1.12.2024. Luettu 2.3.2025.

Cisco. n.d. IoT Security Lab: What is OPC-UA and how does it manage security. Verkkoaineisto. Cisco. <https://www.cisco.com/c/en/us/td/docs/solutions/Verticals/IoT_Security_Lab/OPC-UA_WP.html>. Luettu 10.3.2025.

Claroty Team82. 2023. OPC UA Deep Dive (Part 1): History of the OPC UA Protocol. Verkkoaineisto. Claroty-blogi. <https://claroty.com/team82/research/opc-ua-deep-dive-history-of-the-opc-ua-protocol>. 12.4.2023. Luettu 28.2.2025.

Freedman, M. & Blackwood-Sewell, J. 2025. Timescale architecture for real-time analytics. Verkkoaineisto. Timescale Inc. <https://docs.timescale.com/about/latest/whitepaper/>. Luettu 2.3.2025.

Hoppe, S. 2023. OPC UA – Interoperability for Industrie 4.0 and IoT. Verkkoaineisto. OPC Foundation -esite. <https://opcfoundation.org/wp-content/uploads/2023/05/OPC-UA-Interoperability-For-Industrie4-and-IoT-EN.pdf>. Luettu 1.3.2025.

Kirdan, E., Řezábek, F., Mülbauer, N., Carle, G. & Pahl, M.-O. 2023. Real-Time Performance of OPC UA. Verkkoaineisto. ArXiv:2310.17052. <https://arxiv.org/abs/2310.17052>. 19.11.2023. Luettu 28.2.2025.

Mahmoud, M.S., Sabih, M. & Elshafei, M. 2020. Using OPC technology to support the study of advanced process control. Verkkoaineisto. ISA Transactions. <https://doi.org/10.1016/j.isatra.2014.07.013>. 18.2.2025. Luettu 10.4.2025.

OPC Foundation. 2017. OPC Unified Architecture Specification Part 4: Services. Verkkoaineisto. Versio 1.04. <https://reference.opcfoundation.org/Core/Part4/v104/docs/>. 22.11.2017. Luettu 1.3.2025.

OPC Foundation. n.d. Unified Architecture – Landingpage. Verkkoaineisto. <https://opcfoundation.org/about/opc-technologies/opc-ua/>. Luettu 20.2.2025.

PTC. n.d. What is OPC UA? – OPC Unified Architecture. Verkkolaineisto. PTC verkkosivu.

<https://www.ptc.com/en/technologies/iiot/industrial-automation/opc/opc-ua>. Luettu 20.2.2025.

Roulet-Dubonnet, O. 2018. Python OPC-UA Library Documentation. Verkkoaineisto. FreeOpcUa-projekti. <https://python-opcua.readthedocs.io>. Luettu 20.2.2025.

Timescale. 2023. TimescaleDB Documentation. Verkkoaineisto. <https://docs.timescale.com/>. Luettu 1.3.2025.

Timescale. 2024a. Data retention. Verkkoaineisto. <https://docs.timescale.com/use-timescale/latest/data-retention/>. Luettu 1.3.2025.

Timescale. 2024b. Continuous aggregates. Verkkoaineisto. <https://docs.timescale.com/use-timescale/latest/continuous-aggregates/>. Viitattu 23.4.2025.

Waehner, K. 2022. OPC UA, MQTT, and Apache Kafka – The Trinity of Data Streaming in Industrial IoT. Verkkoaineisto. Blogikirjoitus. <https://www.kai-waehner.de/blog/2022/02/11/opc-ua-mqtt-apache-kafka-the-trinity-of-data-streaming-in-industrial-iot/>. 11.2.2022. Luettu 1.4.2025.

OPC UA -asiakasohjelman käyttöohjeet

Käynnistyksessä vaadittavat ohjelmistot

- Python 3.10+
- PostgreSQL ja TimescaleDB-laajennus (suositeltu)

Asiakasohjelman asennus

1. Kloonaa varasto.

```
git clone https://github.com/jensjvh/thesis_metropolia.git
cd thesis_metropolia
```

2. Aseta Python-ympäristö.

Poetry (suositeltu):

```
# Asenna Poetry tarvittaessa
pip install poetry
# Asenna riippuvuudet
poetry install
```

Pip:

```
# Luo ja aktivoi virtuaaliympäristö
python -m venv venv
source venv/bin/activate # Windowsissa: venv\Scripts\activate

# Asenna riippuvuudet
pip install -r requirements.txt
```

3. Aseta tietokanta.

Asenna PostgreSQL ja TimescaleDB:

```
# Ubuntu/Debian
sudo apt install postgresql
```

```
# Seuraa TimescaleDB:n asennusohjeet osoitteessa https://docs.timescale.com/install/latest/

# Luo tietokanta
sudo -u postgres createuser -P username
sudo -u postgres createdb -O username username

# Ota TimescaleDB-laajennus käyttöön
sudo -u postgres psql -d username -c "CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;"
```

4. Määritä sovellus.

Muokkaa `config.yaml` -tiedostoa PLC-yhteyksien ja tietokantamääritysten määrittämiseksi:

```
plcs:
- name: plc1
  url: "opc.tcp://localhost:4840/freeopcua/server/"
  auto_discover: true

# Lisää muita PLC-määrityksiä tarpeen mukaan

database:
  dbname: username
  user: username
  password: yourpassword
  host: localhost
  port: 5432
```

Asiakasohjelman käyttö

1. Käynnistä OPC UA palvelin testausta varten (valinnainen).

Testausta varten voit käynnistää sisäänrakennetun OPC UA palvelimen:

```
cd src
python server.py
```

2. Käynnistä PLC-tietojen kerääjä.

```
cd src
python run.py
```

Kerääjä yhdistää kaikki määritetyt PLC-laitteet, havaitsee solmut, jos se on käytössä, ja aloittaa tietojen keräämisen.

3. Katso lokitiedot.

Tarkista plc_collector.log -tiedosto yksityiskohtaisia lokeja varten.

Asiakasohjelman määritysvaihtoehdot

PLC-määritykset

Vaihtoehto	Kuvaus	Oletus
`name`	PLC-laitteen yksilöllinen nimi	Pakollinen
`url`	OPC UA -päätepisteen URL	Pakollinen
`auto_discover`	Automaattinen solmujen havaitseminen	False
`discover_depth`	Automaattisen havaitsemisen syvyys	3
`discover_throttle_ms`	Viive havaitsemispyynnöissä (ms)	100
`discover_paths`	Aloituspolku havaitsemiselle	["/Objects"]
`nodes`	Manuaalinen solmujen kartoitus {solmu_id: nimi}	{}
`value_deadband_percent`	%:n muutos jotta arvo tallennetaan	1.0
`value_deadband_absolute`	Absoluuttinen muutos jotta arvo tallennetaan	0.0
`buffer_size`	Maksimimäärä bufferissa ennen tietokannan kirjoittamista	100
`buffer_flush_interval`	Maksimiviive ennen tietokannan kirjoittamista	5
`username`	Tunnistautumistunnus	Ei pakollinen
`password`	Salasana	Ei pakolli-

Vaihtoehto	Kuvaus	Oletus
		nen

Tietokannan määritykset

Vaihtoehto	Kuvaus	Oletus
`dbname`	Tietokannan nimi	Pakollinen
`user`	Tietokannan käyttäjä	Pakollinen
`password`	Tietokannan salasana	Pakollinen
`host`	Tietokannan isäntä	Pakollinen
`port`	Tietokannan portti	Pakollinen
`min_connections`	Minimi ´yhteyksien määrä	1
`max_connections`	Maksimi yhteyksien määrä	5