jamk.fi

Raspberry Pi:n mittausraportin yhdistäminen tuotannonohjausjärjestelmän työtehtävän kanssa ja tallentaminen serverin tietokantaan

Juha Ojala

Opinnäytetyö
Toukokuu 2021
ICT-ala
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Ojala, Juha	Opinnäytetyö, AMK	toukokuu 2021
	Sivumäärä 29	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Raspberry Pi:n mittausraportin yhdistäminen tuotannonohjausjärjestelmän työtehtävän kanssa ja tallentaminen serverin tietokantaan.

Tutkinto-ohjelma

Tieto- ja viestintätekniikka

Työn ohjaaja(t)

Jani Immonen, Matti Mieskolainen

Toimeksiantaja(t)

Tiivistelmä

Tehtaan kokoonpanolinjan osakokoisuuksien testauksissa hyödynnetään yhden piirilevyn tietokoneita, kuten Raspbery Pi. Niihin liitetyt anturit ja testausohjelmat käyttöliittymineen ovat ketteriä muokata tarpeisiin soveltuviksi. Testauksissa syntyviä mittaustuloksia ei kuitenkaan ole tallennettu myöhäisempää analysointi varten. Osittain koska testikoneet eivät ole liitettynä toimistoverkkoon tietoturvasyistä ja toisaalta koska tietojen tallennuksesta ei ole hyötyä ilman mukana kulkevia tuotetietoja ja koodeja.

Toimeksiantajan mukaan tuotetiedot voidaan hakea tuotannonohjausjärjestelmän käyttämistä XML tiedostoista, jotka välittävät tietoa kullekin kokonpanon solulle. Tavoitteena oli yhdistää Python ohjelman avulla tuotetiedot ja mittaustulokset, sekä tallentaa ne tietokantaan tehtaan serverille. Ohjelman pitäisi myös olla ns. vakiomalli, jonka voisi sijoittaa muidenkin testaussolujen tietokoneisiin. Ohjelmalle annetaan parametrinä testaussolun tunnus, jonka perusteella se suodattaa lukemansa XML tiedostot. Tietokantana toimi Microsoftin SQL Server, koska tehtaalla oli se jo käytössä. Ohjelma luo automaattisesti aina uuden taulun uutta testaussolua varten valmiiksi tehtyyn tietokantaan.

Lopputuloksena syntynyt lyhyt Python ohjelma koostuu kolmesta toistorakenteesta: XML skanneri, mittaustietojen luku ja puskuroitu lähetysosa. Testausohjelman mittaustiedot siirretään globaalimuuttujan avulla tietokantaan lähetettäväksi.

Toimistoverkkoon liittämisen edellytyksenä tietoturvan on oltava kunnossa. Koska itse tietokanta sijaitsee paikallisverkossa, on laitteen liikenne estetty paikallisverkon ulkopuolelle sekä omasta, että tehtaan palomuurista.

Avainsanat (asiasanat)

Raspberry Pi, Python, XML, SQL, tietokanta, tietoturva

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

Description

Author(s) Last name, First name	Type of publication Bachelor's thesis	Date May 2021
		Language of publication: Finnish
	Number of pages 29	Permission for web publication: x

Title of publication

Combining of Raspberry Pi's measurement report with manufacturing execution system's XML report and saving to database.

Degree programme

Information and Communication Technologies

Supervisor(s)

Ojala, Juha

Assigned by

Abstract

Single-circuit board computers, such as Raspbery Pi, are used to test the subassemblies of the factory assembly line. The sensors and test programs connected to them and their user interfaces are agile to adapt to the needs. However, the measurement results resulting from the tests have not been saved for later analysis. Partly because the test machines are not connected to the office network for security reasons and partly because data storage is not useful without accompanying product information and codes.

According to the client, the product information can be retrieved from the XML files used by the manufacturing execution system, which transfer information to each assembly cell. The aim was to combine the product data and measurement results with the help of a Python program, and to store them in a database on the factory server. The program should also be so-called standard model which could be placed on computers of other test cells. The program is given a test cell ID as a parameter, based on which it filters the XML files. Microsoft SQL Server was used as database because it was already in use at the factory. The program automatically creates a new table for a new test cell in the pre-configured database.

The resulting short Python program consists of three while structures: XML-scanner, measurement data reader and buffered transmission part. Measurement data from the test program is transferred to the data transmission program using a global variable.

Security must be in order as a prerequisite for connection to the office network. Because the database itself is located on the local network, traffic outside the local network is denied from both its own and the factory firewall.

Keywords/tags (subjects)

Raspberry Pi, Python, XML, SQL, database, information security

Miscellaneous (Confidential information)

Sisältö

Kuviot

Lyhenteet

API	Application Programming Interface
daas	database as a service
IDE	Integrated Development Environment
IoT	Internet of Things
MES	Manufacturing Execution System
SQL	Structured Query Language
UFW	Uncomplicated Firewall
VLAN	Virtual Local Area Network
XML	Extensible Markup Language

1 Johdanto

Laadun varmistamiseksi tehtaan kokoonpanolinjalla syntyviä kokonaisuuksia on testattava. Testauksessa hyödynnetään ketteriä ja edullisia yhden piirilevyn tietokoneita, kuten esim. Raspberry Pi (jatkossa viitataan RasPi) ja Arduino. Tähän mennessä RasPia käytetään jo parissa testaussolussa ja lisää kohteita on suunnitteilla. Puutteena on toistaiseksi ollut, että testeistä syntyviä mittaustuloksia ei tallenneta mihinkään myöhempää analysointia varten, vaan ainoastaan kuitataan testatuksi. Työn tavoitteena on uuden Python ohjelmakoodin avulla yhdistää testattavan kokonaisuuden tuotetiedot tuotannonohjausjärjestelmästä testin mittaustuloksiin, sekä tallentaa ne tietokantaan. Uuden ohjelman pitäisi olla myös ns. vakiomalli, jotta se voidaan liittää mihin tahansa uuteen testitietokoneeseen tallentamaan erilaisia mittaustuloksia. Tarkoituksena on siis luoda edellytykset paremmalle laaduntarkkailulle ja jäljitettävyydelle.

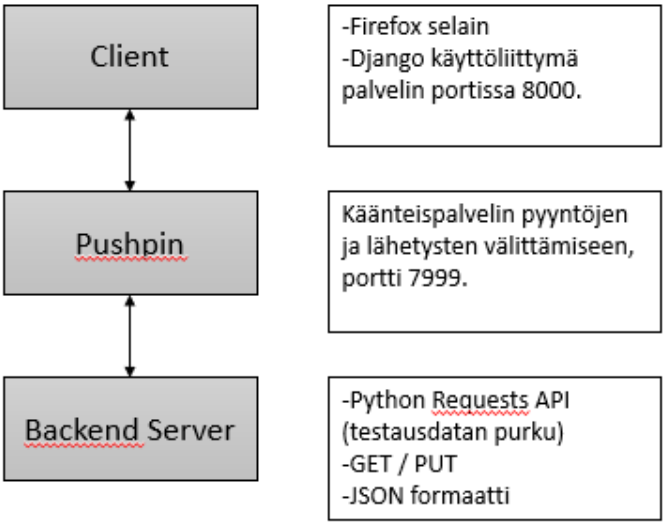
Työssä syntyvä ohjelmakoodi on osa tilaajan kehitystyötä, joten lähdekoodia, lähtötietoaineistoa tai työn tilaajan nimeä ei toimeksiantajan pyynnöstä julkaista tässä raportissa. Tarvittaessa työn tilaajaan viitataan maininnalla Yritys X. Tehtaalla tarkoitetaan Yritys X:n tuotantotehdasta.

2 Lähtökohdat

2.1 Raspberry Pi testauslaitteena

Esimerkkinä jo käytössä olevasta RasPista, jossa testaustapahtuma etenee niin, että kun osakokonaisuus on valmiina paikallaan ja tuotannonohjausjärjestelmän näytöllä on testauspyyntö, niin testaaja käynnistää testin painonappulasta ja etenee RasPin näytöltä saamiensa ohjeiden mukaan. Kun testi on suoritettu, kuitataan se valmiiksi tuotannonohjausjärjestelmän näytöltä. Tässä tapauksessa testauksessa tarvittavat painonappulat ja mittausanturit, sekä ohjelman toiminnallisuus on tehty Arduinossa. Arduino lähettää koodattua testausdataa sarjaliikenneväylää pitkin RasPille. Kuviossa

1 on esitetty RasPin ohjelmointirajapinta, jossa Python kielellä tehty ns. testausdatan purkuohjelma, joka toimii API ohjelmointirajapintana käyttöliittymää varten. Käyttöliittymä puolestaan on toteutettu Pythonin Django ohjelmalla. Päällimmäisenä RasPissa toimii i3wm ikkunoiden hallintaohjelma, joka avaa oletuksena omalle välilehdelle selaimeen käyttöliittymän testausohjelmalle. Toiselta välilehdeltä voi tarkastella terminaali-ikkunoista palveluiden käynnistymistä ja Python ohjelmien raportointia. Python ohjelmien ja palveluiden käynnistyminen on määritelty i3wm konfigurointitiedostossa, malli kuviossa 2. Huomattavaa on, että itse testausohjelmisto mittauksineen voisi olla toteutettu myös RasPissa Arduinon sijaan. Toisaalta itse mittaustapahtuman suunnittelu voi olla ketterämpää Arduinolla tai sopiva lisälaitekortti voi olla paremmin saatavilla siihen, mutta tässä tapauksessa RasPissa olevan käyttöliittymän suunnittelu on haluttu pitää omana kehitystyöprojektinaan.



Kuvio 1. RasPin ohjelmointirajapinta

```
#startup
exec urxvt -e python3 ~/[redacted]tests/manage.py runserver
exec urxvt --hold -e sh -c "sleep 20; python3 ~/[redacted]tests/serialhandler/jarrutesthandler.py"
exec urxvt -e sh -c "sudo mount -a; sudo pushpin --route='* localhost:8000'" &
#exec urxvt -e sudo pushpin --route="* localhost:8000"
#exec urxvt -e sudo mount -a
exec --no-startup-id sh -c " sleep 25; i3-msg workspace 2; sleep 5; firefox"
exec xset s noblank
exec xset s off
exec xset -dpms

# start a terminal
bindsym $mod+Return exec urxvt

# screen lock manual
bindsym $mod+z exec i3lock -c 000000 -n -f
# screen lock auto
exec_always --no-startup-id xautolock -detectsleep -time 600 -locker "i3lock -c 111111" \
-notify 30 -notifier "notify-send -u critical -t 10000 -- 'LOCKING screen in 30 sec'"
```

Kuvio 2. i3wm konfigurointitiedosto

2.2 Tuotannonohjausjärjestelmä ja XML tiedostot

Tehtaan tuotannonohjausjärjestelmä MES huolehtii eri kokoonpanon ja testauksen vaiheista mm. materiaalien keräyksellä ja työohjeiden lähetyksellä eri soluihin oikeaan aikaan, sekä raportoimalla ja tallentamalla tapahtumat tietokantaansa. Tiedonvälitys järjestelmän ja solujen välillä tapahtuu XML tiedostojen avulla, joiden sanomarakenne on vakioitu. Käytännössä solut lukevat XML tiedostoja ohjausjärjestelmän serverin jaetusta kansiosta ja tiedot ovat sitten nähtävissä solun näytöllä.

Käydään esimerkkinä läpi testaussolun XML tiedostot. Yhteen testaustapahtumaan liittyy neljä eri tiedostoa. Tiedostojen nimet sisältävät yhden tai useamman koodin, jolla tiedostot voidaan identifioida samaan tapahtumaan kuuluviksi. Lisäksi aikaleima ja tapahtuman eri vaiheita kuvaava järjestysnumero 1-4. Ensimmäinen vaihe sisältää perustietoja tilauksesta, kuten osaluettelon ja mistä ne ovat peräisin. Toinen vaihe aktivoi testauspyynnön. Tiedosto sisältää testaussolun tunnisteen, minkä perusteella testauspyyntö näkyy oikean testaussolun näytöllä testaajalle. Kolmas vaihe kuittaa ohjausjärjestelmälle, että testauspyyntö on vastaanotettu. Viimeinen neljäs vaihe lähettää tiedon testauksen loppumisesta, kun testaaja on kuitannut sen näytöltä. Ajallisesti väli (1-2) tilauksesta testauksen aloitukseen voi kestää päivän tai pari, kun taas testausväli (3-4) kestää kymmeniä minuutteja.

Työssä keskitytään toisen vaiheen tiedostoon, koska juuri se aktivoi testauksen, jonka mittaustulokset halutaan tallentaa. Tiedosto sisältää myös tarvittavat tiedot mittaustietojen mukana talletettavaksi, kuten tilausnumeron, viivakoodin, testaussolun tunnisteen, sekä tieto suoritetusta testistä. Näiden tietojen perusteella testaustapahtuma pitäisi pystyä myöhemmin jäljittämään ohjausjärjestelmän tietokannasta.

Neljäs vaihe raportoi vain, että testi suoritettu. Alkuvaiheessa vaihtoehtona oli, että mittaustuloksia raportoitaisiin tämän tiedoston mukana. Mutta sitten todettiin, että nykyisen ohjausjärjestelmän tietokanta on jo niin iso, ettei järkevää lisätä tietoa sinne. Lisäksi tietokannan muokkaaminen edellyttää hyväksyttyä hankintaketjua. Myös RasPin uusi tallennusohjelma muodostuu yksinkertaisemmaksi.

2.3 Microsoft SQL tietokanta

Työssä käytettävässä tietokannassa päädyttiin Microsoftin SQL Server relaatiotietokantaan lähinnä siksi, että sitä käytetään tehtaalla jo muutenkin. Ja koska paikallisen serverin kannat ovat paikallisen ylläpitäjän hallinnoimia, on uuden tyhjän kannan luominen hyvin suoraviivaista, asetusten ja versioiden pysyessä yhtenäisinä. Todettiin myös, että useammankin RasPin lähettämän datan määrä ja nopeus on sen verran pientä, ettei se aseta vaatimuksia tietokannan tyypille.

Vaihtoehtona paikalliselle kannalle pohdittiin pilvitallennusta. Tähän on Microsoftilla tarjota Azure SQL relaatiotietokanta, joka on toteutettu daas palveluna. Käyttäjä voi valita, vaikka yksittäisen tietokannan ja maksaa palvelusta käytetyn muistin mukaan, eikä tarvitse huolehtia ohjelmistopäivityksistä tai pilviserverin raudasta. Tietokannan luonnin jälkeen itse konfigurointi ja hallinta tehdään samalla SQL Server Management Studio ohjelmalla, kuin SQL Server tapauksessakin. Lisäksi tiedot ovat helposti siirrettävissä SQL Server ja Azure SQL välillä, koska molemmat käyttävät samaa tietokannan rakennetta.

3 Raspberry:n tietoturvaselvitys

Ohjelman käytön edellytyksenä toimeksiantajan puolelta on, että RasPi saadaan liitettyä toimistoverkkoon turvallisesti.

3.1 Verkkosegmentointi

Yleensä ottaen IoT-laitteet olisi hyvä eristää toimistoverkosta, sillä niiden tietoturvallisuudessa saattaa olla puutteita. Verrattuna normaaleihin työasemiin joita päivitetään keskitetysti järjestelmänvalvojien toimesta, saattavat IoT-laitteet sisältää vanhentuneita ohjelmia ja turhaan auki olevia portteja.

Eristämisen vaihtoehtoja on kuvattu selväsanaisesti esim. Mikrobitin artikkelissa (IoT-laitteet turvallisesti lähiverkkoon, 2019), jonka mukaan eristämiseen voi käyttäää vierasverkkoa, VLAN verkkoa tai jopa kokonaan omaa fyysistä reititintä. Eristetyssä verkossa käytetään myös vahvaa WPA2 salausta. Vierasverkoissa on monesti hyödyllisiä ominaisuuksia, kuten maksimikaistanleveys rajoittaminen, jolloin mahdollinen palvelunestohyökkäys ei tuki koko verkkoa.

Tällä hetkellä tehtaalla ei vielä ollut IoT-laitteille osoitettua omaa verkkoa, mutta verkkouudistus on vireillä. Verkon suojaamisessa käytetään kuitenkin MAC-osoitesuodatusta, jolla sallitaan tietokoneiden pääsy toimistoverkkoon ja internettiin. Mikäli RasPia käytetään paikallisverkossa sijaitsevan tietokannan kanssa, estetään sen ulosmenevä liikenne internettin. Tehtaan laitteistosta riippuen tämä voidaan toteuttaa reitittimen/palomuurin pääsylistoilla ja säännöillä. Tehtaalla saattaakin jo olla koneita, joiden sallitaan liikennöivän vain paikallisverkossa.

3.2 Ohjelmallinen tietoturva

RasPin ohjelmat oli asennettu oletuskäyttäjänimellä Pi. Käyttäjänimi olisi hyvä vaihtaa, koska tunnettu oletuskäyttäjä on suosittu brute-force hyökkäysten kohde. Tär-

keintä on kuitenkin vaihtaa käyttäjän salasana riittävän vahvaksi ja muuttaa sudo komennot kysymään salasanaa. Sudo komennon avulla käyttäjä voi suorittaa komentoja pääkäyttäjäoikeuksin. Vaikka RasPi olisikin joutunut tietomurron kohteeksi, niin hyökkääjä ei pääse suorittamaan komentoja, jotka vaativat pääkäyttäjäoikeudet.

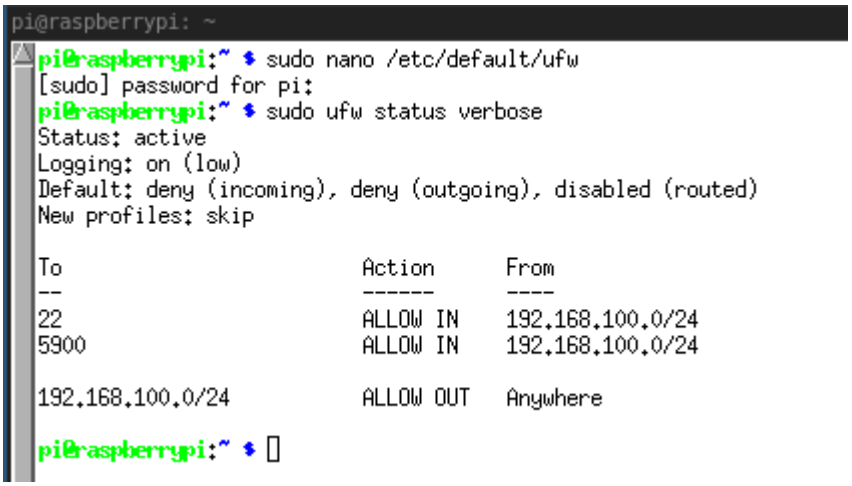
Ikkunoidenhallintaohjelman konfigurointiin lisättiin vielä i3lock skripti, jonka avulla näyttö lukittuu joko näppäinyhdistelmällä tai tietyn ajan kuluttua, kommentoitu tekstillä ”screen lock manual/auto” aikaisemmassa kuviossa 2.

Tietoturvaan kuuluu olennaisena osana myös ohjelmien päivitys. Työssä päivitettiinkin ensin Debianiin pohjautuva Raspbian käyttöjärjestelmä versiosta 9(Stretch) versioon 10(Buster), mikä oli helppo operaatio Raspberryn verkkosivujen ohjeen avulla. Jo asennetut ohjelmat päivitettiin vielä tutuilla sudo apt update ja upgrade komennoilla. Ohjelmat olisi hyvä päivittää jatkossa automaattisesti esim. crontab ajastusohjelman avulla. Mutta mikäli RasPille ei sallita pääsyä internettiin, on ohjelmien päivitys tehtävä säännöllisesti käsin, esim. siirtämällä SD-muistikortti toiseen ylläpitäjän hallitsemaan laitteeseen, jolla on internet yhteys.

Seuraavana suojaustoimena RasPiin asennettiin helppokäyttöinen UFW (Uncomplicated Firewall) palomuuri. Boucheron (2018) kertoo verkkoartikkelissaan, että UFW toimii käyttöliittymänä vaativammalle Iptables palomuurille, joka saattaa olla liian vaativa aloittelijalle. Tässä yksinkertaisen laitteen tapauksessa saadaan suojaus tehtyä helpoilla komennoilla, eikä aikaa tarvitse käyttää vaikeasti ymmärrettäviin sääntöihin.

Palomuurin konfigurointitiedostosta asetettiin tuleva ja lähtevä liikenne oletuksena estetyksi. Tämän jälkeen käynnistettiin palomuuri ja mietittiin tarpeen perusteella lisäsääntöjä. Jos tietokanta sijaitsee tehtaan paikallisverkossa, ei RasPilla ole tarvetta päästä ulkoverkkoon. Sääntöihin siis lisätään aina ehdoksi paikallisverkon IP-alue. Tällä saadaan toteutettua monikerroksinen suojaus, mikäli edellisessä kappaleessa mainittu esto tehtaan reitittimestä/palomuurista on myös toteutettu. Paikallisverkosta tulevasta liikenteestä sallittiin vain testausta varten porttien VNC (5900) ja SSH

(22) yhteydet etämonitorointia ja tiedostojen siirtoa varten. Mikäli ohjelmat jätettäisiin käyttöön, pitäisi tunnistautumisessa käyttää esim. avaimia salasanojen sijaan. Lähtevästä liikenteestä sallittiin kaikista porteista paikallisverkkoon kohdistuva liikenne tietokantaa varten. RasPin käyttöliittymän Firefox selaimesta ei siis saa päästä internettiin. Näin RasPista saatin varsin suljettu kokonaisuus muutamalla komennolla. Kuviossa 3 näkyy yhdellä komennolla verbose parametria käyttäen saatu listaus palomuurin tilasta, oletusasetukset ja luodut säännöt.



Kuvio 3. UFW palomuurin tila

- sudo ufw allow from 192.168.100.0/24 to any port 22
- sudo ufw allow out to 192.168.100.0/24

Linux järjestelmästä löytyy haavoittuvuuksia siinä missä Windows ja Apple järjestelmistäkin. Toisaalta haittaohjelmien tekijöiden pääkohde on edelleen Windows. Lisäksi Linux järjestelmässä haittaohjelman pitäisi saada pääkäyttäjän oikeudet aiheuttaakseen tuhoa. Linux järjestelmässä, joka toimii esim. mailiserverinä ja joka kommunikoi muidenkin käyttöjärjestelmien kanssa, suositellaan käytettäväksi esim. avoimen lähdekoodin virustorjuntaohjelmaa ClamAV. Tässä tapauksessa virustorjunta jätettiin asentamatta.

3.3 Fyysinen tietoturva

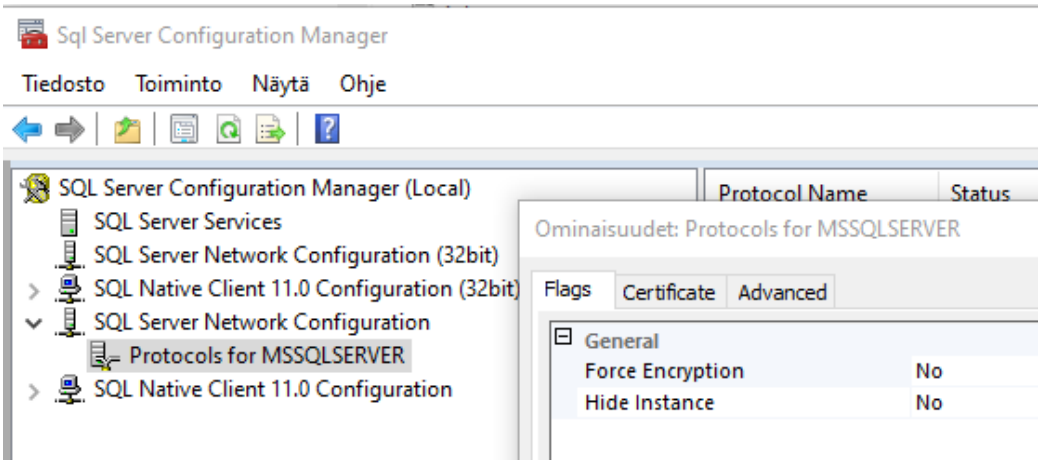
RasPin yksi suurimmista fyysisistä tietoturvapuutteista on helposti irroitettava SD muistikortti. Tähän asennetun linux käyttöjärjestelmän admin salasana on helposti resetoitavissa netistä löytyvillä ohjeilla ja toisella kortinlukijalla varustetulla koneella.

Helpoin tapa suojata RasPi on esim. seinään kiinnitetty lukittava kotelo. Samalla se suojaisi USB-portit vierailta laitteilta. Muistikortilla olevan datan voisi myös kryptata, mutta ongelmana on, että uudelleenkäynnistyksessä jonkun tarvitsee syöttää salasana ja toisaalta taas avaimet pitäisi tallettaa johonkin muualle kuin SD kortille. Vielä enemmän turvaa saisi asentamalla RasPin päälle esimerkiksi Zymbitin valmistaman turvamodulin, joka tutkii kahdella piirillä laitteen fyysistä eheyttä. Murron sattuessa Zymbit voi tarvittaessa lukita laitteen.

RasPia käytetään suljetulla tehdasalueella ja sisätiloissa, joihin vain henkilökunnalla on pääsy. Tällä perusteella voisi arvioida fyysisen tietoturvan uhkan olevan pieni ja riittävä suojaus olisi lukittava kotelo.

3.4 Datan salaaminen

Datan eheydellä tarkoitetaan sen säilymistä muuttumattomana, kun dataa talletetaan tai siirretään kahden pisteen välillä. Tätä varten Microsoftin tietokannan yhteydet voidaan salata ja sertifikaattien avulla sallia vahvistettujen käyttäjien yhteydet. Azure SQL tietokannan tapauksessa yhteyden salaus on aina voimassa ja Azure on jo valmiiksi konfiguroitu tarvittavilla sertifikaateilla. Huomattavaa on, että SSMS ohjelmasta yhteys on myös aina salattu, riippumatta kirjautumisen salausasetuksista. SQL Server tietokannan tapauksessa taas salauksen voi pakottaa päälle instanssikohtaisesti Configuration Manager ohjelmasta, kuvio 4. Käyttäjä asentaa sertifikaatit. Salaus toteutetaan tehtaan omien käytäntöjen mukaan käyttöönottovaiheessa, eikä se vaadi toimenpiteitä ohjelmakoodissa.



Kuvio 4. SQL Serverin salauksen pakotus

4 Ohjelman toimintasuunnitelma

Liitteessä 1 on esitetty ohjelman toimintasuunnitelma vuokaavion avulla. Ohjelman toimintaa hahmoteltiin muutaman ehdon avulla ja tavoitteena oli saada pilkottua toiminta pienempiin kokonaisuuksiin, jolloin ohjelmakoodia voidaan rakentaa ja testata hallitusti pienissä osissa. Ohjelmointityyli on proseduraalinen, eli ohjelma sisältää aliohjelmia, joille lähetetään tietoa tapahtuman suorittamiseen.

4.1 Mittaustulosten tallennus

Olemassaoleva testausdatan ohjelmointirajapinta RasPissa muodostuu lähinnä yhdestä jatkuvasta while toistorakenteesta. Ohjelma päivittää näytölle mm. mittausarvoja ja tietoa testauksen tilasta. Lähtökohtana oli tehdä uusi ohjelma rinnalle tietokantaan tallennusta varten, eikä haluta keskeyttää mittauksen päivitystä. Ja toisaalta muistettava myös se, että itse testausohjelma voi sijaita myös RasPissa. Erillistä ohjelmaa puoltaa myös se, että tallennuksen vikatilalla ei ole vaikutusta testauksen etenemiseen, joka kuitenkin on kriittisin tehtävä.

Testausdatan ohjelmasta otetaan talteen mittaustulokset uuteen muuttujaan, kun testi on tilassa hyväksytty tai hylätty. Tila, joka löytyy kaikista testausohjelmista.

Tämä ohjelma myös asettaa pyynnön tulosten tallentamiselle tietokantaan. Uuden muuttujan sisältö siirretään uudelle ohjelmalle. Jokaisesta testaussolusta tallennetaan erilaisia asioita ja päädyttiin siihen, että tallennusohjelmalle lähetetään aina taulukko tallennettavaksi.

4.2 XML skannaus

Tuotannonohjausjärjestelmän jaettuun kansioon siis ilmestyy jatkuvasti uusia XML tiedostoja eri solujen käytettäväksi. Tarvitaan jatkuvatoiminen skannausohjelma eli tästä syntyy toinen while toistorakenne. Skannerissa olisi hyvä olla jonkinlainen suodatus, ettei tarvitse kaikkien tiedostojen sisältöä parseroida taulukkoon. Ja tiedossa on jo, että testauksen aktivoi vaiheen kaksi tiedosto. Parseroidaan siis vain suodatetut tiedostot ja etsitään niistä ko. testaussolua vastaavaa solutunnusta.

4.3 Tietokantaan tallennus

Lopuksi pitää odottaa hetkeä, jolloin taulukot yhdistetään ja suoritetaan varsinainen tietokantaan tallennusohjelma. Tähän tarvitaan kolmas while toistorakenne, joka odottaa lähetyspyyntö signaalia. Ja kun tallennus suoritettu, nollataan lähetyspyyntö. Tässä tehdään oletus, että viimeisin tallennettu XML vastaa viimeisintä tallennettua mittaustulosta. Muussa tapauksessahan virhe olisi tapahtunut jo tuotannonohjausjärjestelmän XML lähetyksessä. Kolmas toistorakenne tarvitaan myös tapauksessa, jossa ei tulekaan mittaustulosta, eli skanneria ei voida keskeyttää ja jättää odottamaan lähetyspyyntöä. Testitilanteita varten kuitenkin estetään lähetys, jos XML tiedot ovat tyhjiä.

5 Toteutus

5.1 Lähtöaineisto ja työkalut

Työn lähtöaineistoksi sain tilaajalta RasPin, joka oli varmuuskopio jo käytössä olevasta testaussolusta. Lisäksi erikseen sain Arduinon ohjelmakoodit, sekä yhden tapahtuman eli 4 XML-tiedostoa malliksi. Työ alkoi tutustumisella RasPiin tehdyistä ohjelmista, sillä niistä ei ollut toimintakuvausta. Lyhyt yhteenveto sisällöstä esitettiin kappaleessa 2.1 Raspberry Pi testauslaitteena, sekä Liitteen 1 vuokaaviossa.

Testivälineenä toimi siis itse RasPi, johon kopioitiin uusi ohjelmakoodi. Ja tuotannonohjausjärjestelmän jaettuna hakemistona toimi Windows läppärin hakemisto. Käytännössä ohjelmia tehtiin ja testattiin läppärillä PyCharm IDE editorilla. Välillä ohjelmat on kuitenkin siirrettävä RasPiin yhteensopivuuden varmistamiseksi. Varsinkin hardwaren yhteensopivuus on testattava uusien ajureiden tapauksessa.

Koska Python on ohjelmointikielenä hyvin suosittu, löytyy internetistä runsaasti erilaisia oppaita, kursseja ja foorumeja. Koulussa opittujen perusteiden lisäksi hyödynnetään netistä löytyviä keskusteluja ja malliesimerkkejä. Stackoverflow.com on tunnettu ohjelmointiin liittyvä verkkosivusto, jonka foorumilta löytää kommentoituja vaihtoehtoja erilaisiin ongelmiin.

5.2 Mittaustulosten siirto

Kysymykseen kuinka siirtää muuttujia kahden erillisen ohjelman välillä, yleisimpiä ehdotuksia olivat putken rakentaminen multiprocessing kirjastoa käyttäen tai vaikka tallennustiedostolla pickle kirjastoa käyttäen. Nämä eivät tuntuneet kovinkaan järkeviltä tavoilta parin muuttujan siirtämiseen. Mieluiten käytettäisiin globaaleja muuttujia helppouden vuoksi, mutta ne eivät tuntuneet toimivan kahden ohjelmaskriptin välillä. Askel taaksepäin ja vilkaisu koulun kurssimateriaaliin tuotti tulosta. ”Samassa prosessissa olevat säikeet käyttävät samaa muistialuetta.” (Häkkinen 2019, 7) Säikeet ovat prosesseja kevyempiä ratkaisu juuri yhteisen muistialueen vuoksi. Säikeiden

käyttö ratkaisee kaksi ongelmaa. Sen lisäksi, että voidaan käyttää globaalimuuttujia, ratkeaa uusien ohjelmien käynnistys. Liitteen 1 vuokaaviosta nähdään, että uusi ohjelma sisältää kaksi while toistorakennetta. Käynnistetään siis niiden funktiot omina säikeinään olemassaolevasta testausdatan ohjelmointirajapinnasta. Uudet säikeet on merkattu vuokaavioon ja sen lisäksi on käynnistysjärjestyksen listaus, jossa näkyy vain olemassaoleva pääohjelma.

Kuviossa 5 globaalimuuttujat on sijoitettu erilliseen ohjelmaan, joka alustetaan heti pääohjelman käynnistyksessä. Molemmat ohjelmat kutsuvat muuttujia muodossa data_pipe.sendreq. Tällä tavalla globaalit erottuvat selkeästi muista muuttujista ja pysyvät hallinnassa.

data_pipe.py

```
1   def init():
2       # Send Request to database. Auto reset after sending.
3       global sendreq
4       sendreq = False
5       # init empty dictionary for test results data (column name:value).
6       global testdata
7       testdata = {}
8       # init empty dictionary for xml file data (child.tag:child.text).
9       global xmldata
10      xmldata = {}
11
```

Kuvio 5. Globaalimuuttujien alustus

5.3 XML skannaus

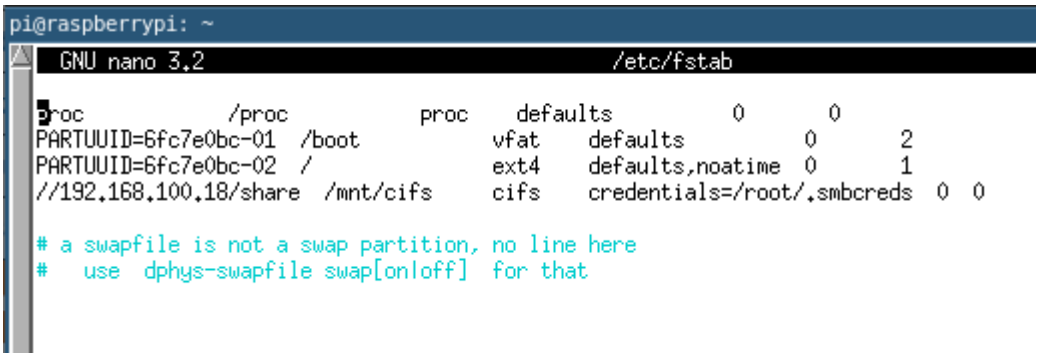
Säikeessä 1 tarvitaan ohjelma, joka tutkii tiedostojärjestelmän muutoksia eli havaitsee uuden XML tiedoston hakemistosta. Vaihtoehtoisia kirjastoja oli useita ja yksinkertaisimmillaan olisi voinut käyttää Pythonin omaa os kirjastoa tekemällä oma olio tarkkailua varten. Etsinnän jälkeen helpoin vaihtoehto oli kuitenkin Watchdog kirjasto. Se sisältää helppokäyttöisen parametrilistan suodatusten tekemiseksi ja sen mainostettiin toimivan tunnetuimmilla käyttöjärjestelmillä. Mastromatteo (2019) on verkkoartikkelissaan kuvannut toimintaa esimerkin avulla. Malliin kuuluu tarkkailija

(Observer), joka sijoitettuna while toistorakenteeseen skannaa määritellyn tiedosto-järjestelmän muutoksia muutaman sekunnin välein. Tarkkailija lähettää tapahtumat tapahtumankäsittelijälle (Event handler), joka käynnistää halutun aliohjelman. Työssä käytettiin suodatusehtona *PTC02*.xml, jolloin vain vaiheen kaksi XML tiedostot käynnistävät parseroinnin.

Ensimmäinen ongelma tuli vastaan, kun luetaan tiedostoja eri käyttöjärjestelmästä. Vona (2019) mukaan Windows käyttää SMB/CIFS protokollaa ja Linux Samba proto-kollaa ja tästä syystä jaettu hakemiston on liitettävä (mount) osaksi Linux järjestel-mää. Hän esitteli verkkoartikkelissaan erilaisia menetelmiä jaetun Windows hakemis-ton liittämiseksi. Jotta liittäminen tapahtuisi automaattisesti järjestelmän käynnistyk-sessä, päädyttiin lisäämään hakemisto järjestelmätiedostoon /etc/fstab. Varmiste-taan kuitenkin ensin, että tarvittavat ohjelmat on asennettu komennolla:

- sudo apt install samba samba-common-bin smbclient cifs-utils

Kuviossa 6 esitetyssä tiedostossa on listattuna kaikki järjestelmän käyttämät levyt ja osiot. Viimeisellä rivillä olevan Windows hakemiston salasanat on tallennettu tiedos-toon .smbcreds. Salasana on kylläkin tavallisessa tekstimuodossa, mutta tiedoston oi-keuksia on muutettu niin, että lukemiseen tarvitaan pääkäyttäjän tunnukset.



Kuvio 6. RasPin fstab järjestelmätiedosto

Jostain syystä hakemisto ei kytkeytynyt käynnistyksessä, vaikka hakemisto oli löydettävissä sillä hetkellä. Ratkaisuna lisättiin kuvissa 2 olevaan i3wm konfigurointitiedostoon uudelleenliittämiskomento ”sudo mount -a” samaan terminaaliin käänteisserverin kanssa.

Toinen ongelma oli, että yleensä liitettyihin kansioihin ei välity tiedot muutoksista tiedostojärjestelmistä. Tähän auttoi Wathcdog kirjaston ohjeista löytynyt vaihtoehtoinen PollingObserver, joka pystyi skannaamaan myös muun käyttöjärjestelmän hakemistoja. Pollaustekniikkaahan olisi käytetty myös Pythonin os kirjaston tapauksessa.

5.4 XML parserointi

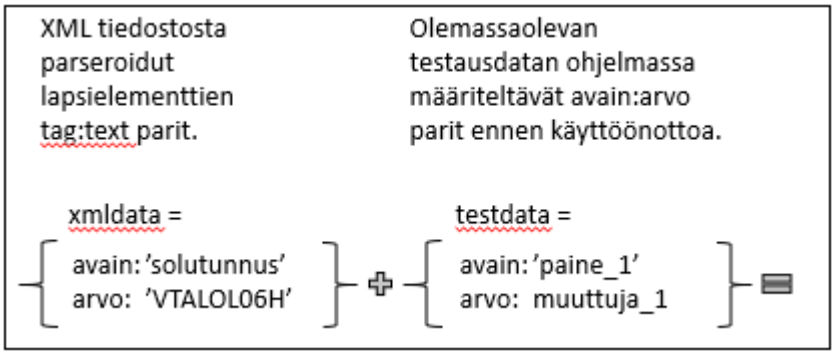
Kun Watchdog on löytänyt uuden XML tiedoston, suoritetaan sen parserointi heti perään. Tavoitteena on siis purkaa tiedot globaalimuuttuja taulukkoon myöhempää yhdistämistä varten. XML tiedostojen käsittelyyn löytyy useita kirjastoja, joista työssä käytetään Pythonin jakeluversioon kuuluvaa ElementTree kirjastoa, koska tiedostot ovat pieniä ja määrä pieni. Tämä myös mahdollistaa XML tiedostojen kirjoittamisen, mikäli tulee tarvetta.

Howson (2018) kuvailee verkkoartikkelissaan XML tiedoston rakenteen muodostuvan elementeistä, joiden sisällä on aloitus- ja lopetustageilla merkitty sisältö. Tiedostossa ylimpänä on aina juurielementti ja sen alapuolella kaikki lapsielementit.

Ohjelma purkaa kaikkien lapsielementtien tiedot for toistorakenteella, niin että tagin nimi on sama kuin taulukon sarakenimi ja sisältö vastaavasti rivin arvo. XML tiedostot eivät myöskään sisällä attribuutteja. Tiedot tallennetaan taulukkoon vain, jos testaussolun tunnus sama kuin RasPiin määritelty. Tagin nimi on säilytettävä samana, jotta myöhemmin tietojen jäljitys tuotannonohjausjärjestelmän tietokannasta olisi helpompaa.

5.5 Tietokantaan tallennus

Säikeen 2 while toistorakenne toimii siis puskurina. Globaalimuuttujiin tallennetut taulukot yhdistetään ja käynnistetään tallennusohjelma, kun lähetyspyyntö on asetettu ja xmldata taulukko sisältää tietoja. Kuviossa 7 on esitetty taulukoiden yhdistäminen. Taulukot ovat itseasiassa Python kielessä luetteloita tai sanakirjoja, mikä on myös SQL kielen suosima formaatti.



Kuvio 7. Tietokantaan tallennettava yhdistetty luettelo

5.5.1 Kirjasto ja ajuri

Myös tietokantaan tallennukselle löytyy useita kirjastoja. Koska tietokanta on Microsoftin, niin etsitään ensisijaisesti Microsoftin suosittelemaa kirjastoa.

Python SQL Driver (2017) ohjeiden mukaan suositeltuja kirjastoja ovat pyodbc ja pymssql, joista ensimmäiseen Microsoft kohdistaa testauksensa. Tämä pyodbc kirjasto hyödyntää Microsoftin ODBC ajuria, jonka asennus onnistuu julkaisulistan mukaan vain AMD prosessoria käyttäviin Debian käyttöjärjestelmiin.

Monet mobiililaitteet ja myös RasPi käyttävät ARM prosessoria, joten tästä syystä työssä käytetään pymssql kirjastoa, joka hyödyntää FreeTDS ajuria. Tämä ajuri on yhteisön ylläpitämä, eikä verkkosivustolla ollut mainintaa yhteensopivista prosessoreista. FreeTDS configuration - Testing the connection ohjeiden mukaan yhteyden muodostumista suoraan ajurin ja tietokannan välillä voidaan testata tsql työkalulla, ilman että pymssql kirjasto vaikuttaa siihen.

Edetään asennuksissa Python SQL Driver - pymssql (2020) ohjeiden mukaan, eli ensin ohjelmien asennus, sitten tietokannan asennus ja viimeisenä testihaku tietokannasta.

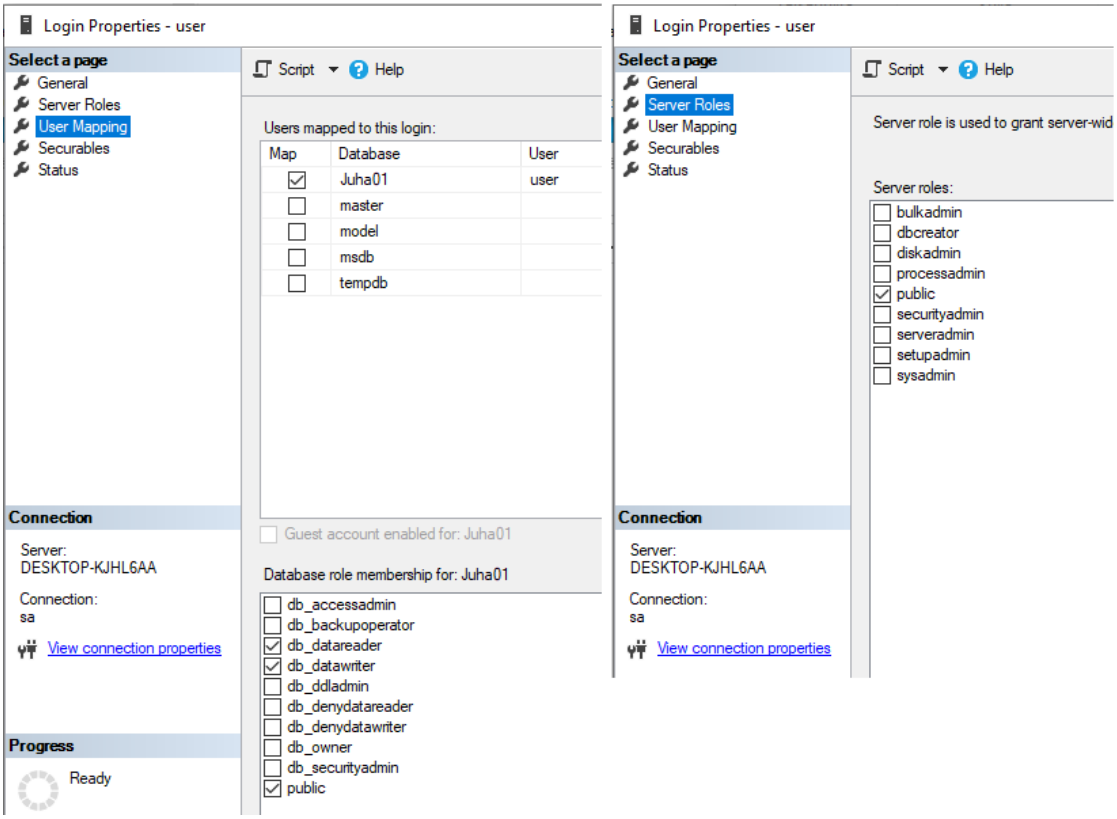
Kirjaston ja ajurin asennus tehdään komennoilla:

- sudo apt-get --assume-yes install freetds-dev freetds-bin
- sudo apt-get --assume-yes install python-dev python-pip
- sudo pip3 install pymssql

5.5.2 SQL Server

Asennetaan omalle läppärille tietokanta SQL Server 2019 Developer Edition. Asennuksessa instanssin nimi on MSSQLSERVER. Valitaan autentikointimenetelmäksi Mixed Mode ja annetaan server administrator (sa) salasana.

Asennetaan SQL Server Management Studio ohjelma SQL Serverin hallintaa varten. Ensimmäisenä toimenpiteenä luotiin uusi user käyttäjä rajoitetuin oikeuksin. Uudella käyttäjällä haluttiin varmistaa, että myös tallennusohjelma toimii näillä oikeuksilla. Kuviossa 8 user käyttäjälle on annettu vain perus luku- ja kirjoitusoikeudet uuteen Juha01 tietokantaan tietoturvallisuutta ajatellen. Eli vaikka käyttäjän salasana paljastuisikin, ei hyökkääjä pysty aiheuttamaan suurempaa vahinkoa.



Kuvio 8. Käyttäjän user oikeudet

Ohjelmasta SQL Server Configuration Manager asetetaan TCP/IP-liikennöinti päälle instanssille MSSQLSERVER.

Kun tietokannan koneen palomuurista on aukaistu TCP portti 1433, voidaan testata FreeTDS ajurin yhteys tietokantaan tsql komennolla (tsql -host -port -database -user -password):

- tsql -H 192.168.100.16 -p 1433 -D Juha01 -U user -P user.

5.5.3 Tallennusohjelma

Yhteyden muodostaminen pymssql kirjastoa hyödyntävästä ohjelmasta tietokantaan on varsin suoraviivaista ja noudatetaan tässä Microsoftin ohjeen testattua esimerkkiä. Tarvittavia tietoja yhteydenmuodostamiseen ovat tietokantaserverin IP-osoite, instanssin nimi (jos useampia ja ei oletus), tietokannan nimi, käyttäjänimi ja salasana. Sen sijaan, että salasana olisi nähtävissä suoraan ohjelmakoodissa, tallennetaan se

käyttöjärjestelmän globaaleihin ympäristömuuttujiin tiedostoon /etc/environment. Tiedosto on tavallisessa tekstimuodossa, mutta kirjoittaminen vaatii pääkäyttäjän tunnukset. Ideana oli, että jos koodia lähetetään tai esitellään muille osapuolille, niin salasana ei olisi nähtävissä.

Ohjelma luo tietokantaan uuden taulun annetulla testaussolun nimellä, mikäli kyseessä on ensimmäinen tallennuskerta. Ei haluta, että tietokantaa pitäisi käydä muuttamassa enää sen jälkeen, kun kanta ja käyttäjät on luotu. Taulun sarakenimet saadaan suoraan aiemmin mainitun yhdistetyn luettelon avaimista siinä järjestyksessä kuin ne ovat. Kaikkien sarakkeiden tietotyypiksi on valittu merkkijono max 25 merkkiä (varchar 25). Tällä tavalla saadaan yksinkertaistettua SQL komentoa ja toisaalta XML dokumentin tiedot ovat jo merkkijonomuodossa. Loppuun lisätään vielä automaattisesti kasvava järjestysnumerosarake IDcol. Tässä vaiheessa sarakkeiden järjestyksen muuttaminen tai poistaminen on työlästä ja se pitäisi tehdä jo aikaisemmissa vaiheissa, mutta toisaalta ei ole tarvettakaan.

Kun taulukko on luotu, voidaan tallentaa datarivi eli yhdistetyn luettelon arvot. Ohjelmaoppaissa arvojen tallennus tehdään yleensä kuvion 9 mukaisella SQL komennolla, mutta tämä on hieman epäkäytännöllinen tapa, joten vaihdetaan kuvion 10 menetelmään. Puretaan yhdistetyn luettelon (datadict) avaimet ja arvot SQL komentoon Pythonin .format metodia hyödyntäen kaarisulkeilla merkittyihin muuttujiin. Taulun sarakenimet pitää syöttää pilkulla erotettuina, ilman lainausmerkkejä ja tämä saavutetaan join metodin avulla. Sarakkeiden arvot puolestaan listataan tuple listaan lainausmerkkeineen.

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Kuvio 9. SQL INSERT peruskomento

```
"INSERT INTO "+database+".dbo."+tablename+" ({columns}) VALUES {values};" .format(
                columns = ', '.join(datadict.keys()),
                values = tuple(datadict.values())
                )
```

Kuvio 10. SQL INSERT .format metodilla

Tallennuksen jälkeen tulostetaan vielä taulukosta viimeisimmän rivin tietoja seurantaa varten. Käytetään hyödyksi IDcol sarakkeen juoksevaa numerointia, sekä IDENT_CURRENT funktiota, joka palauttaa sarakkeen viimeisimmän arvon.

Yhteyden salauksen todentamiseksi voidaan suorittaa kysely, joka kertoo salauksen tilan. Todentaminen testattiin ensin yhteydellä SQL Server kantaan, jossa ei ollut sertifikaattia eikä salauksen pakotusta päällä. Ja sen jälkeen yhteydellä Azure SQL kantaan, jossa taas salauksen pitäisi olla aina päällä. Kysely vaatii kuitenkin palvelimelta sysadmin oikeudet, joten kyselyä ei käytetä tuotannossa.

SELECT encrypt_option FROM sys.dm_exec_connections WHERE session_id = @@SPID;

Ohjelman lopuksi yhteys suljetaan, lähetyspyyntö nollataan ja XML taulukko tyhjennetään.

Virheenkäsittelyä varten tallennusohjelma on sijoitettu try-except Exception rakenteen sisään. Jos ohjelman aikana tapahtuu jokin virhe, esim. yhteys toimistoverkkoon puuttuu, suoritetaan except osiossa lähetyspyynnön nollaus, tulostetaan virheilmoitus ja ohjelma jatkaa seuraavalle riville. Testaustulokset menetetään tältä kerralta.

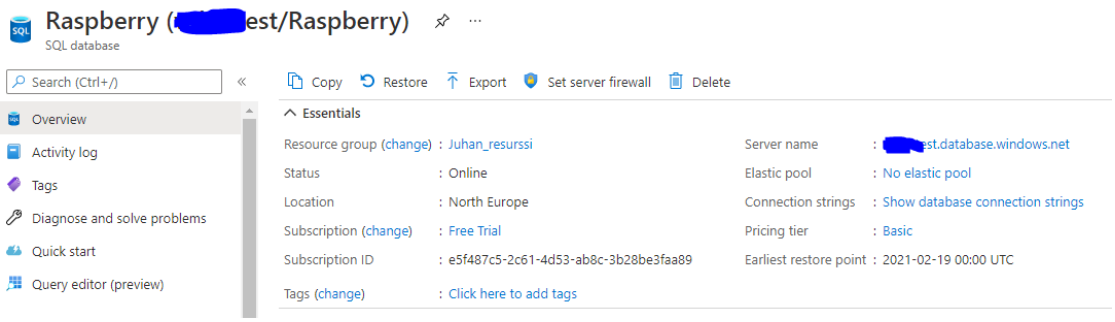
5.5.4 Azure SQL vaihtoehto

Vaihtoehtoisena tallennuspaikkana kokeiltiin pilvitallennusta Azure SQL Database palveluun. Koska itse tietokantaa mainostetaan samana kuin SQL Server versio, niin tarkoituksena oli selvittää, että vaatiiko se muutoksia ohjelmakoodiin.

Palomuurista sallitaan sisään- ja ulosmenevä liikenne oman maantieteellisen Azure Gatewayn IP-alue ja portti alue.

Tietokantaa varten luotiin ilmainen kokeilutili pilvipalveluun, jonka myötä saatiin siis ilmaista käyttöaikaa tietokannalle määräajaksi. Tilille kirjautuminen vaatii vahvan tunnistautumisen. Tämän jälkeen luodaan uusi resurssi ja siihen uusi yksittäinen tietokanta antamalla nimet serverille ja tietokannalle. Kuviossa 11 on esitetty tiedot luodusta tietokannasta ja siitä näkyy myös suoraan osoite yhteyden luomista varten kohdassa Server Name. Osoite on aina muotoa <server>.database.windows.net. Yhteyttä varten määritellään vielä palomuuri välilehdeltä sallittu IP-osoite tai alue. RasPissa ei tarvitse olla kiinteä IP-osoite, vaan palomuuriin määritellään internet palveluntarjoajan antama osoite. Otettaessa yhteyttä SSMS ohjelmalla tai Python ohjelmakoodilla annetaan serverin nimi tuolla em. pitkällä muodolla. Tämän lisäksi on annettava tietokannan nimi. Uuden käyttäjän luonti SSMS:ssä tehtiin Object Explorerin sijaan SQL kyselyn avulla.

Lopputuloksena pilvitallennus ei vaatinut mitään muutoksia ohjelmakoodiin, normaaleja yhteysparametrejä lukuun ottamatta.



Kuvio 11. Azure SQL tietokannan tiedot

5.6 Testaus ja Python tulosterivit

Ohjelmaa testattiin aluksi ihan funktio kerrallaan, mutta jaettiin sitten säikeen 1 ja 2 testauksiin. Säikeen 1 skanneria ja parserointia testattiin kopioimalla XML tiedostoja aluksi ohjelmakansioon ja sitten jaettuun kansioon, eli askel kerrallaan kohti lopullista ratkaisua. Säikeessä 2 taas testattiin ensin yhteyden luomista ja peruskyselyjen tekemistä tietokannasta, jonka jälkeen voitiin siirtyä haastavampiin SQL komentoihin kuten taulukon luontiin ja luettelon tallentamiseen. Viimeisenä yhdistettiin säikeet

globaalimuuttujiin, sekä vaihdettiin testausdatan ohjelman tilalle yksinkertainen simulointiohjelma, joka lähettää valmiin luettelon ja asettaa lähetyspyynnön viiveen kuluttua käynnistyksestä. Simulointiohjelman avulla siis ohitetaan Arduinolta tuleva viesti ja keskitytään vain testausdatan siirtämiseen.

Varsinaista ohjelmakoodia ei raportissa voida esitellä, mutta katsotaan hieman Pythonin terminaaliin tulostuvia rivejä uuden ohjelman osalta kuviossa 12. Näiden rivien avulla saa käsityksen ohjelman kulusta. Skannerin ja puskurin while toistorakenteet kirjoittavat muutaman sekunnin välein tekstiä lähinnä testausmielessä, jotta nähdään säikeiden olevan toiminnassa, mutta poistetaan siis tuotantoversiosta.

Kun skanneri on löytänyt uuden *PTC02*.xml tiedoston, ilmoitetaan siitä tulostamalla tiedoston nimi polkuineen. Parseroinnista tulostetaan juurielementin nimi. Ja jos tiedosto kuuluu halutulle testaussolulle, tulostetaan tietojen siirrosta puskuriin.

Kun lähetyspyyntö on asetettu, tulostuu tieto yhteyspyynnöstä ja yhteyden muodostumisesta. Kun rivi on tallennettu, tulostetaan kannasta viimeisimmän rivin tietoja, yhteyden salauksen tila (vain testi), sekä yhteyden lopetus.

```
scanning...
buffering...
scanning...
Found: .\MFGI3010_PTC02_20201127063423411_LSA484014_S19_3.xml
XML root tag: Event
send xmldata to buffer
buffering...
scanning...
buffering...
scanning...
buffering...
SQL connection request
SQL server connected
Row ID: 44, ProductionOrderNumber: 123150730
Encrypt_option: FALSE
SQL server disconnected
scanning...
buffering...
```

Kuvio 12. Python ohjelman tulosteet terminaalissa

5.7 Käyttöönotto tehtaalla

Monistettaessa RasPia uutta testaussolua varten kopioidaan siis tämän SD-muistikorttia, jolloin RasPiin tehdyt asetukset verkkomäärittelyjen ja salasanan osalta säilyvät. Tietokanta yhteysmäärittelyineen ja käyttäjineen pysyy aina samana molemmissa päissä. Uuden RasPin käyttöönotossa muuttuu testaussolun tunnus ja testauksessa tallennettavat datapisteet. Omana projektinaan uuden testauspisteen tapauksessa on käyttöliittymän suunnittelu Djangolla.

Lyhyt muistilista tarkastettavista asioista:

- testausdatan ohjelmaan lisättävä säikeiden käynnistys, sekä tallennettavat tiedot sarakenimineen Python luetteloon
- testaussolun tunnus tallennusohjelmaan
- tarkasta XML tiedostojen suodatus tallennusohjelmassa
- SQL tietokannan yhteysparametrit tallennusohjelmaan
- SQL tietokannan salasana ympäristömuuttujiin tiedostoon /etc/environment
- suoritusoikeus Python ohjelmille komennolla chmod +x *.py
- testausdatan ohjelman nimi tiedostoon ~/.config/i3/config
- jaetun hakemiston polku ja salasana tiedostoon /etc/fstab
- tarkista palomuurin tila komennolla sudo ufw status verbose

6 Pohdinta

Tavoitteena oli luoda ns. vakiomalli tietojen yhdistämiseksi ja tallentamiseksi. Tuloksena syntyi lyhyt ohjelmakoodi helposti ymmärrettävillä ratkaisuilla. Kun ohjelma on käyttöönotettu tehtaalla ensimmäistä kertaa, niin sen jälkeen tulevissa testaussoluissa uusi ohjelma vain tallennetaan testausohjelman mukana ja sille lähetetään tallennettava materiaali globaalimuuttuja luettelon avulla.

Tehty ohjelma antaa mahdollisuuden paremmalle laaduntarkkailulle. Mutta varsinainen työ on kuitenkin sopivien testaussolujen tunnistaminen, sillä niihin pitää

luoda ensinnäkin testausproseduuri ja lisäksi testausohjelmisto käyttöliittymineen. Tiedossa olevia kohteita on jo muutama tässä vaiheessa. Työn hyödyllisyyttä varten on katsottava valmistusmääriä. Jos yhdellä tehtaalla olisi 5 raportoivaa testaussolua ja lopputuotteita valmistuisi 30 kappaletta päivässä, ei vuositasolla vielä saada suurta määrää dataa analysoitavaksi, mutta sieltä pystynee huomaamaan kun jokin asia on muuttunut. Toisaalta käytetyt osat ovat kalliita, kuten myös takaisinkutsut korjauksia varten.

RasPien ja niiden datan pienestä määrästä johtuen suositus on paikallinen SQL tietokanta. Tietoturva paranee, mikäli RasPeille ei sallita pääsyä internettiin. Eikä tarvitse ostaa uusia pilvipalveluja tms. Tällä tavalla päästään myös kehittämään uusien testaussolujen ohjelmia hieman ketterämmin ja on huomattava, että uusia ohjelmia kehitellään todennäköisesti opiskelijavoimin, jolloin voi myös virheitä sattua herkemmin. Datan nouto analysointia varten onnistuu paikallisesta kannasta siinä missä pilvipalvelustakin ja toisaalta tiedot on myös siirrettävissä pilveen, koska itse tietokanta on sama. Tässä vaiheessa tehtaalla ei ole vielä suunnitelmia datan analysoinnista.

Toimeksiantajalla oli jo esittää jatkokehitysidea, jossa testausparametrit tulisivat XML tiedoston mukana kun käytetään erilaisia osia. Tällä hetkellä parametrit syötetään selaimen parametrit sivulla, josta ne siirretään testausdatan rajapinnan kautta Arduinolle, kun testauksen aloituspainiketta on painettu. Tämä ketju voidaan säilyttää sellaisenaan. Lisätään tallennusohjelmaan XML parserointi parametreille, sitten kun siitä on tehty vakioratkaisu tuotannonohjausjärjestelmässä ja siirretään parametrit globaalimuuttujalla testausdatan rajapinnalle, joka puolestaan voi tallentaa arvot selaimen sivulle. Siirto ohjelmien välillä tehdään, koska ei haluta sekoittaa yksittäisen testaussolun määrittelyjä yleiseen tallennusohjelmaan.

Ohjelmointityön aikana tuli kaksi huomionarvoista asiaa. Ensinnäkin vaihtoehtojen kartoittaminen tietyn asian saavuttamiseksi. Ohjelmoinnissa lähestymistapoja on useita. Paras tapa ei välttämättä ole keskustelupalstoilla mainostettu, vaan se tapa jonka toiminnan itse ymmärrät ja pystyt testaamaan. Siihen kun etsitään sopivaa kirjastoa, on taas selvitettävä vaihtoehdot ja soveltuvuus. Yleisesti ottaen kannattaa

suosia tunnettuja, tuettuja ja päivitettyjä kirjastoja. Toiseksi debuggaus eli virheenjäljitys, johon törmää väkisinkin. Työn aikana huomattiin, että ohjelman pilkkominen pieniin osiin helpotti työtä huomattavasti. Sekä ajureiden testaaminen ennen varsinaisen sovelluksen tekemistä. Alusta lähtien käytettiin paljon muuttujien välitulostuksia oikean sisällön varmistamiseksi tai ihan vaan, että näki missä ohjelma on menossa. Parasta työssä oli monen pienen asian selvittäminen ja ratkaiseminen.

Lähteet

Boucheron, B. 2018. How To Set Up a Firewall with UFW on Ubuntu 18.04. Digitalocean verkkosivusto. Päivitetty 5.7.2018. Viitattu 22.2.2021. https://www.digitalocean.com/community/tutorials/how-to-set-up-a-firewall-with-ufw-on-ubuntu-18-04#prerequisites.

FreeTDS configuration - Testing the connection. Pymssql verkkosivusto. Viitattu 22.2.2021. http://www.pymssql.org/freetds.html#testing-the-connection.

Howson, S. 2018. Python XML with ElementTree: Beginner's Guide. Datacamp -verkkosivusto. Päivitetty 6.3.2018. Viitattu 22.2.2021. https://www.datacamp.com/community/tutorials/python-xml-elementtree.

Häkkinen, A. 2019. Prosessien hallinta opintomoniste. Käyttöjärjestelmät opintojakso 2019. Jyväskylän ammattikorkeakoulu. IT-instituutti.

IoT-laitteet turvallisesti lähiverkkoon. 2019. Mikrobitti lehden opas kesäkuu 2019.

Mastromatteo, D. 2019. How to create a watchdog in Python to look for filesystem changes. The Python Corner -verkkosivusto. Päivitetty 13.1.2019. Viitattu 22.2.2021. http://thepythoncorner.com/dev/how-to-create-a-watchdog-in-python-to-look-for-filesystem-changes/.

Python SQL Driver. 2017. Docs.microsoft verkkosivusto. Muokattu 8.8.2017. Viitattu 22.2.2021. https://docs.microsoft.com/en-us/sql/connect/python/python-driver-for-sql-server?view=sql-server-ver15.

Python SQL Driver - pymssql. 2020. Docs.microsoft verkkosivusto. Muokattu 10.6.2020. Viitattu 22.2.2021. https://docs.microsoft.com/en-us/sql/connect/python/pymssql/python-sql-driver-pymssql?view=sql-server-ver15.

Vona, S. 2019. How to Mount a Windows Share in Linux. Putorious -verkkosivusto. Päivitetty 12.10.2019. Viitattu 22.2.2021. https://www.putorius.net/mount-windows-share-linux.html.

Liitteet

Liite 1. Vuokaavio, ohjelman toimintasuunnitelma