Olli Hernesniemi

MQTT-viestintäprotokollan käyttö ohjelmoitavien logiikoiden kanssa

Opinnäytetyö
Syksy 2018
SeAMK Tekniikka
Automaatiotekniikka

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikan yksikkö

Tutkinto-ohjelma: Automaatiotekniikan tutkinto-ohjelma

Suuntautumisvaihtoehto: Sähköautomaatio

Tekijä: Olli Hernesniemi

Työn nimi: MQTT-viestintäprotokollan käyttö ohjelmoitavien logiikoiden kanssa

Ohjaaja: Jarkko Pakkanen

Vuosi: 2018 Sivumäärä: 39

Tämän opinnäytetyön aiheena on tutkia MQTT-viestintäprotokollaa ja kuinka sen avulla voidaan lähettää viestejä PLC-laitteilta pilvipalvelun kautta eteenpäin. Tutkimuksen keskeisenä osana on Beckhoffin TwinCAT 3 XAE -ohjemointiympäristön Tc3_IoTBase -koodikirjasto. Työssä käytettävä pilvipalvelu on Amazonin AWS-palvelu.

Teoriassa kerrotaan yleisesti ohjelmoitaviin logiikoihin, teolliseen internetiin ja pilvipalveluihin liittyvistä asioista. Siinä selvennetään, mitä MQTT-viestintäprotokolla tarkoittaa ja miten sitä käytetään.

Lopputuloksena on Android-pohjainen kännykkäsovellus, johon saa muuttujien arvoja reaaliaikaisesti PLC-laitteelta AWS-palvelun välittäjän kautta. Työssä käydään läpi sekä TwinCATillä tehty PLC-sovellus, että Android Studiolla tehty Android-sovellus ja niiden koodaukseen liittyviä asioita.

Avainsanat: IoT, MQTT, pilvipalvelu, Amazon, Beckhoff, Android Studio, ohjelmoitavat logiikat

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Automation Engineering

Specialisation: Electric Automation

Author: Olli Hernesniemi

Title of thesis: Using MQTT-Communication Protocol with Programmable Logics

Supervisor: Jarkko Pakkanen

Year: 2018 Number of pages: 39

The subject of this thesis was to research the MQTT-communication protocol and how to use it for sending messages from PLC-devices to a cloud computing service and forward. The research was based on the Tc3_IoTBase -PLC programming library, and particularly the part of Beckhoff's TwinCAT 3 XAE -programming environment, which is a part of Microsoft's larger Visual Studio -development environment. The cloud computing service used in this thesis was Amazon's AWS.

The theoretical part of the thesis concentrated on the basics of programmable logics, industrial internet and cloud computing. It examined the MQTT-communication protocol and how to use it.

The result of this thesis was an Android-based smart phone app, which shows the values of variables brought from a PLC-device through an AWS-service broker. The thesis introduced the PLC-program made using TwinCAT, the Android-app made with Android Studio, and the basics of coding related to the program and the app.

Keywords: IoT, MQTT, cloud computing, Amazon, Beckhoff, Android Studio, programmable logic controllers

SISÄLTÖ

Kuvaluettelo

Käytetyt termit ja lyhenteet

Anturi	Laite, joka muuntaa mitattavan prosessisuureen arvon siihen verranolliseksi viestiksi, joka sitten muutetaan ihmiselle ymmärrettäväksi arvoksi.
API	Application Programming Interface. Ohjelmointirajapinta.
AWS-Endpoint	AWS-palvelussa käytetty verkko-osoite, jolla yhdistäminen tiettyyn tiliin onnistuu
Big Data	Nimitys suurelle määrälle dataa, jota kerätään kaikkialta. Sitä kerätään mitä erilaisemmissa muodoissa ja keräämisen vauhti kiihtyy koko ajan.
DoS-hyökkäys	Palvelunestohyökkäys. Tarkoitus kaataa palvelu suurella määrällä liiallista liikennettä, jolloin palvelu ei pysty palvelemaan kaikkia asiakkaita.
I/O	Input ja Output. Sisääntulo ja ulostulo.
IDE	Integrated Develpoment Enviroment. Integroitu ohjelmistonkehitysalusta.
Metodi	Koodinpätkä, jolla on tietty toiminnallisuus ja se on helppo kopioida ja käyttää useita kertoja.
Payload	MQTT-viestintäprotokollassa viestistä käytetty nimi.
PUBACK	Viesti, jonka MQTT-protokollassa vastaanottaja lähettää QoS 1 -tasolla, kun viesti on saapunut perille.
PUBCOMP	Viesti, jonka MQTT-protokollassa vastaanottaja lähettää QoS 2 -tasolla, kun se on vastaanottanut PUBREL-viestin.
PUBREC	Viesti, jonka MQTT-protokollassa vastaanottaja lähettää QoS 2 -tasolla, kun viesti on saapunut perille.

PUBREL	Viesti, jonka MQTT-protokollassa alkuperäisen viestin lähettäjä lähettää QoS 2 -tasolla, kun se on vastaanottanut PUBREC-viestin.
PLC	Programmable Logic Controller. Ohjelmoitava logiikka.
Repository	Varastosivusto sovelluspakkauksille, josta valmiita sovelluspakkauksia on helppo hakea ja asentaa.
TCP/IP	Transmission Control Protocol/Internet Protocol. Internetliikenteen tietoliikenneprotokollien yhdistelmä.

1 JOHDANTO

1.1 Työn tausta

PLC-laitteilta ja antureilta kerättyä dataa halutaan siirtää nykyään mielummin suoraan pilveen, jotta sitä voidaan tallentaa ja jatkojalostaa sekä höydyntää siellä. Beckhoff on yksi suurista automaatioyrityksistä, joka on kehittänyt omaan TwinCAT-ohjelmointiympäristöönsä koodikirjaston, joka mahdollistaa tämän. Tästä aiheesta ei ole juuri materiaalia saatavana, varsinkaan suomen kielellä, joten sitä on hyvä tutkia ja selventää mahdollista hyötykäyttöä varten.

1.2 Työn tavoite

Työn tarkoituksena on tutustua Beckhoffin Tc3-IoTBase -koodikirjastoon, MQTT-viestintäprotokollaan ja miten näiden avulla saadaan lähetettyä viestiä pilvipalveluun, tässä työssä Amazonin AWS-palveluun. Tarkoitus on myös tutkia, kuinka ja millä tavoin tätä dataa voitaisiin hyödyntää.

1.3 Työn rakenne

Työn alussa käydään läpi teolliseen internetiin, pilvipalveluihin ja ohjelmoitaviin logiikoihin liittyvää yleistä teoriaa. Tämän jälkeen on yleistä tietoa työhön liittyvistä palveluista ja työssä käytetyistä sovelluksista sekä niiden tekijöistä. Luvussa 4 käydään ensin läpi MQTT-protokollaa ja sen jälkeen käsitellään läpi käytännön osuutta TwinCAT-sovelluksen osalta. Seuraavassa luvussa tutustaan käytännön osuuden Android-sovellukseen ja lopussa on yleistä pohdintaa sekä ideoita mahdollisiin jatkokehitysmahdollisuuksiin.

2 TEOLLINEN INTERNET, PILVIPALVELUT JA OHJELMOITAVAT LOGIIKAT

2.1 Teollinen Internet

Martinsuon ja Kärrin (2017, 10) mukaan teollisella internetillä tarkoitetaan komponenttien, tuotteiden, laitteiden, prosessien ja kokonaisten tuotantojärjestelmien sekä niihin liittyvien ihmisten kytkeytymistä toisiinsa ja internetiin siten, että niihin liittyvää informaatiota voidaan seurata ja jopa ohjata reaaliaikaisesti.

Yleismaailmallinen nimi teolliselle internetille on Internet of Things (IoT), Asioiden Internet. Myös Esineiden Internet on paljon käytetty termi kuvaamaan tätä ilmiötä.

Collin ja Saarelainen (2016, 43-44) esittävät, että teollisen internetin syntymää viimeisen kymmenen vuoden aikana on auttanut:

- Anturien hinnan romahdus, pienempi virrankulutus ja koko, parempi suorituskyky
- Internet on kaikkialla oleva hyödyke, verkotettujen laitteiden määrän räjähdysmäinen kasvu
- Big data, tallennuksen hinnan romahdus, analytiikka
- Kuluttajistuminen, käyttöliittymät
- Liiketoiminnan kiihtyvä muutosvauhti.

Toimiva teollisen internetin järjestelmä on monen osan yhteistyötä. Ensimmäisenä toimivat anturit, jotka tuottavat dataa laitteistosta ja sen lähiympäristöstä. Verkkoyhteys vie dataa eteenpäin, joko langallisesti tai langattomasti. Saatu data pitää tallentaa ja se tapahtuu tietokannan avulla, yleensä pilvipalveluun, joka vapauttaa yrityksen omien tietovarastojen ylläpitämisestä. Varastoidusta datasta analytiikkaohjelman algoritmit pyrkivät etsimään toiminnan kannalta hyödyttävää tietoa. Sovellus ja sen käyttöliittymä tuo ihmisten nähtäville saatua tietoa. Sovelluksia voidaan käyttää joko tietokoneen selaimen kautta tai kännykän mobiilisovelluksena.

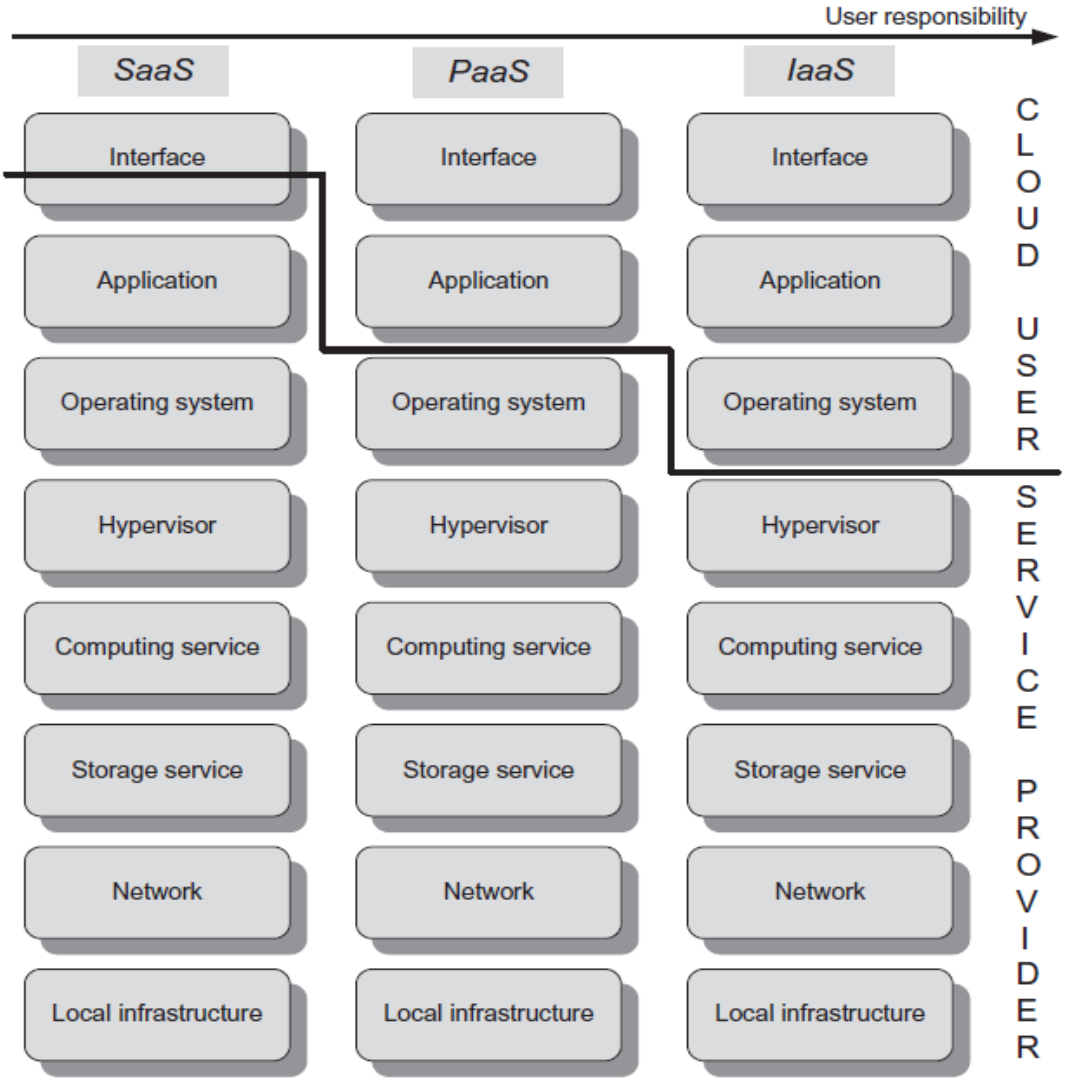
Kaikki tämä yhdistetään niin sanotulla alustalla. Viimeisenä on tietoturvajärjestelmä joka varmistaa, että ulkopuoliset eivät pääse käsiksi koko järjestelmään ja siellä oleviin tietoihin. (Collin & Saarelainen 2016, 143.)

Teollisella internetillä on useita sovellusalueita. Sen avulla voidaan valvoa ja hallita laitteistoa etänä, jolloin myös laitteiston optimointi ja päivittäminen onnistuu olematta laitteiston luona. Koneissa olevien anturien kautta saatua dataa analysoimalla voidaan esimerkiksi saada tietoa siitä kuinka koneen osat kuluvat, jolloin ne voidaan vaihtaa ennen kuin osan kanssa tulee ongelma. Koneista saatavaa dataa voidaan myös myydä eteenpäin, jolloin siitä tulee yritykselle aivan uusi liiketoiminta mahdollisuus. Älykäs tehdas on tulevaisuuden teollisen internetin visio, jossa tehtaan laitteet toimivat yhdessä itse itseään valvoen ja säätäen, jolloin ihminen vapautuu näistä tehtävistä kokonaan. (Collin & Saarelainen 2016, 61-62.)

2.2 Pilvipalvelut

Pilvipalveluilla tarkoitetaan internetin kautta hankittua tietokonekapasiteettia, sovelluksia tai muita palvelusuoritteita (Heino 2010, 32).

Pilvipalveluilla on viisi ominaispiirrettä. Yksi niistä on itsepalvelullisuus eli kyky hankkia palveluja palveluntarjoajilta ilman tarvetta kommunikoida ihmisen kanssa. Toinen on laaja saavutettuus verkon välityksellä eli pilvipalvelu on saatavilla verkon kautta ja sen käyttö onnistuu erilaisilla päätelaitteilla kuten tietokoneen ja kännykän kautta standardisoiduilla mekanismeilla. Kolmas on resurssien yhteiskäyttö eli kyky jakaa resursseja eri asiakkaille yhtä aikaa heidän tarpeidensa mukaan. Neäljäs on nopea joustavuus eli kyky skaalautua asiakkaiden tarpeen mukaan nopeallakin aikataululla. Viides on käytön tarkka mitattavuus eli kyky laskea resurssien käyttöä läpinäkyvästi esimerkiksi laskemalla käytetty tallennustila ja laskuttamijnen tarkalleen käytettyjen resurssien mukaan. (Mell & Grance 7.10.2009.)



Kuva 1. SaaS-, PaaS- ja IaaS-palvelun käyttäjän ja palvelun tuottajan vastuut (Marinescu 2013, 93)

Pilvipalvelut jaetaan kolmeen ryhmään teknisen toteutustavan perusteella. Kuva 1 havainnollistaa niiden erot käyttäjän ja palvelun tuotajien vastuiden avulla. SaaS eli Software as a Service on yksinkertaisin ja helpoin vaihtoehto käyttäjälle. Siinä käyttäjä saa käyttöönsä pilvipalvelusovelluksen, joka yleisesti on verkkoselainpohjainen. Kaikki muu jää palvelun tuottajan hoidettavaksi. Käyttäjällä ei kuitenkaan ole mahdollisuutta vaikuttaa taustalla olevaan infrastrukturiin. Erilaiset verkkoselainpohjaiset sähköpostiohjelmat, kuten Googlen Gmail, kuuluvat tähän ryhmään. PaaS eli Platform as a Service on sovellusalusta, jolle käyttäjä voi siirtää omia tai kolmannen osapuolen sovelluksia. Käyttäjä ei voi vaikuttaa taustalla olevaan infrastrukturiin, mutta sovelluksiin voi vaikuttaa haluamallaan tavalla. IaaS eli Infrastructure as a

Service on näistä vaihtoehdoista laajin. Siinä palvelun tuottaja lohkoaa virtuaalisista konesaleista tietyn kokoisia ja hintaisia osioita, joihin asiakas voi sitten asentaa oman käyttöjärjestelmänsä ja laittaa siihen haluamansa sovellukset. Tämä on käyttäjän kannalta vaativin vaihtoehto, koska se vaatii käyttäjältään paljon tuntemusta ja kokemusta aiheesta. Amazon Web Service (AWS) on tunnettu IaaS-palvelu. (Heino 2010, 50-54.)

Pilvipalvelut voidaan jakaa myös neljään ryhmää sen mukaan millaiselle käyttäjäkunnalle sitä jaetaan. Yksityinen pilvi on yhden organisaation käytössä ja sen ylläpidosta vastaa kyseisen organisaation oma tekninen tuki tai ulkoinen osapuoli. Yhteisöllinen pilvi on usean organisaation käytössä ja sen ylläpidosta vastaa joku organisaatioista tai ulkoinen osapuoli. Julkinen pilvi on kaikkien tai useiden eri yritysten käytössä, omistajuus kuuluu palvelun toimittajalle. Hybridipilveen kuuluu useampi pilvi, ne voivat olla kaikkia kolmea edellä mainittua mallia. Niitä sitoo yhteen standardisoitu ja patentoitu teknologia, jolla niiden välisten sovellusten ja datan siirto onnistuu. (Mell & Grance 7.10.2009.)

Yksi pilvipalveluiden pelätyin ongelma on tietoturva. Perinteisiksi uhaksi sanotaan sellaisia uhkia, joita ilmenee silloin kun mikä tahansa järjestelmä liitetään internetiin. Vastuu tietoturvan järjestämisestä pilvipalvelun toimittajan ja käyttäjän välillä on usein hämärä, mikä saattaa johtaa huonosti toteutettuun tietoturvaan. Yleinen uhka on myös tunnistautuminen ja auktorisointiprosessi. Tämä koskee yrityksiä ja organisaatioita, joissa kaikkilla ihmisillä ei ole tarvetta eikä valtuuksia kaikkeen tietoon, joten pilvipalveluun pitää järjestää eri tasoisia turvaluokituksia, joilla on erilaiset rajoitukset datan käyttöön. Kuten normaalit internetsivut, myös pilvipalvelujen tarjoajat ovat kohteita DDoS-hyökkäyksille eli palvelunestohyökkäyksille. (Marinescu 2013, 274.)

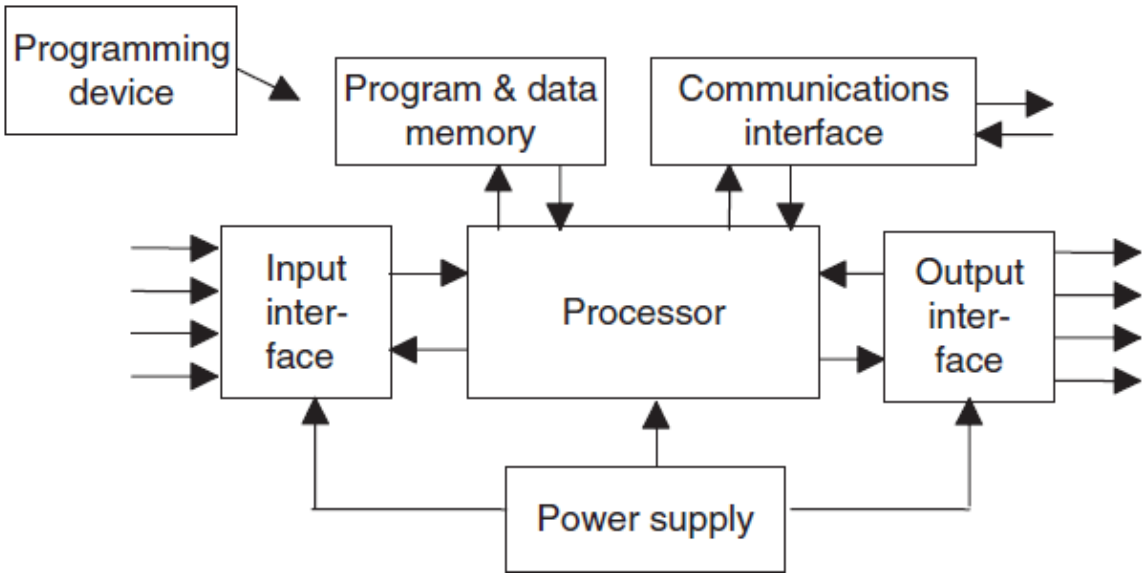
Toinen yleisesti huolta aiheuttava ongelma on pilvipalveluiden saatavuus. Pilvipalvelut voivatkin yllättäen olla käyttäjän ulottumattomissa, jos palvelun tuottajalla on tapahtunut sähkökatkos tai järjestelmissä on häiriö. Palvelun tarjoajat antavat kuitenkin usein palvelutasolupauksen. Se on prosenttiluku ja tarkoittaa sitä, kuinka suuren osa vuodesta, kuukaudesta tai päivästä pilvipalvelu on saatavilla tai kuinka hyvin data säilyy kyseisessä palvelussa. Tällä hetkellä palvelutasolupaukset ovat luokka 99,90—99,95 % saatavuuden osalta, mikä tarkoittaa, että keskimäärin pilvipalvelu

voi olla pois käytöstä noin minuutin päivässä. Säilyvyydessä eli pilvipalvelun kyvyssä säilyttää dataa ilman, että sitä katoa tallentuksen aikana, puhutaan 99,99 %:n lupauksista. (Salo 2014, 106.)

2.3 Ohjelmoitavat logiikat

Ensimmäiset ohjelmoitavat logiikat tulivat markkinoille vuosien 1968 ja 1969 aikana. Patentin ja ensimmäiset ohjelmoitavat logiikat kehitti Bedford Associates. Bedford Associates myös käytti ensimmäisenä englanninkielistä nimitystä PLC (Programmable Logic Controller), jota nyt yleisesti käytetään ohjelmoitavista logiikoista puhuttaessa. (Keinänen, Kärkkäinen, Metso & Putkonen 2000, 241-242.)

Ohjelmoitavan logiikan käyttötarkoitus on ohjata erilaisia reaaliaikaisia automaatioprosesseja. Se tuli korvaamaan entiset järjestelmät, jotka koostuivat suuresta määristä ajastimia ja releitä. Nyt tämä onnistuu yhdellä laitteella. (Keinänen ym. 2000, 241.)



Kuva 2. Havainnekuva ohjelmoitavan logiikan toiminnasta
(Bolton 2015, 4)

Kuva 2 havainnollistaa ohjelmoitavan logiikan toimintaa ja sen sisällä olevia komponentteja. Prosessori (Processor), joka on keskellä kuvaa, on kaikkea toimintaa

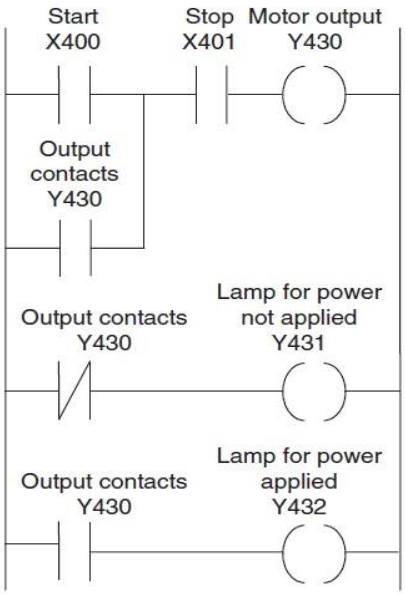
ohjaava komponentti. Se vastaanottaa ja tulkitsee tuloihin (Input) saapuvan datan, joka tulee antureilta. Antureilta saadun datan ja logiikan sisään ohjelmoidun logiikkaohjelman mukaisesti jaetaan toimintakäskyt lähtöjen (Output) kautta toimilaitteille. Logiikkaohjelma on tallennettu muistiyksiköön (Program & data memory). Virtalähde (Power supply) muuttaa sisääntulevan 230 voltin vaihtojännitteen laitteelle sopivammaksi 5—24 voltin tasajännitteeksi. Ohjelmointilaiteena (Programming device) voi toimia erillinen TouchPad-laite tai tietokone, josta ohjelmoitu logiikkaohjelma siirretään kaapelia pitkin muistiyksikköön. Kommunikointirajapinnan (Communication interface) kautta logiikka voi keskustella samassa verkossa olevien muiden logiikoiden kanssa. (Bolton 2015, 4-5.)

Logiikat voidaan jakaa parilla eri tavalla ryhmiin. Ne voidaan jakaa joko kompakteihin ja modulaarisiin logiikoihin tai pieniin, keskisuuriin ja suuriin logiikoihin. Kompaktit logiikat ovat pieniä logiikkoja, joilla ei ole tarkoitus ohjata kuin yhtä pienikokoista laitetta. Ne ovat halpoja, mutta niissä ei ole monia tuloja/lähtöjä ja niitä voi laajentaa rajallisesti. Modulaarinen logiikka on käytännössä loputtomiin asti laajentuva logiikka. Se koostuu erilaisista yksiköistä, joita voi lisätä ja poistaa tarpeiden mukaan. Pienen, keskisuuren ja suuren logiikan erot määritellään prosessorin kyvyllä käsitellä tuloja ja lähtöjä eli Inputien/Outputien (I/O) määrällä. Pienet logiikat pystyvät käsittelemään alle sata tuloa ja lähtöä, keskisuuret noin 100—500 ja suuret 500:sta ylöspäin. Kaikki logiikat kuitenkin koostuvat samoista osista, jotka kuvasta 2 löytyvät. Siihen päälle voi asentaa erilaisia erikoisyksiköitä kuten väyläyksiköitä. (Fonselius, Pekkola, Selosmaa, Ström & Välimaa 1996, 105-106.)

Logiikoiden ohjelmointi tehdään nykyään joko tietokoneella tai ohjelmointi/TouchPad-laitteella. Ohjelmointikieliä ja -tapoja on paljon erilaisia, monilla valmistajilla on omansa. Yleisimmät ohjelmointitavat ovat Ladder, Function Block Diagram ja Structured Text. (Bolton 2015, 111)

Ladderilla tehdyt ohjelmat koostuvat askelmista, joiden pitää alkaa Input-tiedolla ja loppua Output-lähtöön. Ohjelma lukee sitten askeleet ylhäältä alas päin järjestyksessä. Seuraavassa esimerkissä on yksinkertainen ladder-ohjelma. Kun start-nappulaa X400 painetaan, virta siirtyy Motor-outputiin ja moottori lähtee käyntiin. Kuvassa Stop-nappi on virheellisesti kuvattu normaalisti avoinna olevalla kontaktorilla,

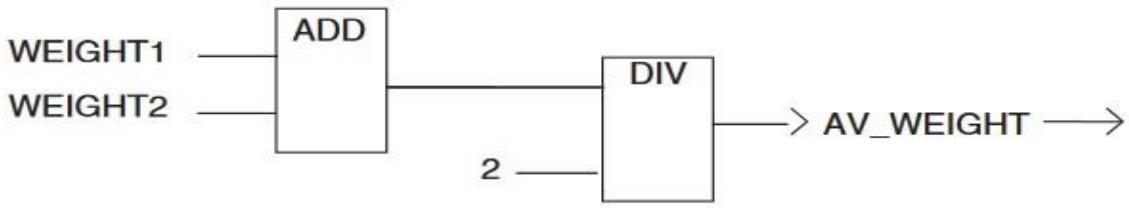
oikeasti siinä käytetään normaalisti kiinni olevaa kontaktoria, joka Stop-nappia painettaessa katkaisee virtapiirin. Tällöin moottorille ei enää tule virtaa ja se sammuu. Kun virtanappi on kerran painettu pohjaan, sitä ei enää tarvitse pitää pohjassa, koska sen alla oleva Output contacts Y430 pitää virran kiertämässä virtapiirissä niin kauan kuin moottori on päällä. Ala olevat kaksi riviä ovat lampuille, jotka kuvaavat, onko moottori käynnissä vai ei. Ylemmässä kahdesta alimmasta rivistä on normaalisti kiinni oleva kontaktori moottorin ulostulosta, jolloin tämä lamppu palaa kun moottori ei ole käynnissä. Alempi kahdsta alimmasta rivistä taas on päinvastainen eli kun moottori on käynnissä, niin tämä lamppu palaa. (Bolton 2015, 111-126.)



Kuva 3. Esimerkki Ladder-ohjelmointitavasta
(Bolton 2015, 124)

Function Block Diagramissa ohjelmaa kuvataan laatikoilla, joista data liikkuu läpi. Näihin laatikoihin tulee tuloja, jotka esitetään viivoina laatikon vasemmalla puolen ja lähtöjä, jotka esitetään viivoina laatikon oikealla puolen. Laatikon ylälaidassa on function blockin nimi, josta määrittyy sen sisäinen toiminta. Laatikon sisällä tulojen ja lähtöjen kohdalla voi olla lyhenteet, jotka kertovat, mitä tietoa niihin pitää tulla tai mitä tietoa niistä lähtee. Tuloista tulevien tietojen mukaan function block tekee tyypistään riippuvat toimenpiteet. Kun tehtävät on tehty, tarvittava tieto lähtee lähtöjen kautta eteenpäin. Seuraavassa kuvassa on kaksi painoa WEIGHT1 ja WEIGHT2, jotka laskettaan ensin yhteen. Sitten saatu arvo jaetaan kahdella. Näin saadaan

kahden painon keskiarvo AV_WEIGHT, tämä tieto lähtee vielä eteenpäin. (Bolton 2015, 126-134.)



Kuva 4. Function Block Diagramilla tehty kahden painon keskiarvon laskeminen (Bolton 2015, 127)

Structured Text -ohjelmointityyli muistuttaa vahvasti monia tavallisessa ohjelmoinnissa käytettyjä ohjelmointikieliä kuten Pascalia ja C-kieliä. Structured Textissä ohjelmaa luetaan ylhäältä alaspäin rivi kerrallaan. Seuraavassa kuvassa on yksinkertainen esimerkki ST-ohjelmointityylistä. Siinä käytetään IF-lausetta, jolla on helppo ohjelmoida esimerkiksi sellaiset tapaukset, joissa halutaan erilaisia lopputuloksia riippuen useiden input-signaalien tilasta. Kun Limit_switch1- ja Workpiece_Present-signaalit ovat päällä, Gate1 on auki ja Gate2 on kiinni. Kaikissa muissa tapauksissa Gate1 on kiinni ja Gate2 on auki. (Bolton 2015, 160-167.)

```
IF (Limit_switch1 AND Workpiece_Present) THEN
    Gate1 := Open;
    Gate2 := Close;
ELSE
    Gate1 := Close;
    Gate2 := Open;
END_IF;
```

Kuva 5. Esimerkki Structured Text -ohjelmointityylistä (Bolton 2015, 164)

3 KÄYTETYISTÄ PALVELUISTA JA SOVELLUKSISTA SEKÄ NIIDEN TEKIJÖISTÄ

3.1 Amazon.com

Amazon.com, Inc on perustettu vuonna 1994 Yhdysvalloissa Washingtonin osavaltiossa. Se on yksi maailman suurimmista nettiverkkokaupoista. Nettiverkkokauppa avautui vuonna 1995. Sillä on myös omia tuotteita kuten Kindle, e-kirjojen lukulaite ja Amazon Web Services -pilvipalvelualusta. (Amazon 2013, 5.)

3.2 AWS-palvelu

Amazon Web Services (AWS) on Amazonin pilvipalvelualusta. Se tarjoa laskentakapsiteettia, tiedon varastointia ja analysointia sekä tietokantoja. Siellä on myös kehitystyökalut tavalliselle koodille, koneoppimiselle, peleille, kännykkäsovelluksille sekä Internet of Things -työkaluja. (Amazon 2018a.)

3.3 AWS IoT- ja AWS IoT Core -palvelut

AWS Iot tarjoaa turvallisen kaksisuuntaisen kommunikaation internetiin yhdistettyille laitteille kuten antureille ja älylaitteille. Sen avulla voidaan kerätä dataa useista laitteista ja säilöä sekä analysoida koottua dataa. Sen tärkeimpiä osia ovat message broker, jonka avulla voi viestiä turvallisesti MQTT-protokollaa käyttäen ja rules engine, joka prosessoi viestejä ja mahdollistaa eri AWS-palveluiden yhdistämisen toisiinsa ja datan lähettämisen niiden kesken. (Amazon 2018c, 1-2.)

AWS IoT Core on yksi Amazon Web Servicen palveluista. Sen avulla on helppoa ja turvallista yhdistää laitteita pilvipalveluun ja muihin laitteisiin. Siihen voi liittää miljoonia laitteita ja niillä voi lähettää miljardeja viestejä turvallisesti ja luotettavasti AWS-päätepisteisiin ja muihin laitteisiin. Sen avulla on helppo käyttää muita AWS-palveluita ja rakentaa Internet of Things -sovellus, joka kerää, käsittelee, analysoi ja toimii laitteilta saadun datan mukaan. (Amazon 2018b.)

3.4 Beckhoff

Beckhoff Automation GmbH & Co. KG on maailmanlaajuisesti toimiva automaatiojärjestelmiä ja -komponentteja tekevä yritys. Se pohjaa järjestelmänsä PC-pohjaiseen ohjaustekniikkaan, sen tuotteisiin kuuluvat teollisuustietokoneet, kenttäväyläkomponentit, ohjauspaneelit, liikkeenohjaustuotteet sekä automaatiosovellusten ohjelmistot. Sen pääkonttori sijaitsee Saksan Verlissä ja Suomessa se toimii neljällä eri paikkakunnalla. (Beckhoff 6.9.2018.)

3.5 TwinCat 3 XAE

Twincat 3 XAE (eXtended Automation Engineering) on Beckhoffin PC-pohjainen ohjemointiympäristö, joka on integroitu osaksi isompaa Microsoftin Visual Studio -ohjelmistonkehitysympäristöä. Tämän ansiosta automaatiolaitteet pystytään ohjelmoimaan, konfiguroimaan, parametroimaan ja diagnosoimaan yhdellä ohjelmistolla. (Beckhoff 2012.)

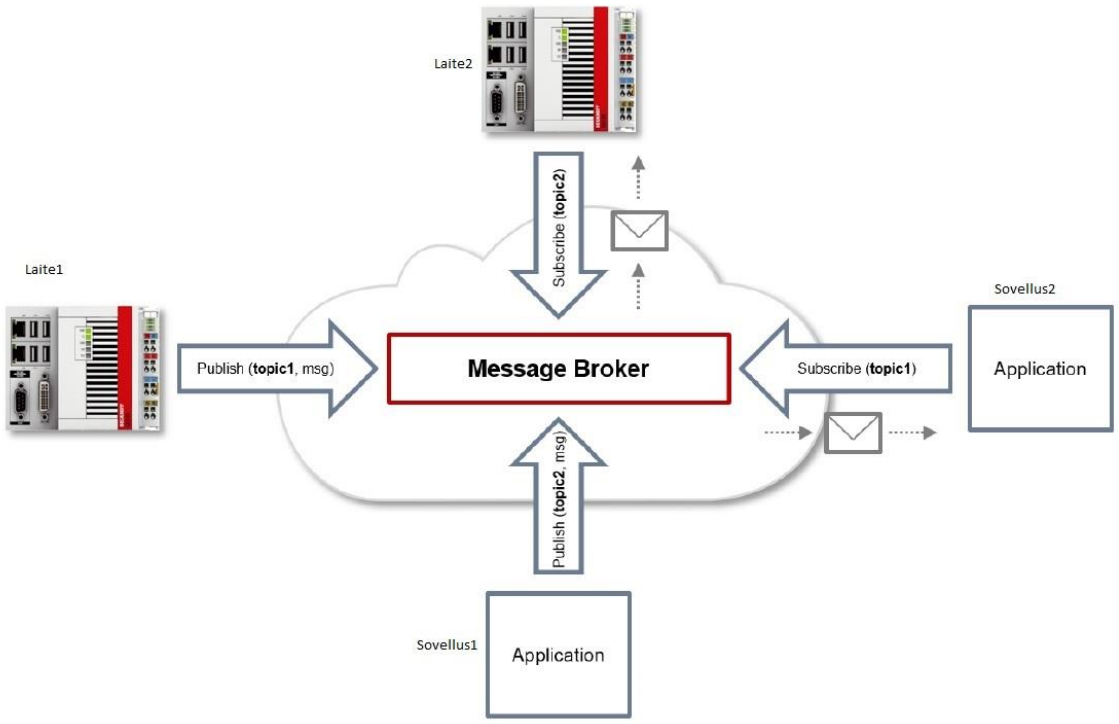
3.6 Android Studio

Android Studio on IntelliJ IDEA -alustaan pohjautuva virallinen integroitu ohjelmistonkehitys alusta (Integrated Development Enviroment, IDE) Android-sovelluksille. IntelliJ-alustan tehokkaan koodieditorin ja ohjelmoitityökalujen lisäksi se sisältää yhtenäisen alustan, jolla voi kehittää sovelluksia kaikille Android-laitteille. Android Studio sisältää emulaattorin, jolla voidaan emuloida mitä tahansa Android-pohjaista laitetta ja kaikkia eri Android-käyttöjärjestelmäversioita. Sillä ohjelmointiin voidaan käyttää Java-, Kotlin- ja C++-ohjelmointikieliä. (Developers 8.3.2018.)

4 MQTT JA SILLÄ YHTEYDEN LUOMINEN

4.1 MQTT

MQTT (Message Queueing Telemetry Transport) on julkaisija/tilaajaperustainen viestintäprotokolla. Keskiössä siinä on niin sanottu message broker (viestin välittäjä), joka toimittaa viestin lähettäjältä tai itsenäisestä ohjelmasta viestin tilaajalle. Message broker yhdistää julkaisijan ja tilaajan ilman, että kummankaan tarvitsee tietää toistensa verkko-osoitteita. Kommunkoinnin aikana laitteet eivät keskustele toistensa kanssa, vaan kaikki viestit kulkevat message brokerin kautta. MQTT-viestin sisältöä kutsutaan Payloadiksi, se voi olla mitä tahansa tiedon muotoa, kuten tekstiä, numeroita tai kokonaisia informaatiorakenteita. (Beckhoff 2017, 13.)



Kuva 6. Message broker ja sen kautta viestien välittäminen (Beckhoff 2017,13)

MQTT-protokollassa viestit organisoidaan niin sanottujen topicien eli aihepiirien mukaan. Tilaaja tilaa tietyn aihepiirin, jolloin ainoastaan sen aihepiirin viestit tulevat tilaajalle. Kuvassa 6 laite1 julkaisee topic1-aihepiiriin ja saman aihepiirin on tilannut sovellus2. Sovellus1 taas julkaisee topic2-aihepiirin ja laite2 on tilannut saman ai-

hepiirin. Viestit eivät sekoitu keskenään, koska ne kuuluvat eri aihepiireihin. Aihepiirit voidaan jakaa useisiin tasoihin, jolloin tietoa on helpompia jaotella, Koulu/Kerros2/Huone202/Lämpötila on esimerkki tästä, aihepiiri on Koulu, jolla on alatasot: kerrokset, kerroksissa olevat huoneet ja huoneista tulevat data. Kun tilaa aihepiirin, samalla tilaa kaikki alemmat tasot. (Beckhoff 2017, 13-14.)

MQTT-protokollassa viestien lähettämisen takaamisesta vastaa QoS (Quality of Service) -toiminto. Sekä lähettäjä että vastaanottaja valitsevat itselleen sopivan tason, mutta käytönnössä taso valikoituu viestin vastaanottajan tason mukaan. QoS jaetaan kolmeen tasoon:

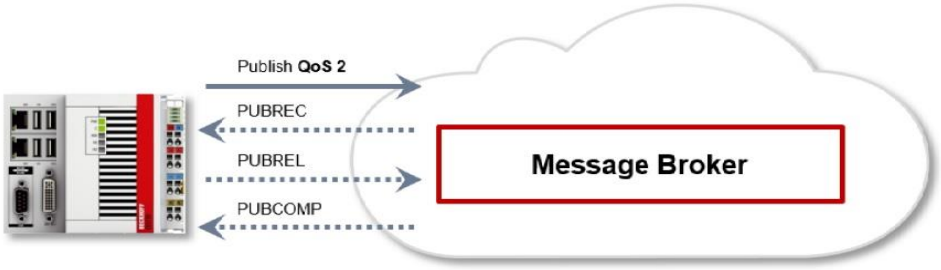
- 0 – ei useammin kuin kerran
- 1 – vähäintään kerran
- 2 – tasan kerran. (Beckhoff 2017, 13-14.)

QoS-taso 0 tarkoittaa sitä, että viestin lähettäjä ei saa kuittausta viestiin vastaanottamisesta. Viestiä ei lähetetä uudestaan. (Beckhoff 2017, 15.)

QoS-taso 1 varmistaa, että viesti saapuu vastaanottajalle vähintään kerran. Lähettäjä säilyttää viestin, kunnes on saanut PUBACK-viestin vastaanottajalta. Jos PUBACK-viesti ei saavu tietyn ajan sisällä, viesti lähetetään uudelleen. (Beckhoff 2017, 15.)

QoS-taso 2 on kaikista varmin, mutta myös hitain. Kun vastaanottaja saa viestin, se kuittaa sen PUBREC-viestillä. Kun lähettäjä on vastaanottanut PUBREC-viestin, lähettäjä voi hylätä alkuperäisen julkaisutiedon, koska se tietää, että vastaanottaja on saanut viestin kertalleen. Lähettäjä muistaa PUBREC-viestin sisäisesti ja lähettää PUBREL-viestin. Kun vastaanottaja on saanut PUBREL:in, se voi hylätä aiemmin muistetut tilat ja vastata PUBCOMP-viestillä. Viestinlähettäjä muistaa viestin sisäisesti, kunnes se on saanut PUBCOMP-viestin. Jos viestin kulku katkeaa jossain

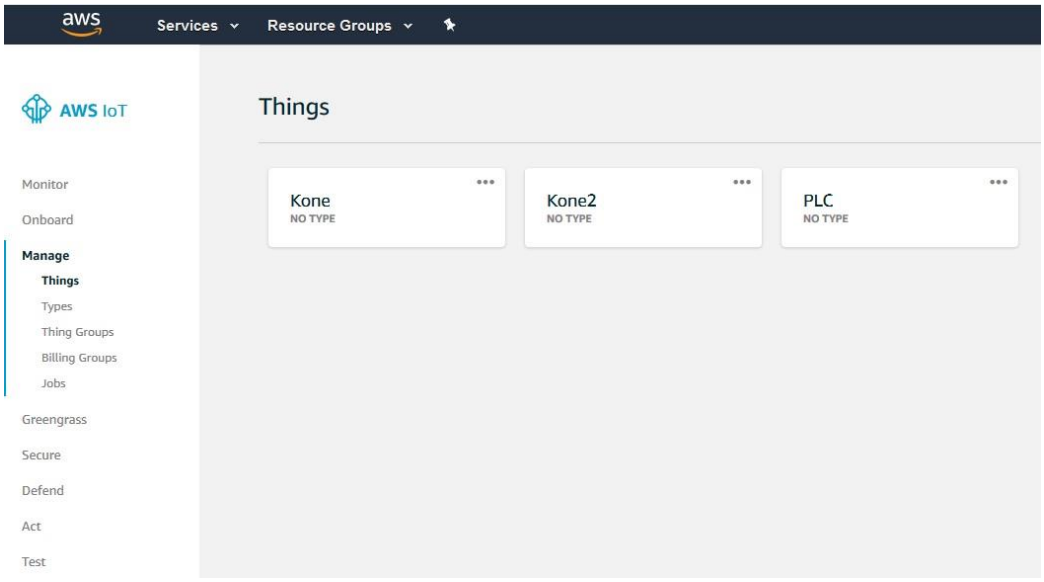
välissä, on lähettävän osapuolen tehtävä lähettää viimeisin viesti uudelleen tietyn ajanjakson jälkeen. (Beckhoff 2017,16.)



Kuva 7. Havainnekuva QoS-tasosta 2
(Beckhoff 2017, 16)

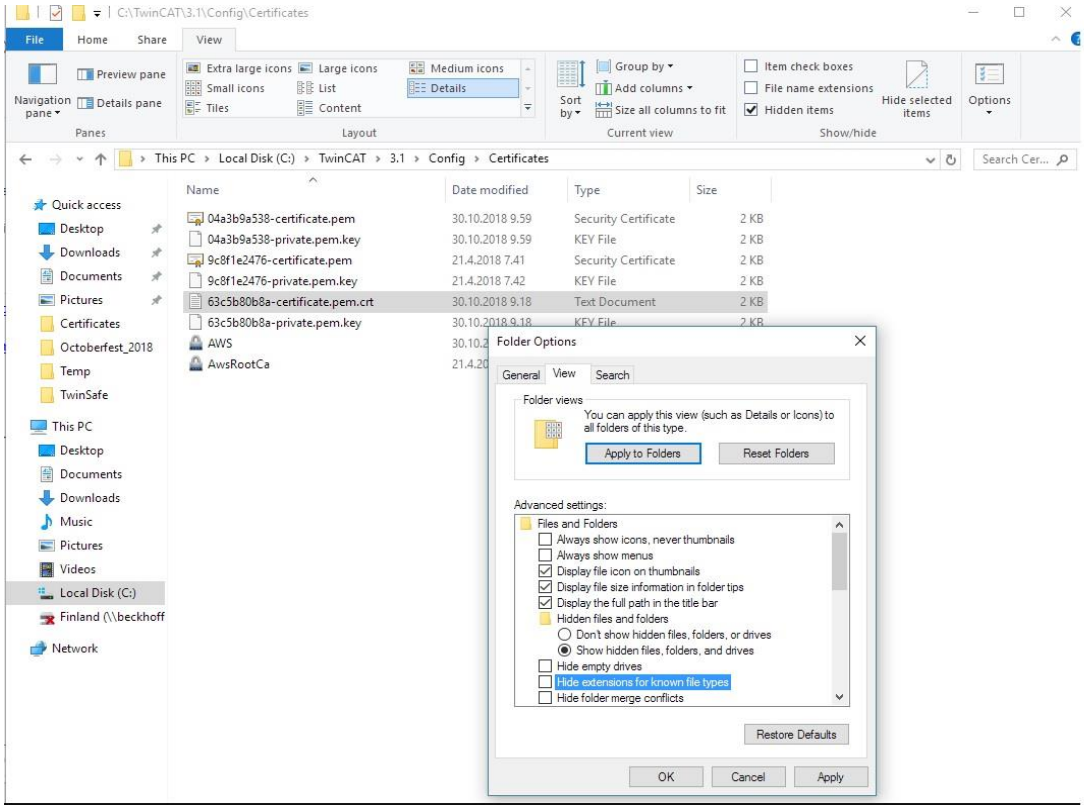
4.2 Yhteyden luonti AWS-palveluun

Yhteyden luominen AWS-palvelu vaatii Amazon Web Services -tilin. Se on ilmainen tietyin rajoituksin, suurinta osaa palveluista voi käyttää ilman, että joutuu maksamaan. Tärkein palvelu yhteyden luomiseen on IoT Core -palvelu. Siellä pitää luoda niin sanottu Thing, joka on virtuaalinen esitysmuoto fyysistä laitteesta tai loogisesta kokonaisuudesta, kuten PLC-sovelluksesta. Palvelusta löytyy ohjeet Thingin luomiseen.



Kuva 8. AWS IoT Coren -käyttöliittymä ja Things-valikko

Yhteyden luomisesta on haluttu tehdä mahdollisimman turvallista ja sen takia yhteys AWS-palvelun kanssa vaatii yhdistävältä laitteelta sertifikaatteja ja Policyn. AWS IoT käyttää yhteyksiensä suojaamiseen julkiseen avaininfrastruktuuriin standardin mukaisia X.509 digitaalisia sertifikaatteja. AWS Iot luo tarvittavan sertifikaattitiedoston ja avaintiedoston Thingin luonnin yhteydessä. Jokaisella Thingillä kannattaa olla omat sertifikaattitiedosto. Lisäksi tarvitaan, CA-sertifikaatti jonka voi ladata AWS-palvelun sivuilta. CA-sertifikaatti voi olla sama kaikilla Thingeillä. Tässä kohtaa kannattaa huomioida, että AWS-palvelusta ladattu sertifikaattitiedosto tulee tekstimuotoisena, jolloin se ei toimi. Tietokoneen kansioiden asetuksista pitää tarkistaa, onko seuraavassa kuvassa oleva valinta ”Hide extensions for known file types” valittuna. Jos se on valittuna, ei tiedoston perässä oleva piilotettu .txt-pääte näy. Kun tiedoston nimestä poistaa .txt-päätteen, se muuttuu sertifikaattitiedostoksi.



Kuva 9. Tietokoneen kansioiden asetukset

Toinen turvallisuuteen liittyvä asia on niin sanottu Policy. Sen avulla päätetään mitä resursseja ja toimintoja laitteet ja käyttäjät voivat käyttää. Sen avulla annetaan oi-

keudet yhdistää AWS-palveluun, tilata aihepiiri, julkaista aihepiiriin sekä vastaanottaa viestejä ja nämä samat oikeudet voidaan kieltää tietyiltä käyttäjiltä tai laitteilta. (Amazon 2018c, 138-141.)

Seuraavassa kuvassa on Policy, joka antaa tietylle käyttäjälle oikeuden yhdistää AWS IoT –palveluun sekä samalle käyttäjälle oikeuden julkaista aihepiirin topic/foo/bar. AWS-palvelun sivuilta löytyy ohjeet Policyjen luomiseen.

```
{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": ["iot:Connect"],
        "Resource": [
            "arn:aws:iot:us-east-1:123451234510:client/${iot:ClientId}"
        ]
    },
    {
        "Effect": "Allow",
        "Action": ["iot:Publish"],
        "Resource": [
            "arn:aws:iot:us-east-1:123451234510:topic/foo/bar/${iot:ClientId}"
        ]
    }]
}
```

Kuva 10. Esimerkki Policystä
(Amazon 2018c, 141)

Jotta Thing, Sertifikaatti ja Policy toimisivat yhdessä, ne pitää yhdistää toisiinsa AWS-palvelussa. Sertifikaatin päältä hiiren oikeaa näppäintä painamalla tulee esiin valikko, jossa on mahdollista yhdistää Thing ja Policy sertifikaattiin.

5 TWINCAT-SOVELLUS

TwinCatissä tehty sovellus pohjautuu Beckhoffin esimerkkisovellukseen joka on ladattavissa osoitteesta: https://infosys.beckhoff.com/. Sen toimiminen vaatii Tc3_IoTBase-kirjaston TwinCatissä. Sovelluksen idea on lähettää neljän muuttujan arvo AWS-palvelun välittäjän kautta Android-pohjaiseen älypuhelinsovellukseen, josta tietoa voi seurata reaaliaikaisesti. Koodi on tehty ST-ohjemointityylillä.

```
IF bSetParameter THEN
    bSetParameter := FALSE;
    fbMqttClient.stTLS.sCA := 'c:\TwinCAT\3.1\Config\Certificates\AWS.pem';
    fbMqttClient.stTLS.sCert := 'c:\TwinCAT\3.1\Config\Certificates\63c5b80b8a-certificate.pem.crt';
    fbMqttClient.stTLS.sKeyFile := 'c:\TwinCAT\3.1\Config\Certificates\63c5b80b8a-private.pem.key';
    fbMqttClient.sHostName := 'akmqkhv4enijl-ats.iot.eu-central-1.amazonaws.com';
    fbMqttClient.nHostPort:= 8883;
    fbMqttClient.sClientId:= 'Kone';
    fbMqttClient.ipMessageQueue := fbMessageQueue;
END_IF
```

Kuva 11. Kuvakaappaus TwinCat-sovelluksen AWS-palveluun yhdistämiseen vaadittavista parametreistä

Kuvassa 11 on parametrit, jotka vaaditaan AWS-palveluun yhdistämiseen. MQTT-sovellusta ja sen yhteysparametrejä pitää kutsua syklisesti, jotta yhteys välittäjään saadaan luotua ja ylläpidettyä. Tästä johtuen bSetParameter, joka on apumuuttuja parametrien sykliseen kutsumiseen, vaihdetaan välittömästi epätodeksi. fbMqttClient on function block, joka hoitaa kommunikaation MQTT-välittäjän kanssa ja pitää huolen, että yhteys on vain tasan yhteen välittäjään eikä sen enempään. Muut parametrit ovat sen inputteja eli niillä syötetään sille yhteyden luontiin tarvittavat tiedot.

bSetParameterin jälkeen seuraavat kolme muuttujaa ovat sertifikaattitiedostot. Ensimmäinen on CA-sertifikaattitiedosto, toinen on käytössä olevan Thingin sertifikaattitiedosto ja kolmas on sertifikaattitiedoston avaintiedosto. sHostName on AWS-tilin domain-nimi, jolla varmistetaan, että yhdistetään oikeaan tiliin. Se löytyy AWS IoT Core -palvelusta ja on aina sama samaan tiliin yhdistettäessä. nHostPort on aina 8883 AWS-palveluun yhdistettäessä yhteyden suojaamiseksi. MQTT-protokolla käyttää suurimmassa osassa yhteyksiä kahta eri TCP/IP-porttia: 1883 on suojaa-

mattomille yhteyksille ja 8883 on suojatuille yhteyksille. sClientId on nimi, jota yhdistävä laite käyttää, joten se on hyvä olla sama kuin Thingin nimi sekaannuksien välttämiseksi.

```
fbMqttClient.Execute(bConnect);
```

Kuva 12. Execute-metodi

Tämän jälkeen, kun parametrit on laitettu, pitää kutsua fbMqttClientin metodi Execute(), joka taustalla varmistaa yhteyden MQTT-välittäjään TwinCATin kanssa. bConnect on muuttujien esittelyn yhteydessä kirjoitettu aina olemaan tosi.

```
IF fbMqttClient.bConnected THEN
    IF NOT bSubscribed THEN
        bSubscribed := fbMqttClient.Subscribe(sTopic:=sTopicSub, eQoS:=TcIotMqttQos.AtMostOnceDelivery);
```

Kuva 13. IF-lause, joka varmistaa, ettei mitään tapahdu ilman, että yhteys välittäjään on luotu ja Subscribe-metodi

Itse toiminnalisuutta tehdessä kannattaa toiminnallisuus kokonaisuudessaan laittaa IF-lauseen sisään, jossa lauseen on ehtona fbMqttClient-function blockin output bConnected totena oleminen, kuten kuvassa 13. bConnected kertoo, onko yhteys välittäjään saatu luotua. Näin sovellus ei tee mitään ilman, että yhteys toimii ensin. Sovellus voi tilata aihepiirin Subscribe-metodilla, jos halutaan saada tietoa muualta ja muuttaa sovelluksen toimintaa saadusta tiedosta riippuen. Subscribe-metodi tarvitsee parametrit sTopic eli aihepiirin nimi ja eQoS eli haluttu QoS-taso.

```
fbMqttClient.Publish(   sTopic:= sTopicPubl,
                        pPayload:= ADR(sPayloadPubl), nPayloadSize:= TO_UDINT(LEN(sPayloadPubl))+1,
                        eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:= FALSE );
```

Kuva 14. Publish-metodi

Publish-metodin avulla tapahtuu viestien lähettäminen aihepiireihin. Siihen tarvittaan parametrit sTopic, pPayload, nPayloadSize ja eQoS. sTopic on haluttu aihepiiri. pPayload on Payloadiksi muutetun viestin osoite ja nPayloadSize on Payloadin koko bitteinä. eQoS on haluttu QoS-taso. bRetain ja bQueue eivät ole pakollisia parametrejä.

```
fbTimer(IN:=TRUE);
IF fbTimer.Q THEN // publish new payload every second
    fbTimer(IN:=FALSE);
    i := i + 1;
    i2 := i2 - 1;
    i3 := i3 + 1;
    i4 := i4 - 1;
    sPayloadPub1 := UDINT_TO_STRING(i);
    sPayloadPub2 := UDINT_TO_STRING(i2);
    sPayloadPub3 := UDINT_TO_STRING(i3);
    sPayloadPub4 := UDINT_TO_STRING(i4);
    //sPayloadPub := CONCAT('MyMessage', TO_STRING(i));
    fbMqttClient.Publish(   sTopic:= sTopicPub1,
                            pPayload:= ADR(sPayloadPub1), nPayloadSize:= TO_UDINT(LEN(sPayloadPub1))+1,
                            eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:= FALSE );
    fbMqttClient.Publish(   sTopic:= sTopicPub2,
                            pPayload:= ADR(sPayloadPub2), nPayloadSize:= TO_UDINT(LEN(sPayloadPub2))+1,
                            eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:= FALSE );
    fbMqttClient.Publish(   sTopic:= sTopicPub3,
                            pPayload:= ADR(sPayloadPub3), nPayloadSize:= TO_UDINT(LEN(sPayloadPub3))+1,
                            eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:= FALSE );
    fbMqttClient.Publish(   sTopic:= sTopicPub4,
                            pPayload:= ADR(sPayloadPub4), nPayloadSize:= TO_UDINT(LEN(sPayloadPub4))+1,
                            eQoS:= TcIotMqttQos.AtMostOnceDelivery, bRetain:= FALSE, bQueue:= FALSE );
    IF fbMqttClient.bError THEN
        // add your error logging here
        hrErrorOccurred := fbMqttClient.hrErrorCode;
    END_IF
END_IF

IF fbMessageQueue.nQueuedMessages > 0 THEN
    IF fbMessageQueue.Dequeue(fbMessage:=fbMessage) THEN
    fbMessage.GetTopic(pTopic:=ADR(sTopicRcv), nTopicSize:=SIZEOF(sTopicRcv) );
    fbMessage.GetPayload(pPayload:=ADR(sPayloadRcv), nPayloadSize:=SIZEOF(sPayloadRcv), bSetNullTermination:=FALSE);
    END_IF
END_IF
```

Kuva 15. TwinCAT-sovelluksen koodin loppuosa

Yhdistämistä varten tarvittavat parametrit sekä Execute-metodi pitää löytyä jokaisesta sovelluksesta. Loppu on sovelluskohtaista. Kuvassa 15 on loput tässä työssä käytetystä koodista. Siinä joka sekunnin välein tapahtuu seuraavat asiat: neljän muuttujan i, i2, i3 ja i4 arvo joko kasvaa tai laskee yhdellä. Niiden annetut lähtöarvot muuttujien esittelyssä on 0, 10000, 2500 ja 2500. Sitten sen hetkinen arvo muutetaan lukuarvosta string-muotoiseksi, jotta se voidaan lähettää eteenpäin. Sen jälkeen on Publish-metodit, joissa muuttujien arvot lähetetään aihepiireihin. Lopussa

on vielä viestien vastaanoton mahdollistava koodi, mutta se ei ole käytössä tässä työssä. Seuraavassa kuvassa on esitelty sovelluksen muuttujat.

PrgMqttCom*

```
1  PROGRAM PrgMqttCom
2  VAR
3      fbMqttClient : FB_IotMqttClient;
4      bSetParameter : BOOL := TRUE;
5      bConnect : BOOL := TRUE;
6
7      sTopicPub1 : STRING(255) := 'topic/Lampo';
8      sTopicPub2 : STRING(255) := 'topic/Kosteus';
9      sTopicPub3 : STRING(255) := 'topic/Valmiit';
10     sTopicPub4 : STRING(255) := 'topic/Osat';
11     sPayloadPub1 : STRING(255);
12     sPayloadPub2 : STRING(255);
13     sPayloadPub3 : STRING(255);
14     sPayloadPub4 : STRING(255);
15     i: UDINT;
16     i2: UDINT := 10000;
17     i3: UDINT := 2500;
18     i4: UDINT := 2500;
19     fbTimer : TON := (PT:=T#1S);
20
21     bSubscribed : BOOL;
22     sTopicSub : STRING(255) := 'topic';
23     {attribute 'TcEncoding':='UTF-8'}
24     sTopicRcv : STRING(255);
25     {attribute 'TcEncoding':='UTF-8'}
26     sPayloadRcv : STRING(255);
27     fbMessageQueue : FB_IotMqttMessageQueue;
28     fbMessage : FB_IotMqttMessage;
29
30
31     hrErrorOccurred : HRESULT; // contains the latest occurred error
32 END_VAR
```

Kuva 16. TwinCAT-sovelluksen muuttujien esittely

6 ANDROID-SOVELLUS

6.1 Yleistä Android-sovelluksesta

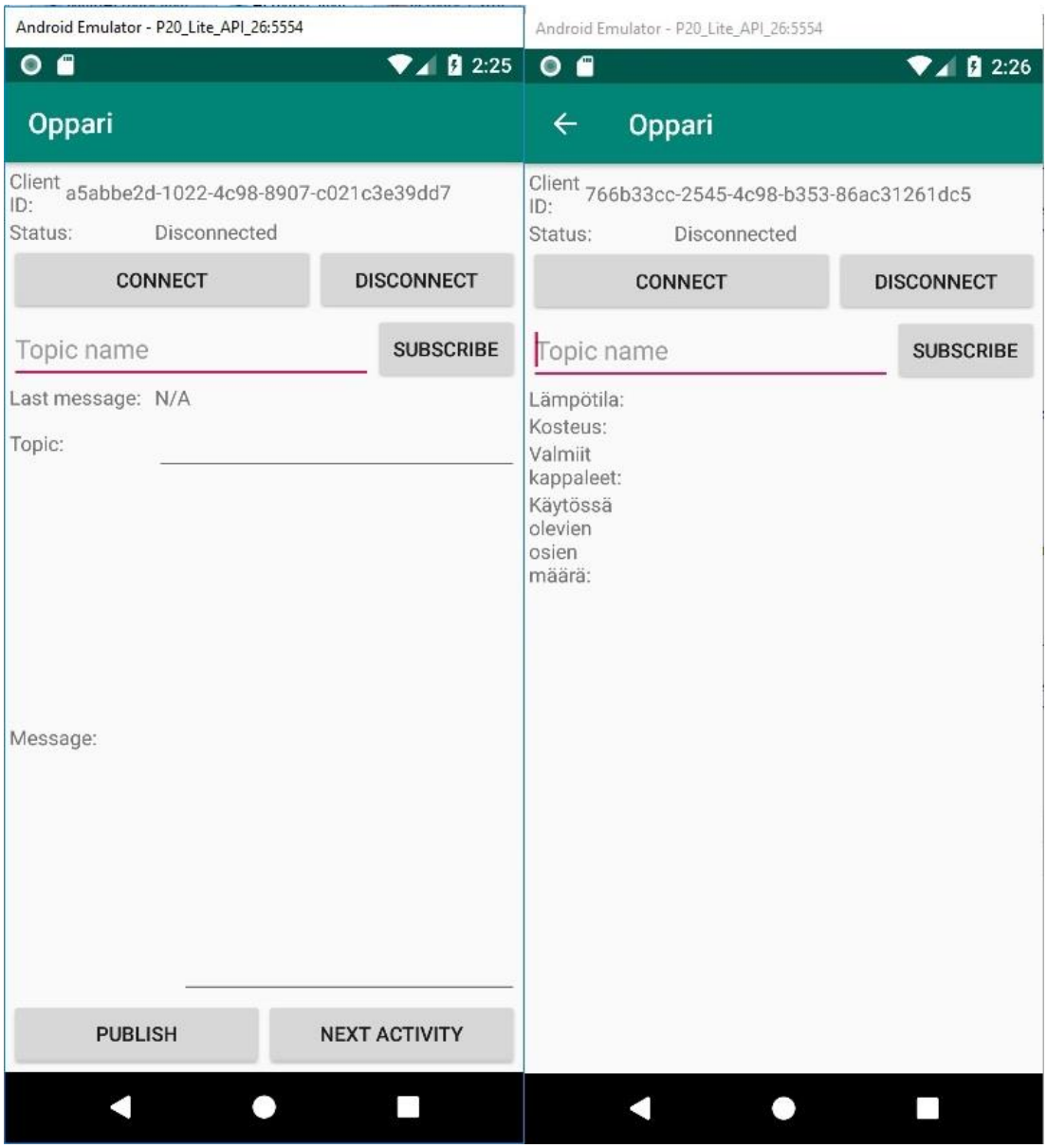
Android Studiolla tehty sovellus pohjautuu aws-sdk-android-samples -repositoryssä olevaan AndroidPubSub-mallisovellukseen. Samassa repositoryssä on myös muihin AWS-palveluihin liittyviä esimerkkisovelluksia. Se on saatavissa osoitteessa: https://github.com/awslabs/aws-sdk-android-samples.

Kuvassa 17 on Android-sovelluksen molemmat sivut vierekkäin. Ensimmäistä sivua käytetään yhteyden testaamiseen. Sillä voi tilata aihepiirin sekä nähdä uusimman lähetetyn viestin, joka saapuu tilattuun aihepiiriin tilaamisen jälkeen. Sovelluksella pystyy myös lähettämään haluamaansa aihepiiriin viestin. Toisella sivulla voi tilata aihepiirin ja nähdä reaaliajassa aihepiirin ala-aihepiirien /Lampo, /Kosteus, /Valmiit ja /Osat arvot. Tässä työssä käytetään aihepiirinä topicia, jolloin ala-aihepiirit ovat topic/Lampo, topic/Kosteus, topic/Valmiit ja topic/Osat.

Yläosa käyttöliittymästä on molemmilla sivuilla sama. Käyttöliittymä on järjestetty lineaarisesti tasoihin. Ensimmäisellä tasolla on Client ID:. Sen jälkeen näkyy sovelluksen satunnaisesti generoima merkkijoukko, joka on jokaisella sovelluksen avauskerralla eri. Toisella tasolla on Status:. Sen perässä oleva sana kertoo onko yhteys poikki, yhdistäminen kesken, uudelleen yhdistäminen kesken vai toimiiko yhteys. Kolmannella tasolla on Connect-nappi, jota painettaessa yhteys AWS-palveluihin luodaan, sekä Disconnect-nappi, jolla tämän yhteys katkaistaan.

Neljännellä ja viimeisellä yhteisellä tasolla on tekstikenttä, johon kirjoitetaan aihepiirin nimi, joka halutaan tilata, ja Subscribe-nappi, joka tallentaa aihepiirin nimen sovellukseen. Näiden pitää olla molemmilla sivulla, koska yhteys katkeaa sivua vaihdettaessa. Ensimmäisellä sivulla on tämän jälkeen viidennellä tasolla Last Message:. Sen jälkeen näkyy viimeisin viesti tilatusta aihepiiristä, joka on siihen lähetetty tilaamisen jälkeen. Sen alla on Topic:. Sen perässä on tekstikenttä, johon kirjoitetaan aihepiirin nimi, johon halutaan lähettää viesti. Sen alla on Message:. Tähän perään kirjoitetaan viesti, joka halutaan lähettää. Alimmaisena on Publish-nappi,

jolla kirjoitettu viesti lähetetään aihepiiriin ja Next Activity -nappi, jolla siirrytään toiselle sivulle. Toisella sivulla on viidennellä tasolla Lämpötila:. Sen perään tulee reaaliaikainen arvo valitun aihepiirin ala-aihepiiristä /Lampo. Seuraavilla tasoilla tapahtuvat samat asiat, mutta vain ala-aihepiiri muuttuu.



Kuva 17. Android-sovelluksen käyttöliittymän sivut 1 ja 2

```
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

import com.amazonaws.auth.CognitoCachingCredentialsProvider;
import com.amazonaws.mobileconnectors.iot.AWSIotKeystoreHelper;
import com.amazonaws.mobileconnectors.iot.AWSIotMqttClientStatusCallback;
import com.amazonaws.mobileconnectors.iot.AWSIotMqttLastWillAndTestament;
import com.amazonaws.mobileconnectors.iot.AWSIotMqttManager;
import com.amazonaws.mobileconnectors.iot.AWSIotMqttNewMessageCallback;
import com.amazonaws.mobileconnectors.iot.AWSIotMqttQos;
import com.amazonaws.regions.Region;
import com.amazonaws.regions.Regions;
import com.amazonaws.services.iot.AWSIotClient;
import com.amazonaws.services.iot.model.AttachPrincipalPolicyRequest;
import com.amazonaws.services.iot.model.CreateKeysAndCertificateRequest;
import com.amazonaws.services.iot.model.CreateKeysAndCertificateResult;

import java.io.UnsupportedEncodingException;
import java.security.KeyStore;
import java.util.UUID;
```

Kuva 18. Anrdoid-sovelluksen tarvitsemat pakkaukset

6.2 Sovelluksen koodista

Android Studiossa voi käyttää koodaukseen useita eri kieliä, mutta tähän työhön valikoitui käytettäväksi kieleksi Java, koska se on niistä yleisin. Javassa käytetään yleisesti valmiiksi ladattavia pakkauksia, jotka ovat yhteen kerättyjä class-tiedostoja. Näin kaikkea ei tarvitse lähteä tekemään nollasta ja keksiä asioita uudelleen. Kuvassa 18 on pakkaukset, joita tämä sovellus käyttää.

```
32      static final String LOG_TAG = MainActivity.class.getCanonicalName();
33
34      // --- Constants to modify per your configuration ---
35
36      // IoT endpoint
37      // AWS Iot CLI describe-endpoint call returns: XXXXXXXXXX.iot.<region>.amazonaws.com
38      private static final String CUSTOMER_SPECIFIC_ENDPOINT = "fgdj432fdsf13-ats.iot.eu-central-1.amazonaws.com";
39      // Cognito pool ID. For this app, pool needs to be unauthenticated pool with
40      // AWS IoT permissions.
41      private static final String COGNITO_POOL_ID = "eu-central-1:04548f3d2-2779-40f9-dj36-fd793bd566d2";
42      // Name of the AWS IoT policy to attach to a newly created certificate
43      private static final String AWS_IOT_POLICY_NAME = "test2";
44
45      // Region of AWS IoT
46      private static final Regions MY_REGION = Regions.EU_CENTRAL_1;
47      // Filename of KeyStore file on the filesystem
48      private static final String KEYSTORE_NAME = "iot_keystore";
49      // Password for the private key in the KeyStore
50      private static final String KEYSTORE_PASSWORD = "password";
51      // Certificate and key aliases in the KeyStore
52      private static final String CERTIFICATE_ID = "default";
```

Kuva 19. Android-sovelluksen parametrit AWS-palveluun yhdistämiseen

Android Studio tarvitsee hieman eri parametrit AWS-palveluun yhdistämiseen kuin TwinCAT. Endpoint, joka on sama kuin sHostName TwinCATissä, on ainoa sama parametri. COGNITO_POOL_ID-parametriä varten pitää AWS Gognito -palvelussa luoda uusi identity pool, joka avulla annetaan tarvittavat oikeudet käyttää AWS-palveluita väliaikaisille käyttäjille, jotka sovellus aina luo kun sitä käytetään. POLICY_NAME on sama asia kuin aiemmin luvussa 4.2 mainittu Policy. Siinä voi käyttää samaa Policyä kuin Thing käyttää tai luoda kokonaan uuden. MY_REGION on AWS-palvelusta valittava palvelin, jota haluataan käyttää. Eri palvelimet eivät voi keskustella keskenään, kaikien toiminnan pitää tapahtua saman pavelimen sisällä. Loput parametrit eivät ole pakollisia, niillä voi käyttää omia sertifikaatteja ja avaintiedostoja. Parametrien jälkeen on muuttujien esittely. Se on periaatteeltaan täysin samanlainen kuin TwinCATissä. Seuraavassa on kuva, jossa esitellään Android Studion muuttujia.

```
54      EditText txtSubcribe;
55      EditText txtTopic;
56      EditText txtMessage;
57
58      TextView tvLastMessage;
59      TextView tvClientID;
60      TextView tvStatus;
61
62      Button buttonConnect;
63      Button buttonSubscribe;
64      Button buttonPublish;
65      Button buttonDisconnect;
66      Button buttonNext;
67
68      AWSIotClient mIotAndroidClient;
69      AWSIotMqttManager mqttManager;
70      String clientID;
71
72      CognitoCachingCredentialsProvider credentialsProvider;
73
74      KeyStore clientKeyStore = null;
75
76      String keystorePath;
77      String keystoreName;
78      String keystorePassword;
79
80      String certificateId;
```

Kuva 20. Muuttujien esittely Android Studiossa

Tämän jälkeen alkaa metodi onCreate, joka on koko toiminnan pohja. Sen sisälle kirjoitetaan suurin osa koodista. Kuvassa 21 näkyy alkuosa onCreate-metodin sisällä olevasta koodista. Siinä on kirjoituskenttijen muuttujien, muuttuvien tekstikenttijen muuttujien ja nappijen muuttujien määrittely ja yhdistäminen käyttöliittymän vastaavaan muuttujaan sekä nappien toiminnallisuutta määrittelevien metodien suorittavat komennot. Siinä myös täysin satunnaisen ClientID:n luova komento sekä osa AWS-palveluun yhteyden luomiseen tarvittavista komennoista.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    nextActivity();

    txtSubcribe = (EditText) findViewById(R.id.txtSubcribe);
    txtTopic = (EditText) findViewById(R.id.txtTopic);
    txtMessage = (EditText) findViewById(R.id.txtMessage);

    tvLastMessage = (TextView) findViewById(R.id.tvLastMessage);
    tvClientID = (TextView) findViewById(R.id.tvClientID);
    tvStatus = (TextView) findViewById(R.id.tvStatus);

    buttonConnect = (Button) findViewById(R.id.buttonConnect);
    buttonConnect.setOnClickListener(connectClick);
    buttonConnect.setEnabled(false);

    buttonSubscribe = (Button) findViewById(R.id.buttonSubscribe);
    buttonSubscribe.setOnClickListener(subscribeClick);

    buttonPublish = (Button) findViewById(R.id.buttonPublish);
    buttonPublish.setOnClickListener(publishClick);

    buttonDisconnect = (Button) findViewById(R.id.buttonDisconnect);
    buttonDisconnect.setOnClickListener(disconnectClick);

    // MQTT client IDs are required to be unique per AWS IoT account.
    // This UUID is "practically unique" but does not _guarantee_
    // uniqueness.
    clientID = UUID.randomUUID().toString();
    tvClientID.setText(clientID);

    // Initialize the AWS Cognito credentials provider
    credentialsProvider = new CognitoCachingCredentialsProvider(
            getApplicationContext(), // context
            COGNITO_POOL_ID, // Identity Pool ID
            MY_REGION // Region
    );

    Region region = Region.getRegion(MY_REGION);

    // MQTT Client
    mqttManager = new AWSIotMqttManager(clientID, CUSTOMER_SPECIFIC_ENDPOINT);

    // Set keepalive to 10 seconds.  Will recognize disconnects more quickly but will also send
    // MQTT pings every 10 seconds.
    mqttManager.setKeepAlive(10);
```

Kuva 21. Android-sovelluksen ensimmäisen sivun onCreate-metodin alkuosa

Loppuosassa koodia on loput AWS-palveluun yhdistämiseen vaadittavista komennoista, toiminnallisuus omien sertifikaattien käyttämiselle sekä uuden sertifikaatin ja avaintiedoston luominen, jos omia ei ole. onCreate-metodin ulkopuolelle on jätetty ainoastaan nappien toiminnallisuutta ohjaavat metodit. Koska ne on siirretty sinne, koodista tulee selkeämmin luettavaa. Tämä ei vaikuta koodin toimintaan mitenkään. Kuvassa 22 on yksi näistä metodeista, Connect-napin toiminnallisuutta ohjaava metodi.

```
View.OnClickListener connectClick = new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        Log.d(LOG_TAG, msg: "clientID = " + clientID);

        try {
            mqttManager.connect(clientKeyStore, new AWSIotMqttClientStatusCallback() {
                @Override
                public void onStatusChanged(final AWSIotMqttClientStatus status,
                                            final Throwable throwable) {
                    Log.d(LOG_TAG, msg: "Status = " + String.valueOf(status));

                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            if (status == AWSIotMqttClientStatus.Connecting) {
                                tvStatus.setText("Connecting...");

                            } else if (status == AWSIotMqttClientStatus.Connected) {
                                tvStatus.setText("Connected");

                            } else if (status == AWSIotMqttClientStatus.Reconnecting) {
                                if (throwable != null) {
                                    Log.e(LOG_TAG, msg: "Connection error.", throwable);
                                }
                                tvStatus.setText("Reconnecting");
                            } else if (status == AWSIotMqttClientStatus.ConnectionLost) {
                                if (throwable != null) {
                                    Log.e(LOG_TAG, msg: "Connection error.", throwable);
                                }
                                tvStatus.setText("Disconnected");
                            } else {
                                tvStatus.setText("Disconnected");

                            }
                        }
                    });
                }
            });
        } catch (final Exception e) {
            Log.e(LOG_TAG, msg: "Connection error.", e);
            tvStatus.setText("Error! " + e.getMessage());
        }
    }
};
```

Kuva 22. Android-sovelluksen Connect-napin toiminnalisuutta ohjaava metodi

Kuva 23 on pieni ote käyttöliittymän koodista. Käyttöliittymän rakentaminen on helppoa Android Studiolla, koska haluamiaan käyttöliittymän osia kuten erilaisia nappuloita ja tekstikenttiä voi vain raahata tyhjälle sivulle. Android Studio luo niille koodin samalla valmiiksi. Sitten ei tarvi kuin muuttaa osien koko ja muut parametrit haluamikseen. Tässä sovelluksessa eri osat on järjestetty lineaarisesti tasoihin, jotta on saatu miellyttävämpi käyttöliittymä.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <TextView
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="3"
        android:gravity="center_vertical"
        android:text="Client ID:" />

    <TextView
        android:id="@+id/tvClientID"
        android:layout_width="250dp"
        android:layout_height="match_parent"
        android:layout_weight="5"
        android:gravity="center_vertical" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <TextView
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="2"
        android:gravity="center_vertical"
        android:text="Status:" />
```

Kuva 23. Esimerkki Android-sovelluksen käyttöliittymän koodista

7 POHDINTA

Työ osoittautui mielenkiintoiseksi ja haastavaksi. Työn tekijä sai hyvän kuvan TwinCAT Tc3-IoTBase -koodikirjaston toiminnasta sekä siitä kuinka MQTT-viestintäprotokolla toimii. Ongelmien takia datan hyödyntämistä ei sen sijaan ehditty kunnolla tutkia.

Haasteita aiheuttivat tiedon vähyys, koska aihe on uusi ja se on vielä enemmän yrityksille suunnattu. Koska selkeää tietoa ei ollut saatavilla, pari pientä ongelmaa aiheuttivat pitkät katkokset työn etenemisessä ja näin ollen veivät aikaa pois monimuotoisemman TwinCAT-sovelluksen ja Android-sovelluksen tekemisestä. Myös tuntematon ohjelmointiympäristö ja ohjelmointikieli Android Studiossa aiheuttivat vaikeuksia. AWS-palvelun käyttö ja sen käyttöohjeet sen sijaan olivat selkeitä.

Tehdyn sovelluksen ideana oli saada reaaliaikaista dataa PLC-logiikasta ja tässä onnistuttiin. Jatkojalostettuna sovellus voi toimia kannettavana valvomona, josta esimerkiksi yrityksen huoltohenkilökunnan on helppo seurata laitteiden toimintaa.

Ongelmista johtuneiden pitkien katkojen vuoksi sekä TwinCAT-sovelluksen että kännykkäsovelluksen kehittäminen jäi vähälle, siihen jää huomattavasti kehitysvaraa AWS-palveluiden kattavan tarjonnan vuoksi. Mahdollisia kehityskohteita on muuttujien datan tallentaminen pilvipalveluun ja niiden lajittelu sinne päiväkohtaisesti. Myös Lambda-palvelun kanssa olisi hyvät kehittämismahdollisuudet. Mahdollinen kehityskohde olisi lämpötilan ja muun tarvittavan tiedon seurantasovellus, jossa esimerkiksi lämpötilan laskiessa liian matallalle PLC-laitteella tuleva hälytys käynnistää pilvessä olevan Lambda-koodin, joka lähettää Push-ilmoituksen käyttäjän kännykkään, jolloin koko seurantasovelluksen ei tarvitse olla käynnissä hälytyksen tapahtumisaikana. Myös viestien vastaanottaminen TwinCAT-sovelluksella jäi ongelmien vuoksi testaamatta.

Työ antaa hyvän perustan opetusmateriaalille aiheesta, jos tätä aihepiiriä koskeva kurssi joskus toteutetaan. Siihen jää myös hyviä jatkokehitysideoita ja mahdollisia opinnäytetöiden alkuja.

LÄHTEET

Amazon. 2013. Form 10-K, Annual Report. [PDF-tiedosto]. Yhdysvallat: Amazon.com, Inc. [Viitattu 13.11.2018]. Saatavissa: http://pdf.secdatabase.com/1562/0001193125-13-028520.pdf

Amazon. 2018a. Amazon Web Services. [Verkkosivu]. Yhdysvallat: Amazon.com, Inc. [Viitattu 20.11.2018]. Saatavissa: https://aws.amazon.com/

Amazon. 2018b. Amazon Web Services Iot Core. [Verkkosivu]. Yhdysvallat: Amazon.com, Inc. [Viitattu 20.11.2018]. Saatavissa: https://aws.amazon.com/iot-core/

Amazon. 2018c. AWS IoT Developer Guide. [PDF-Tiedosto]. Yhdysvallat: Amazon.com, Inc. [Viitattu 21.11.2018]. Saatavissa: https://docs.aws.amazon.com/iot/latest/developerguide/iot-dg.pdf?shortFooter=true#what-is-aws-iot/

Beckhoff. 2012. TwinCAT 3 – eXtedended Automation (XA). [PDF-tiedosto]. Beckhoff Automation GmbH. [Viitattu 23.11.2018]. Saatavissa: http://download.beckhoff.com/download/document/catalog/Beckhoff_TwinCAT3_042012_e.pdf

Beckhoff. 2017. TC3 IoT Communication (MQTT). [PDF-tiedosto]. Beckhoff Automation GmbH & Co. KG. [Viitattu 13.11.2018]. Saatavissa: https://infosys.beckhoff.com/

Beckhoff. 06.09.2018. New Automation Technology. [Verkkosivu]. Beckhoff Automation GmbH & Co. KG. [Viitattu 23.11.2018]. Saatavissa: https://beckhoff.fi/

Bolton, W. 2015. Programmable Logic Controllers. [E-kirja]. Newnes. [Viitattu 04.07.2018]. Saatavissa Ebsco eBook Collection tietokannasta. Vaatii käyttöoikeuden.

Collin, J. ja Saarelainen, A. 2016. Teollinen Internet. Helsinki: Talentum.

Developers. 08.03.2018. Meet Android Studio. [Verkkosivu]. [Viitattu 23.11.2018]. Saatavissa: https://developer.android.com/studio/intro/

Fonselius, J., Pekkola, K., Selosmaa, S., Ström, M. & Välimaa, T. 1996. Koneautomaatio Automaatiolaitteet. Helsinki: Oy Edita AB.

Heino, P. 2010. Pilvipalvelut. Helsinki: Talentum.

Keinänen, T., Kärkkäinen, P., Metso, T. & Putkonen, K. 2000. Koneautomaatio 2 Logiikat ja ohjausjärjestelmät. Vantaa: WSOY Konetekniikka.

Marinescu, D. 2013. Cloud computing: theory and practice. [E-kirja]. [Viitattu 14.05.2018]. Saatavissa: Ebsco eBook Collection tietokannasta. Vaatii käyttöoikeuden.

Martinsuo, M. & Kärri, T. 2017. Teollinen internet uudistaa palveluliiketoimintaa ja kunnossapitoa. Helsinki: Kunnossapitoyhdistys Promaint ry.

Mell, P. & Grance, T. 7.10.2009. The NIST Definition of Cloud Computing. [Verkkojulkaisu]. Information Technology Laboratory: National Institute of Standards and Technology. [Viitattu 26.04.2018]. Saatavissa: http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf.

Salo, I. 2014. Big data & pilvipalvelut. Jyväskylä: Docendo Oy.