,Maija Somero

Uima-allasjärjestelmän mittaustiedon keruu ja analyysi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Smart Systems

Insinöörityö

3.11.2021

Tiivistelmä

Tekijä:	Maija Somero
Otsikko:	Uima-allasjärjestelmän mittaustiedon keruu ja analyysi
Sivumäärä:	32sivua + 14 liitettä
Aika:	3.11.2021
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintä tekniikka
Ammatillinen pääaine:	Smart Systems
Ohjaajat:	Lehtori Keijo Länsikunnas Manageri Sampo Nurmentaus

---

Tässä työssä käydään läpi sulautetun järjestelmän luominen ja sen tuottaman tiedon analyysi data-analyysillä. Käytettyjä teknologioita ovat Raspberry Pi 3 B, 2-wire-lämpötila-anturi, Amazonin AWS-pilvipalvelu ja sen sisällä mm. Thing-olio, Analytics-palvelu, QuickSIght-tiedon kuvantamispalvelu ja Cloud Watch -metriikkojen tarkkailupalvelu. Työ on tehty Voltigo Oy:lle, ja sen kautta loppuasiakas on Allas Sea Pool Helsinki. Työ toteutettiin kesällä 2021 ja tehtiin tiimityönä Metropolian R&D-osaston alla. Olin kesäharjoittelussa Metropolialla. Edellä mainittujen tekniikoiden lisäksi työssä pohditaan muita mahdollisia ratkaisuja, eli miten asiat olisi ainakin teoriassa voinut tehdä toisin. Myös AWS:n tietoturvaa ja skaalautuvuutta käsitellään jonkun verran. Näkökulmat ovat sekä yleinen että työhön liittyvä tarkastelu.

Avainsanat: sulautetut järjestelmät, data-analyysi

Abstract

Author:	Maija Somero
Title:	Pool system data collection and analysis
Number of Pages:	31 pages + 14 appendices
Date:	3 December 2021
Degree:	Bachelor of Engineering
Degree Programme:	Smart Systems
Professional Major:	Embedded Systems
Supervisors:	Sampo Nurmentaus, Project Manager
	Keijo Länsikunnas, Principal Lecturer

This Thesis goes through the creation of an embedded system and the analysis of the data it produces with data analysis. Technologies used include Raspberry Pi 3 B, 2-wire temperature sensor, Amazon's AWS cloud service and within it, technologies such as Thing instance, Analytics service, QuickSIght data visualization service and Cloud Watch metrics monitoring service. The project was commissioned by Voltigo Oy and the customer is Allas Sea Pool Helsinki. The work was carried out in summer 2021 and was done as a teamwork under Metropolia's R&D department. The author of the Thesis did her summer internship at Metropolia. In addition to the above-mentioned techniques, the work considers other possible solutions, i.e. how things could have been done differently, at least in theory. AWS's security and scalability are also addressed to some extent, both in general and in relation to the study.

Keywords: Embedded Systems, Data Analysis

Sisällys

Lyhenteet

AWS: Amazonin pilvipalvelu

Raspi: Raspberry Pi (tässä tapauksessa model 3 B).

QS: Amazonin QuickSight käytetty BI-työkaluna datan kuvaamiseen.

1 Johdanto

Tämä insinöörintyö on tehty Metropolian harjoitteluna sVoltigo Oy:lle siten, että loppuasiakas on Allas Sea Pool Helsinki. Aihe on uima-allasjärjestelmän mittaustiedon keruu ja analyysi. Alkuasetelma oli seuraava: Allas Sea Pool halusi uima-altailtaan mittaustietoa altaiden lämpötiloista ja veden energian kulutuksesta. Tähän tehtävään valittiin prototyypin omaisesti RaspberryPi-rauta ja 1-wire-tyyppisiä lämpötila-antureita. Kun rautapuoli saatiin valmiiksi, alettiin miettiä tiedon tuontia jollekin alustalle. Tarkoituksena on analysoida tietoa. Tähän valittiin Amazonin AWS, josta löytyy Internet of Things -osuus. AWS:n puolelle pystyi luomaan omia itsenäisiä Thing-olioita. Käytimme tätä yhdessä MQTT-protokollan kanssa. Altaalla olevalla raudalla oli Python-koodi, joka lähetti tietoa altaan olosuhteista suoraan AWS:lle. Meillä oli putki rakennettu altaalta AWS:ään. Tietysti myös lämpötilan lukuun tarvittiin Python-koodi. AWS:n puolella tiedot siirtyivät automaattisesti tietovarastoon, josta niitä luettiin AWS:n QuickSight-nimiseen toimintoon. QuickSight on eräänlainen BI-työkalu, jolla voi muodostaa analyysejä tiedosta ja visualisoida dataa. Työn aikana pohdittiin paljon AWS:n hyödyllisyyttä ja käytettävyyttä. Oli myös tilanteita, jossa mietittiin, pitäisikö systeemi tehdä kokonaan ilman AWS-tuotteita. Osin meillä oli haasteita päästä mukaan AWS:n toimintalogiikkaan nopeasti. Kukaan tiimin jäsenistä ei ollut tehnyt ennen mitään AWS:llä. Myös projektiin kuuluivat energian kulutuksen laskut, jotka saatiin veden lämpötilan muutoksista. Lisäksi kokeilimme CloudWatch-omianisuutta, jossa voi muodostaa hälytyksen, jos joku arvo ylittää asetetun rajan.

Lisäksi työ käsittelee hitusen AWS:n tietoturvakysymyksiä ja skaalautuvuutta. Molemmat ovat hyvin tärkeitä ominaisuuksia pilvipalvelussa. Lisäksi esimerkiksi EU-alueella on astunut voimaan laki tietosuojasta, joka vaikuttaa tällä alueella toimiviin toimijoihin suoraan lain puitteissa. Tietoturvakysymykset ovat myös Amazonin AWS:lle hyvin tärkeitä. Kun yleinen trendi on ollut, että yritykset siirtä-

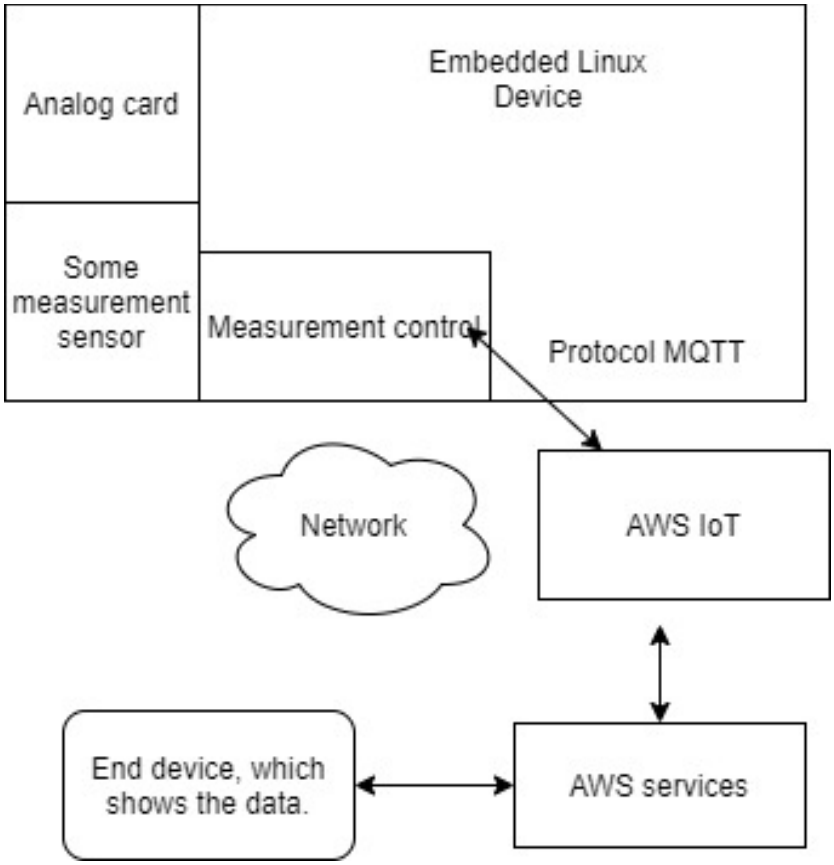
vät toimintaansa pilveen, niin on syntynyt hyvin laaja käsitys ja tarpeet, mitä tulee tietoturvaan. Loppupuolella on pohdintaa kesken jääneistä ominaisuuksista ja siitä, mitä ei toteutettu, mutta pohdittiin kesken projektin.

Työskentelymetodina meillä oli löyhä Kanban. Pdimme yleensä puhelimitse yhden tiimitapaamisen alkuviikosta. Tiimissä työskenteli Petteri Koivu (projekti-insinööri Metropolialla), minä ja koodaava manageri Sampo Nurmentaus (Metropolia).

Haluan kiittää työn ohjaamisesta Sampo Nurmetausta ja Keijo Länsikunnasta, hyvistä neuvoista työssä Petteri Koivua ja itse Allas Sea Pool Helsinkiä ja Voltigo Oy:tä mielenkiintoisista hetkistä it-tuotannon parissa.

2 Arkkitehtuuri, systeemin osat, Python

Ensimmäisenä työpäivänä minulle kuvailtiin järjestelmää, jonka olisimme tulossa tekemään. Tämän jälkeen minulle annettiin tehtäväksi piirtää karkea kuva systeemistä. Tuo kuva on nyt liitetty tähän työhön. En muista, että olisin ennen tehnyt arkkitehtuurikuvaa yksin mistään järjestelmästä.

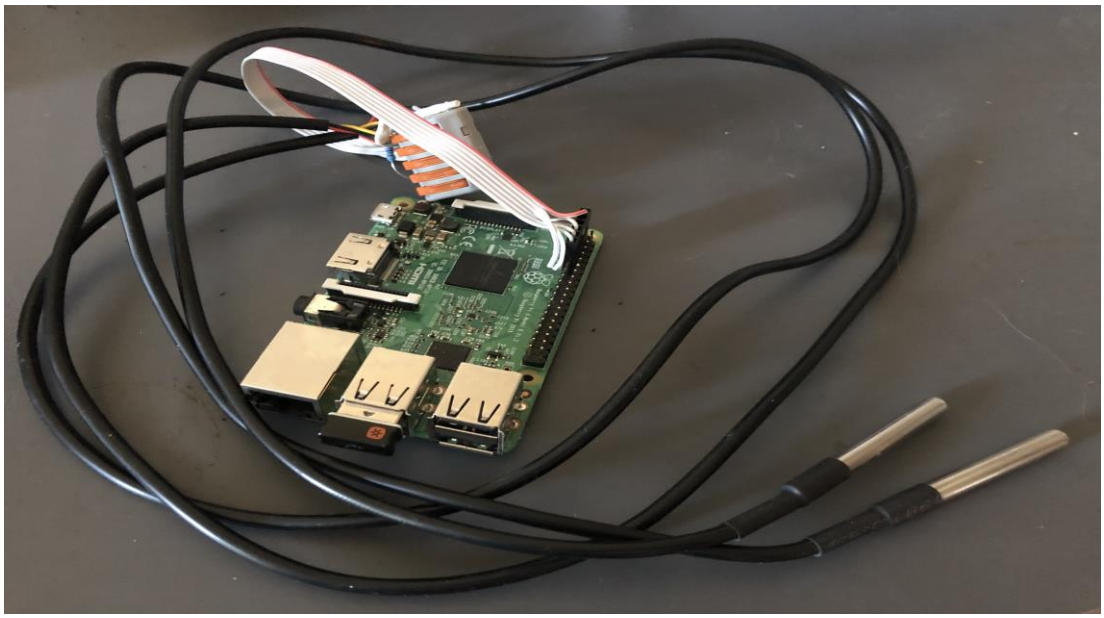


Kuva 1. Arkkitehtuurikuva järjestelmästä, tällä kuvalla lähdimme liikkeelle projektissa

2.1 RaspberryPi mikrokontrolleri ja 1-wire-anturit

Työssä käytettiin mikrokontrollerina RasperryPi-rautaa, joka on hyvä prototyypittämiseen. Raspberry Pi on yhden piirilevyn tietokone. Sen on kehittänyt brittiläinen Raspberry Pi Foundation, jonka tehdas on Walesissa. Puoliväliin 2016 mennessätietokonetta oli myyty noin 10 miljoonaa kappaletta.[1.] Maaliskuuhun 2021 mennessä RasPin malleja on myyty yli neljäkymmentä miljoonaa.[2; 3.]

Raspi on halpa ja suhteellisen tehokas ARM-pohjainen tietokone. Erikoista siinä on, että yleensä käytetään (esim. HDMI yhteydellä) ulkoista erillistä näyttöä. Näin teimme mekin. Ensimmäisellä tiimitapaamisellamme tapasin tiimin jäsenet ensimmäistä kertaa, ja me saimme muutaman Raspin ja 1-wire-lämpötila-anturia mukaan. Petteri Koivu teki meille lämpötila-anturiasetukset. Kuvassa 2 on Raspi ja anturit, kun ne on kytketty oikein ja systeemi on toimintakuntoinen.



Kuva 2. Raspi 1-wire-antureilla

Raspiin tarvittiin SD-kortti, jolla on jokin käyttöjärjestelmä ladattuna, jotta se toimii. Itse latasin Raspin sivuilta imagerin, jolla latasin SD-kortille oikean käyttöjärjestelmän (Debian) ja vain avasin Raspin käyttöön. Imageri ei ollut paras mahdollinen. Se saattoi välillä pysähtyä ja näin ollen tarvitsi aloittaa alusta. Mutta sain silti muodostettua yhden käyttökelpoisen kuvan SD-kortille. Kuvassa 3 on Raspista teknisiä tietoja.

Specifications

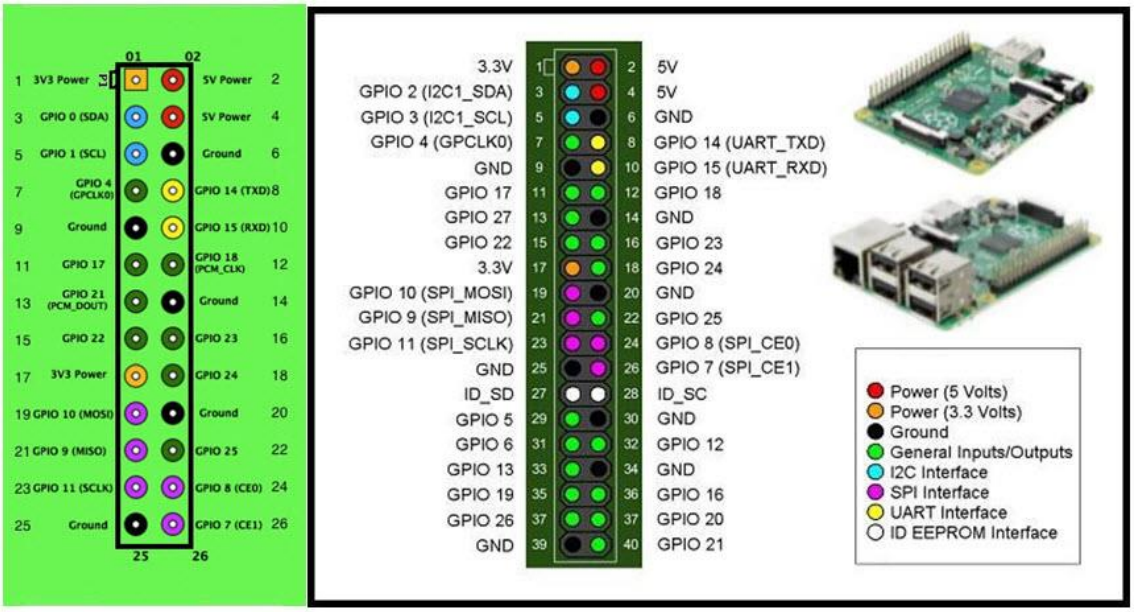
Processor:	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
Memory:	1GB LPDDR2 SDRAM
Connectivity:	■ 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE ■ Gigabit Ethernet over USB 2.0 (maximum throughput 300Mbps) ■ 4 × USB 2.0 ports
Access:	Extended 40-pin GPIO header
Video & sound:	■ 1 × full size HDMI ■ MIPI DSI display port ■ MIPI CSI camera port ■ 4 pole stereo output and composite video port
Multimedia:	H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
SD card support:	Micro SD format for loading operating system and data storage
Input power:	■ 5V/2.5A DC via micro USB connector ■ 5V DC via GPIO header ■ Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment:	Operating temperature, 0–50°C
Compliance:	For a full list of local and regional product approvals, please visit www.raspberrypi.org/products/raspberry-pi-3-model-b+
Production lifetime:	The Raspberry Pi 3 Model B+ will remain in production until at least January 2023.

raspberrypi.org

Kuva 3. Raspin teknisiä tietoja

Lämpötila-anturi työssä on siis 1-wire-tyyppinen. Kuten kuvasta näkyy, siinä on kaksi johtoa, jotka mittaavat lämpötilaa. Eri tarpeisiin tulleita antureita palloteltiin jonkun verran. Suunnitelma on liitteessä 1.

Lämpötila-antureita asennettaessa piti ottaa huomioon GPIO-asetelmat. Käytimme kuvan 4 pin-kohita 1, 7,9,1. Meillä oli siis virta, maa ja kaksi GPIO-piniä: GPIO 4 ja GPIO 17.



Kuva 4. Raspi 3 B GPIO pinnit

Saadaksemme tämän asetelman toimimaan tarvitsi ajaa seuraavia asetusskriptjä komentopäätteeltä. Etsittiin listauksesta kyseessä olevat kaksi lämpötila-anturi-ID:tä. Seuraava kymmenalkuinen koodi on lämpötilamittarin ID.

Ja näin edellisellä komennolla saadaan näytölle lämpötila kerrottuna tuhannella. Tämä on ensimmäinen vaihe kohti lämpötilakoodia. Alla komentorivikomennot.

sudo modprobe w1-gpio

sudo modprobe w1-therm

cd /sys/bus/w1/devices/

ls

```
cat /sys/bus/w1/devices/10-000802b4ba0e/w1_slave
```

.

2.2 Amazon pilvipalvelu AWS

IT-jätti Amazon perusti oman pay-as-go-maksumetodin pilvipalvelun vuonna 2000. Se on hyvin käytetty ja tunnettu palvelu. Amazon markkinoi tuotteitaan helppokäyttöisyydellä, menojen skaalautuvuudella, vähenevillä materiaalikustannuksilla (ei omaa palvelinsalia) ja mahdollisuudella ruveta globaaliksi toimijaksi silmän räpäyksessä.

Meille AWS valikoitui projektin luonteen takia. Haluttiin jotain toiminnallista valmiiksi nopeasti ilman suuria rahallisia panoksia. Pitkin projektia meillä oli monta pohdintaa, halutaanko tätä palvelua oikeasti käyttää vai ei. Jo alkumetreillä tuntui hieman vastahankaiselta. Amazonin AWS-palvelut eivät olleet ensikertalaiselle niin helposti lähestyttäviä kuin olisi voinut toivoa.

AWS:stä käytimme Internet of Things -osiota (Thing-olioita säilömään tietoa), Iot Corea, jossa Thingit muodostettiin, QuickSightia visualisoimaan dataa ja CloudWatchia tarkkailemaan raja-arvoja (esim. veden lämpötila) sekä siihen liittyvää SNS-palvelua, jolla voi lähettää muun muassa sähköpostia, jos jokin raja ylittyy.

Lisäksi tietysti Docker cotainerissa olevaa laskentakonetta ja S3-bukettia. S3 on tapa tallentaa tietoa AWS:n puolella.

Kun projekti eteni, kävi selväksi, että ainakin prototyypittämiseen AWS todella oli halpa. Kuukaudelta tuli maksuja noin muutaman dollarin verran. Lisäksi projektissa oli ajatuksena, että jos Helsingin altaille saadaan hyvä toteutus, niin Allas Sea Poolin laajennus ajatuksissa tätä systeemiä voisi käyttää myös muualla, minne Allas Sea Pool olisikin laajentumassa. Näihin tarpeisiin tuntui AWS sopivalta kaikista epäilyksen siemenistä huolimatta.

Thing instanssi

AWS IoT Core

Kuva 5. AWS:n logot Iot Corelle ja Thing-oliolle

AWS manosti kovasti sivuillaan, että kehittäjä pääsee alkuun jo muutaman kymmenen minuutin tutoriaalilla. Eikä tuo täysin tekaistua ollutkaan. Suurimmat ongelmat kohtasimme AWS:n puolella oikeuksien hallinnassa, joka osoittautui suhteellisen soiseksi maastoksi.

2.3 Python-kieli

Python on tulkattava ohjelmointikieli. Tulkattavalla ohjelmointikielellä tarkoitetaan, että ohjelmaa ei käännetä, joten se on heti valmis suoritettavaksi. Tämä nopeuttaa ohjelmoijan työtä. Tosin sitten kolikon toinen puoli on, että ohjelman suorituskyky ei vastaa käännettävän esim. C-kielisen ohjelman toimintaa. Nykyään useat yliopistot käyttävät Pythonia ensimmäisenä ohjelmointikielenä, jota opetetaan fukseille, mukaan luettuna Helsingin yliopisto. Pythonin syntaksi on suhteellisen ”kevyt”, joten se on helppo oppia. Lisäksi Pythonia käytetään paljon tieteelliseen laskentaan esim. NumPy- ja SciPy-kirjastoilla. Pythonille on rajapintoja tietokantoihin (MySQL, PostgreSQL). Myös verkkosivujen tekeminen Pythonilla on yleistynyt ja siihen on olemassa kehitysympäristöjä (esim. Django).

Kuva 6. Python-kielen symboli. [6.]

Pythonia voi käyttää olio-ohjelmointiin, funktionaalisesti ja proseduraalisesti. Olio-ohjelmoinnin ytimessä on, että toistettavat ominaisuudet tehdään luokkina, joilla on metodit ja attribuutit eli ominaisuudet ja toiminnot. Pääohjelmassa kutsutaan luokan ilmentymiä ja niiden toiminnallisuuksia ja ominaisuuksia. Olio-ohjelmointi kehitettiin poistamaan ongelmia, joita oli aiemmissa tyyleissä, mutta olio-ohjelmointi ei ole tie taivaaseen. Siinä on omat kipupisteensä.

Funktionaalinen ohjelmointi lähtee siitä, että kaikki ominaisuudet ja toiminnot ohjelmassa on esitetty funktioiden avulla. Tällaista lähestymistapaa on paljolti käytetty myös projektin koodeissa. Ensin luodaan kasa funktioita def-määreen avulla, ja sitten pääohjelmassa kutsutaan näitä funktioita ja siten rakennetaan ohjelma.

Proseduraalinen ohjelmointi on ohjelmointiparadigma, jossa ohjelma jaetaan aliohjelmiin, joita voi kutsua pääohjelmasta tai muista aliohjelmista. Aliohjelmat ovat proseduureja. Tällaista ohjelmointia on esimerkiksi C-kielessä tyypillisesti. Itse toisessa harjoittelussani Konecranesilla tein pari Scrum-jakson mittaista tehtävää tämäntyyppistä ohjelmointia. (Paradigmalla tarkoitetaan yleisesti hyväksyttyä tapaa ajatella ja mahdollisesti toimia tietyssä tilanteessa, jos määritellään löyhästi. Myös tieteessä on paradigmoja. Ne ovat eräänlaisia kattoteorioita, jotka kasaavat alleen ison määrän tieteellistä tietoa. Paradigmoille on tyypillistä,

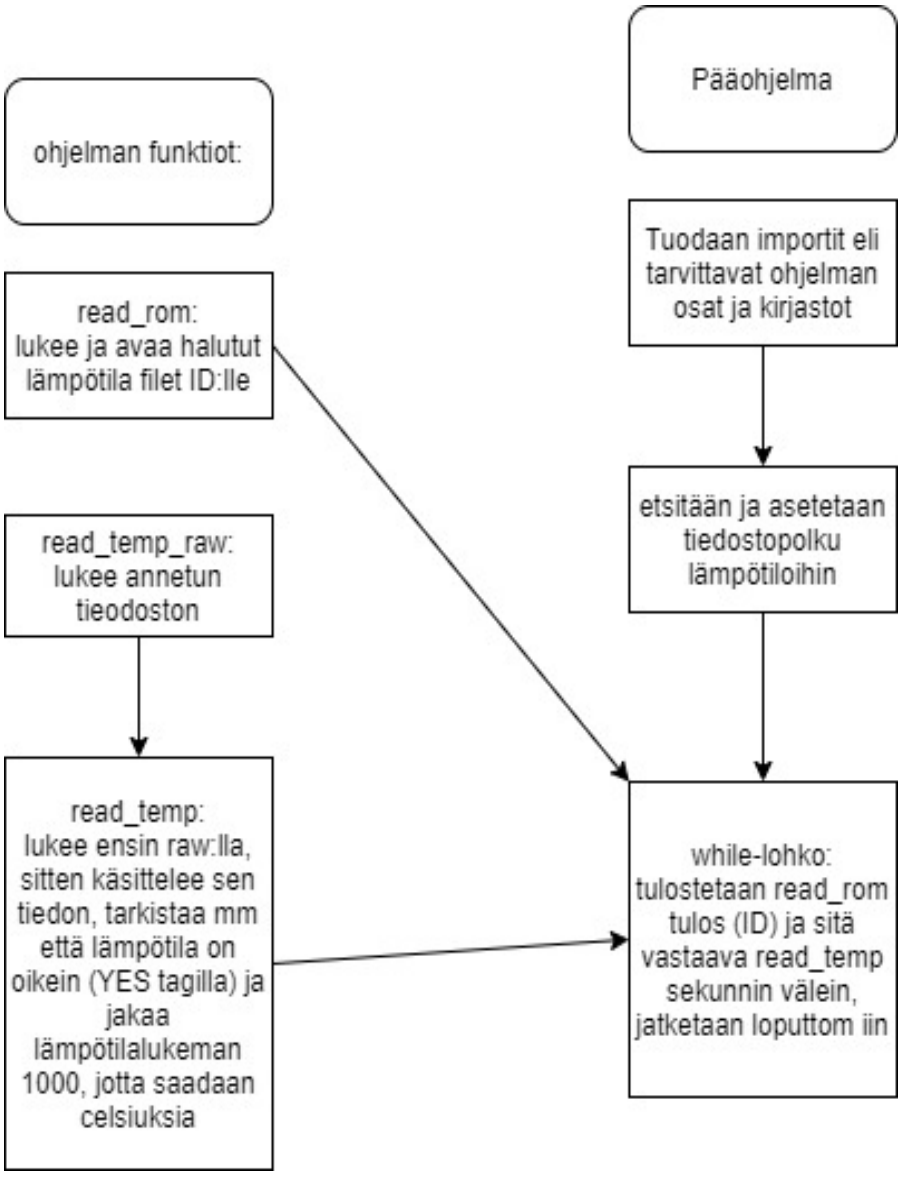
että ne eivät helposti muutu. Esimerkiksi Einstein muutti fysiikan paradigman kertaheitolla. Suureet yllättävät löydöt tieteissä tekevät näin.)

2.4 Lämpötilakoodi

Työskentelimme seuraavasti lämpötilanlukukoodin kanssa. Minä tein prototyypin, jonka perusteella Petteri Koivu viimeisteli tuotantoon menneen version. Käytin apuna internettiä ja sieltä löytyneitä tiedonläheteitä. Valitettavasti en kaikkia kerännyt samalla ylös enkä löytänyt myöhemmin haulla samoja lähteitä, joten lähteet jäävät valitettavasti hämärän peittoon. Ohjelmointiin käytimme Pythonia. Tämä ratkaisu tehtiin, sillä Python-kielellä oli tarjolla paljon puolivalmista materiaalia ja mahdollinen ohjelmiston jatko olisi helpompi, sillä lienee paljon helpompaa löytää Pytho- osaaja kuin C-ohjelmoija. AWS tarjoaa kehittäjille SDK:ta eli valmiita paketteja, jotka lataamalla on helppoa työskennellä koodin ja AWS:n välillä. Näitä paketteja oli tarjolla aina C-kielestä JavaScriptiin. Myös Raspissa on valmis tuki Python-kielelle. Lämpötilan lukukoodin proto löytyy liitteistä. Kuva 7. on kaaviokuva koodista.

Kuvassa on ohjelman funktiot kuvattu sanallisesti vasemmalla, ja ohjelman pääohjelman suorituslinja on kuvattu oikealla.

Ohjelman suoritukseen on tarvittu taitoja tiedostonluvusta, tiedostopoluista, funktioiden luomisesta ja kutsumisesta, mutta myös muuttujista ja ehtorakenteista sekä silmukoista (niin while kuin for, sekä erityisesti niiden eroavuudet).



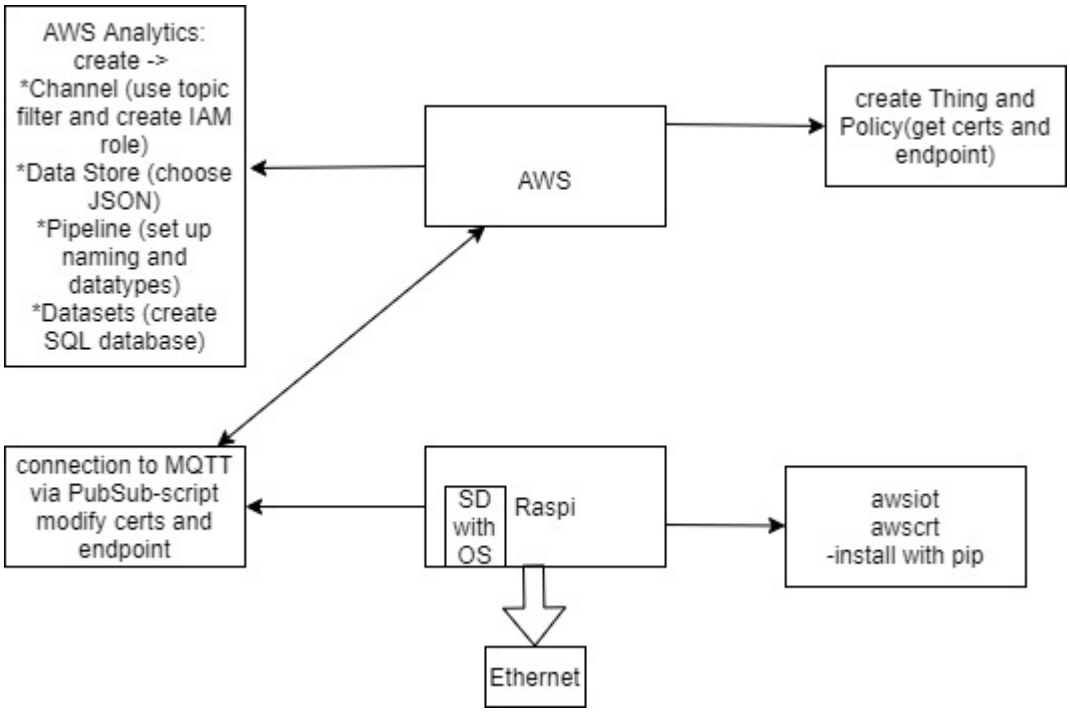
Kuva 7. Lämpötilanluku koodin kaavio

3 AWS-putki

3.1 Rakenne

AWS-putki on pipeline, jolla saadaan altaan tiedot Raspista AWS:n Analyticsin puolelle tietovarastoon, mistä sitä voidaan käsitellä QuickSight:in BI-välineillä visuaalisesti mielekkääseen muotoon. Kuvassa 8. on lyhyesti kuvailtu tarvittavat välineet ja työvaiheet, joita putken luomiseen tarvitaan. Aluksi pitää olla toimiva Raspi internetyhteydellä. Siihen asennetaan pip-työkalulla komentoriviltä kaksi

aws-pakettia: awsiot ja awscrt. Sitten luodaan AWS:n puolella Thing-olio, jolla on loppupiste (endpoint) ja sertifikaatit. Nämä tiedot upotetaan Python-koodiin, joka huolehtii yhteydestä. Koodissa pitää olla viittaus sertifikaatteihin ja endpointiin, jotta yhteys voidaan muodostaa. Kun MQTT-yhteys on saatu, rupeaa AWS-sivuille ilmestymään lähetettyä tietoa. Tieto tulee olla JSON-muodossa lähettäessä. AWS Analyticsin puolelta asetetaan kanava, tietovaranto, putki ja tietojoukko, joihin MQTT- viestit varastoidaan.



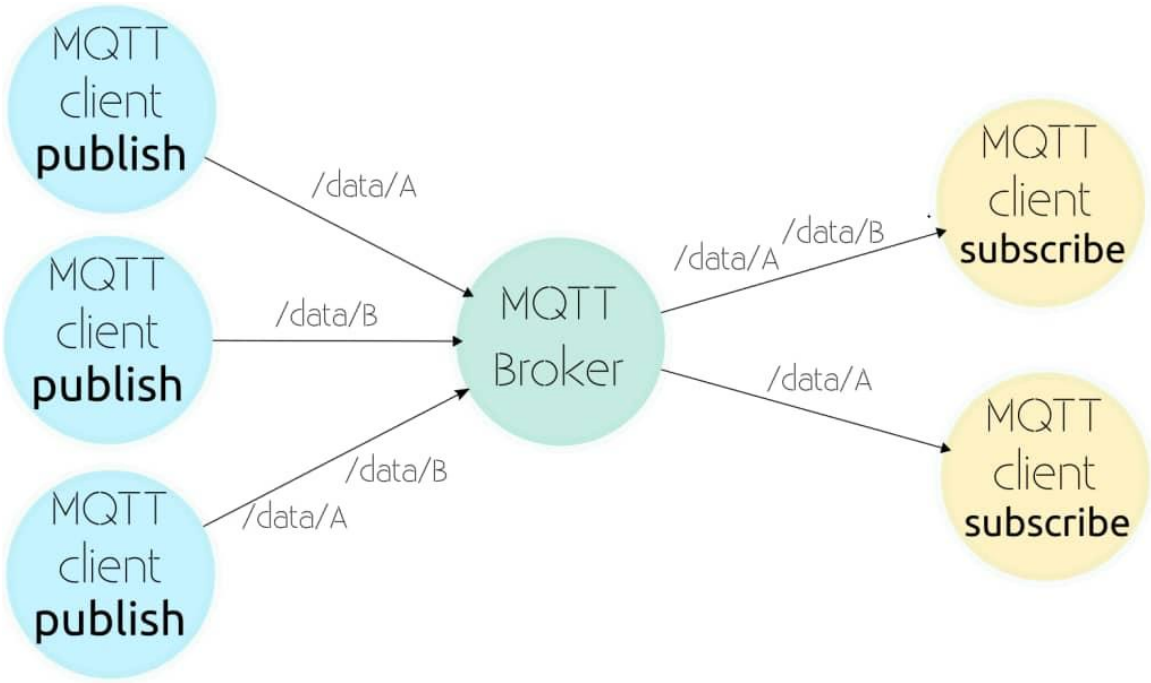
Kuva 8. Kaaviokuva AWS-Raspi putken luomisesta

Liitteessä on tarkempi dokumentaatio putken rakentamisesta. Aiemmin on jo käsitelty Raspia ja sen toimintaa. Seuraavassa puhutaan muun muassa MQTT viestiprotokollasta.

3.2 MQTT

Viestintäprotokollana Raspin ja AWS:n välillä käytettiin MQTT-protokollaa, mikä on erilaisissa Internet of Things -systeemeissä yksi suurimpia tekijöitä. MQTT on 1990-luvulla Andy Stanford-Clark (IBM) ja Arlen Nipper (Cirrus Link) keksimä viestiprotokolla, jota he käyttivät kaasuputkien seuraamiseen satelliitilla. [4.]

MQTT määrittelee kahdentyyppisiä viestimen välineitä: viestin välittäjä (broker) ja x kappaletta käyttäjiä (client). Välittäjä ottaa kaikki käyttäjien viestit vastaan ja välittää viestit oikealle käyttäjälle. MQTT-käyttäjä voi olla mitä vain mikrokontrollerin ja palvelimen välillä, kunhan siinä on tarpeelliset MQTT-kirjastot linkitettynä ja se on verkon välityksellä yhteydessä välittäjään. Viestien välityksen perustana ovat viestiaiheet (topic). Jos käyttäjällä on kirjattuna, että se kuuntelee jotain tiettyä aihetta, se saa kaikki viestit tähän aiheeseen liittyen.



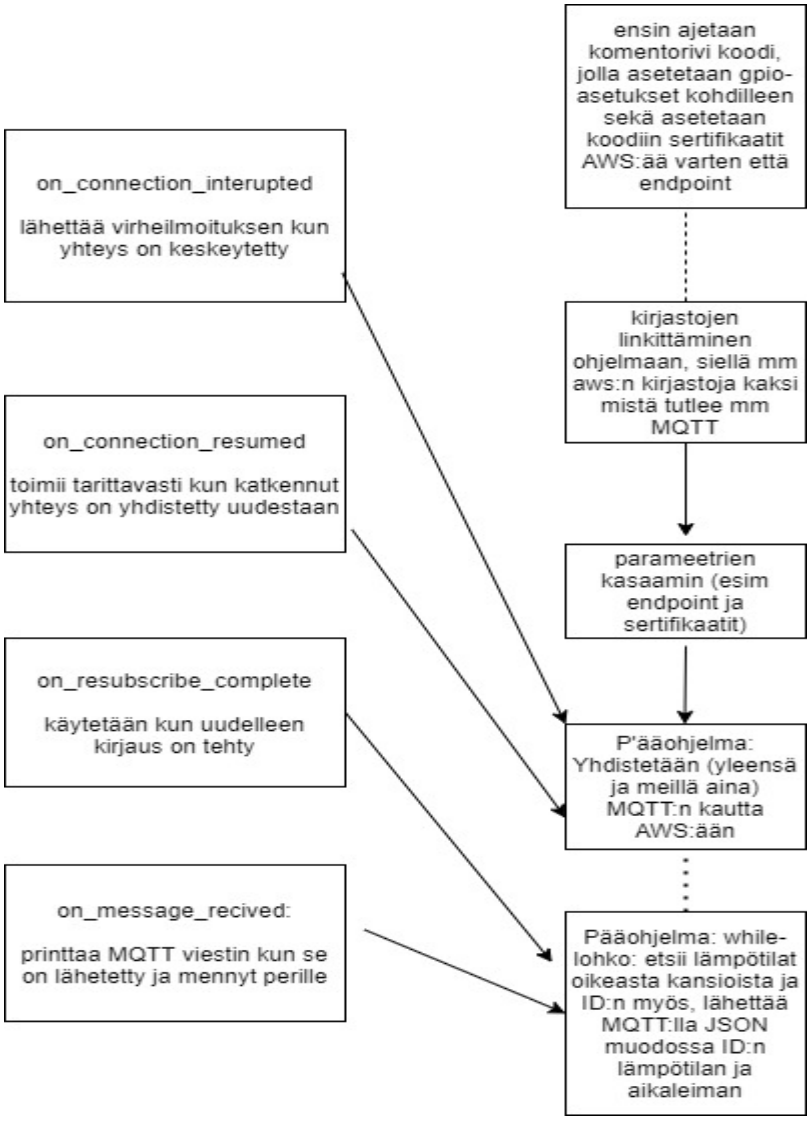
Kuva 9. Kuva MQTT liikenteestä

Systeemissä julkaisevien (publish) ja kuuntelijoiden(subscribe) ei tarvitse tietää toisistaan sen enempää. Jos A julkaisee aiheeseen /data/A ja B kuuntelee aihetta /data/A, niin toisistaan tietämättä B saa viestin A:lta. Eli kuuntelija ottaa vastaan ne kaikki aiheet, joihin se on kirjautunut. Kuvassa x ylempi kuuntelija ottaa vastaan aihetta data/A ja data/B. Alempi vain ottaa vastaan data/A:n. Sama käyttäjä voi sekä julkaista että kuunnella viestejä, mutta tämä on harvinaista.

MQTT-protokollassa on myös määriteltävä tietoa lähettäessä QoS eli Quality of Service, palvelun laatu. Tasoja on 0, 1 ja 2. [4.]

Aiheissa (topic) voi käyttää myös wild cards -merkintöjä. +-merkkiä voi käyttää missä päin vain aihetta, #-merkkiä voi käyttää aiheen lopussa. [4.]

MQTT-viestinnästä meillä huolehtii koodi, joka on liitteissä. Alla on kaavio sen toiminnasta, vasemmalla kuvassa funktiot, oikealla linkitykset sekä pääohjelma.

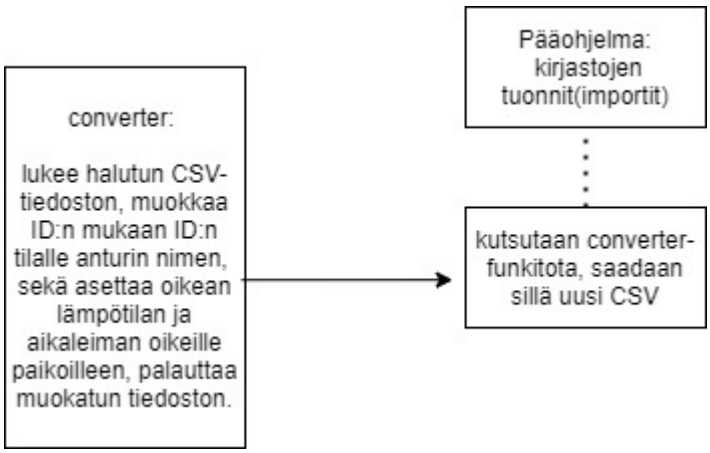


Kuva 20. Lämpötila-MQTT koodi kaavio

Koodin käyttö vaatii ymmärrystä MQTT-portokollasta, Python-kielen perusteiden osaamista (for, while, tiedoston luku, ehto-lausekkeet) sekä JSON-formaatin tuntemista ja jotain käsitystä websogetista.

3.3 QuickSigth - datan visualisointi

Aloitin QS:n kanssa heinäkuussa, jolloin Petteri Koivu oli lomilla. Meillä oli kaksi AWS-tiliä, minun testikäyttööni tarkoitettu ja Petteri Koivun ns. tuotantoversio. Altaalta tieto meni suoraan Koivun puolelle ja niinpä se minulle piti siirtää CSV-tiedostomuodossa QuickSightiin. Kun tietojen lataus tehtiin näin, en päässyt asettamaan tietojen muodostumiseen liittyviä asetuksia (data ei tullut omalta AWS tililtäni), joten tein koodin, jolla muokkasin CSV:n muotoon, jolla sain helposti QuickSightista ulos haluamaani dataa. Alla on kuvaus koodista. Koko koodi on liitteissä.



Kuva 31. Kaaviokuva CSV:n muokkauskoodista

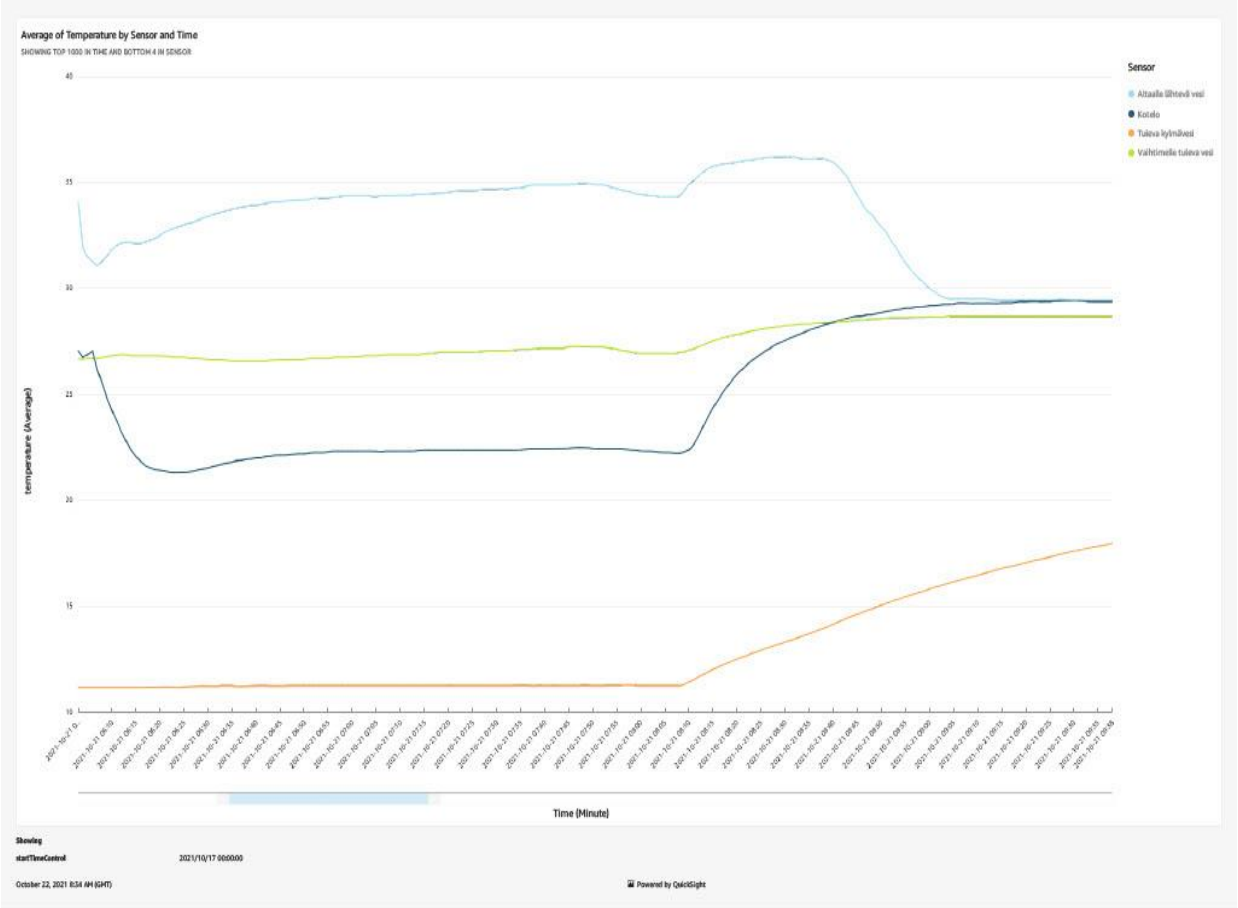
Koodissa annetaan nimet ID:n perusteella, jotta ne näkyisivät oikein kuvaajissa. Koodissa käytetään Pandas-kirjastoa, joka on hyvä CSV-tiedostojen käsittelyyn ja erityisesti siis data science -töihin. Kuten edellisten sivujen kuvauksen perusteella on kerrottu, on lämpötilatieto saatu täten AWS:ään, niin sieltä se siirretään QuickSightiin visualisoitavaksi.

QS on Amazonin omien sanojen mukaan vapaasti käännettynä seuraavaa: ”QS antaa helposti luoda ja julkaista interaktiivisia ”työpöytiä” kuin myös ottaa vastauksia sekunneissa luonnollisella kielellä tehdyissä kyselyissä. QS-työpöydän voi saavuttaa miltä tahansa laitteelta ja upottaa se sovellukseen, web-sivuun yms. QS on serveritön ja skaalaa automaattisesti kymmeniä tuhansia käyttäjiä ilman ympäristö- tai kapasiteettimuutoksia. Se on myös ensimmäinen BI-palvelu, joka tarjoaa maksa-pre-sessiohinnoittelun, jolloin joutuu maksamaan vain silloin, kun käyttäjä käyttää palveluita, mikä tekee siitä kilpailukykyisen hinnaltaan ison kaavan produktioissa.” [7.] .

Työkaluna QS oli suhteellisen helppo, mutta vastoin AWS:n väitteitä se ei ollut minusta intuitiivinen. Tutustuin eri tavoin työkaluun ennen sen käyttöä: selailin tutoriaalia ja katsoin videoita. Itse kuvaajien luomiseen meni muutama päivä. Minun tekemiäni kuvia ei ole tässä työssä. Jouduin myöhemmin tuhoamaan testitilin erään tapauksen vuoksi, joten menetin pääsyn tekemiini kuviin.

Mietimme kuitenkin muita datankäsittelytyökaluja. Mainittakoon niistä Jupyter-Notebook ja Pytho-kieli Pandas/NumPy-kirjastoilla. QS osoittautui jonkin verran kömpelöksi ja siellä kaikki kontrolli ei ole käyttäjällä.

Seuraavan Koivun tuotannossa käytetty kuva lämpötiloista, jotka on tehty QS:llä.

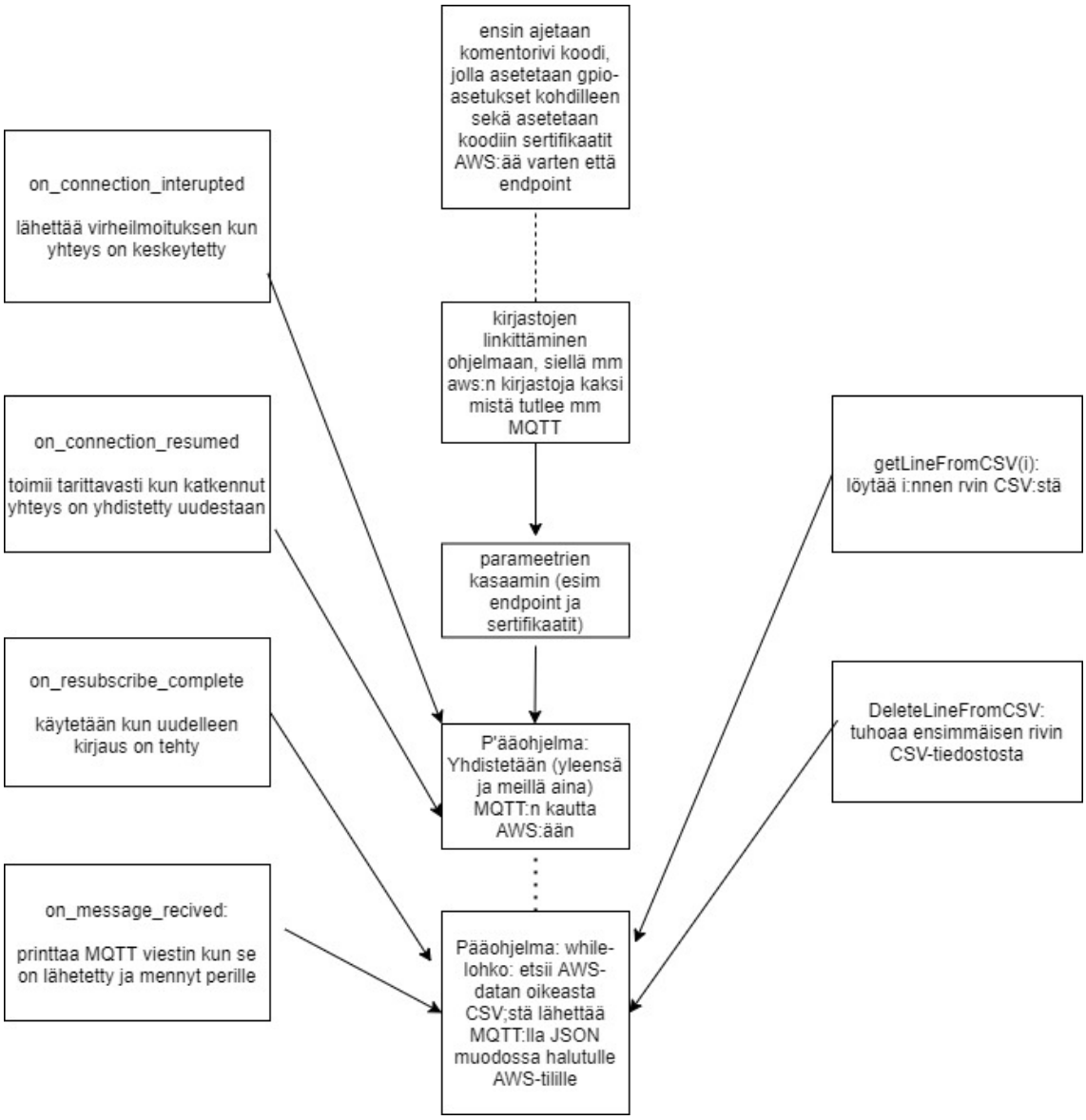


Kuva 42. Lämpötilan visualisointi kuva

3.4 CloudWatch raja-arvojen seuraaminen

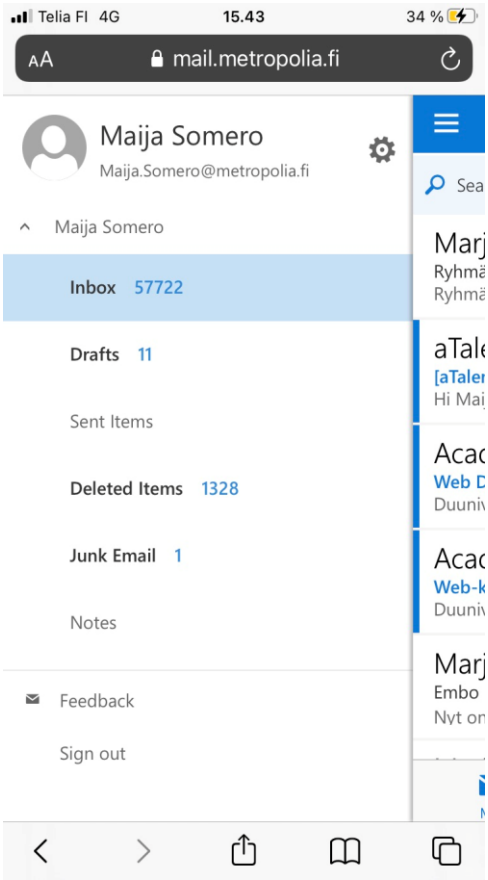
Amazon itse sanoo CloudWatchista seuraavaa: Amazon CloudWatch on monitorointi- ja tarkkailupalvelu rakennettu DevOps-insinööreille ja SR-insinööreille sekä IT-managereille. Cloud Watch tarjoaa data- ja toiminnallisuusmonitorointia sovelluksesta, jossa vastataan järjestelmälevveydeltä suorituskykymuutoksiin, optimoituun resurssien järjestämiseen ja antamaan yksiselitteisen kuvan toiminnallisesta kunnosta. CW kerää monitorointi- ja toiminnallisuustietoa lokeihin, mittareihin ja tapahtumiin, mikä tarjoaa yhtenäisen kuvan AWS-resursseista, sovelluksista ja palveluista, jotka ovat AWS:llä. CW:tä voi käyttää etsimään väärää toimintaa ympäristössä, asettaa hälytyksiä, kuvata lokeja ja mittareita, tehdä automatisoituja askeleita, etsiä virheitä, löytää keinoja, joilla pitää sovellus ajamassa hyvin. [8.].

Kun olin loppupuolella harjoittelua ja meillä tuli jo dataa altaalta AWS:stä QS:ään, haluttiin asettaa lämpötiloille tiettyjä raja-arvoja, jos jotain dramaattista tapahtuu, saadaan siitä hälytys. Minä selvitin ja rakensin systeemin, joka testimielessä lähetti jokaisesta lämpötila-arvosta, joka oli alle 20 celsiusastetta, hälytyksen sähköpostitse. Systeemin rakentamisen ohjeet löytyvät liitteistä. Kuten kuvaan kuuluu, piti systeemi myös testata testitilillä. Sain muilta altaalta ison CSV-tiedoston täynnä oikeaa lämpötiladataa. Tein koodin, joka lukee CSV:ltä AWS:ään MQTT:llä tietoa. Kuvassa x kuvaus koodista CSV:stä AWS:ään (MQTT:n kautta). Liitteissä on koodi kokonaisuudessa.



Kuva 53. CSV tiedosto MQTT:n kautta AWS-palveluun koodi

Koodi on muutoin sama kuin lämpötilanlukukoodi MQTT:lla, mutta MQTT:n pääluokan while-lohkossa on tiedonhaku CSV:stä ja rivin lähetys sekä poisto. Jos poistoa ei haluta käyttää, niin rivien luku vaatii indeksimuuttujan, joka on toki helppo lisätä, jos haluaa säilyttää alkuperäisen CSV:n, sillä rivien luku on tehty indeksimuuttujalla.



Kuva 64. Kuva sähköpostistani päälle jääneen testiajon jälkeen

Testiajosta voi sanoa, että toimii hyvin. Minulta jäi vahingossa CSV:n luku AWS:ään päälle koneelle. Kun oli kesä ja lämpötilarajaksi asetettu 20 astetta, niin hälytyksiä tuli lähes joka sekunti. Testiajo jäi päälle perjantaina ja huomasin tämän vasta sunnuntaina. Tästä lopputulemana sain omaan sähköpostiini noin 55 000 hyvin lyhyttä hälytysviestiä.

Oma sähköpostini etusivu näytti kovastikin kiireiseltä tuon viikonlopun jälkeen. Kun sunnuntaina huomasin tilanteen, säikähdin. Pelkäsin, että olen onnistunut tekemään miljoonalaskun. Niinpä paniikissa suljin koko testitilin kokonaan. Onneksi hupi maksoi vain 2 dollaria. Tämä selvisi paniikin hälvettyä hintatietoja etsimällä.

Ohjeet sähköpostihälytyksen tekemiseen ovat liitteessä.

4 AWS:n tietoturva ja skaalautuvuus

4.1 AWS:n näkökulma

AWS on dokumentaatiossaan ilmoittanut, että se jakaa pilven tietoturvavastuut: Amazon on vastuussa pilvestä, eli esimerkiksi palvelinsalien turvallisuudesta, kun taas asiakas, joka käyttää palvelua, on vastuussa pilvessä, tehdyistä turvallisuuskonfiguraatioista. Seuraavassa on AWS:n sivuilta vapaasti käännettynä tietoa tietoturvasta.

Tiedon suojaus:

AWS tarjoaa palveluita, jotka auttavat asiakasta suojelemaan tietoa, tilejä ja työtaakkaa asiattomalta pääsyltä. Ne tarjoavat salausta ja avainhallintaa sekä uhkien jäljitystä, jotka jatkuvasti seuraavat ja suojelevat asiakkaan tiliä ja työtaakkaa.

Tunnistautuminen ja pääsyn hallinta

AWS-identiteettipalvelut mahdollistavat turvallisesti identiteettien, resurssien ja lupien hallinnan skaalaten. AWS:llä asiakkaalla on identiteettipalveluita työvoimalla ja asiakasrajapinnan sovelluksille, mitkä voi aloittaa nopeasti, joilla hallita työvoiman ja sovelluksen pääsyjä.

Verkon ja sovelluksem suojelu

Verkon ja sovelluksen suojauspalvelut mahdollistavat asiakkaalle käyttää hienojakoisia turvallisuuskäytänteitä verkkokontrollipisteissä ympäri asiakkaan organisaatiota. AWS-palvelut auttavat asiakasta tarkkailemaan ja suodattamaan liikennettä ja kieltämään asiattoman resursseihin pääsyn palvelin-, verkko- ja sovellustasoilla.

Vikaseuranta ja jatkuva valvonta

AWS tunnistaa uhkia jatkuvalla valvonnalla verkkoaktiivisuuden ja tilikäyttäytymisen puolesta asiakkaan pilviympäristössä.

Sisäinen valvonta ja tiedon yksityisyys

AWS antaa asiakkaalle hyvän kuvan sisäisestä tilasta ja sen statuksesta ja jatkuvasti valvoo asiakkaan ympäristöä, käyttää automatisoituja sisäisen tarkastuksen pistokokeita perustuen AWS:n käytänteisiin ja teollisuuden standardeihin. [9.]

4.2 Tietoturvariskit

Tämä luku referoi Pete Cheslockin blogikirjoitukseen AWS:n tietoturvariskeistä. Käsittelen nuo yleisimmät ja sitten peilaan niitä projektiimme nähdäkseni, teimmekö kaiken oikein ja suositusten mukaan.

Cheslockin mukaan, vaikka AWS on jo hetken ollut keskuudessamme, niin yritykset vielä kamppailevat sen kanssa, mitkä olisivat parhaat turvallisuuskäytänteet. Hänen mukaansa AWS on jo kahdeksan vuotta ollut käytetyin IaaS-palvelu. IaaS tulee sanoista ”infrastructure as a service”, eli konkreettisten laitteiden sijaan asiakas saa samat palvelut sovelluksena.

Asiakkaiden erilaisten teollisuudenalojen ja yrityskulttuurien ja yritysideoiden seasta on löydettävissä ainakin kolme pääkysymystä, jotka ohjaavat tietoturvallisuutta:

- Kenellä on pääsy mihinkin sovellukseen ja milloin?
- Kuinka voidaan valvoa avain tiedostojen muutoksia?
- Tiedotetaanko ajoissa, kun jotain poikkeavaa tapahtuu?

Eräs ongelma myös yrityksillä on tietoturva, kun tietoa, joka voi olla hyvinkin monimutkaista ja moninaista sillä välin, kun sitä siirretään AWS:ään. Tietoturva-riskikohdat, jotka Pete Cheslock nimeää, ovat seuraavat 9 kohtaa:

Laitetaan tietoturvastrategia kontrollien ja työkalujen edelle

Yksi isoja kysymyksiä AWS:n suhteen on ollut erilaisilla keskustelupalstoilla, että miten ihan aluksi suhtautua tietoturvaan. Vaikka kysymys on keskeinen ja tuntuu lähes triviaalilta, niin vastaus ei ole täysin yksinkertainen. Yhdeksässä tapauksessa kymmenestä tietoturvastrategian pitäisi tulla ensin. Laittamalla strategia etusijalle saadaan myös kaikki liiketoimintaosat integroitua strategiaan. Tämä voi olla suuri apu jatkuvalle integraatiolle. Esimerkiksi jos käytetään jatkuvan integraation työkaluja, saadaan sovitettua tietoturvastrategia jo heti alusta. Sama pätee mihin tahansa liiketalousprosessiin tai työkaluun.

Turvallisuusläpinäkyvyyden puutteen (pilvessä) selättäminen

Pilvi sovellusten laajuuden ja erilaisten sisäänkirjautumisten ja kontrollien määrä on aiheuttanut sen, että on lähes mahdotonta tarkkailla kaikkia sisäänkirjautumisia ja erityisesti pahantahtoisia tai epätyypillisiä sisäänkirjautumisia. Tämä turvallisuusläpinäkyvyys nousee erityisesti esiin, jos yrityksellä ei ole turvallisuus strategiaa. Saavuttaakseen paremman läpinäkyvyyden AWS:ssä voi käyttää seuraavia menetelmiä:

- Läpikotainen tarkastelunäkökulma:

Jos ei tiedetä mitä tapahtuu isännällä tai työmäärissä, tarvitaan enemmän tietoa kuin IDS-logi voi tarjota. Tarvitaan ratkaisu, joka näyttää tietyt tapahtumat

ajassa tietyllä palvelimella. Jutussa viitataan heidän rakentamaansa Threat Stackiin.

- Ylitetään lokitiedot

Vaikka lokitiedot ovat tärkeitä, ne tarjoavat vain kapean kuvan järjestelmän toiminasta. Vertauskuvallisesti: on eri asia tietää, kuka tuli sisään tai meni ulos rakennuksesta ja milloin, kuin että mitä he siellä sisällä tekivät. Kirjoittajan mukaan NIDS (network-based intrusion detection) ei tarjoa kovin hyvää ratkaisua ongelmaan, kun taas HIDS (host-based intrusion detection)-metodissa, jossa tietoturva on upotettu isäntätasolle, saadaan tietoa, mitä, milloin, missä, ennen, aikana ja jälkeen hyökkäyksen.

- Suojaudutaan sisäisiltä uhilta

Joskus uhat tulevat sisältä, ja ne voivat olla seuraavia: epätyypillinen internetin käyttö, luvattomat lataukset, epätyypilliset sisäänkirjautumisen yritykset tai epäonnistumiset, avaintiedostojen muutokset.

Luotettavuuden parantaminen pilvipalvelun palveluntarjoajan tietoturvassa

Vaikka AWS tarjoaa monia hyviä turvallisuus palveluita kuten AWS CloudTrail ja Amazon Cloud Watch, joilla voi esimerkiksi tarkkailla, on hyvä tietää, mihin Amazonin vastuu loppuu ja mistä alkaa käyttäjän vastuu, erityisesti mitä tulee arkaluontoiseen materiaaliin.

Historia on nähnyt yrityksiä, jotka ensin tutkivat tietoturvakysymykset ja vasta sitten ottavat AWS:n käyttöön, jos ottavat. On tullut yleisemmäksi tutustua AWS:n lisäksi tietoturvapalveluita tarjoaviin yrityksiin, kuten Therat Stackiin. Pääkysymyksiä ovat: kuinka varmistaa sisäinen valvonta, kuinka käsitellä vahinko tilanne ja kuinka päästää lokitiedostoihin. Nämä ovat hyvin olennaisia kysymyksiä. Näitä pohtimalla ja tekemällä tapauskohtaisia arvioita, on mahdollista saavuttaa parempi luotettavuus.

Määritellään, kuka on vastuullinen

Vastuullisuus on kuuma aihe pilvipalveluissa yleensä. Sillä jos sattuu jokin vahinko, niin tarvitaan tieto, kenen vastuulla on tehdä toimintoja liittyen vahinkoon. AWS ja sen kaltaiset toimijat ovat ottaneet enemmän yhteisöllisen vastuullisuuden aina virtuaalikonetason yläpuolella. Mutta käyttäjän pitää vielä silti ottaa valvontaa, kontrollia ja lokeja saadakseen tietää, kenellä on pääsy kirjautumaan. Yritykset voivat ottaa ennakoivan lähestymistavan sisäänpääsyn tavoissa ja valvonnan aktiivisuudessa yli verkon. Jotta jos ja kun jotain sattuu, yritykset voivat tarkasti osoittaa, kenen vastuulla asia on.

Ymmärtää miksi hyökkääjät ovat kiinnostuneet pilvestä

Yritykset luottavat paljon AWS:n kaltaisiin toimijoihin antamalla heille esim. terveys-, luottokortti- ja liiketaloudellista tietoa. Tämän johdosta ovat kyseiset palveluntarjoajat myös hyvä kohde pahaa tarkoittaville taholle. Tästä huolimatta suurimmat vahingot Cheslockin mukaan tapahtuvat kuitenkin sisäänkirjautumistietojen hukkuessa tai joutuessa vääriin käsiin, ei niinkään palvelimelle kohdistuneesta hyökkäyksestä. Cheslock antaa pari esimerkkiä, missä muun muassa jouduttiin sulkemaan kokonainen yritys kadonneiden sisäänkirjautumistietojen vuoksi. Nyrkkisääntönä voi pitää, että kirjautumistiedoissa käytetään MFA:ta (multi factor authentication), tarkkaillaan epätyypillisiä sisäänkirjautumisia, toteutetaan lokipalvelut isäntätasolle ja AWS Secret Manageria tai jotain vastaavaa palvelua.

Puolustaudutaan uteliaita urkkijoita vastaan monikäyttäjäympäristössä

Teoriassa monikäyttäjäympäristö on riskialttiimpi sisäänkirjautumistieto vuodoille. Mutta oikeasti tämä riippuu, kuinka tietoturvallinen järjestelmä on. Todellinen riski on, kun kokematon henkilökunta tai prosessi työskentelee ja valvoo virtuaalista järjestelmää. Jutun mukaan useammat yritykset pelkäävät, että monikäyttäjyys on suuri tietoturvariski. AWS on ottanut tämän huomioon palveluissaan. Suuri pelko on, että tietoa yrityksestä vuotaa jonnekin muualle. Tietoturvan kypsyyttä voi tarkastella seuraavilla viidellä alueella: systeemiin pääsy ja

käyttäjät, muutostietojen ja heikkouksien hallinta, järjestelmän hallinta, verkko-asiat, ajonaikaisuus ja palvelut.

Asettaa sisäisen valvonnan säädöksiä

Huoli sisäisestä valvonnasta kaikuu niin pienistä kuin suurista yrityksistä. Erityisesti viimeisillä GDPR-säädöksillä (Euroopan unionin tietosuoja asetukset) AWS on esittänyt resursseja, joilla suojata tietoa. Vaikka toimijat kuten AWS tarjoavat tietyn verran suojausta tiedolle, ne eivät voi tarjota jokaista sisäisen valvonnan näkökulmaa. AWS pystyy ja myös tarjoaa suojauksia kuten tietyn tyyppinen salaus, mutta se ei jatkuvasti valvo tiedon epätyypillisiä käyttäytymisiä ja tarjoa isäntätason katsantoa, joka voisi olla ongelman ytimessä. Ei ole helppoa sanoa, missä AWS:n sisäisen valvonna työkalu loppuu ja mistä pitäisi alkaa jonkun muun palveluntarjoajan työ.

C:n mukaan tästä huolimatta iso osa yrityksiä uskaltaa ja haluaa lähteä pilveen mm. taloudellisten etujen tähden. Hänen mukaan AWS:ää voi luottaa, mutta aina kannattaa tarkastaa ratkaisut ensin.

Tämä artikkeli tuntui puhuvan enemmän isomman yrityksen näkökulmasta. Meillä toteutui ainakin MFA-kirjautuminen ja Koivu, joka oli varsinaisen tilin pääkäyttäjä, painotti monia turvallisuuskriteerejä, kuten että oletussalasanat tuli vaihtaa heti. Meilläkin oli yrityksen luottokortti kiinni tilissä, ja sitä kautta sensitiivistä tietoa pilvessä. Käytimme myös hänen mainitsemaansa Cloud Watchia sisäiseen valvontaan.

4.3 Skaalautuvuus

Mediumissa on kirjotettu juttu AWS:n skaalautuvuudesta. Skaalautuvuudella tarkoitetaan sitä, miten palvelut niin sanotusti joustavat, kun esimerkiksi palvelu

kasvaa. Kirjoituksen mukaan on erilaisia vaiheita, joista ensimmäinen on palvelun pystyyn laittaminen. Eli juuri se mitä teimme kesällä. Tällaisen asian pystyy tekemään alle 10 dollarin, ja palvelu on olemassa. Jutun mukaan seuraava vaihe on, kun tuotetta viedään tuotantoon. Itse emme täysin tähän vaiheeseen päässeet. Silloin tulee vastaan uusia haasteita, kun palvelu on auki uusille käyttäjille. Juttu listaa muutaman näkökulman: virheen sieto, tietokantojen varmuuskopiot, suorituskyky, skaalautuvuus. Jos käyttö lisääntyy ja tulee ns. käyttöpiikkejä, jolloin suoritusteho on kovemmalla kuin jos on hiljaisempi kausi menossa, niin AWS:llä on tähän Load Balancing-palvelu, joka on maksullinen mutta joka pitää huolen siitä, että palvelun tehot on käytetty ekonomisesti. Toinen palvelu on Auto scaling, joka myös tasapainottaa resursseja käytön mukaan. Tämäkin palvelu on maksullinen.

Meidän projektissamme skaalautuvuus ei niin vahvasti tule esiin. Käyttäjiä palvelulla on joka tapauksessa vain muutama (Sea Poolin/Voltigon vastuu henkilöt tai omistajat). Ainoa kasvun suunta tuotteelle on leveys: eli jos jonnekin rakennetaan uusi allassysteemi, niin sille voidaan asentaa tämä tuotettu ohjelma vastuuhenkilöiden käyttöön. [10.]

5 Tuotteen tulevaisuus, mahdolliset toiset ratkaisut ja jälkisanat

5.1 Energialaskut

Yksi asia, joka jäi kesken kesällä, olivat energialaskut. Eli miten vedenkulutuksesta ja lämpötiloista helposti lasketaan energiankulutus.

Lämpöenergia

Johtuu kappaleen molekyylien värähtelyliikkeestä. Riippuu kappaleen ominaislämpökapasiteetista c, massasta m ja kappaleen lämpötilan ja vertailulämpötilan erotuksesta $\Delta T = T2 - T1$.

$$E = c*m*\Delta T$$

[6.]

Lasku on yksinkertaisia kertolaskuja ja lämpötilojen erotus. Koska meillä on vain yksi aine, vesi, c on vakio ja m saadaan mittaustuloksista veden virtaamasta. Lämpötilojen erotuksen pystyy laskemaan mittaustuloksista.

Tätä toteutusta pohdimme jonkin aikaa ja minä sain idean, että nuo yksinkertaiset laskut voisi tehdä AWS:n puolella SQL-tietokannassa suoraan. Manageri Sampo Nurmentaus piti tätä hyvänä idean ja laittoi minut selvittelemään laskujen onnistumista SQL:n puolella. Niinpä minä sitten tein testi-SQL-joukon olemassa olevasta datasta ja kokeilin, että erotus toimii, kertolasku toimii ja niiden yhdistäminen toimii. Normaalissa SQL-kannassa tämä olisi ollut itsestäänselvyys, mutta emme olleet aivan varmoja AWS:n SQL-kantatoteutuksesta. Tehtävä oli melko yksinkertainen, mutta silti yksi hassu virhe sattui aluksi. Merkitsin desimaalit suomalaisittain pilkulla laskuissani (testasin ns. ominaislämpökapasiteetilla kertomista). Lopputulos oli pyöristettyjä arvoja. Kun tajusin laittaa pisteen, niin laskut toimivat halutusti.

Eli saimme tuloksen, että on mahdollista toteuttaa lämpöenergiaan liittyvät laskut AWS:n SQL-puolella. AWS:n SQL-dataan pystyy asettamaan ns. sääntöjä, jotka määrittävät SQL:llä, miten muodostetaan uusi taulu.

SQL-tietokanta on relaatiotietokanta, ja tieto on esitetty riveinä ja sarakkeina tauluissa, joiden välillä on yhteyksiä.

Toinen vaihtoehto olisi ollut tehdä laskut QS:n puolella visualisoidessa dataa. Mutta tämä vaihtoehto jäi vain ajatukseksi heti alussa, sillä koimme QS:n suhteellisen kankeaksi.

5.2 AWS:n korvaaja

Jo alkumetreillä Metropolian vakituiset työntekijät Nurmentaus ja Koivu miettivät projektin tekemistä jollain muulla kuin AWS:llä. Syitä oli monia. AWS:n epäintuitivinen käyttöliittymä, spekulaatio siitä, säilyykö Internet of Things osuus AWS:ssä loputtomiin, eli onko se osuus AWS:ää todella tuettu koko tuotteen elinkaaren. Vaihtoehtoinen toteutus olisi ollut myös Raspilla ja sensoreilla (toki). Sitten Nurmentaus pohti, että tiedot voisi lähettää PostreSQL-tietokantaan, jota voisi ylläpitää itse. Sieltä tiedot saataisiin itse reaaliajassa web-sivulle, jotka tehtäisiin itse JavaScriptillä ja Reactilla, jota myös ylläpidettäisiin itse.

Tämän ratkaisun vahvuudet olisivat siinä, että kaikki kontrolli olisi meillä tekijöillä, mutta tietysti ratkaisu vaatisi ylläpitoa aivan eritasolla kuin AWS:n käyttö. Asiakkaan toiveesta jäimme AWS-ratkaisuun.

5.3 QuickSightin tilalle Pandas?

Toinen mitä mietittiin ja mikä olisi myös edellisen ratkaisun osa, oli QS:stä luopuminen ja esimerkiksi JupyterNotebookin käyttö Python-kielellä ja Pandas- ja NumPy-kirjastoilla. Edellisen kannalla olin itsekin jonkin verran. Olen käyttänyt koulussa tilastotieteen kurssilla näitä työkaluja data science -tyyppiseen työhön. Lisäksi minulla toisessa harjoittelussa toissa kesänä Konecranesilla oli harjoittelijoiden projekti, jossa teimme data science -työn.

Näillä yllä mainituilla työkaluilla saa tehtyä kenties laadukkaampaa data-analyysiä kuin QS:llä. Tällöin tekijällä on enemmän kontrollia. Tämä ratkaisuehdotus kuten edellinenkin, jäivät vain ajatuksen tasolle eikä niitä koskaan toteutettu.

5.4 Shadow´n käyttö projetkissa AWS:ssä

Yksi mahdollisuus tehdä MQTT-toteutus olisi ollut AWS: shawod´n avulla. Tämä otus on kirjaimellisesti kuin varjo. Se kuvaa systeemin tilaa kullakin hetkellä vii-

meisimmän tiedon valossa. Shadow´n tila vastaa siis systeemin kulloistakin tilaa. Tähän totutusmalliin olisi ollut paljon valmista koodia käytössä, mutta tuon ominaisuuden käyttö olisi ollut maksullista. Muistaakseni jokainen päivitys shadow-olioon olisi kustantanut jotain pientä. Eli tekemällä MQTT-toteutus ilman varjoa säästi mukavasti projektin rahaa. Lisäksi meillä omassa toteutuksessa oli MQTT jatkuvalla syötöllä päällä, joten teoriassa emme edes tarvinneet tätä ominaisuutta, sillä saimme viimeisimmän tiedon raa'alla MQTT:lla suoraan.

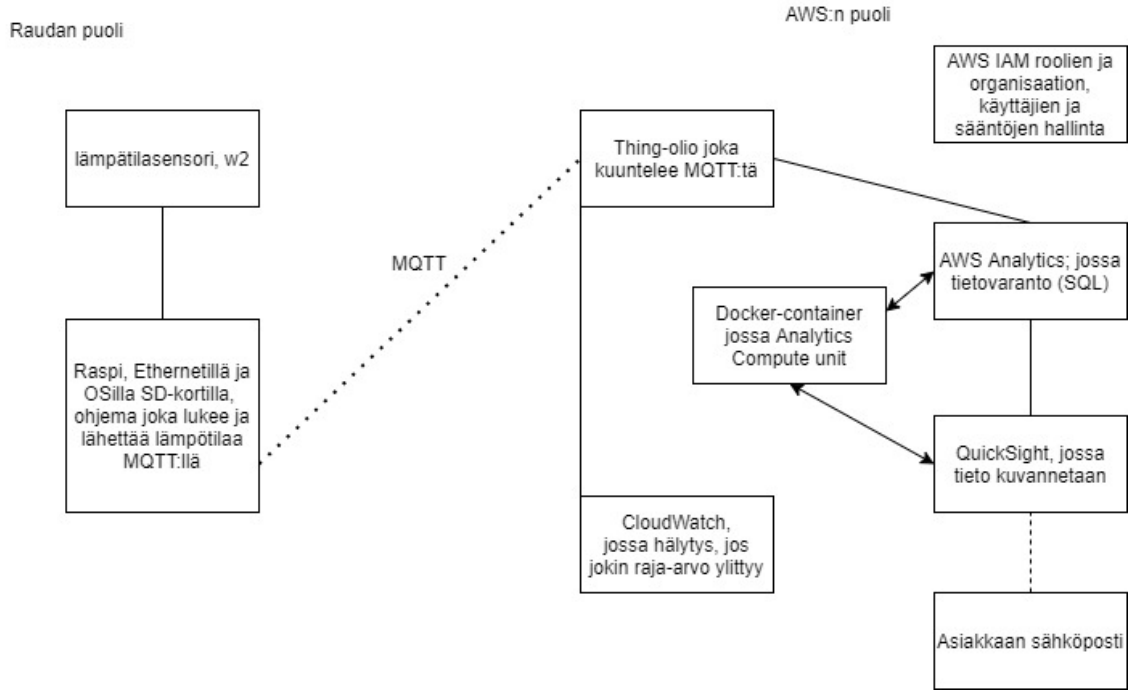
5.5 Tuotteen tila kirjoittaessa opinnäytetyötä

Kun projektia lähdettiin viemään eteenpäin, oli ajatus, että tämä projekti tehdään Allas Sea Pool Helsingille. Mutta kuulin myös, että Allas Sea Poolilla oli laajenemisaikeita kansainvälisesti. Jos tekemämme tuote olisi riittävän hyvä, se tulisi myös muualle käyttöön. Kun lopetin määräaikaisen työsuhteeni Metropolialle saattaakseni opintoni loppuun, oli projekti vielä hieman kesken. Kuten yllä mainitsin, asiakas olisi halunnut myös veden lämpöenergiankulutuskuvaajat QS:sta. Tämä on Petteri Koivun sanojen mukaan vaiheessa. Isossa kuvassa koko projekti on laitettu hetkellisesti jäihin eli tauolle.

5.6 Mitä tehtiin

Eli fyysisellä puolella on Raspi ja lämpötila-anturi, jotka ovat Ethernetin välityksellä kiinni internetissä ja sieltä Raspi kommunikoi MQTT:n välityksellä AWS:n Thing-oliota, joka hälyttää, jos asetetut raja-arvot ylittyvät. Oliosta tieto menee

Analyticsiin, josta Docker container se



Kuva 75. Kuva rakennetusta järjestelmästä

n ohjaa QS:ään. QS tuottaa halutulla kaavalla kuvaajia, jotka saatetaan loppukäyttäjän tietoon esimerkiksi sähköpostilla.

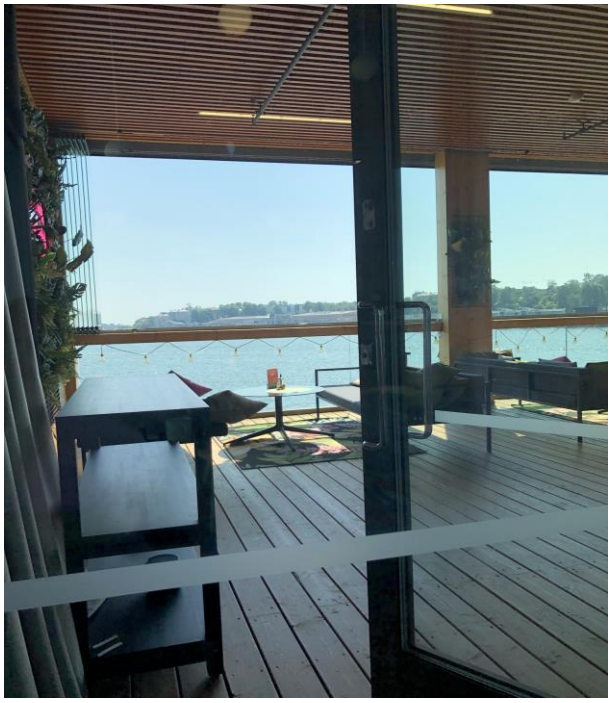
5.7 Projekti prosessina

Oma kokemukseni projektista oli suhteellisen ristiriitainen. Pidin ideasta ja osittain sulautettujen järjestelmien lisäksi data-analyysin puolelle menevästä toteutuksesta. Osittain silti tuntui, etten saanut parasta mahdollista suoritusta irti itsestäni.

Pidin meidän löyhää Kanbania hyvänä: soittelimme aina viikon alussa tiimipuhelun, jossa käytiin läpi projektin kuulumiset, seuraavan viikon tehtävät ja muut projektiin liittyvät juoksevat asiat. Toki johonkin väliin mahtui vitsi poikineen. Ihan projektin alussa pidimme tiimitapaamisia minun kotini pihalla (liikun ke-

peillä ja siirtyminen haasteellisempaa). Tällaisia tapaamisia oli muutama. Lisäksi loppupuolella projektia meillä oli useita tapaamisia Voltigon kanssa, joissa käytiin läpi projektin tilaa ja uusimpia tehtäviä.

Kun projekti eteni, niin kävimme asiakkaamme Voltigo Oy:n (joka taasen oli Allaa Sea Poolin asiakas) kanssa Allas Sea Pool Helsingissä paikan päällä katsomassa ja asentamassa rautaa. Tapaamiseen kuului lounas Voltigon kanssa Allas Sea Poolilla.



Kuva 16. Kuva asiakastapaamisesta

5.8 Mitä olisi voinut tehdä paremmin

Ehkäpä minun ja Petteri Koivun työnjako olisi voinut olla hieman selkeämpi jo alussa, tuli jonkun verran tehtyä päällekkäistä työtä. Olisin itse voinut kysellä tarkemmin työtehtävistä jo alussa, mutta tekevälle sattuu. Opinpahan, että kannattaa kysellä innokkaasti jo alkumetreillä. Tavallaan minut heitettiin aika heti syvempään päätyyn ja katsottiin, osaanko uida. Omissa työtavoissani olisin voinut hakea puhtaan kokeilun sijaan hanakammin tietoa internetistä. Tosin lähes

kaikki virheilmoitukset, joita AWS antoi, olivat ns. maagisia: samalle virheelle löytyi internetistä monta erilaista ratkaisuvaihtoehtoa, eikä aina ollut niin selvää mikä ratkaisu sopisi juuri meidän tilanteeseen.

5.9 Tuotteen käytettävyys

Jos tulevaisuudessa halutaan luoda uusi yksikkö, joka toimii suunnittelemallamme tavalla, se on helppoa, eikä itse materiaalitkaan maksa kovinkaan paljon (Raspirauta n. 35 €, sensorihinnasta ei ole tarkkaa tietoa, mutta eivät ole kalliita). Jäljelle jää vain AWS:n puolella Thing-olion luominen ja Analyticsin puolella tarvittavien järjestelmien kasaaminen sekä liittäminen QS:ään, kuten työssä on jo esitetty. Jos lopullisesta versiosta tulee sellainen, kuin asiakas haluaa, niin laajentaminen toisiin kohteisiin on todella helppoa. Toteutus on ainakin kustannustehokas. Jos tosiaan energialaskut saadaan tulevaisuudessa toimimaan, niin data-analyysi mahdollistaisi energian ja veden kulutuksen ennustamisen jollain mittakaavalla. Erityisesti mittaustietoa on jo ehtinyt kertyä pidemmältä aikaväliltä.

. .

Lähteet

Lisää lähteet siinä järjestyksessä, kuin ne on mainittu tekstissä.

1. Eben Upton. 2016. Ten millionth Raspberry Pi, and a new kit. Verkkoainesto. <Ten millionth Raspberry Pi, and a new kit > 8.9.2016 Luettu 19.10.2021.
2. Eben Upton. 2018.Raspberry Pi 3 Model B+ on sale now at 35$. Verkkoaineisto. < Raspberry Pi 3 Model B+ on sale now at $35 > 14.3.2018. Luettu 19.10.2021.
3. EbenUpton. 2019. Raspberry Pi numbers get stale fast. Verkkoaineisto. "@lisn92 @bateskecom @arturo182 @Raspberry_Pi Raspberry Pi numbers get stale fast. We sold our thirty-millionth unit…" (Tweet). Luettu 25.2.2020 – via Twitter.
4. Python kuva. Verkkoainesto. < python - Bing images > Luettu 20.10.2021
5. Mary Brickenstein Hofschen. 2021. The New MQTT 5 Protocol – MQTT 5 Essentials Part 1. Verkkoainesto. < The New MQTT 5 Protocol - MQTT 5 Essentials Part 1 (hivemq.com) > Luettu 20.10.2021.
6. MQTT kuva. Verkkoaineisto. < mqtt2-1.jpg (1016×604) (domopi.eu) > Luettu 20.10.2021.
7. Amazon. 2021. Amazon QuickSight – Business Intelligence Service – Amazon Web Services. Verkkoaineisto. <Amazon QuickSight - Business Intelligence Service - Amazon Web Services >. Luettu 27.10.2021.
8. Amazon. 2021. Amazon CloudWatch – Application and Infrasturucture Monitoring. Verkkoaineisto.<Amazon CloudWatch - Application and Infrastructure Monitoring> Luettu 27.10.2021.
9. Pete Cheslock. 2021. Cloud Security, Identity, and Compliance Products – Amazon Web Services. Verkkoaineisto. <Cloud Security, Identity, and Compliance Products – Amazon Web Services (AWS)> .Luettu 28.10.2021.
10. Williams O. 2021. Scalability in Amazon Web Services. Verkkoaineisto. <Scalability in Amazon Web Services | by Williams O | Medium> Luettu 2.11.2021.

11. Lämpöenergia, Ominaislämpökapasiteetti. Verkkoaineisto. <Ominaislämpökapasiteetti (peda.net)> Viitattu 2.11.2021.

Liitteet

Altaan anturi pohdintaa

Lämpötila (Allas)

- DS18B20 tai vastaava 1-wire anturi toimii varmaan parhaiten. Etenkin jos vain yksi anturi per väylä. Useampi vaatii vähän lisäsäätöä.

Tuuli (=sääasema)

- Sivulla http://weewx.com/hardware.html listatut laitteet saataisiin todennäköisesti toimimaan Raspin kanssa melko helposti. Itse WeeWX-ohjelmistoakin voitaneen käyttää erillisenä komponenttina vaikka se onkin GPL 3.

Pulssi- ja kärkitiedot

- Toteutettavissa, mutta mielellään optoerottimilla. Valmiita hattuja tuntuisi olevan heikosti saatavilla, joten joutuu ehkä tekemään itse. Tämä riippuu paljon mitä halutaan yhdistää.

Lämpötilakoodi proto

```
import os
import glob
import time

base_dir = '/sys/bus/w1/devices/'
# Get all the filenames in the path base_dir.
device_folders = glob.glob(base_dir + '*')
device_file[]
n = 0;
for i in device_folders:
    device_file[n] = device_folders[n] + '/w1_slave'
    n += 1

def read_rom(folder):
    name_file=folder+'/name'
    f = open(name_file,'r')
    return f.readline()

def read_temp_raw(file):
    f = open(file, 'r')
    lines = f.readlines()
    f.close()
```

```
    return lines

def read_temp(file):
    lines = read_temp_raw(file)
    # Analyze if the last 3 characters are 'YES'.
    if lines[0][-4] == 'Y' and  lines[0][-3] == 'E' and lines[0][-
2] == 'S':
        time.sleep(0.1)
    else:
        return -273, -273.15
    # Find the index of 't=' in a string.
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        # Read the temperature .
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        return temp_c



while True:

    n = 0
    for i in device_folders:
        print(' rom: '+ read_rom(device_folders[n]))
        print('C=%3.3f'% read_temp(device_file[n]))
        print('\n')
        n += 1



    time.sleep(1)
```

AWS Part.1

- Create **thing object and IoT Policy**(iot-dg.pdf, p. 34-37)
- Remember to store keys and certificates

Raspberry (Python ed.)

- Download and RaspberryPiOs and flash it to micro SD card
- Put file “ssh” (contents do not matter) under /boot/
- Insert SD card and Ethernet cable and power up Raspberry Pi
- Connect via SSH (pi:raspberry)

- enable SSH permanently (sudo raspi-config)
- Update (sudo apt update followed by sudo apt upgrade) and reboot (sudo shut-down -r now)
- Install the required tools and libraries for the AWS IoT Device SDK (iot-dg.pdf, p. 53)
- Install AWS IoT Device SDK (iot-dg.pdf, p. 53-54)
- Set up monitoring of topic “topic_1” at AWS console: Viewing MQTT messages in the MQTT client (iot-dg.pdf, p. 61-63)
- Install and run the sample app (pub_sub_t1.py) (iot-dg.pdf, p. 56-58)
    - Use pubsub_t1.py and launch script (test_pubsub.sh) instead of pub-sub.py mentioned in the iot-dg.pdf
    - **Remember to edit correct paths for the keys and certificates in test_pubsub.sh script!**
    - Check that you can receive the messages at AWS console

AWS Part.2

- More info in iotanalytics-ug.pdf
- Go to IoT Analytics section
- Select Channel->create channel
    - Specify channel details: Insert unique and relevant name for the channel -> Next
    - Choose storage option: Service Managed Storage (for now) -> Next
    - Configure source:
        - IoT Core Topic filter: topic_1
        - IAM Role: Create New (First time) -> Next
    - Click create channel

Select Data Stores -> create data store

- Specify data store details: Insert unique and relevant name for the data store -> Next
- Configure storage type: Service Managed Storage (for now) -> Next
- Configure data format: choose JSON -> Next
- Click Create Data Store

Select Pipelines -> Create pipeline

- Setup pipeline ID and sources:
    - Pipeline name: Insert unique and relevant name
    - Pipeline source: Select channel that was just created
    - Pipeline output: Select data store just created -> Next
- Infer message attributes: Add the following:
    - name:dev_id, value:0, datatype: number
    - name:timestamp, value:0, datatype: number
    - name:value, value:0, datatype: number
    - -> next
- Add a pipeline activity: Select “Select Attributes from the message” -> add
    - select the three attributes from last step (dev_id etc.) ->Next
- -> Create pipeline

Select Datasets -> create datasets

- Select a type: -> Create SQL
- Specify dataset details
    - Dataset name: Insert unique and relevant name
    - Select data store source: Select data store made previously ->Next
- Author query: Here you can set contents of the data set

        - Example: SELECT * FROM [data store name here] LIMIT 10 ->Next
    - Data selection filter: -> Next
    - Set query schedule: -> Next
    - Configure the results of your analytics: -> Next
    - Configure the delivery rules of your analytics results: -> Next
    - (Create dataset)

Somewhere you create a rule for a channel with SQL syntax: SELECT dev_id, timestamp, value FROM 'topic_1'

- Possibly Iot Core -> Rules -> create -> add action -> Send a message to IoT Analytics

Test dataset by going to datasets-> click the dataset created -> Contents -> click “run now”. Downloadable result (.csv) should appear below

Shell skripti joka ajetaan ennen MQTT-lämpötila koodia

```
#!/bin/bash
if test ! -e /sys/bus/w1/devices/
then
  sudo dtoverlay w1-gpio gpiopin=4 pullup=0
fi
python3 /home/pi/test/pub_temps.py --topic pool_temps --root-
ca ~/aws_certs/G2-RootCA1.pem --cert ~/aws_certs/1b24705559-certifi-
cate.pem.crt --key ~/aws_certs/1b24705559-private.pem.key --end-
point a1bzrqjclf6dej-ats.iot.eu-central-1.amazonaws.com
```

MQTT-lämpötila koodi

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0.

import argparse
from awscrt import io, mqtt, auth, http
from awsiot import mqtt_connection_builder
import sys
import threading
import time
from uuid import uuid4
import random
import glob
```

```
# This sample uses the Message Broker for AWS IoT to send and re-
ceive messages
# through an MQTT connection. On startup, the device con-
nects to the server,
# subscribes to a topic, and begins publishing messages to that topic.
# The device should receive those same messages back from the mes-
sage broker,
# since it is subscribed to that same topic.

parser = argparse.ArgumentParser(description="Send and receive mes-
sages through and MQTT connection.")
parser.add_argument('--endpoint', required=True, help="Your AWS IoT cus-
tom endpoint, not including a port. " +
                                                       "Ex: \"abcd123456wx
yz-ats.iot.us-east-1.amazonaws.com\"")
parser.add_argument('--port', type=int, help="Specify port. AWS IoT sup-
ports 443 and 8883.")
parser.add_argument('--cert', help="File path to your client certifi-
cate, in PEM format.")
parser.add_argument('--key', help="File path to your pri-
vate key, in PEM format.")
parser.add_argument('--root-ca', help="File path to root certificate au-
thority, in PEM format. " +
                                      "Neces-
sary if MQTT server uses a certificate that's not already in " +
                                      "your trust store.")
parser.add_argument('--client-id', default="test-
" + str(uuid4()), help="Client ID for MQTT connection.")
parser.add_argument('--topic', default="test/topic", help="Topic to sub-
scribe to, and publish messages to.")
parser.add_argument('--message', default="Hello World!", help="Mes-
sage to publish. " +
                                                             "Spec-
ify empty string to publish nothing.")
parser.add_argument('--count', default=5, type=int, help="Number of mes-
sages to publish/receive before exiting. " +
                                                          "Spec-
ify 0 to run forever.")
parser.add_argument('--use-websocket', default=False, ac-
tion='store_true',
    help="To use a websocket instead of raw mqtt. If you " +
    "specify this option you must specify a region for signing.")
parser.add_argument('--signing-region', default='us-east-
1', help="If you specify --use-web-socket, this " +
    "is the region that will be used for computing the Sigv4 signature")
parser.add_argument('--proxy-host', help="Hostname of proxy to con-
nect to.")
```

```
parser.add_argument('--proxy-port', type=int, de-
fault=8080, help="Port of proxy to connect to.")
parser.add_argument('--verbosi-
ty', choices=[x.name for x in io.LogLevel], default=io.LogLevel.No-
Logs.name,
    help='Logging level')

# Using globals to simplify sample code
args = parser.parse_args()

io.init_logging(getattr(io.LogLevel, args.verbosity), 'stderr')

received_count = 0
received_all_event = threading.Event()
base_dir = '/sys/bus/w1/devices/'
# Get all the filenames begin with 28 in the path base_dir.
device_folders = glob.glob(base_dir + '28-*')

# Callback when connection is accidentally lost.
def  (connection, error, **kwargs):
    print("Connection interrupted. error: {}".format(error))


# Callback when an interrupted connection is re-established.
def on_connection_resumed(connection, return_code, session_pre-
sent, **kwargs):
    print("Connection resumed. return_code: {} session_present: {}".for-
mat(return_code, session_present))

    if return_code == mqtt.ConnectReturnCode.ACCEPTED and not ses-
sion_present:
        print("Session did not persist. Resubscribing to existing top-
ics...")
        resubscribe_future, _ = connection.resubscribe_existing_topics()

        # Cannot synchronously wait for resubscribe result be-
cause we're on the connection's event-loop thread,
        # evaluate result with a callback instead.
        resubscribe_future.add_done_callback(on_resubscribe_complete)


def on_resubscribe_complete(resubscribe_future):
        resubscribe_results = resubscribe_future.result()
        print("Resubscribe results: {}".format(resubscribe_results))

        for topic, qos in resubscribe_results['topics']:
            if qos is None:
```

```
                sys.exit("Server rejected resubscribe to topic: {}".for-
mat(topic))


# Callback when the subscribed topic receives a message
def on_message_received(topic, payload, dup, qos, retain, **kwargs):
    print("Received message from topic '{}': {}".format(topic, payload))
    global received_count
    received_count += 1
    if received_count == args.count:
        received_all_event.set()

if __name__ == '__main__':
    # Spin up resources
    event_loop_group = io.EventLoopGroup(1)
    host_resolver = io.DefaultHostResolver(event_loop_group)
    client_bootstrap = io.ClientBootstrap(event_loop_group, host_re-
solver)

    proxy_options = None
    if (args.proxy_host):
        proxy_options = http.HttpProxyOp-
tions(host_name=args.proxy_host, port=args.proxy_port)

    if args.use_websocket == True:
        credentials_provider = auth.AwsCredentialsProvider.new_de-
fault_chain(client_bootstrap)
        mqtt_connection = mqtt_connection_builder.websockets_with_de-
fault_aws_signing(
            endpoint=args.endpoint,
            client_bootstrap=client_bootstrap,
            region=args.signing_region,
            credentials_provider=credentials_provider,
            http_proxy_options=proxy_options,
            ca_filepath=args.root_ca,
            on_connection_interrupted=on_connection_interrupted,
            on_connection_resumed=on_connection_resumed,
            client_id=args.client_id,
            clean_session=False,
            keep_alive_secs=6)

    else:
        mqtt_connection = mqtt_connection_builder.mtls_from_path(
            endpoint=args.endpoint,
            port=args.port,
            cert_filepath=args.cert,
            pri_key_filepath=args.key,
            client_bootstrap=client_bootstrap,
```

```
            ca_filepath=args.root_ca,
            on_connection_interrupted=on_connection_interrupted,
            on_connection_resumed=on_connection_resumed,
            client_id=args.client_id,
            clean_session=False,
            keep_alive_secs=6,
            http_proxy_options=proxy_options)

    print("Connecting to {} with client ID '{}'...".format(
        args.endpoint, args.client_id))

    connect_future = mqtt_connection.connect()

    # Future.result() waits until a result is available
    connect_future.result()
    print("Connected!")

    # Subscribe
 #   print("Subscribing to topic '{}'...".format(args.topic))
  #  subscribe_future, packet_id = mqtt_connection.subscribe(
   #     topic=args.topic,
    #    qos=mqtt.QoS.AT_LEAST_ONCE,
     #   callback=on_message_received)

    #subscribe_result = subscribe_future.result()
    #print("Subscribed with {}".format(str(subscribe_result['qos'])))

    # Publish message to server desired number of times.
    # This step is skipped if message is blank.
    # This step loops forever if count was set to 0.
    if args.message:
        if args.count == 0:
            print ("Sending messages until program killed")
        else:
            print ("Sending {} message(s)".format(args.count))

#       publish_count = 1
        while True:
            n=0
            for i in device_folders:
                ff = open("{}/temperature".format(device_folders[n]),"r")
                temp = float(ff.read()) / 1000
                split_str = device_folders[n].split("/")
                #print("{}\n".format(split_str[5]))
                message = "{\n" + "  \"dev_id\": \"{}\",\n".for-
mat(split_str[5]) + "  \"timestamp\": {},\n".for-
mat(int(time.time())) + "  \"temperature\": {}".format(temp) + "\n}"
                ff.close()
```

```
                mqtt_connection.publish(
                topic=args.topic,
                payload=message,
                qos=mqtt.QoS.AT_LEAST_ONCE)
                n += 1
                time.sleep(0.05)
#        while (publish_count <= args.count) or (args.count == 0):
#            #message = "{} [{}]".format(args.message, publish_count)
#            temp = random.randint(0, 9)
#            mes-
sage = "{\n" + "  \"dev_id\": 1001,\n" + "  \"timestamp\": {},\n".for-
mat(int(time.time())) + "  \"value\": {}".format(temp) + "\n}"
#            print("Publishing message to topic '{}':\n{}".for-
mat(args.topic, message))

 #           time.sleep(1)
 #           publish_count += 1

    # Wait for all messages to be received.
    # This waits forever if count was set to 0.
    if args.count != 0 and not received_all_event.is_set():
        print("Waiting for all messages to be received...")

    received_all_event.wait()
    print("{} message(s) received.".format(received_count))

    # Disconnect
    print("Disconnecting...")
    disconnect_future = mqtt_connection.disconnect()
    disconnect_future.result()
    print("Disconnected!")
```

CSV-tiedosto haluttuun muotoon

```
import pandas as pd
import datetime
import os
import csv
```

```
def converter1(csv):

    data = pd.read_csv(csv)

    names = [None] * 119315
    for i in range(0, len(data)):
        row = data.iloc[i]['dev_id']
        a = str(row)
        if(a == "28-01193275fee0"):
            names[i] = "Anturi 4, Laitekotelo / Altaan täyttö"
        if (a == ("28-01193237af61")):
            names[i] = "Anturi 1, Tuleva kylmävesi"
        if (a == "28-011932301a67"):
            names[i] = "Anturi 3, Altaalle lähtevä vesi"
        if (a == "28-0119323e5a73"):
            names[i] = "Anturi 2, Vaihtimelle tuleva vesi (kierrätettyä)"

    data["names"] = pd.Series(names)

    temps1 = [None] * 119315
    temps2 = [None] * 119315
    temps3 = [None] * 119315
    temps4 = [None] * 119315
    for i in range(0, len(data)):
        row = data.iloc[i]['dev_id']
        a = str(row)
        if(a == "28-01193275fee0"):
            temps4[i] = data.iloc[i]['temperature']
        if (a == ("28-01193237af61")):
            temps1[i] = data.iloc[i]['temperature']
        if (a == "28-011932301a67"):
            temps3[i] = data.iloc[i]['temperature']
        if (a == "28-0119323e5a73"):
            temps2[i] = data.iloc[i]['temperature']

    data["Anturi 1, Tuleva kylmävesi"] = pd.Series(temps1)
    data["Anturi 2, Vaihtimelle tuleva vesi(kierrätettyä)"] = pd.Se-
ries(temps2)
    data["Anturi 3, Altaalle lähtevä vesi"] = pd.Series(temps3)
    data["Anturi 4, Laitekotelo / Altaan täyttö"] = pd.Series(temps4)

    dates = [None] * 119315
    for i in range(0, len(data)):
        row = data.iloc[i]['timestamp']
        a = int(row)
        date = datetime.datetime.utcfromtimestamp(a)
        targetDate = date.strftime("%Y-%m-%d %H:%M:%S")
        dates[i] = targetDate
```

```
    data["date"] = pd.Series(dates)

    print(data['names'])
    data.to_csv('modifyied.csv', index=False, header=True)

    return data


def main():
    data = converter1('modifyied.csv')


if __name__ == "__main__":
    main(
```

CVS:stä AWS:ään koodi niiltä osin, kun eroaa MQTT-lämpötila koodista

```
def deleteLineFromCSV():

    data = pd.read_csv('VTS5.csv')

    data.drop(data.columns[[1]], axis=1)

    data.to_csv('VTS5.csv', index=False, header=True)


def getLineFromCSV(i):

    data = pd.read_csv('VTS5.csv')
    str1 = data.iloc[i]['dev_id']
    str2 = data.iloc[i]['timestamp'],
    str3 = data.iloc[i]['temperature']

    return str1, str2, str3

##this is just the whie from MQTT which is only dif-
frence wtih the def's above

 publish_count = 1
        while (publish_count <= args.count) or (args.count == 0):
            split_str = getLineFromCSV(1)
            deleteLineFromCSV()
            message = "{\n" + "  \"dev_id\": \"{}\",\n".for-
mat(split_str[0]) + "  \"timestamp\": {},\n".format(
```

```
                split_str[1]) + "  \"temperature\": {}".for-
mat(split_str[2]) + "\n}"
            print("Publishing message to topic '{}':\n{}".format(
                args.topic, message))
            mqtt_connection.publish(
                topic=args.topic,
                payload=message,
                qos=mqtt.QoS.AT_LEAST_ONCE)
            time.sleep(1)
            publish_count += 1
```

Sähköposti hälytyksen luominen

Creating an e-mail alarm for values that go under or above a limit on data set:

Pre-condition: IoTCore and IoTAnalytics set up done.

With this task working IAM user needs to have the following roles.

• Create a Role for AWS IoT Events, on IAM services. Use desired policies for IoTEvents (for test case I

used AllAccess but it might be a security issue).

• Create a Role for AWS SNS, on IAM services.

• Create SNS topic on SNS services.: From Details choose Standard version, and name it.

• On test version other fields where left blank, but in production it might be recomended that the subscribe

access could be restricted.

• Got to IoT Events and create detector model.

• Create Input: name it and load example json, where is in json format the analysed data

• In our case json wtih dev_id, timestamp, temperature etc.

• On Event diagram (on state),click events, there appears a choose bar, take OnInput: add Event

• Name event, add a condition where should be used the former created input

• Click on Add Action. Choose ”Send SNS message”

• Choose formerly created SNS topic from menu SAVE

• Test Analysis with Run

• If no errors, Publish

• Go to IoT Core, there Set For the detected topic a Rule under bar Act.

• Click Rules → Create → set name and SQL formated rule, Actions: Send IoTEventInput, choose the

created input.

• On SNS topics choose the created topic and create a Subscription, take from menu protocol Email →and the

person’s email that is to be reached. → create The subscription must be confirmed via email.

Now when limit is reached the person will get an email...