Joonatan Kallio

Julkiset pilvialustat IoT-datan keruussa

Tietotekniikan pro gradu -tutkielma

5. lokakuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Joonatan Kallio

Yhteystiedot: joonatan.g.kallio@student.jyu.fi

Ohjaaja: Timo Hämäläinen

Työn nimi: Julkiset pilvialustat IoT-datan keruussa

Title in English: Public cloud platforms in IoT data collection

Työ: Pro gradu -tutkielma

Opintosuunta: Tietotekniikka

Sivumäärä: 96+15

Tiivistelmä: Pilvialustat tarjoavat hyvän pohjan kehittää sovelluksia, jotka hyödyntävät IoT-laitteita ja niillä kerättyä dataa. Tässä tutkimuksessa tutustutaan kolmeen julkiseen pilvialustaan IoT-laitteilla tapahtuvan datan keruun näkökulmasta. Pilvialustat ovat Amazonin AWS, Microsoftin Azure ja Googlen Cloud Platform. Tutkimuksessa myös vertaillaan pilvialustojen omaksuttavuutta sekä niiden tarjoamia työkaluja. Mikään alustoista ei ole yksiselitteisesti muita parempi, vaan niiden vahvuudet vaihtelevat työkalujen välillä. IoT-laitteiden kanssa kommunikointi sekä datan keruu ja tallennus onnistuvat kuitenkin kaikilla pilvialustoilla.

Avainsanat: pilviteknologiat, pilvialustat, PaaS, sovellusalusta palveluna, IoT, esineiden internet, Raspberry Pi

Abstract: Cloud platforms offer a good basis for developing applications that utilize IoT devices and the data they collect. This study explores three public cloud platforms from the perspective of data collection from IoT devices. The cloud platforms are Amazon's AWS, Microsoft's Azure and Google's Cloud Platform. The study also compares the adoptability of the cloud platforms and the tools they offer. None of the platforms is unequivocally better than the others, but their strengths vary between their tools. Communicating with IoT devices as well as data collection and storage can be done on all cloud platforms.

Keywords: cloud technologies, cloud platforms, PaaS, platform as a service, IoT, internet of things, Raspberry Pi

Esipuhe

Kiitokset ohjaajalleni Timo Hämäläiselle kannustavasta ohjauksesta kirjoitusprosessin aikana. Suurimmat kiitokset myös rakkaalle puolisolleni Jennalle tuesta ja kannustuksesta koko työn aikana sekä pojalleni Väinölle kärsivällisyydestä.

Jyväskylässä 5. lokakuuta 2021

Joonatan Kallio

Termiluettelo

ACID	*Atomicity, Consistency, Isolation and Durability*. Atomisuus, eheys, eristyneisyys ja pysyvyys ovat tietokantatransaktioiden ominaisuuksia, jotka takaavat datan eheyden myös virhetilanteen sattuessa.
AMQP	*Advanced Message Queuing Protocol* on erityisesti finanssialaa varten kehitetty M2M-viestiprotokolla.
API	*Application Programming Interface*, ohjelmointirajapinta. Rajapinta, jonka kautta ohjelmistot pystyvät kommunikoimaan keskenään itsenäisesti.
AWS	*Amazon Web Services* on Amazonin PaaS-mallinen pilvialusta.
Azure	Microsoftin PaaS-mallinen pilvialusta.
CoAP	*Constrained Application Protocol* on pienitehoisille laitteille kehitetty M2M-viestiprotokolla.
CRUD	*Create, Read, Update, Delete*. Lyhenne, jolla viitataan tyypillisiin datankäsittelyoperaatioihin — datan luonti, luku, päivitys ja poisto.
DTLS	*Datagram Transport Layer Security* on yhteydettömän protokollan (esim. UDP) päällä toimiva salausprotokolla.
GCP	*Google Cloud Platform* on Googlen PaaS-mallinen pilvialusta.
HTTP	*Hypertext Transfer Protocol* on hypermedian välitystä varten kehitetty pyyntö/vastaus-tyyppinen protokolla.
I2C-väylä	De facto -standardi laitteistotason tiedonsiirtoväylä.
IaaS	*Infrastructure as a Service*, infrastruktuuri palveluna.
IoT	*Internet of Things*, esineiden internet.
JSON	*JavaScript Object Notation* on yleiskäyttöinen, tekstipohjainen ja ihmisluettava datansiirtoformaatti.
JWT	*JSON Web Token* on tiivis formaatti esimerkiksi autentikaatiossa tarvittavien vaateiden kommunikointiin. Vaateita ovat esimerkiksi pääsyoikeuden myöntämis- ja erääntymisajankohdat.

M2M	*Machine to Machine*, laitteelta laitteelle.
MQTT	*Message Queue Telemetry Transport* on julkaise/tilaa-tyyppinen M2M-viestiprotokolla.
NoSQL-tietokanta	*Not only SQL* -tietokanta on tietokanta, joka ei noudata perinteistä relaatiotietokantamallia. Se voi olla esimerkiksi paremmin skaalautuva ja vaatia vähemmän ylläpitoa kuin relaatiotietokanta, mutta toisaalta ei välttämättä takaa ACID-ominaisuuksia datalle.
Osiointiavain	Tietokannan tietoalkioista löytyvä attribuutti, jonka perusteella tietoalkiot voidaan jakaa loogisiin osioihin.
PaaS	*Platform as a Service*, sovellusalusta palveluna.
Pääavain	Tietokannan tietoalkioista löytyvä attribuutti tai attribuuttijoukko, joka yksilöi tietoalkion.
REST	*Representational State Transfer* on ohjelmistoarkkitehtuurityyli, joka on suunniteltu hajautetuille hypermediajärjestelmille. Monessa tapauksessa HTTP-protokollaa käytetään REST-tyylisissä järjestelmissä, mutta REST ei ole protokollasidonnainen.
SaaS	*Software as a Service*, ohjelmisto palveluna.
SDK	*Software Development Kit* on asennettava paketti, joka sisältää ohjelmistokehityksessä tarvittavia työkaluja.
TCP	*Transmission Control Protocol* on TCP/IP-pinon kuljetuskerroksen protokolla. Mahdollistaa yhteydellisen tietoliikenteen kahden laitteen välillä.
TLS	*Transport Layer Security* on luotettavan yhteysprotokollan (esim. TCP) päällä toimiva salausprotokolla.
UDP	*User Datagram Protocol* on TCP/IP-pinon kuljetuskerroksen protokolla. Mahdollistaa yksinkertaisen yhteydettömän tietoliikenteen laitteiden välillä.
URI	*Uniform Resource Identifier* on resurssin yksilöivä tunniste. Se ei välttämättä kerro resurssin sijaintia. Esimerkiksi verkkosi-

	vun URL-osoite.
URL	*Uniform Resource Locator* on URI, joka yksilöinnin lisäksi kertoo resurssin sijainnin esimerkiksi verkossa.

Kuviot

Sisältö

1 Johdanto

Nykyään ohjelmistokehitykseen on tarjolla useita suosittuja ja monipuolisia julkisia pilvialustoja. Suuren tarjonnan keskellä haasteeksi nousee, miten alustoista valitaan sopivin, jos tavoitteena on kehittää IoT-järjestelmä. Tähän haasteeseen useat tutkijat ovat tarjonneet ratkaisuksi taulukkomuotoisia ominaisuusvertailuja (ks. esim. Bastos 2019; Hejazi ym. 2018; Pflanzner ja Kertesz 2016). Nämä kuitenkin näyttäisivät keskittyvän monesti pintapuoliseen pilvialustojen ominaisuuksien listaukseen ja vertailuun sitä kautta. Usein varsinkin kehittäjien näkökulma ja esimerkiksi pilven käyttöönoton helppous uudelle kehittäjälle jää taka-alalle. Myös pilvialustojen tarjoamien kehitystyökalujen vertailu näyttäisi jäävän usein vähäiseksi. Näihin näkökulmiin tämä tutkimus pyrkii kokoamaan tietoa.

Tässä tutkimuksessa etsitään vastausta kysymyksiin

- kuinka keskeisimpiä julkisia pilvialustoja voidaan hyödyntää IoT-datan keruussa sekä
- kuinka nämä pilvialustat eroavat toisistaan käytännön kehitystyössä, kun kyseessä on IoT-dataa keräävä järjestelmä.

Tutkimuksessa kartoitetaan prototyypin kautta muutaman käytetyimmän PaaS-pilvialustan tarjoamia ominaisuuksia. Erityisesti keskitytään näiden yhtäläisyyksiin ja eroihin käytännössä. Tähän kuuluvat esimerkiksi erot käyttöönotossa ja käytössä datan tallennuksen osalta sekä erot tarjotuissa API:ssa (so. engl. *Application Programming Interface*, ohjelmointirajapinta).

Tämän tutkimuksen teoriaosiossa (luvut 2 ja 3) käsiteltävistä aiheista kerätään tietoa etsimällä kuvailevalla kirjallisuuskatsauksella (Salminen 2011, luku 2.1) olemassaolevia tutkimuksia ja julkaisuja. Empiirisessä osiossa (luvut 4 ja 5) käsiteltävistä PaaS-alustoista kerätään tietoa alustojen palveluntarjoajien dokumentaatioista sekä käytännön prototyypin kautta.

Tutkimuksen empiirisessä osiossa käytetään tutkimusmetodina konstruktiivista tutkimusta (Kasanen, Lukka ja Siitonen 1991; Hevner ym. 2004). Konstruktiivinen tutkimus sopii menetelmäksi tähän, koska tutkimuksen tarkoituksena on tuottaa yleiskatsaus/vertailu (artefakti) kolmen keskeisimmän PaaS-pilvialustan tarjoamista työkaluista IoT-datan keräämiseen

käytännössä. Tässä pilvialustojen käytännön testauksen ja vertailun apuna käytetään Raspberry Pi -tietokoneen päälle rakennettua prototyyppiä.

Tutkielman luvussa 2 käydään kevyesti läpi aiempaa tutkimusta aihealueelta. Luvussa 3 käsitellään esineiden internetiin ja pilvipalveluihin liittyviä käsitteitä tämän tutkimuksen kannalta olennaisimmista näkökulmista. Luvussa 4 tutustutaan kolmeen keskeiseen julkiseen pilvialustaan sekä rakennetaan dataa keräävä ja pilveen tallentava IoT-prototyyppi. Luvussa 5 katsotaan IoT-prototyypin kehittämisestä saatujen kokemusten pohjalta, miten tutkittavat pilvialustat eroavat toisistaan. Luvussa 6 pohditaan saatuja tuloksia ja tehdään johtopäätöksiä. Luvussa 7 päätetään tutkielma vetämällä käsitellyt asiat yhteen.

2 Aiempi tutkimus

Tutkielmaa varten haettiin aiempaa tutkimusta kuvailevalla kirjallisuuskatsauksella (Salminen 2011, luku 2.1). Käytännössä tämä tapahtui käyttäen ensisijaisesti hakukoneita “IEEE Xplore” (2021), “Google Scholar” (2021) ja “Scopus - Document Search” (2021). Hakukoneilla haettiin joitakin sanojen ”iot”, ”data collecting”, ”cloud services” ja ”survey” yhdistelmiä.

Löydetyistä artikkeleista suuri osa käsittelee IoT-pilvijärjestelmiä konstruktiivisella tutkimuksella käytännön sovelluksen kautta. Toisin sanoen niissä rakennetaan artefakti, jota evaluoidaan reaalimaailman kontekstissa. Aiheiden osalta monessa artikkelissa tietoturva nousee keskeiseksi teemaksi. Myös reuna- ja sumulaskenta sekä ylipäätään reunalla olevien IoT-laitteiden keräämän datan pelkistys (engl. data reduction) esiintyy monessa artikkelissa keskeisenä aiheena.

IoT-laitteita ja pilviteknologioita käsittelevien artikkeleiden tyypillisimpiä sovellusalueita näyttäisivät olevan esimerkiksi terveydenhuolto (Winnie, E ja Ajay 2018), älykaupungit (Iasio ym. 2019; Serrano, Baldassarre ja Stroulia 2016)ja maatalous (Bojan ym. 2015; Yuyanto ja Liawatimena 2018). Joukosta löytyy myös älykoteihin liittyviä artikkeleita (Dutta ym. 2020). Tässä tutkimuksessa keskitytään erityisesti älykoteihin sekä muihin saman kokoluokan kohteisiin IoT:n sovellusalana.

Entuudestaan tutkittavasta aiheesta tiedetään yleisellä tasolla, että pilviteknologioita hyödynnetään nykyään IoT-laitteiden datan keräämisessä myös älykotijärjestelmissä. Sekä teollisuuden että tutkijoiden näkökulmasta aihealueella kiinnostaa myös teknisemmät asiat, kuten minkälaisia palveluita käytännön tasolla julkiset pilvialustat tarjoavat. Näiden alustojen tarjoamista ominaisuusjoukoista löytyykin useita tutkimuksia. Tällaiset tutkimukset kuitenkin keskittyvät pääasiassa ominaisuuksien listaukseen kartoituksenomaisesti ottamatta syvällisemmin kantaa pilvialustojen toiminnallisuuksiin käytännön IoT-pilvijärjestelmien kehitystyössä. Seuraavissa tekstikappaleissa esitellään lyhyesti muutama tällainen tutkimus.

Bastos (2019) teki katsauksen muutaman keskeisimmän julkisen PaaS-mallisen pilvipalvelualustan tarjoamiin tietoturvaominaisuuksiin sekä niiden tukemiin teknologioihin ja proto-

kolliin. Teknologioilla tässä viitataan esimerkiksi ohjelmointikieliin ja laitteistotukeen. Katsauksessa olivat mukana “AWS IoT Core” (2021), “Azure IoT Hub” (2021), “Google Cloud IoT Core” (2021), “IBM Watson IoT” (2021) ja “ARM Mbed Pelion” (2021). Katsauksessa nousi esille, että nämä palveluntarjoajat ottavat tietoturvakysymykset tosissaan, mutta tietoturvaominaisuuksien tarjonnan laajuudessa on vielä eroja. Katsauksessa mainitaan, että näistä palveluntarjoajista AWS (engl. *Amazon Web Services*), Azure ja IBM ovat tehneet useita onnistuneita tapaustutkimuksia IoT-järjestelmien käyttöönotosta omille pilvialustoilleen. Googlella ja ARM:lla ei ole vielä katsauksen julkaisun aikoihin ollut vastaavia portfolioita.

Pflanzner ja Kertesz (2016) tekivät katsauksen, jossa listataan PaaS-pilvialustojen ominaisuuksia IoT-laitteiden näkökulmasta. Katsauksessa pilvialustoista kerrotaan vaihtelevalla tarkkuudella keskittyen isompiin toimijoihin, mutta mukana on myös useita pienempiä pilvialustoja. Suurimmaksi osaksi katsauksessa keskitytään siihen, mitä ominaisuuksia pilvialustoilla on olemassa, eikä pilvialustojen käytöstä käytännössä juuri puhuta. Tämän katsauksen julkaisusta on tutkielmaa kirjoittaessa jo lähes viisi vuotta aikaa, minkä takia alustoilla on luultavasti tapahtunut muutoksia ja ominaisuuksia on todennäköisesti tullut lisää.

Hejazi ym. (2018) kokeilivat ja listasivat 20 eri pilvialustaa ominaisuuksineen. Listaukseen valittujen ominaisuuksien pääpaino oli suurten IoT-järjestelmien vaatimissa toiminnoissa, mutta tässäkään ei erityisesti puhuttu alustoista käytännön työssä.

Ominaisuuskartoitustutkimusten lisäksi pilvialustoista IoT:n saralla on tehty tutkimuksia keskittyen esimerkiksi niiden hintoihin (Kalmar ja Kertesz 2017). Myös hyvän pilvialustan valintamenetelmän löytämisen eteen esimerkiksi älyrakennuksia varten on tehty tutkimusta (Tulenkov, Parkhomenko ja Sokolyanskii 2019). Lisäksi yleiskuvaa on pyritty luomaan pilvi-IoT-kentästä. Tästä esimerkkinä Botta ym. (2016) luovat tutkimuksessaan laajan kokonaiskuvan pilvi-IoT-paradigmasta (engl. *CloudIoT paradigm*) sekä esimerkiksi sen syntyyn vaikuttaneista tekijöistä.

Mainituissa tutkimuksissa ei kuitenkaan suoraan oteta kantaa pilvialustojen käyttöön ja tässä esiintyviin eroihin käytännön esimerkkien kautta. Kirjallisuuskatsauksessa löydetyistä tutkimusartikkeleista Hellbe ja Bohlin (2018) pääsevät yhteisessä kandidaatintutkielmassaan lähimmäksi tätä tavoitetta. He suunnittelevat ja rakentavat prototyypin älyrakennuk-

sen lämmitys- ja ilmanvaihtojärjestelmien ohjausta varten käyttäen sekä AWS-alustaa että Microsoftin Azurea. Prototyypin perusteella he vertailevat näiden kahden pilvialustan soveltuvuutta tähän tarkoitukseen. Ruotsalaisten kandidaatintutkielmasta poiketen tässä tutkielmassa vertaillaan useampaa pilvialustaa ja jätetään esimerkiksi web-sivuihin liittyvät toiminnallisuudet pois vertailusta. Lisäksi tässä tutkielmassa pääpaino on datan keräämisessä kokonaisvaltaisen etäohjaustyyppisen järjestelmän sijaan.

3 Esineiden internet ja pilvialustat

Tässä luvussa käydään läpi keskeisiä IoT-pilvijärjestelmiin liittyviä käsitteitä ja määritelmiä. Käsiteltävästä teoriasta kerrotaan olemassaoleviin tutkimuksiin ja julkaisuihin pohjautuen. Aiheiden käsittelyssä keskitytään tutkielman kannalta olennaisiin näkökulmiin. Aliluvussa 3.1 tarkastellaan esineiden internetiin keskeisesti liittyviä käsitteitä. Aliluvussa 3.2 tutustutaan pilvialustojen maailmaan. Viimeisenä aliluvussa 3.3 käydään läpi verkkoprotokollia, jotka mahdollistavat IoT-laitteiden ja pilvialustojen välisen kommunikaation.

3.1 Esineiden internet

CERP-IoT (2010, luku 3.1.1) määrittelee esineiden internetin (IoT, engl. *Internet of Things*) olevan standardeihin kommunikaatioprotokolliin perustuva dynaaminen maailmanlaajuinen fyysisistä ja virtuaalisista ”esineistä” (engl. *things*) koostuva verkko. Määritelmän mukaan näillä esineillä voi olla kyky esimerkiksi havainnoida ympäristöään tai reagoida saamiinsa viesteihin suorittamalla toiminto, joka vaikuttaa ympäristöönsä. Hyvä esimerkki havainnoivasta esineestä on automaattinen sääasema, joka kerää lämpötila- ja kosteusdataa ympäristöstään. Reagoiva esine sen sijaan voisi olla kerrostalon ulko-ovi, joka aukeaa, kun talon asunnosta painetaan avauspainiketta.

Tässä tutkimuksessa keskitytään havainnoiviin esineisiin. Toisin sanoen keskeisenä teemana on datan kerääminen sensoreiden avulla. Luvussa 4 kehitetään Raspberry Pi 2 -tietokoneeseen perustuva prototyyppi, jolla kerätään dataa ympäristöstä sensoreiden avulla. Sensorit kommunikoivat laitteistotasolla Raspberry Pi:n kanssa I2C-väylän avulla. Raspberry Pi on yhteydessä internetiin ja siirtää kerättyä dataa pilvialustoille. Tätä käsitellään tarkemmin luvuissa 3.2 ja 3.3. Seuraavissa aliluvuissa käydään pintapuolisesti läpi I2C-väylän sekä Raspberry Pi:n rakennetta ja toimintaa. Läpikäynti näiden osalta jätetään vähäiseksi, sillä niiden syvällinen ymmärtäminen ei ole olennaista tämän tutkimuksen kannalta.

3.1.1 I2C-väylä

I2C-väylä on maailmanlaajuisesti käytetty de facto -standardi laitteistokommunikaatiossa (“UM10204 I2C-Bus Specification and User Manual” 2014). Väylän kehitti Philips Semiconductors (nykyään NXP Semiconductors) vuonna 1982. Väylän rakenne on yksinkertainen, sillä se sisältää vain kaksi johdinta — sarjadatajohtimen (SDA) ja sarjakellojohtimen (SCL). Tästä huolimatta väylään voi liittää suuren määrän laitteita, sillä jokaisella laitteella on oma yksilöllinen osoitteensa, jonka perusteella viestit välitetään. Laitteet toimivat viestien välityksen aikana väylässä joko isäntinä tai orjina. Isännät ovat väylässä aktiivisia toimijoita, jotka voivat joko lähettää dataa orjille tai pyytää dataa näiltä. Orjat toimivat isäntien pyyntöjen mukaan. Isäntiäkin voi olla väylässä useita, jos käytetään I2C-väyläprotokollaa, joka tätä tukee.

I2C-spesifikaatio (“UM10204 I2C-Bus Specification and User Manual” 2014) määrittelee useita väylässä käytettäviä protokollia, jotka eroavat toisistaan pääasiassa vain suurimmilta siirtonopeuksiltaan. Normaalitila (engl. *Standard-mode*) tukee siirtonopeuksia 100 kbit/s asti, Nopea tila (engl. *Fast-mode*) tukee 400 kbit/s asti, Nopea tila Plus (engl. *Fast-mode Plus*) tukee 1 Mbit/s asti ja Suurnopeuksinen tila (engl. *High-speed mode*) tukee 3,4 Mbit/s asti. Ultranopea tila (engl. *Ultra Fast-mode*) tukee siirtonopeuksia 5 Mbit/s asti, mutta aiemmista poiketen tämä protokolla on yksisuuntainen ja sallii siksi vain yhden isännän liitämisen väylään viestitörmäysten ehkäisemiseksi. Tämän takia isäntälaite ei voi pyytää orjalaitteita lähettämään viestejä itselleen. Toisin sanoen kyseinen protokolla soveltuu parhaiten esimerkiksi LED-valojen ohjaussignaalin lähettämiseen valojen ohjausyksiköille.

3.1.2 Raspberry Pi ja IoT-käyttöjärjestelmät

Raspberry Pi -tietokonesarja on Raspberry Pi Foundationin (2021) kehittämä. Tietokoneet ovat pienikokoisia ja niille asennetaan yleensä Linux-pohjainen käyttöjärjestelmä. Lisäksi ne pohjautuvat ARM-prosessoreihin ja erikoisuutena niissä on useita ohjelmallisesti ohjattavia GPIO-tappeja (so. engl. *General-Purpose Input/Output*) (“Raspberry Pi Hardware - Raspberry Pi Documentation” 2021). Nämä mahdollistavat esimerkiksi yksinkertaisten digitaalisten sensoreiden tai LED-valojen liittämisen suoraan johtimilla tietokoneeseen. Niiden avulla

myös laitteistotason väyliä pystyy käyttämään helposti. Tästä esimerkkinä luvussa 3.1.1 käsitelty I2C-väylä, jota tämän tutkimuksen empiirisessä osuudessa käytetään. Pienen kokonsa, monipuolisuutensa ja edullisen hintansa takia Raspberry Pi -tietokoneet soveltuvatkin varsin hyvin esimerkiksi prototyyppien rakentamiseen.

Tutkimuksessa Raspberry Pi -tietokoneen käyttöjärjestelmäksi asennetaan Raspberry Pi OS (aiemmin Raspbian) ("Raspberry Pi OS - Raspberry Pi Documentation" 2021). Tämä valikoitui käyttöjärjestelmäksi, koska se on normaalikäyttöön suositeltu käyttöjärjestelmä Raspberry Pi:lle ja se soveltuu myös tässä tutkimuksessa rakennettavan prototyypin pohjaksi. Käyttöjärjestelmästä käytetään Lite-versiota, jossa ei tule graafista käyttöliittymää mukana, sillä sitä ei tässä tutkimuksessa tarvita. Tutkimuksessa käsiteltävien pilvialustojen IoT-palveluille on saatavilla virallisia SDK-paketteja (engl. *Software Development Kit*) Python-ohjelmointikielelle, jota tässä tutkimuksessa käytetään. Nämä paketit toimivat myös Raspberry Pi OS:llä.

Tutkimuksessa käsiteltävät pilvialustat tarjoavat myös omia käyttöjärjestelmiään IoT-laitteille. Ne ovat kuitenkin joko tiukemmin sidoksissa palveluntarjoajien omiin pilvialustoihin tai tarkoitettu käytettäväksi esimerkiksi erityisen pienitehoisissa laitteissa. Seuraavaksi käydään lyhyesti läpi muutama tällainen käyttöjärjestelmä.

"FreeRTOS" (2021) on Amazonin ylläpitämä ja alan de facto -standardiksi muodostunut mikrokontrollereille ja pienitehoisille mikroprosessoreille tarkoitettu reaaliaikainen käyttöjärjestelmä (RTOS, engl. *Real-time Operating System*). Reaaliaikaisen käyttöjärjestelmän tarkoitus on suorittaa määritellyt operaatiot mahdollisimman tarkasti valittuina ajanhetkinä tai valituilla aikaväleillä (Tanenbaum 2015, luku 1.4.8). Näissä järjestelmissä aika on keskeinen parametri toisin kuin yleisemmissä (rinnakkaisissa) käyttöjärjestelmissä, joissa odotusajat voivat venyä pitkiksikin. FreeRTOS-käyttöjärjestelmä olisi mahdollista saada toimimaan myös Raspberry Pi:llä (Hull 2021), mutta tätä ei yritetä tämän tutkimuksen puitteissa.

Amazonin lisäksi myös Microsoft tarjoaa RTOS-käyttöjärjestelmää. Microsoftin ylläpitämä "Azure RTOS" (2021) toimii esittelysivun mukaan suosituimmilla 32-bittisillä mikrokontrollereilla. Azureen tottuneille kehittäjille tässä käyttöjärjestelmässä on etuna tiivis integraatio Azuren IoT-pilvipalveluun. Microsoft ylläpitää lisäksi raskaampaa, mutta monipuolisempaa

Windows-pohjaista IoT-käyttöjärjestelmää. “Windows 10 IoT” (2021) on graafisen käyttöliittymän sisältävä tehokkaammille IoT-laitteille suunnattu käyttöjärjestelmä, jota voi käyttää esimerkiksi polttoaineen tankkausasemien maksupäätteissä tai teollisuusautomaatiossa. Käyttöjärjestelmästä löytyy tuki myös esimerkiksi koneoppimismallien suoritukselle tai konenäön hyödyntämiselle.

Myös Google on kehittänyt oman IoT-käyttöjärjestelmän. “Android Things” (2021) on erityisesti Android-kehittäjille suunnattu IoT-laitteiden käyttöjärjestelmä, joka tukee suoraan useita Googlen pilvipalveluita. Käyttöjärjestelmän aktiivinen kehitys ei-kaupalliseen käyttöön on kuitenkin lopetettu, eikä uusia projekteja ole voinut luoda 5.1.2021 jälkeen. Kaupallinen käyttö on rajattu vain Googlen sopimusasiakkaisiin.

3.2 Pilvialustat

Mell ja Grance (2011) määrittelevät NIST:n (National Institute of Standards and Technology) alaisuudessa pilvilaskennan (engl. *cloud computing*) olevan malli, jolla päästään vaadittaessa ja helposti käsiksi laskentaresursseihin paikasta riippumatta. Laskentaresursseihin luetaan esimerkiksi tietoverkot, palvelimet ja tallennuspalvelut. Tähän malliin kuuluu myös, että käytössä olevia laskentaresursseja voidaan lisätä ja vähentää hyvin pienellä vaivalla — jopa automaattisesti kysynnän muuttuessa. Baun ym. (2011, luku 1.2) tarkentavat omassa määritelmässään pilvilaskennan perustuvan virtualisoituun laskentaan ja tallennuspalveluihin. Näiden avulla abstrahoiduista palveluista saadaan skaalautuvia ja niistä myös maksetaan yleensä käytön mukaan. Skaalautuvalla palvelulla tässä tarkoitetaan sellaista palvelua, joka suoriutuu hyvin käyttäjämäärän suurestakin kasvusta lyhyessä ajassa ilman, että palvelun laatu heikkenee esimerkiksi merkittävän hidastelun tai kaatuilun vuoksi.

Määritelmien mukaan pilvimallin voi jakaa ainakin kolmeen palvelumalliin ja kolmeen käyttöönottomalliin (Mell ja Grance 2011; Baun ym. 2011, luku 3). Käyttöönottomallit ovat yksityinen pilvi, julkinen pilvi ja hybridipilvi. Yksityinen pilvi tarkoittaa mallia, jossa pilvi-infrastruktuuri on kokonaan tarkoitettu vain yhden organisaation käyttöön. Fyysisesti tämän infrastruktuurin ei tarvitse kuitenkaan sijaita organisaation tiloissa ja sen hallinnointikin voi kuulua kolmannelle taholle. Julkinen pilvi sen sijaan on esimerkiksi yrityksen, akateemisen

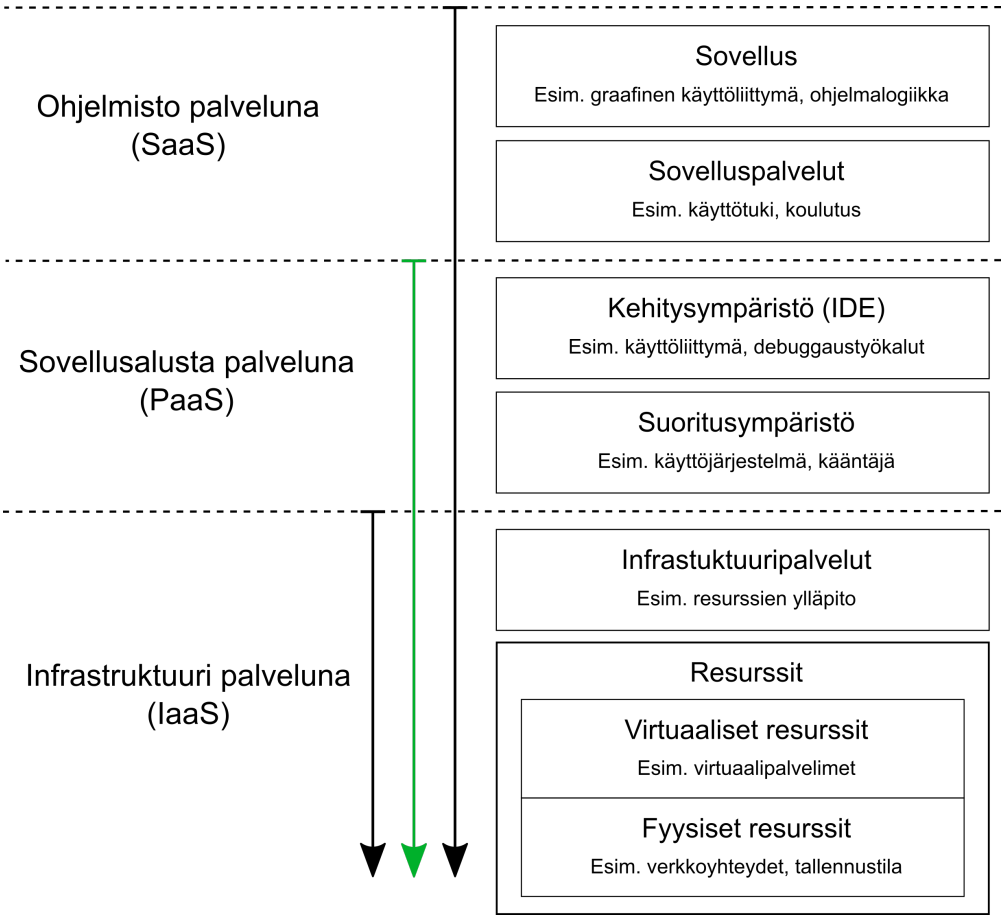
toimijan tai valtion tarjoama kaikille avoin palvelu. Julkisen pilvilaskentapalvelun hallintaan tarjotaan yleensä itsepalveluun perustuva web-portaali, joka mahdollistaa myös palvelun skaalautuvuuden. Hybridipilvi on nimensä mukaan muiden käyttöönottomallien sekoitus. Mell ja Grance (2011) lisäävät määritelmässään tähän joukkoon vielä neljännen käyttöönottomallin, yhteisöpilven, joka sijoittuu yksityisen ja julkisen pilven väliin. Yhteisöpilvessä pilvi-infrastruktuuria voi käyttää useampi organisaatio, joilla on yhteinen pilveä koskeva vaatimus. Tämä voisi olla esimerkiksi jokin tietoturvavaatimus tai erityinen toimintamalli. Tässä tutkimuksessa rajoitutaan käyttöönottomalliltaan julkisiin pilvipalveluihin.

Mellin ja Grancen (2011) sekä Baunin ym. (2011, luku 3.2) määrittelemät pilvilaskennan palvelumallit ovat infrastruktuuri palveluna (IaaS, engl. *Infrastructure as a Service*), sovellusalusta palveluna (PaaS, engl. *Platform as a Service*) ja ohjelmisto palveluna (SaaS, engl. *Software as a Service*). Näitä käsitellään tarkemmin seuraavissa aliluvuissa samoihin lähteisiin pohjautuen. Baun ym. (2011, luku 3.2) lisäävät palvelumallien joukkoon vielä neljännen mallin, ihmiset palveluna (HuaaS, engl. *Humans as a Service*), mutta tätä ei käsitellä tässä tutkimuksessa. Kuviossa 1 esitetään tässä luvussa käsiteltävät pilvipalvelumallit mukaillen Baunin ym. (2011) kirjan kuviota 3.2. Kuvion oikeassa reunassa on esitetty palvelumalleihin kuuluvia kokonaisuuksia sekä muutamia esimerkkejä konkreettisista palveluista tai toiminnoista, joita niihin voi kuulua. Nuolilla on merkitty kokonaisuudet, jotka kuuluvat kuhunkin pilvipalvelumalliin.

3.2.1 IaaS

Infrastruktuuri palveluna, IaaS, on pilvipalvelumalli, jossa asiakkaalle tarjotaan mahdollisuus varata käyttöönsä haluamansa määrän laitteistoresursseja muun muassa palvelimien muodossa yleensä web-portaalin kautta (ks. kuvio 1). Laitteistoresursseihin kuuluu esimerkiksi tallennuskapasiteetti, (virtuaalisen) palvelimen prosessoriytimien määrä ja kellotaajuus sekä muistin määrä. Palvelimille voi asentaa vapaavalintaisen käyttöjärjestelmän, jonka päällä on mahdollista ajaa mielivaltaisia ohjelmia, kuten millä tahansa tietokoneella. Samoin tallennustilaan voi tallentaa tietoa esimerkiksi ohjelmallisesti.

Teknisyytensä vuoksi IaaS-mallisten pilvipalveluiden kohderyhmänä on monessa tapaukses-



Kuvio 1. Pilvipalvelumallit mukaillen Baunin ym. (2011) kirjan kuviota 3.2. PaaS-pilvipalvelumalliin keskitytään tässä tutkimuksessa eniten, ja se on korostettu kuviossa vihreällä.

sa kehittäjät ja tietojärjestelmien ylläpitäjät. Tarkemmin sanottuna tästä mallista saattaisivat hyötyä eniten sellaiset kehittäjät ja ylläpitäjät, jotka tarvitsevat täyden kontrollin ajoympäristöstä, mutta joilla ei ole tarvetta tai resursseja omien palvelinlaitteiden ylläpitoon.

Etuna tässä mallissa verrattuna itse ylläpidettävään laitteistoon on, että asiakkaan ei tarvitse huolehtia fyysisistä palvelinlaitteista, ja esimerkiksi teknistä tukea voi olla — palveluntarjoajasta riippuen — helposti saatavilla. Lisäksi laitteistoresurssit ovat yleensä nopeasti asiakkaan käytössä. Poikkeuksena tähän voisi olla esimerkiksi erityisen suuri tarve laitekapasiteetille, jolloin palveluntarjoaja joutuisi tekemään fyysisiä laitehankintoja täyttääkseen asiakkaan tarpeet.

Haasteeksi IaaS-mallissa voi muodostua se, että asiakas maksaa palvelusta varaamiensa laitteistoresurssien mukaan riippumatta siitä, käyttääkö hän niitä. Tästä syystä asiakkaalle saattaa syntyä turhia kustannuksia. Tapauksesta riippuen myös käyttöjärjestelmien ja sovellusten ajoympäristöjen ylläpito voi kuluttaa asiakkaan resursseja tarpeettomasti.

Esimerkkejä IaaS-mallisista pilvipalveluista ovat “Amazon Simple Storage Service (S3)” (2021), “DigitalOcean” (2021) ja “Microsoft Azure Virtual Machines” (2021).

3.2.2 PaaS

Sovellusalusta palveluna, PaaS, on pilvipalvelumalli, jossa asiakas saa käyttöönsä kokonaisen sovellusalustan, jonka päälle hän pystyy rakentamaan omia sovelluksiaan alustan tukemilla teknologioilla (ks. kuvio 1). Taustalla toimivat palvelimet, käyttöjärjestelmät ja ohjelmistojen ajoympäristöt on kokonaan piilotettu asiakkaalta. Sovellusalusta voi kuitenkin tarjota asetuksia sovellusalustan konfigurointiin korkealla tasolla. Konfigurointi ja sovellusalustan käyttöönotto tapahtuu yleensä web-portaalin kautta pienellä työllä ja nopeasti. PaaS-malliset pilvipalvelualustat on suunnattu erityisesti sovelluskehittäjille.

Etuna palvelimien ja taustaohjelmistojen piilotuksessa on, että asiakkaan ei tarvitse huolehtia näiden ylläpidosta. Tämä vähentää kuormaa verrattuna kokonaan itse ylläpidettäviin laitteistoihin tai IaaS-mallisiin pilvipalveluihin. Taustajärjestelmien piilotus sekä nopea konfigurointi ja käyttöönotto mahdollistavat myös palvelun paremman skaalautuvuuden. Tätä voidaan perustella sillä, että kaikkia varsinaiseen sovellukseen suoraan kuulumattomien teknologiapinon taustajärjestelmien resursseja voidaan lisätä (tai vähentää) hyvin pienellä viiveellä ja ilman ylimääräistä ylläpitotyötä. Etuna PaaS-mallissa on myös, että asiakas maksaa vain oikeasti käyttämästään suoritusajasta.

PaaS-mallisessa pilvipalvelussa sovellusalustan tarkat rajoitukset esimerkiksi käytettäville teknologioille voivat kuitenkin muodostua haasteeksi. Lisäksi sovellusalustan päälle rakennetut sovellukset ovat usein riippuvaisia alustastaan, joten sovellusta voi olla myöhemmin haastavaa siirtää toiselle (kilpailevalle) sovellusalustalle. Näistä syistä myös esimerkiksi IaaS-mallisen pilvipalvelun päälle rakennettua sovellusta voi olla vaikeaa siirtää suoraan PaaS-malliselle sovellusalustalle.

Esimerkkejä PaaS-mallisista pilvipalveluista ovat “Amazon Web Services (AWS)” (2021), “Microsoft Azure” (2021) ja “Google Cloud Platform (GCP)” (2021). Tässä tutkimuksessa keskitytään erityisesti tätä palvelumallia tukeviin pilvipalveluihin, ja näistä käytetään termiä ”pilvialusta”.

3.2.3 SaaS

Ohjelmisto palveluna, SaaS, on pilvipalvelumalli, jossa asiakkaalle tarjotaan kokonainen sovellus tai ohjelmisto, jota voi käyttää suoraan esimerkiksi web-selaimesta (ks. kuvio 1). Etuna tässä on, että palvelu on aina käytettävissä lähes miltä laitteelta tahansa. Lisäksi ohjelmiston konfiguraatio, asetukset ja data eivät ole riippuvaisia käyttäjän päätelaitteesta, vaan ne on tallennettu toisaalla olevalle palvelimelle. Tämä mahdollistaa sen, että käyttäjä voi vaihtaa päätelaitettaan toiseen ilman, että sovellusta tarvitsisi asentaa tai dataa ja asetuksia siirtää uuteen laitteeseen.

Etuna päätelaiteriippumattomuuden lisäksi SaaS-mallisessa ohjelmistossa on, että taustalla olevat laitteistoresurssit ja taustaohjelmistot on piilotettu asiakkaalta. Tämän ansiosta niiden ylläpitotyö jää pois, jolloin asiakkaan tarvitsee huolehtia ainoastaan suoraan ohjelmistoon liittyvistä asioista sekä esimerkiksi päätelaitteiden selaimista, joilla ohjelmistoa käytetään. Toisaalta palveluntarjoajalle tämä mahdollistaa jatkuvan tulovirran kuukausilaskutuksen muodossa.

SaaS-mallisessa pilvipalvelussa on myös haasteita. Tällainen ohjelmisto vaatii verkkoyhteyden toimiakseen ja ohjelmiston toiminta riippuu täysin palveluntarjoajasta. Jos esimerkiksi palveluntarjoaja lopettaisi ohjelmiston tarjoamisen, asiakas ei välttämättä pystyisi sitä enää käyttämään. Myös esimerkiksi tietoturva- ja yksityisyyskysymykset voivat muodostua haasteeksi, jos esimerkiksi palvelimien sijaintia ei tiedetä tai ne ovat väärässä paikassa.

Esimerkkejä SaaS-mallisista pilvipalveluista ovat “Microsoft 365” (2021) -toimistosovellusperhe, “Google Maps” (2021) -karttapalvelu ja “MindMeister” (2021) -miellekarttatyökalu. Verrattuna PaaS- ja IaaS-mallisiin pilvipalveluihin SaaS-malliset palvelut on suunnattu loppukäyttäjille, kun taas PaaS ja IaaS ovat kehittäjien ja muiden asiantuntijoiden työkaluja.

3.3 IoT-pilvikommunikaatio

IoT:n esineiden ja pilvipalveluiden välistä kommunikaatiota varten tarvitaan protokollia, jotta viestejä ja dataa pystytään välittämään niiden välillä. Viestit välitetään internetin kautta, joka pohjautuu TCP/IP-pinoon. Kuviossa 2 pino on esitetty alalle tyypillisellä, mutta yksinkertaistetulla tavalla (Peter Loshin 2003, luku 5.3). Todellisuudessa TCP/IP-pinon kerrosrajat eivät ole tiukkoja, eivätkä kaikki protokollat noudata niitä. Tässä tutkimuksessa ko. TCP/IP-pinon kuvaus on kuitenkin riittävä ja kiinnostavimpia siinä ovat erityisesti sovelluskerroksen protokollat.

Kuvio 2. TCP/IP-pino perustuen Peter Loshinin (2003, kuvio 5-1) kirjaan ja muutama protokolla eri tasoilta. Kirjassa on mainittu, että tämä on tyypillinen, mutta yksinkertaistettu tapa esittää TCP/IP-pino. Todellisuudessa tässä mallissa protokollat voivat rikkoa kerrosten rajat.

Seuraavissa aliluvuissa käydään tarkemmin läpi MQTT-, AMQP-, CoAP- ja HTTP-protokollia, minkä lisäksi XMPP- ja DDS-protokollia sivutaan hieman. Nämä valikoituivat tutkimukseen, sillä ne ovat laajasti hyväksyttyjä protokollia, jotka soveltuvat IoT:n vaihteleviin laitteelta laitteelle (M2M, engl. *Machine to Machine*) keskittyviin viestinvälitystarpei-

siin (Bandyopadhyay ja Bhattacharyya 2013; Mishra ja Kertesz 2020).

3.3.1 MQTT

MQTT (engl. *Message Queue Telemetry Transport*) on standardoitu julkaise/tilaa-tyypinen protokolla viestinvälitykseen palvelimeen yhteydessä olevien asiakkaiden välillä (OASIS Standard 2014, 2019). Julkaise/tilaa-tyyppisellä protokollalla tarkoitetaan protokollaa, jonka avulla varsinainen data välitetään ”viesteinä” (engl. *message*) laitteiden (tai ohjelmien) välillä. Jokainen viesti kuuluu johonkin ”aiheeseen” (engl. *topic*), jotka esittävät viestivirtoja. Laitteet voivat tilata jonkin aiheen. Tällaisia laitteita kutsutaan ”tilaajiksi” tai ”kuluttajiksi” (engl. *subscriber*, *consumer*). Laitteet voivat myös julkaista johonkin aiheeseen liittyviä viestejä. Tällaisia laitteita kutsutaan ”julkaisijoiksi” tai ”tuottajiksi” (engl. *publisher*, *producer*). MQTT:ssä julkaisijoiden ja tilaajien välillä toimii palvelin, jota kutsutaan ”viestinvälittäjäksi” (engl. *message broker*). Viestinvälityksen aikana julkaisijoiden ja tilaajien tulee olla jatkuvassa yhteydessä viestinvälittäjään.

MQTT tukee kolmea viestiliikenteen palvelutasoa (QoS, engl. *Quality of Service*) (OASIS Standard 2019). Korkeammalla palvelutasolla viestin saapuminen perille kerran on varmempaa, mutta viestin välitykseen liittyy enemmän ylimääräistä rasitetta esimerkiksi lähetettävien kuittausten muodossa. ”Korkeintaan kerran” -tasolla (taso 0) viestin välityksen onnistuminen riippuu taustalla olevan verkon tilasta. Toisin sanoen viestin saaja ei lähetä kuittausta lähettäjälle, eikä uudelleenlähetystä yritetä, minkä takia viesti saapuu kerran tai ei ollenkaan. ”Vähintään kerran” -tasolla (taso 1) viestin saaja lähettää kuittauksen lähettäjälle viestin saatuaan, mutta viesti saattaa saapua useamman kerran, jos lähettäjä ei saa kuittausta ennen uudelleenlähetystä. ”Tasan kerran” -tasolla (taso 2) viesti toimitetaan perille tasan kerran. Nelitiekättelystä johtuen tähän tasoon liittyy kuitenkin selvästi enemmän rasitetta sekä viestin lähettäjälle että vastaanottajalle.

MQTT-protokolla on laajimmalle levinnyt M2M/IoT-protokolla ja sitä on myös tutkittu paljon (Mishra ja Kertesz 2020). MQTT-protokolla julkaistiin ensimmäisen kerran vuonna 1999, minkä jälkeen siitä on julkaistu useita versioita. Näistä version 3.1.1 on standardoinut OASIS (2014) ja ISO (2016). Uusin standardi on versionumeroltaan 5 ja sen on standardoinut

OASIS (2019). Koska MQTT vaatii toimiakseen alleen protokollan, joka tarjoaa järjestetyn ja häviöttömän kaksisuuntaisen yhteyden (esim. TCP/IP-pinon TCP-protokolla), se ei suoraan sovellu langattomiin sensoriverkkoihin (esim. ZigBee-pohjaiset verkot). Tätä tarkoitusta varten MQTT-protokollasta on kehitetty versio nimeltään MQTT-SN (engl. *MQTT for Sensor Networks*) (Stanford-Clark ja Truong 2013). Tämä protokolla mahdollistaa myös toimintavarmuudeltaan epäluotettavissa verkoissa toimimisen. Esimerkiksi häviöllisen UDP-protokollan käyttäminen kuljetuskerroksen protokollana TCP/IP-verkoissa on mahdollista MQTT-SN-protokollan kanssa. Tätä protokollaversiota ei käsitellä tässä syvällisemmin, koska se ei ole olennaista tämän tutkimuksen kannalta.

MQTT-protokollan kanssa voidaan käyttää tietoturvaominaisuuksia melko vapaasti, mutta tarjolla olevat ominaisuudet riippuvat protokollan implementaatiosta (OASIS Standard 2019). MQTT-liikenteen salauksessa voidaan käyttää esimerkiksi TLS-protokollaa (Rescorla ja Dierks 2008). Autentikaatiota varten MQTT määrittelee käyttäjätunnus- ja salasanakentät protokollan *CONNECT*-pakettiin (OASIS Standard 2019). Salasanakentässä voi kuljettaa myös tunnisteita (engl. *token*), jolloin perusautentikaation (käyttäjätunnus ja salasana) sijaan voidaan käyttää esimerkiksi OAuth-valtuutusta (engl. *authorization*) (Hardt 2012), joka sallii tarkemmat käyttöoikeusmääritykset.

Haasteena MQTT-protokollassa on, että standardi määrittelee tuen vain yhdelle viestinvälittäjälle. Tämän takia protokolla ei standardinmukaisena skaalaudu erityisen suuriin rinnakkaisiin järjestelmiin, eikä se voi kaikilta osin hyödyntää esimerkiksi reunalaskentaa (Mishra ja Kertesz 2020). Toinen asia, joka voi muodostua haasteeksi on aiemmin mainittu MQTT:n vaatimus yhteydellisestä alusprotokollasta. Käytännössä tämä tarkoittaa TCP/IP-verkon tapauksessa riippuvuutta TCP-protokollasta. Tämä voi hidastaa viestinvaihtoa verrattuna UDP-protokollaan johtuen TCP-protokollan yhteysominaisuuksista, kuten yhteyden muodostuksesta ja ruuhkanhallinnasta (Bandyopadhyay ja Bhattacharyya 2013).

Mainituista haasteista huolimatta luvussa 4 käytetään MQTT-protokollaa IoT-prototyypin ja pilvipalveluiden välisessä kommunikaatiossa. Tähän päädyttiin erityisesti MQTT:n yksinkertaisuuden vuoksi sekä siksi, että kaikki testattavat pilvipalvelut tukevat protokollaa suoraan (Bastos 2019).

3.3.2 CoAP

CoAP (engl. *Constrained Application Protocol*) on monikäyttöinen protokolla, joka on suunniteltu erityisesti pienitehoisille laitteille M2M-tyyppiseen kommunikaatioon (Shelby, Hartke ja Bormann 2014). Tämä protokolla on suunniteltu toimimaan WWW:ssä (engl. *World Wide Web*) käytetyn HTTP-protokollan kanssa suoraviivaisen siltauksen kautta. Tämän mahdollistaa CoAP-protokollan tuki WWW:ssä käytetyille konsepteille, kuten yksittäiset resurssit tunnistaville URI:lle (engl. *Uniform Resource Identifier*), REST-tyyppiselle (engl. *Representational State Transfer*) (Fielding 2000) arkkitehtuurille ja viestien sisältötyyppien (engl. *Content-type*) määrittelylle. CoAP-protokolla soveltuu käytettäväksi varsinkin kohteissa, joissa helppo integroituvuus HTTP-pohjaisen verkon kanssa on tärkeää. Hyviä sovelluskohteita ovat esimerkiksi älykäs energia sekä rakennusautomaatio (Shelby, Hartke ja Bormann 2014).

CoAP-protokollassa toimintamallina käytetään pyyntö/vastaus-mallia (engl. *request/response*), joka on myös HTTP-protokollan käyttämä malli (Shelby, Hartke ja Bormann 2014). Tässä mallissa asiakas lähettää pyynnön palvelimelle ja palvelin vastaa pyyntöön parhaan kykynsä mukaan. Tämän lisäksi CoAP tukee resurssi/tarkastelija-mallia (engl. *resource/observer*), joka on muunnelma luvussa 3.3.1 esitellystä julkaise/tilaa-mallista. Erona näillä on, että resurssi/tarkastelija-mallissa käytetään URI:a resurssien tunnisteena aiheen sijaan.

Etuna CoAP-protokollassa verrattuna esimerkiksi MQTT- tai HTTP-protokollaan on, että se tukee asynkronista viestinvaihtoa (Shelby, Hartke ja Bormann 2014). Lisäksi se on suunniteltu toimimaan yhteydettömän UDP-protokollan päällä, mutta sitä on mahdollista käyttää myös TCP:n päällä. Tämä tekee CoAP-protokollasta MQTT-protokollaa kevyemmän ja tehokkaamman, sillä CoAP ei ole riippuvainen TCP:n raskaista luotettavuutta parantavista varmistusmekanismeista, kuten yhteydenmuodostuksesta (Bandyopadhyay ja Bhattacharyya 2013). CoAP sisältää kuitenkin omia tarvittaessa käytettäviä varmistusmekanismeja, kuten viestien kuittauksen, mutta on myös näiden osalta kevyempi kuin TCP. CoAP-protokollan tehokkuutta lisää myös se, että pakettien tunnistetiedot (engl. *header*) on yksinkertaisempi ja nopeampi parsia kuin HTTP:ssä (Bandyopadhyay ja Bhattacharyya 2013).

CoAP-protokolla tukee viestiliikenteessään erityyppisiä viestejä (Shelby, Hartke ja Bormann

2014), jotka vastaavat osittain MQTT:n QoS-tasoja. *Confirmable*-tyyppiset viestit kuitataan toimituksen onnistumisen varmistamiseksi. Tämä vastaa MQTT:n QoS-tasoa 1. Vastaavasti *Non-confirmable*-tyyppisiä viestejä ei kuitata. Tämä vastaa MQTT:n QoS-tasoa 0. Tietoturvan osalta CoAP-protokolla ei itse tarjoa salaus- tai autentikaatio-ominaisuuksia, mutta näitä varten protokollan kanssa voidaan käyttää esimerkiksi IPsec- tai DTLS-protokollaa (Shelby, Hartke ja Bormann 2014).

Kaikki luvussa 4 testattavat pilvipalvelut eivät suoraan tue CoAP-protokollaa (ks. esim. "Azure IoT Hub Communication Protocols and Ports" 2021). Tuki olisi kuitenkin mahdollista lisätä esimerkiksi lisäosilla (ks. esim. "CoAP Receiver" 2021), mutta tässä tutkimuksessa pitäydytään pilvipalveluiden suoraan tukemissa protokollissa.

3.3.3 AMQP

AMQP (engl. *Advanced Message Queuing Protocol*) on erityisesti finanssialaa varten suunniteltu protokolla (OASIS Standard 2012). Protokolla on monimutkaisempi, mutta samalla monipuolisempi kuin MQTT (Uy ja Nam 2019). Suunnittelussa on keskitytty protokollan nopeuteen, luotettavuuteen ja tietoturvaominaisuuksiin (Foster 2015). Protokolla tukee esimerkiksi viestien luotettavaa jonotusta, niiden joustavaa reititystä sekä transaktioita. Toimintamallinaan AMQP tukee sekä pyyntö/vastaus-mallia että julkaise/tilaa-mallia (Naik 2017).

AMQP vaatii toimiakseen yhteydellisen alusprotokollan (Vinoski 2006), kuten MQTT:kin. Yleensä alusprotokollana käytetään TCP-protokollaa. AMQP tukee kolmea QoS-tasoa, jotka vastaavat MQTT:n tasoja (ks. luku 3.3.1) (Foster 2015). Protokolla käyttää viestiliikenteen salaukseen TLS-protokollaa ja autentikaatioon SASL-protokollaa (Foster 2015; OASIS Standard 2012).

Verrattuna luvuissa 3.3.1 ja 3.3.2 käsiteltyihin MQTT- ja CoAP-protokolliin, AMQP on pääasiassa näitä raskaampi esimerkiksi viestiensä koolta, latenssiltaan ja virrankulutukseltaan (Naik 2017). Lisäksi, kuten CoAP-protokollankin kohdalla, kaikki luvussa 4 testattavat pilvipalvelut eivät tue suoraan AMQP-protokollaa (ks. esim. "Protocols | Cloud IoT Core Documentation" 2021). Sen sijaan esimerkiksi tietoturvaominaisuuksiltaan AMQP on muita mainittuja protokollia monipuolisempi (Naik 2017).

3.3.4 HTTP

HTTP (engl. *Hypertext Transfer Protocol*) on erityisesti hajautettua hypermedian välitystä varten suunniteltu protokolla (Mogul ym. 1997). Protokollasta standardoitiin versio 1.1 vuonna 1997. Versio 2 standardoitiin vuonna 2015 (Belshe, Thomson ja Peon 2015), mutta Varvellon ym. (2016) rakentaman mittaussivuston mukaan versiota HTTP/1.1 käytetään vielä laajasti. HTTP on paljon käytetty protokolla erityisesti WWW-sivustojen yhteydessä, mutta se on suosittu myös IoT-laitteiden välisessä kommunikaatiossa.

HTTP on toimintamalliltaan pyyntö/vastaus-tyyppinen. Asiakkaat voivat tehdä resursseihin liittyen pyyntöjä palvelimelle URI:en perusteella. Pyynnöt voivat olla REST-tyyppisen arkkitehtuurin mukaisesti esimerkiksi jonkin resurssin lisäyksiä, hakuja, päivityksiä tai poistoja. Alusprotokollana HTTP käyttää MQTT:n ja AMQP:n tavoin yleensä TCP:tä (Mogul ym. 1997; Belshe, Thomson ja Peon 2015). HTTP ei määrittele QoS-tasoja ollenkaan, vaan luottaa TCP:n yhteydenhallintaan. Yhteyden salaukseen protokollan kanssa voidaan käyttää TLS:ää. Autentikaatiossa ja valtuutuksessa tarvittavia tietoja varten HTTP määrittelee tunnistetietokentän. Toisin sanoen tässä kentässä voidaan kuljettaa esimerkiksi perusautentikaatioon tarvittavia tietoja tai vaihtoehtoisesti esimerkiksi OAuth-valtuutuksen vaatimaa tunnistetta.

HTTP:n suosiosta huolimatta erityisesti versiossa 1.1 on useita heikkouksia (pienitehoisten) IoT-laitteiden näkökulmasta (Bziuk ym. 2018). Koska HTTP/1.1 käyttää tarpeettoman suuria tekstimuotoisia tunnistetietoja (engl. *header*), on niiden jäsentäminen ja siirtäminen verkon yli raskasta. Tämä lisää — mahdollisesti akkukäyttöisten — IoT-laitteiden energiankulutusta. Lisäksi HTTP/1.1-protokollalla tapahtuvassa kommunikaatiossa on pitkä viive. Tämä johtuu osittain julkaise/tilaa-tyyppisen toimintamallin puutteesta, sillä synkronisen kommunikaation takia pyynnön lähettänyt IoT-laite joutuu odottamaan palvelimen vastausta. Viive johtuu myös osittain alusprotokollana käytettävän TCP:n ominaisuuksista, kuten ruuhkanhallinnasta ja yhteydenmuodostuksesta. Näiden lisäksi haasteeksi voi muodostua *Head-of-line blocking* (HOL) -ongelma. HTTP/1.1 mahdollistaa saman TCP-yhteyden käyttämisen useampaa pyyntöä varten, ettei TCP-yhteydenmuodostusta tarvitsisi tehdä useaan kertaan. Pyyntöihin täytyy kuitenkin vastata järjestyksessä, joten alussa epäonnistunut pyyntö voi viivästyttää kaikkia sen jälkeen tehtävien pyyntöjen vastauksia. Tämä puolestaan lisää vii-

vettä entisestään. HTTP/2-protokolla parantaa tilannetta näiden ongelmien osalta (Belshe, Thomson ja Peon 2015).

HTTP on tuettu protokolla kaikissa luvussa 4 testattavissa pilvipalveluissa. Tästä huolimatta tämän tutkimuksen puitteissa keskitytään MQTT-protokollan käyttämiseen erityisesti edellä mainittujen HTTP:n heikkouksien takia.

3.3.5 Muita protokollia

Edellisissä luvuissa läpikäytyjen protokollien lisäksi muitakin IoT/M2M-käyttöön soveltuvia protokollia on kehitetty. Tässä käydään lyhyesti läpi kaksi tällaista protokollaa — XMPP ja DDS. XMPP (engl. *Extensible Messaging and Presence Protocol*) on XML-tietovirtojen jakamiseen kehitetty protokolla (Saint-Andre 2011). Protokolla mahdollistaa hajautetun asiakas-palvelinverkon toteuttamisen. Tämä tarkoittaa, että samassa verkossa on tietovirtoja tarjoavia ja kuluttavia asiakkaita sekä tarvittaessa useita palvelimia, jotka välittävät näitä virtoja. Hajautuksen ansiosta verkko ei kaadu, vaikka jokin palvelimista hajoaisi. Protokolla mahdollistaa tietovirtojen avulla reaaliaikaisen kommunikaation, eikä siksi vaadi asiakkaita kyselemään palvelimilta toistuvasti uutta dataa, kuten HTTP vaatisi. XMPP-protokollaa ei ole alun perin suunniteltu pienitehoisille laitteille, mutta siitä on kehitetty kevyempiä malleja, jotka soveltuvat paremmin IoT-järjestelmiin (Wang ym. 2017). Hajautetun arkkitehtuurinsa ansiosta XMPP voisi soveltua suuriin rinnakkaisiin IoT-järjestelmiin.

DDS (engl. *Data Distribution Service*) on Object Management Groupin (OMG) standardoima väliohjelmisto, joka mahdollistaa datakeskeisen reaaliaikaisen kommunikaation asiakkaiden välillä (Object Management Group 2015). DDS on julkaise/tilaa-tyyppinen järjestelmä. Chen ja Kunz (2016) vertailivat DDS:ää esimerkiksi MQTT- ja CoAP-protokolliin ja huomasivat, että luotettavuutensa sekä pienen viiveensä ansiosta DDS voisi soveltua näitä paremmin esimerkiksi langattomiin lääketieteellisiin sovelluksiin. DDS käytti kuitenkin verkkokaistaa selvästi verrokkejaan enemmän, joten muut protokollat voivat olla monessa tapauksessa parempi valinta.

4 IoT-datan keruu pilvialustalle

Tästä luvusta alkaa tutkimuksen empiirinen osuus. Luvussa vastataan ensimmäiseen tutkimuskysymykseen: Kuinka keskeisimpiä julkisia pilvialustoja voidaan hyödyntää IoT-datan keruussa? Tarkemmin sanottuna tässä luvussa rakennetaan Raspberry Pi 2 -tietokoneeseen pohjautuva prototyyppi, jolla kerätään ilmankosteus-, ilmanpaine- ja lämpötiladataa ympäristöstä sensorin avulla. Kerätty data lähetetään Raspberry Pi:lla kolmelle eri pilvialustalle käsittelemättömänä ja tallennetaan alustojen tarjoamiin dokumenttipohjaisiin NoSQL-tietokantoihin. Käyttöönotettuja pilvialustoja vertaillaan luvussa 5 useammasta näkökulmasta prototyypistä saatujen kokemusten pohjalta.

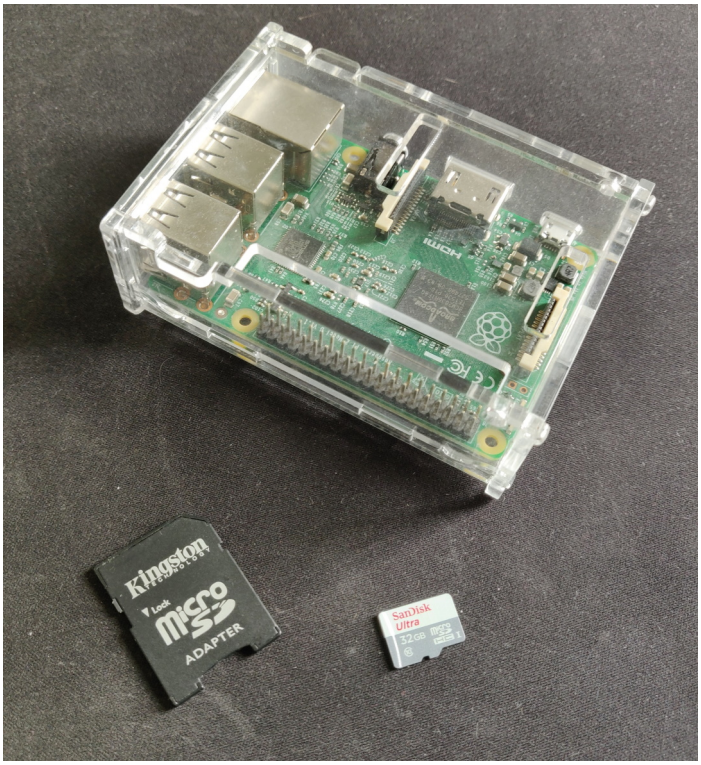
Ensimmäiseksi luvussa 4.1 tarkastellaan fyysisellä tasolla, minkälainen sensori prototyyppiin kytketään ja miten tämä kytkeminen käytännössä tehdään. Seuraavaksi luvussa 4.2 tutustutaan Raspberry Pi:lle toteutettuun ohjelmaan, jolla data kerätään sensoreilta ja lähetetään pilvialustoille. Viimeiseksi luvussa 4.3 katsotaan tarkemmin tutkittavien pilvialustojen käyttöönottoprosesseja alusta kerrallaan.

4.1 IoT-laitteiston toteutus

Seuraavaksi käydään läpi tutkimuksen prototyypissä käytetty fyysinen IoT-laitteisto. Fyysisen laitteiston sijaan prototyypissä olisi mahdollista käyttää myös PC:n avulla simuloitua laitteistoa tai ohjelmaa, joka generoi ”mittausdataa” pyydettäessä ja lähettää sen pilvialustoille. Tutkimusta varten on kuitenkin käytettävissä myös fyysistä laitteistoa, joten sitä hyödynnetään tutkimuksen autenttisuuden vuoksi.

4.1.1 Käytetty IoT-laitteisto

Tutkimuksen prototyypin fyysisenä verkkoon kytkettynä IoT-laitteena käytetään *Raspberry Pi 2 Model B V1.1* -tietokonetta, joka on esitetty kuviossa 3. Kuviossa näkyy myös tietokoneen massamuistina käytettävä 32:n gigatavun MicroSD-muistikortti sekä SD-MicroSD-muistikorttiadapteri, jota hyödynnetään käyttöjärjestelmän kirjoituksessa muistikortille.

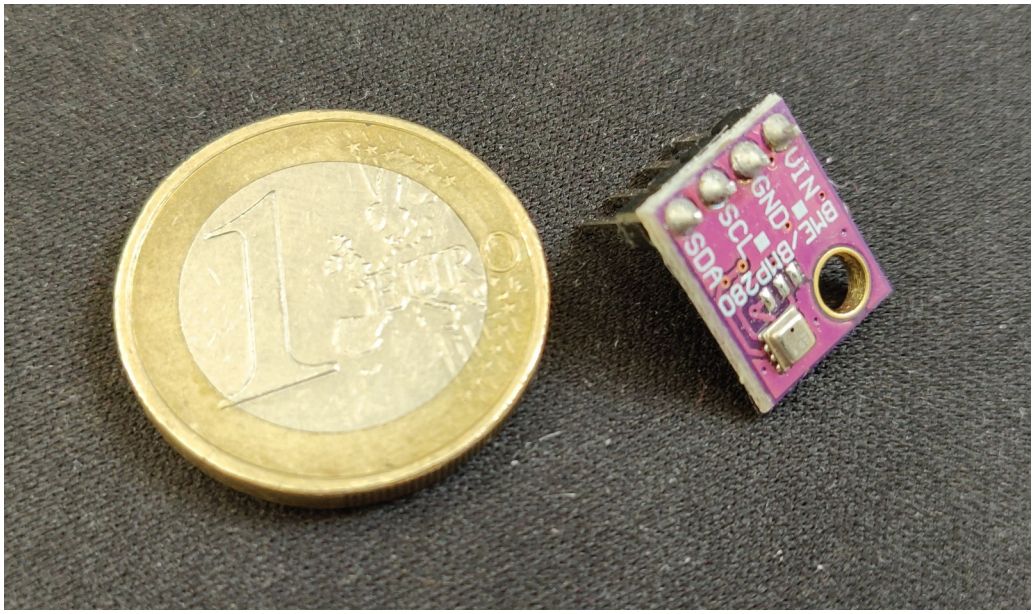


Kuvio 3. *Raspberry Pi 2 Model B V1.1* -tietokone, tietokoneen massamuistina käytettävä 32:n gigatavun MicroSD-muistikortti sekä SD-MicroSD-muistikorttiadapteri käyttöjärjestelmän kirjoitusta varten.

Dataa keräävänä sensorina tutkimuksessa käytetään *BME280*-lämpötila-, kosteus- ja ilmanpainesensoria, joka on esitetty kuviossa 4. Sensori on juotettu pienelle piirilevylle, joka tuo I2C-väylässä (ks. luku 3.1.1) käytettävät johtimet saataville. Sensori tukee myös SPI-väylää (*BME280 Data Sheet* 2020), mutta piirilevyllä ei kaikkia SPI-väylän johtimia ole saatavilla.

4.1.2 Kytkennät

Koska prototyypissä sensorin ja Raspberry Pi -tietokoneen välillä käytetään I2C-väylää, ovat kytkennät suoraviivaisia. Sensorin ja tietokoneen väliset kytkennät näytetään kuviossa 5. Näissä on käytetty apuna Raspberry Pi:n GPIO-dokumentaatiota (2021), jossa kerrotaan kunkin tapin käyttötarkoitus sekä esimerkiksi I2C- ja SPI-väylissä käytettävät GPIO-tapit. Sensorin jännitejohdin (VIN) on kytketty punaisella johtimella Raspberry Pi:n 3,3 voltin jännitetappiin, joka on fyysiseltä tappinumeroltaan 1. Sensorin maajohdin (GND) on kytket-



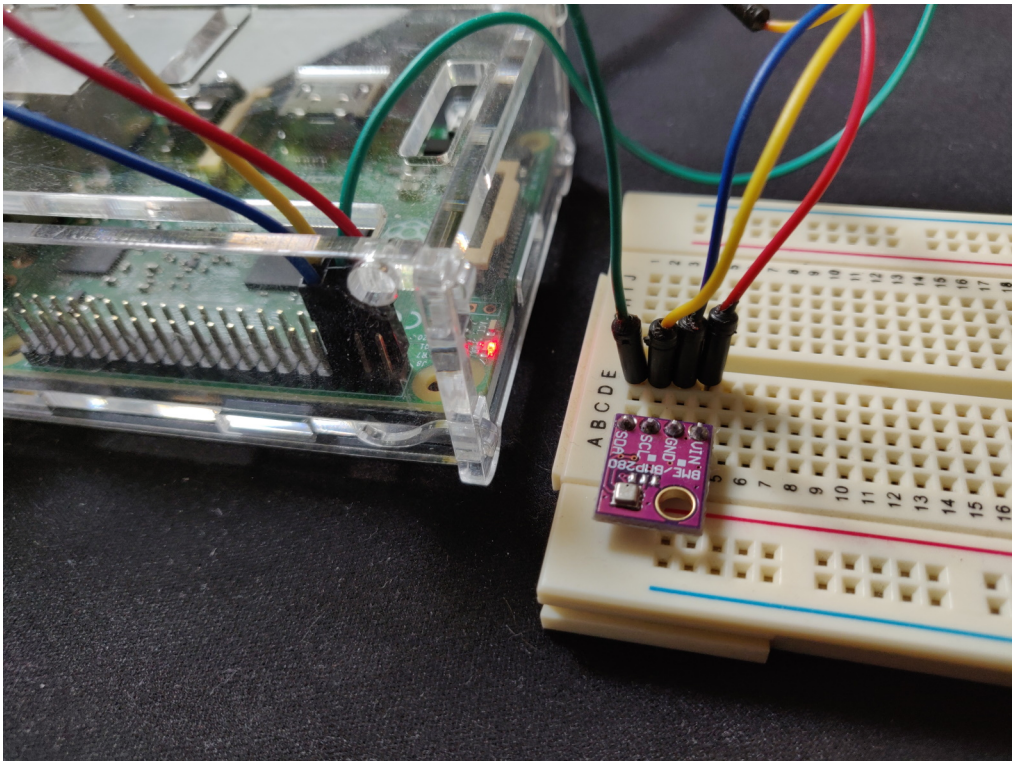
Kuvio 4. *BME280*-lämpötila-, kosteus- ja ilmanpainesensori, jota käytetään tutkimuksen IoT-prototyypissä datan keräämiseen. Euron kolikko on kuviossa mittakaavan hahmotusta varten.

ty sinisellä johtimella yhteen Raspberry Pi:n maatapeista. Tässä käytetään tappia, joka on fyysiseltä numeroltaan 6. I2C-väylän sarjakellojohdin (SCL) on yhdistetty keltaisella johtimella Raspberry Pi:n GPIO 3 -tappiin, joka on fyysiseltä tappinumeroltaan 5. Viimeiseksi I2C-väylän sarjadatajohdin (SDA) on yhdistetty vihreällä johtimella Raspberry Pi:n GPIO 2 -tappiin, joka on fyysiseltä tappinumeroltaan 3.

Sensorikytkentöjen lisäksi Raspberry Pi on kytketty verkkoon Ethernet-kaapelilla etäohjausta ja pilviyhteyttä varten. MicroSD-muistikortti on liitetty käyttöjärjestelmän asennuksen jälkeen tietokoneen pohjassa olevaan muistikorttipaikkaan. Käyttöjärjestelmän asennusta muistikortille käsitellään tarkemmin luvussa 4.2. Viimeisenä Raspberry Pi -tietokoneeseen kytketään virrat liittämällä viiden voltin virtalähteeseen kytketty Micro-USB-kaapeli tietokoneen virtaliittimenä toimivaan Micro-USB-liittimeen.

4.2 IoT-laitteen ohjelmisto

Tässä luvussa tutustutaan prototyyppin rakennusprosessiin Raspberry Pi:n ohjelmiston osalta sekä tutkimusta varten ohjelmoituun sensoridatan keräysohjelmaan. Käydään ensin tii-



Kuvio 5. I2C-väylän mukaiset kytkennät BME280-sensorin ja Raspberry Pi -tietokoneen välillä.

vistetysti rakennusprosessia läpi. Prosessissa ensimmäisenä Raspberry Pi:n massamuistina käytettävälle muistikortille asennetaan Raspberry Pi OS Lite toisella tietokoneella. Asennetusta käyttöjärjestelmästä aktivoidaan samalla SSH-palvelin (engl. *Secure Shell*), joka mahdollistaa tietoturvallisen etäkäytön tekstipohjaisen komentorivikäyttöliittymän kautta. Näin Raspberry Pi:hin ei tarvitse kytkeä näyttöä ja näppäimistöä missään vaiheessa. Tämän jälkeen muistikortti liitetään Raspberry Pi:hin ja se käynnistetään. Raspberry Pi:hin otetaan SSH-yhteys toiselta samassa verkossa olevalta tietokoneelta. Lopuksi asennetaan tarvittavia ohjelmia sekä tehdään tietoturvaa lisääviä konfiguraatioita. Näitä ovat esimerkiksi oletussalasanan ja -käyttäjätunnuksen vaihto sekä palomuurin asennus. Raspberry Pi:n asennuksen ja valmistelun yksityiskohtaiset vaiheet on listattu liitteessä A.1.

4.2.1 Datan keruu sensorilta

Sensoridatan keräämistä ja pilveen lähetystä varten kirjoitetaan ohjelma. Ohjelmointiprosessin aikana kirjataan havaintoja ja kokemuksia ylös erityisesti datan pilvilähetystä koskevista vaiheista. Ohjelmointikieleksi datankeruuohjelmaa varten valitaan Python 3, koska kaikki testattavat pilvialustat tukevat sitä IoT-laitteiden kielenä. AWS ja Azure tarjoavat omat Python-SDK-paketit IoT-laitteille, ja Googlella on esimerkkejä avoimen lähdekoodin "Eclipse Paho" (2021) MQTT-kirjaston käyttämisestä tällä kielellä IoT-laitteilla. Lisäksi Raspberry Pi:lle on kehitetty avoimen lähdekoodin Python-kirjasto BME280-sensorin lukemista varten ("Rm-Hull/Bme280" 2021).

Ennen ohjelmointia varmistetaan, että Raspberry Pi:lle on asennettu Python 3. Varmistuksen jälkeen seurataan BME280-sensorikirjaston asennusohjeita ennakkovaatimuksista alkaen. Näihin sisältyvät esimerkiksi I2C-kerneliajurin kytkeminen päälle tarvittaessa, I2C-väylän siirtonopeuden nosto nopeaan tilaan (400 kbit/s) sekä sensorin I2C-väyläosoitteen tarkistus. Yksityiskohtaiset vaiheet löytyvät liitteestä A.2. Sensorikirjaston asennuksen jälkeen tätä voidaan testata Python-ohjelmassa esimerkiksi näin:

```
import smbus2
import bme280

# Esitiedot
i2c_port = 1
bme280_address = 0x76

# Sensoriyhteyden alustus
bus = smbus2.SMBus(i2c_port)
calibration_params = bme280.load_calibration_params(
    bus, bme280_address)

# Sensoriarvojen lukeminen tapahtuu sample-metodilla.
data = bme280.sample(bus, bme280_address, calibration_params)
print(data)
```

Edellinen esimerkkikoodi on kirjoitettu mukaillen BME280-sensorikirjaston käyttöesimerkkiä ("Rm-Hull/Bme280" 2021).

Kun BME280-sensorikirjasto on asennettu siirrytään ohjelmoimaan datankeruuohjelmaa. Ohjelman pohjaksi otetaan kirjaston käyttöesimerkki sen GitHub-sivulta. Esimerkin rakennetta muokataan siirtämällä sensoriyhteyden alustus metodiin `init_sensors()` ja datan tulostus metodiin `print_data(data)`. Datan luku siirretään uuteen päättymättömään `while`-silmukkaan, johon lisätään myöhemmin myös datan lähetys pilveen. Silmukkaan lisätään jokaiselle kierrokselle yhden sekunnin viive `time.sleep(1)`-kutsulla ja se siirretään uuteen `main()`-metodiin. Tähän metodiin lisätään myös `KeyboardInterrupt`-poikkeuksen kiinniotto, jotta päättymättömästä silmukasta voisi poistua *Ctrl* + *C* -näppäinkomennolla kaatamatta ohjelmaa. Näiden muutosten jälkeen ohjelman `main()`-metodi näyttää tältä:

```
def main():
    init_sensors()

    try:
        while True:
            # the sample method will take a single reading and return
            # a compensated_reading object
            data = bme280.sample(
                bus, BME280_ADDRESS, calibration_params)
            print_data(data)

            time.sleep(1)
    except KeyboardInterrupt:
        print('Stopped')
```

Tässä vaiheessa ohjelma lukee lämpötila-, ilmankosteus- ja ilmanpainearvot sensorilta ja tulostaa ne näytölle kerran sekunnissa.

4.2.2 Datan lähetys pilveen

Seuraavaksi käydään läpi yleisellä tasolla, kuinka sensoridata lähetetään pilveen. Yksityiskohtaisempi läpikäynti jokaisen pilvialustan osalta erikseen tehdään luvussa 4.3. Datankeruuohjelmaan lisätään selkeyden vuoksi jokaista pilvialustaa kohden asiakasluokka, jonka kautta pilvialustalle voidaan lähettää sensoridataa yhtenäisellä tavalla. Asiakasluokat lähet-

tävät datan käyttämällä pilvialustojen suosittelemia IoT-laitteiden SDK-paketteja tai MQTT-kirjastoja, joiden API-rajapinnat eroavat toisistaan. Asiakasluokkien olioinstansseja on tarkoitus käyttää kutsumalla `connect()`-, `disconnect()`- ja `send_message(...)`-metodeja. Nämä luovat ja katkaisevat yhteyden pilvipalveluun sekä lähettävät sinne viestin tässä järjestyksessä. Viestinlähetysmetodin parametrilistaus vaihtelee hieman pilvialustasta riippuen, sillä alustojen mahdollistamat IoT-toiminnot vaihtelevat. Tästä huolimatta jokaiselle pilvialustalle lähetetään sama sensoridata formatoituna JSON-muotoon (engl. *JavaScript Object Notation*). JSON on yleiskäyttöinen, tekstipohjainen ja ihmisluettava datansiirtoformaatti ("JSON" 2021). Tätä formaattia käytetään, koska AWS, Azure ja GCP tukevat JSON-muotoa IoT-datan siirtoformaattina ja tämä muoto helpottaa myös datan viemistä NoSQL-tietokantoihin.

Asiakasluokat lisätään datankeruuohjelmaan ja niistä luodaan olioinstanssit `main()`-metodin alussa. Pilviyhteydet muodostetaan kutsumalla jokaiselle olioinstanssille heti luonnin jälkeen `connect()`-metodia. Päättymättömään `while`-silmukkaan lisätään datan lähetys kullekin pilvialustalle `send_data_to_*(...)`-muotoisilla kutsuilla. Nämä kuorimetodit piilottavat JSON-formatoinnin ja alustakohtaiset erot datan lähetyksestä. Silmukan alapuolelle lisätään pilviyhteyksien katkaisut `disconnect()`-kutsuilla. Lisäksi datankeruuohjelmaan lisätään komentoriviargumenttien jäsentäminen. Komentoriviargumenttien avulla ohjelman käyttäjä voi syöttää pakollisia ja oletuksista poikkeavia arvoja ohjelmalle. Pakollisia arvoja ovat esimerkiksi GCP-pilvialustalla määritelty yhdistettävän IoT-laitteen tunniste sekä tiedostopolku, josta löytyy epäsymmetrisessä autentikaatiossa tarvittava yksityinen avain. Oletuksista poikkeavia arvoja voivat olla esimerkiksi viestinlähetystiheys sekunteina tai lokitustaso, jota käytetään AWS-pilvialustan kanssa kommunikoidessa. Näiden lisäysten jälkeen datankeruuohjelman `main()`-metodi näyttää tältä:

```
def main(args):
    bus, calibration_params = init_sensors(args)

    aws = AWSClient(args)
    aws.connect()
    azure = AzureClient(args)
    azure.connect()
    gcp = GCPClient(args)
```

```
    gcp.connect()

    try:
        while True:
            data = bme280.sample(
                bus, args.bme280_address, calibration_params)
            send_data_to_aws(args, aws, data)
            send_data_to_azure(args, azure, data)
            send_data_to_gcp(args, gcp, data)

            time.sleep(args.msg_freq)
    except KeyboardInterrupt:
        print("Stopping...")
    finally:
        aws.disconnect()
        azure.disconnect()
        gcp.disconnect()

    print("Stopped")
```

Ohjelman lähettämä data sisältää lämpötila-, ilmankosteus- ja ilmanpainearvojen ja näiden yksiköiden lisäksi aikaleiman arvojen mittausajanhetkeltä. Aikaleima on Python-kielen virallisen *datetime*-kirjaston tarjoama POSIX-aikaleima millisekunneissa UTC-aikavyöhykkeeltä. POSIX-aikaleima approksimoi aikayksiköiden lukumäärää ajanhetkestä 1.1.1970 kello 00.00.00 alkaen. Data sisältää lisäksi ohjelmalle syötetyn IoT-laitteen sijainnin merkkijonona sekä `id`- ja `device_id`-arvot, joita osa pilvialustoista tarvitsee. `id` on merkkijonoksi muutettu aikaleima ja `device_id` on pilvialustalla määritelty laitetunniste. JSON-formaatissa data näyttää esimerkiksi tältä:

```
{
    "id": "1627031690425",
    "device_id": "RPi2",
    "timestamp": 1627031690425,
    "location": "jyvaskyla",
    "temperature": {
        "value": 26.21,
        "unit": "\u00b0C"
```

```
    },
    "pressure": {
        "value": 999.49,
        "unit": "hPa"
    },
    "humidity": {
        "value": 30.804,
        "unit": "%rH"
    }
}
```

Tässä esitetyn ohjelman lähdekoodit ovat saatavilla GitHub-varastossa “MrCliff/Iot-Cloud-Connection-Proto” (2021). Varastossa olevaan versioon on tehty myös muutamia tässä mainitsematta jääneitä lisäyksiä, kuten käytettävän pilvialustan valinta komentoriviargumentilla.

4.3 Pilvialustojen käyttöönotto

Tähän mennessä on käyty tutkimuksen IoT-prototyyppiä läpi IoT-laitteen näkökulmasta ja viimeisimmäksi myös datan lähetystä pilvialustoille yleisellä tasolla. Seuraavaksi tarkastellaan aloittelevan pilvisovelluskehittäjän näkökulmasta pilvialustojen käyttöönottoprosesseja, kun tarkoituksena on rakentaa telemetriadataa keräävä IoT-järjestelmä. Telemetriadatalla tässä tarkoitetaan IoT-laitteiden lähettämää tapahtumadataa, joka voi olla esimerkiksi mittauksia ympäristöstä. Painotus siirtyy seuraavissa aliluvuissa IoT-laitteesta pilvialustojen päähän, mutta IoT-laitteella suoritettavaa datankeruuohjelmaa tarkastellaan myös hieman.

Pilvialustoista on saatavilla hyvin monen tyyppistä ja tasoista opiskelumateriaalia niin uusille kuin kokeneemmillekin pilvisovelluskehittäjille. Pilvialustojen käyttöä sovelluskehityksessä voi opiskella esimerkiksi itsenäisesti alustojen tarjoamien dokumentaatioiden ja tutoriaalien kautta. Kaikille tässä tutkimuksessa testatuille pilvialustoille on saatavilla myös muuta materiaalia, kuten videoluentoja, yhteisön tekemiä tutoriaaleja sekä maksullisia kouluttajan vetämiä koulutuksia (ks. esim. “Google Cloud Courses and Training” 2021; “Azure IoT Hub” 2021). Tästä materiaalin runsaudesta johtuen tutkimuksessa käytetään mahdollisimman yhtenäistä opiskelu- ja käyttöönottoprosessia jokaisen testattavan pilvialustan kohdalla. Luvussa 4.3.1 tehdään yleiskatsaus tässä tutkimuksessa käytettävään pilvialustojen

käyttöönottoprosessiin. Tämän jälkeen luvuissa 4.3.2, 4.3.4 ja 4.3.6 tutustutaan yksityiskohtaisemmin kuhunkin pilvialustaan, ja luvuissa 4.3.3, 4.3.5 ja 4.3.7 käydään läpi Raspberry Pi:n liittämistä niihin.

4.3.1 Käyttöönottoprosessi yleisesti

Dataa keräävän IoT-laitteen liittäminen pilvialustaan lähtee liikkeelle kyseisen alustan IoT-palvelun verkkosivun etsimisestä. Tähän tutkimuksessa käytetään Google-hakukonetta. Löydetyltä IoT-palvelun esittelysivulta varmistetaan, että palvelu voisi sopia haluttuun käyttötarkoitukseen. Tämä tapahtuu lukemalla mm. esimerkkejä palvelun ominaisuuksista ja asiakaskokemuksista. IoT on laaja käsite ja IoT-laitteita on hyvin monen tasoisiin käyttötarkoituksiin, minkä takia pilvialustoilla voi olla tarjolla useita erilaisia IoT-palveluja. Esimerkiksi Azure tarjoaa tässä tutkimuksessa käytetyn *Azure IoT Hub* -palvelun lisäksi *Azure IoT Edge* ja *Azure IoT Central* -palveluita. Näistä ensimmäinen on tarkoitettu reunalaskentaa sisältävien IoT-järjestelmien osaksi ja jälkimmäinen on Azure IoT Hubin päälle rakennettu IoT-sovellusalusta, joka tarjoaa valmiita malleja IoT-sovellusten ja -järjestelmien rakentamisen pohjaksi (“Azure Internet of Things (IoT) Technologies and Solutions” 2021). Tässä tutkimuksessa keskitytään ilman valmiita malleja tai sovelluspohjia rakennettuihin vapaammin mukautettaviin IoT-järjestelmiin. Tällaisten rakentamiseen kaikki tutkittavat pilvialustat tarjoavat suhteellisen yhdenmukaiset työkalut. IoT-laitteiden yhdistämis- ja hallintapalveluna Azuressa käytetään *Azure IoT Hub* -palvelua, AWS:ssä *AWS IoT Core* -palvelua ja GCP:ssä *Google Cloud IoT Core* -palvelua.

Kun ollaan varmistuttu siitä, että pilvialusta tarjoaa todennäköisesti sopivan IoT-palvelun, luodaan alustalle käyttäjätunnukset ja etsitään virallinen kehittäjädokumentaatio. Kehittäjädokumentaation avulla tutustutaan tarkemmin IoT-palvelun konsepteihin sekä sen tarjoamiin teknisiin ominaisuuksiin. Konsepteilla tässä tarkoitetaan esimerkiksi IoT-palvelun toimintaperiaatteita sekä tyypillisiä rakenteita ja arkkitehtuureja, joita tällä palvelulla rakennetuissa IoT-järjestelmissä voisi käyttää. Näihin tutustuminen auttaa hahmottamaan, mitä IoT-palvelulla on mahdollista tehdä ja miten tämä tapahtuisi. Samalla tutustutaan tarpeellisilta osin myös koko pilvialustan yleisiin konsepteihin ja esimerkiksi käsitteisiin. Eri palveluntarjoajien pilvialustat käyttävät joiltain osin samoista tai vastaavista asioista eri termejä, joten

käsitteisiin tutustuminen voi olla tarpeen, vaikka olisi aiemmin kehittänyt sovelluksia käyttäen toista vastaavaa pilvialustaa.

Pilvialustan ja IoT-palvelun konseptien lisäksi kehittäjädokumentaatioista löytyy niin pinnallisia pika-aloitustutoriaaleja (engl. *quickstart tutorial*, *getting started tutorial*) kuin syvällisempiä tiettyihin aihealueisiin liittyviä tutoriaaleja. Konseptien läpikäynnin jälkeen tai samassa yhteydessä tutustutaan pilvialustaan tekemällä ensin IoT-palvelun pika-aloitustutoriaali ja sitten jokin syvällisempi telemetrian keräämiseen liittyvä tutoriaali. Nämä opastavat käytännön esimerkkien avulla, kuinka IoT-palvelua käytetään, kuinka IoT-laite saadaan kommunikoimaan sen kanssa sekä miten IoT-palveluun siirrettyä dataa voidaan viedä eteenpäin pilvialustan muihin palveluihin, kuten tietokantaan. Tutoriaalien jälkeen tutustutaan esimerkiksi tietoliikenteen ja IoT-laitteiden tietoturvaan liittyviin dokumentaatiosivuihin sekä tarvittaessa muihin teknisiin sivuihin ymmärryksen lisäämiseksi. Tietoturva-asioista tutkimuksen kannalta kiinnostavia ovat esimerkiksi käytettävissä olevat salaus- ja tunnistautumismenetelmät.

Dokumentaatioon tutustumisen ja tutoriaalien suorituksen jälkeen siirrytään implementoimaan pilviyhteyden muodostusta IoT-prototyypin datankeruuohjelmaan. Tässä käytetään apuna erityisesti syvällisten telemetrian keräämiseen keskittyvien tutoriaalien esimerkkejä. Lisäksi pilviyhteyden muodostuksessa käytettävien ohjelmointikirjastojen API-dokumentaatioita hyödynnetään.

Kun data on saatu kulkemaan pilvialustan IoT-palveluun, siirrytään tutustumaan pilvialustan tarjoamiin dokumenttipohjaisiin NoSQL-tietokantoihin (engl. *document-oriented Not only SQL database*), joihin data on tarkoitus lopulta kuljettaa. Verrattuna perinteisiin relaatiotietokantoihin NoSQL-tietokannat ovat monessa tapauksessa esimerkiksi paremmin skaalautuvia ja toiminnaltaan nopeampia eivätkä ne tarvitse yhtä aktiivista ylläpitoa (Nayak 2013). Toisaalta NoSQL-tietokannoilla ei ole keskenään yhtenäistä kyselykieltä eivätkä esimerkiksi dokumenttipohjaiset tietokannat takaa ACID-ominaisuuksia (engl. *Atomicity, Consistency, Isolation and Durability*) datalle. Dokumenttipohjaiset tietokannat on suunniteltu tallentamaan esimerkiksi JSON-muotoisiin dokumentteihin muotoiltua dataa, joten ne soveltuvat hyvin tämän tutkimuksen IoT-prototyypin käyttötarkoitukseen. Tästä syystä jokaisella pilvialustalla tietokannaksi valitaan dokumenttipohjainen NoSQL-tietokanta. Jos tällaisia tie-

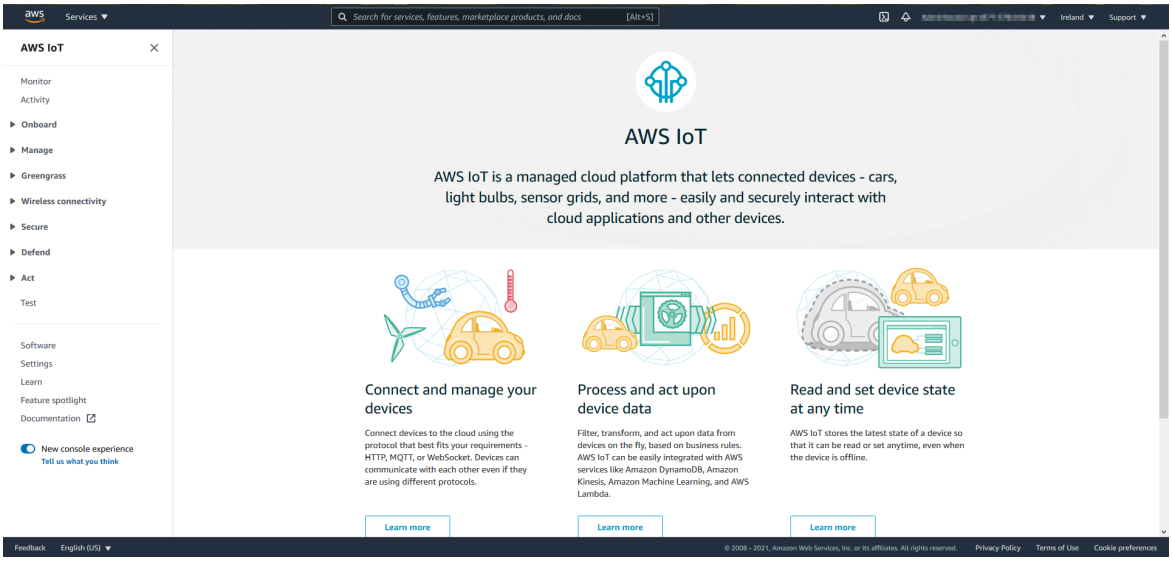
tokantoja on useita, näistä valitaan se, joka on tutkimushetkellä kyseisen alustan dokumentaation mukaan suositelluin valinta uusille projekteille. Esimerkiksi GCP:lla on mahdollista käyttää tässä tutkimuksessa valitun *Firestore*-tietokannan sijaan vanhempia *Datastore* tai *Firebase Realtime Database* -tietokantoja ("Choosing between Native Mode and Datastore Mode | Firestore" 2021). Firestore on näiden kahden tietokannan seuraaja ja siksi suositeltu uusiin projekteihin. AWS:llä tietokantana käytetään *DynamoDB*-tietokantaa ja Azurella *Cosmos DB*-tietokannan *SQL API* -versiota.

Tietokannan valinnan jälkeen jokaisella pilvialustalla tutustutaan tarpeellisilta osin tarjottuihin alustan sisäisiin dataväyläpalveluihin. Dataväyläpalveluilla tarkoitetaan tässä pilvialustojen tarjoamia palveluita, joilla tutkimuksen IoT-prototyypissä kuljetetaan dataa IoT-palveluista tietokantoihin. Tähän lukeutuu mm. GCP:n Cloud Functions -palvelu, sillä tutkimuksessa se toimii vain datan välittäjänä. Todellisuudessa sitä voi käyttää moneen muuhunkin tarkoitukseen. Tavoitteena on löytää IoT-prototyypin käyttötarkoitukseen sopiva tapa kuljettaa IoT-palveluun saapuvat dataviestit pilvialustan sisällä ja tallentaa ne valittuun NoSQL-tietokantaan. Jos järkeviä dataväyliä on useita, valitaan sellainen, joka on uuden käyttäjän näkökulmasta todennäköisesti helppokäyttöisin tai muuten käyttötarkoitukseltaan IoT-prototyyppiin sopivin. Dataväylän löydyttyä se otetaan käyttöön yhdessä tietokannan kanssa, minkä jälkeen IoT-prototyyppiä pystyy testaamaan kokonaisuudessaan. Samassa yhteydessä korjataan prototyyppiä mahdollisten datan kuljetuksessa ilmenevien virheiden osalta sekä parannellaan sitä tarvittaessa, kunnes data saadaan siirtymään tietokantaan halutussa muodossa. Tästä voisi jatkaa IoT-prototyypin parantelua ja laajentamista aina mahdollisiin mobiilisovelluksiin ja datan hyödyntämiseen asti, mutta resurssirajoitteiden vuoksi pitäydytään tässä datan keräämisessä ja tallennuksessa. Seuraavaksi tutustutaan tutkimuksessa toteutuneisiin pilvialustojen käyttöönottoprosesseihin alusta kerrallaan.

4.3.2 Amazon Web Services

AWS-pilvialustaan ja sen IoT-palveluun tutustuminen lähtee liikkeelle Google-haulla "aws iot", jolla AWS:n IoT-ominaisuuksien esittelysivu löytyy ensimmäisenä. AWS IoT -sivulta löytyy lyhyt esittely mm. AWS IoT Core -palvelusta, jonka sanotaan mahdollistavan esimerkiksi IoT-laitteiden kommunikoinnin AWS-pilvialustan palveluiden kanssa. AWS IoT Co-

re -palvelulla on myös oma esittelysivunsa, jolla siitä kerrotaan tarkemmin. Tämän sivun ylälaidassa keskeisellä paikalla on ”Get started with IoT Core for free” -linkki, joka ohjaa AWS-tunnusten luonnin kautta IoT Coren web-pohjaisen hallintakonsolin etusivulle (ks. kuvio 6). Web-konsolia käytetään tässä tutkimuksessa vaihtoehtoisen komentorivikäyttöliittymän (CLI) sijaan. Tutkimuksen aikaan AWS IoT:n web-konsolin käyttökokemusta ollaan päivittämässä. Tässä käytetään uutta web-konsolia, joka on oletuksena aktivoitu uusille käyttäjille.



Kuvio 6. AWS IoT Core -palvelun web-hallintakonsolin etusivu kuvattuna 4.5.2021.

AWS IoT Coren web-konsolin vasemmassa laidassa on linkki sen dokumentaation etusivulle, josta löytyy linkkejä useisiin aihealueisiin liittyviin materiaaleihin. Ensimmäinen linkki vie AWS IoT:n kehittäjädokumentaation aloitussivulle “What Is AWS IoT? - AWS IoT Core” (2021). Tämä sivu löytyy myös Googlen kautta aiemmin mainituilla hakusanoilla ”aws iot”. Sivulla on esitelty AWS IoT -palvelua kevyesti teknisten ominaisuuksien osalta. Näitä ovat esimerkiksi listaus tuetuista protokollista sekä työkaluista, joiden avulla palveluun voi yhdistää. Sivulla on linkki aloitusohjeistukseen (“Getting Started with AWS IoT Core - AWS IoT Core” 2021), jossa listataan mm. palvelun konsepteista kertovia sivuja sekä kerrotaan konkreettisesti AWS IoT Coren käyttöönotosta AWS-käyttäjätunnusten luonnista alkaen. Seuraavissa tekstikappaleissa käydään keskeisiä aloitustoimenpiteitä läpi kyseisen käyttöönotto-ohjeistuksen mukaisessa järjestyksessä.

Kun AWS:ään rekisteröidytään, luodaan samalla root-tunnukset, jonka käyttöoikeudet esimerkiksi uusien palveluiden käyttöönottoon ovat lähes rajoittamattomat. Root-tunnusten lisäksi AWS suositteleekin erillisten IAM-käyttäjätunnusten (engl. *Identity and Access Management*) luontia päivittäisiä toimia varten. IAM-käyttäjälle voidaan antaa laajat pääkäyttäjäoikeudet, mutta niitä voidaan tarvittaessa rajoittaa suoraviivaisesti IAM-käytännöillä (engl. *IAM Policy*).

IAM-käyttäjätunnusten luonnin jälkeen kokeillaan käyttöönotto-ohjeistuksessa mainittua interaktiivista demoa, joka esittelee AWS IoT Core -palvelun toimintaperiaatteita havainnollisella tavalla. Demo on pelinomainen sovellus, jossa käyttäjä voi esimerkiksi klikkailla hiirellä ”IoT-laitteen” painikkeita, jotka laukaisevat viestin IoT-laitteelta IoT Core -palveluun. Tämä puolestaan ohjaa viestin lamppua kuvaavalle ”IoT-laitteelle”, joka vaihtaa viestin saapuessa väriään. Demo auttaa hahmottamaan korkealla tasolla esimerkiksi IoT Coren käyttämän MQTT-protokollan toimintaa.

Interaktiivisesta demosta siirrytään kokeilemaan oikean IoT-laitteen pikayhdistämistä, joka toimii AWS IoT Coren ”Hello world” -esimerkkinä. Tämä tutoriaali esittelee pinnallisesti IoT-esineen liittämistä pilvialustaan automaattisesti generoidun yhdistämispaketin avulla. Tutoriaalin avulla selviää, onnistuuko yhteyden muodostus oman IoT-laitteen, kuten Raspberry Pi:n ja AWS IoT Coren välillä. Yhteyden muodostuksessa käytetään yhdistämispaketissa mukana tulevaa esimerkkiohjelmaa, johon ei kuitenkaan tutustuta koodin tasolla. Sen sijaan seuraavaksi suoritettavassa laajemmassa tutoriaalissa valmista yhdistämispakettia ei ole, vaan kaikki askeleet tehdään itse testausohjelman ohjelmointia lukuun ottamatta. Tässä tutoriaalissa esimerkiksi sertifikaattien ja salausavainten tarve tulee konkreettisesti esille, sillä ne generoidaan ohjeistetusti ja ilman niitä yhdistäminen IoT Coreen ei ole mahdollista.

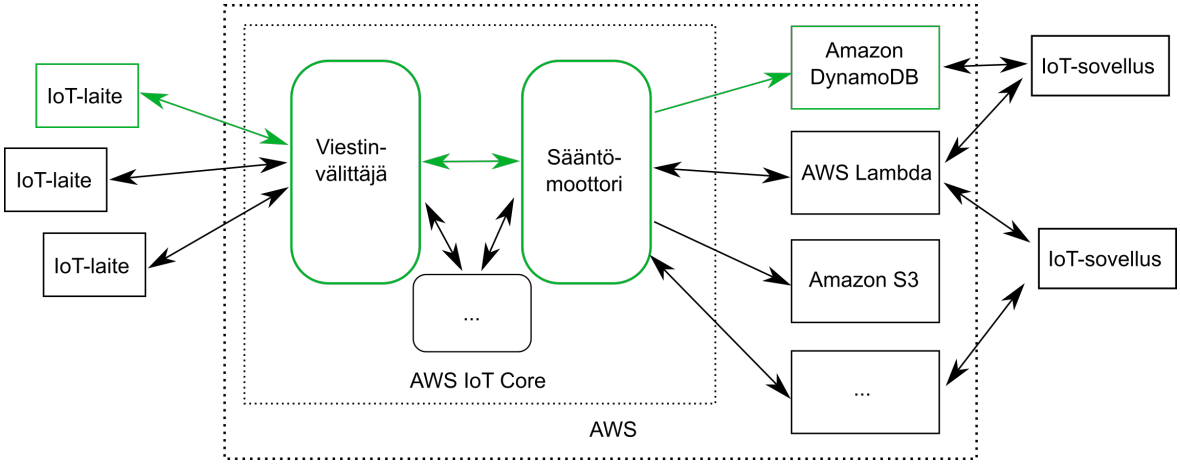
Viimeiseksi käyttöönotto-ohjeistuksessa käydään läpi AWS IoT Coren web-konsolista löytyvää MQTT-asiakasohjelmaa, jolla voi testata luomiaan sovelluksia sekä debugata niiden MQTT-viestiliikennettä. Kommunikaatio AWS IoT Coren kanssa tapahtuukin suureksi osaksi MQTT-protokollan avulla. Vaikka HTTPS on myös tuettu, AWS suosittelee MQTT-protokollan käyttöä, sillä IoT-laitteiden ei ole mahdollista vastaanottaa tässä pilvipalvelussa niille osoitettuja viestejä HTTPS:llä (“Device Communication Protocols - AWS IoT Core” 2021). Lisäksi MQTT on kevyempi protokolla (ks. luvut 3.3.1 ja 3.3.4).

Käyttöönotto-ohjeistuksen lisäksi AWS IoT Coren dokumentaatiossa on paljon materiaalia sen tarjoamista toiminnoista aina korkeatasoisista konseptiesittelyistä yksityiskohtaisiin esimerkkeihin ja API-dokumentaatioihin asti. Myös tutoriaaleja löytyy lisää useista aihealueista. Tämän tutkimuksen kannalta “Connect a Device to AWS IoT Core by Using the AWS IoT Device SDK - AWS IoT Core” (2021) -tutoriaali on kiinnostava, sillä siinä tutustutaan yksityiskohtaisesti IoT-laitteiden ja AWS IoT Coren väliseen kommunikaatioon. Tämä tapahtuu IoT-laitteille optimoidun AWS IoT Device SDK -paketin avulla. Tutoriaalissa käydään lähdekoodin tasolla läpi, kuinka kyseistä SDK-pakettia käytetään. Paketti hyödyntää MQTT-protokollaa viestien välityksessä, ja se tarjoaa API:n vain IoT-laitteille hyödyllisiin toimintoihin piilottaen samalla MQTT-protokollan yksityiskohdat. AWS tarjoaa lisäksi yleiskäyttöisempiä AWS SDK -paketteja sekä AWS CLI -komentorivityökalun AWS-resurssien hallintaan (“Connecting to AWS IoT Core Service Endpoints - AWS IoT Core” 2021). Näillä voi esimerkiksi automatisoida IoT-laitteiden rekisteröinnin AWS IoT Coreen, mutta niitä ei ole tarkoitettu käytettäväksi IoT-laitteen ja IoT Coren välisessä viestinvaihdossa.

IoT-laitteet viestivät AWS IoT Coren sisäisen viestinvälittäjän (engl. *Message broker*) kanssa (ks. kuvio 7). Viestinvälittäjä vastaa toiminnaltaan suurelta osin MQTT-standardin mukaista viestinvälittäjää (ks. luku 3.3.1), mutta erojakin on. MQTT:n QoS-tasoa 2 ei tueta, ja IoT Coren viestinvälittäjälle voi lähettää viestejä MQTT:n lisäksi HTTPS-protokollalla, kuten aiemmin mainittiin. Erona on myös IoT Coren sääntömoottori (engl. *Rules engine*), joka voi mm. muokata viestejä ennen kuin viestinvälittäjä toimittaa ne vastaanottajilleen.

Sääntömoottori mahdollistaa viestien muokkaamisen lisäksi IoT Coren suoran integraation lukuisiin muihin AWS:n palveluihin. Sen avulla on mahdollista viedä esimerkiksi JSON-muotoiset viestit IoT-laitteilta suoraan Amazon DynamoDB -tietokantaan, kuten tässä tutkimuksessa myöhemmin tehdään. AWS IoT Core sisältää myös muita IoT-järjestelmissä tarpeellisia ominaisuuksia, kuten laitevarjot (engl. *Device Shadows*), joiden avulla IoT-laitteen tilaa voidaan ylläpitää, vaikka laiteen yhteys pilvialustalle katkeaisi. Näitä ei kuitenkaan käsitellä tässä tutkimuksessa tarkemmin.

IoT-prototyypin keräämän telemetriadatan tallennukseen käytetään Amazon DynamoDB -palvelua, joka on mukana kuviossa 7. Tämä on skaalautuva NoSQL-tietokantapalvelu, joka tarjoaa esimerkiksi automaattisen datan salauksen levossa (“What Is Amazon DynamoDB?



Kuvio 7. AWS IoT Coren toimintamalli mukaillen “How AWS IoT Works - AWS IoT Core” (2021) -sivun kuvaa AWS IoT Coren tarjoamista palveluista. Kuviossa on mukana keskeisimmät osat telemetriadataa keräävien IoT-järjestelmien osalta. Tutkimuksen IoT-prototyypissä käytetyt osat on korostettu vihreällä.

- Amazon DynamoDB” 2021). DynamoDB:ssä data on jaettu tauluihin (engl. *table*), tietoalkioihin (engl. *item*) ja attribuutteihin (engl. *attribute*). Taulut sisältävät tietoalkioita, joissa puolestaan on attribuutteihin tallennettuja arvoja. Taulut ovat rakenteettomia (engl. *schemaless*), mikä tarkoittaa sitä, että tallennetut tietoalkiot voivat olla attribuuteiltaan keskenään erilaisia. Lisäksi, vaikka kaksi tietoalkiota sisältäisi saman attribuutin, voi tämän arvon tietotyyppi olla niissä eri. Toisessa voi olla esimerkiksi lukuarvo samalla, kun toisessa on lista merkkijonoista.

DynamoDB:n taululle tulee määrittää luonnin yhteydessä pääavain (engl. *primary key*), joka koostuu osiointiavaimesta (engl. *partition key*) ja valinnaisesti järjestysavaimesta (engl. *sort key*) (“Core Components of Amazon DynamoDB - Amazon DynamoDB” 2021). Jokaisella tietoalkiolla tulee olla uniikki osiointiavain taulun sisällä. Jos myös järjestysavain on määritelty, voi usealla tietoalkiolla olla sama osiointiavain, kunhan niiden järjestysavaimet poikkeavat toisistaan. Osiointiavaimet mahdollistavat tietokannan automaattisen rinnakkaistamisen, sillä data jaetaan niiden perusteella osioihin, joiden avulla data voidaan jakaa fyysisesti eri tallennuslevyille. Jos järjestysavainkin on määritetty, samalle osiolle tallennettavat tietoalkiot järjestetään sen mukaan. Osioinnin ja järjestämisen ansiosta tietokantakyselyistä saadaan hyvin tehokkaita datan määrästä riippumatta. Toisaalta ilman toissijaisia indeksejä

jokaisessa tietokantakyselyssä täytyy määritellä vähintään haettavien tietoalkioiden osiointiavaimet. Ilman osiointiavainta joudutaan tekemään kyselyn (engl. *query*) sijaan täysi skannaus (engl. *scan*), joka tarkoittaa taulun kaikkien tietoalkioiden lukua. Suurella datamäärällä tämä on kallis operaatio niin suorituskyvyn kuin siitä veloitettavan rahamääränkin kannalta. Jotta pääavaimeen kuuluvien sarakkeiden lisäksi taulun muillakin sarakkeilla voisi tehdä tehokkaita kyselyitä ilman pääavainta, tulee taululle lisätä toissijaisia indeksejä. Näille määritellään pääavaimen tapaan osiointiavain ja valinnainen järjestysavain. Indeksit päivitetään automaattisesti tietokantatauluun tehtyjen tietoalkiolisäysten, -muutosten ja -poistojen mukaan.

Tietoalkioiden luontien, lukujen, päivitysten ja poistojen (CRUD, engl. *Create, Read, Update, Delete*) tekemiseen voi käyttää DynamoDB:n perinteistä CRUD API:a (engl. *Classic CRUD API*), jossa jokaiselle toiminnolle on oma käskynsä. Vaihtoehtoisesti DynamoDB tarjoaa rajapinnan SQL-kieltä jäljittelevän PartiQL-kyselykielen käyttämiseksi. Tämä voi olla houkuttelevampi vaihtoehto relaatiotietokantoja käyttäneelle. PartiQL-kyselyiden haasteena on, että kehittäjän täytyy muistaa lisätä kyselyyn rajaus tietyllä osiointiavaimella. Jos tämä unohtuu, seurauksena on kyselyn sijaan koko taulun skannaus.

Datan vienti AWS IoT Coren viestinvälittäjältä Amazon DynamoDB -tietokantaan tapahtuu IoT Coren sääntömoottorilla (ks. kuvio 7). Tämän avulla voi määritellä sääntöjä, jotka koostuvat muun muassa MQTT-aihesuodattimesta ja viestin saapuessa suoritettavista toiminnoista. Aihesuodatin määrittää nimensä mukaan IoT Coren viestinvälittäjän MQTT-aiheet, joihin saapuvien viestien perusteella säännön toiminnot suoritetaan. Toimintovaihtoehtoja on useita ja näistä kaksi mahdollistaa MQTT-viestin sisällön tallentamisen DynamoDB-tietokantaan (“AWS IoT Rule Actions - AWS IoT Core” 2021). *DynamoDB*-toiminto mahdollistaa uuden tietoalkion luonnin DynamoDB-tauluun siten, että viestin sisältö tallennetaan kokonaisuudessaan tai osittain yhteen ennalta määrättyyn attribuuttiin. Tämä on hyödyllistä, jos MQTT-viestin sisältö ei ole JSON-muodossa tai tietokantaa ei tarvitse indeksoida JSON-muotoisen viestin sisällön perusteella. JSON-muotoisen viestin tallentamiseen voi käyttää myös *DynamoDBv2*-toimintoa, jonka avulla viestin sisällön voi kokonaan tai osittain jakaa useisiin tietoalkion attribuutteihin.

4.3.3 Raspberry Pi:n yhdistäminen AWS:ään

Siirrytään seuraavaksi takaisin IoT-prototyyppiin ja käydään läpi vaadittavia toimenpiteitä Raspberry Pi:n yhdistämiseksi AWS:ään. Yhteyden luomista varten AWS IoT Coreen lisätään IoT-käytäntö (engl. *IoT Policy*) sekä esineolio (engl. *thing object*) Raspberry Pi:lle. IoT-käytännön avulla rajataan IoT-laitteiden käyttöoikeuksia IoT Coren resursseihin. Sillä voi esimerkiksi määritellä MQTT-aiheet, joihin IoT-laitteiden on sallittu lähettää viestejä. Esineolio vastaa puolestaan yksittäistä IoT-laitetta IoT Coren rekisterissä. Esineoliolla on yksilöllinen nimi ja sille täytyy lisätä myös sertifikaatti. Tämä tarvitaan, sillä ilman sitä IoT Core ei voi varmentaa siihen yhdistävää laitetta, eikä yhdistäminen siksi ole tässä tilanteessa mahdollista. Sertifikaattiin liitetään yksi tai useampi IoT-käytäntö.

IoT-käytännön, sertifikaattien ja esineolion luonnin jälkeen IoT-laite voidaan yhdistää AWS IoT Coreen esimerkiksi AWS IoT Device SDK -paketin avulla. Python-ohjelmointikielelle se on saatavilla `awsiotsdk`-nimisen PyPI-paketin kautta. Kun tämä paketti on asennettu, siirrytään lisäämään datankeruuohjelmaan yhdistämistoiminnallisuutta AWS IoT Corelle. Tässä käytetään apuna aiemmin mainittua “Connect a Device to AWS IoT Core by Using the AWS IoT Device SDK - AWS IoT Core” (2021) -tutoriaalia. Ohjelmaan lisätään ensimmäiseksi `awscrt`- ja `awsiot`-kirjastojen importit. Seuraavaksi luodaan `AWSClient`-luokka, jonka konstruktorissa hoidetaan MQTT-yhteyden muodostukseen tarvittavien olioiden alustukset. Yksi näistä olioista on `ClientBootstrap`, joka hoitaa sokettiviestinnän erillisessä säikeessä tapahtumasilmukan (engl. *event loop*) avulla. Tämän ansiosta kehittäjän ei tarvitse huolehtia aktiivisesti verkkoliikenteestä esimerkiksi silmukalla tai itse luodulla säikeellä. Importit ja luokan konstruktori on esitetty seuraavassa koodikatkelmassa. Epäolennaisia rivejä on jätetty pois kohdista, joissa on ”`# ...`”-kommentti.

```
# API: https://awslabs.github.io/aws-crt-python/
from awscrt import io, mqtt
# API: https://aws.github.io/aws-iot-device-sdk-python-v2/index.html
from awsiot import mqtt_connection_builder

# ...
class AWSClient():
    """Class for all AWS Cloud specific functionality."""
```

```
    def __init__(self, args):
        # ...
        event_loop_group = io.EventLoopGroup(1)
        host_resolver = io.DefaultHostResolver(event_loop_group)
        client_bootstrap = io.ClientBootstrap(event_loop_group,
            host_resolver)
        self.mqtt_connection = mqtt_connection_builder.mtls_from_path(
            endpoint=self.endpoint,
            cert_filepath=self.cert,
            pri_key_filepath=self.key,
            client_bootstrap=client_bootstrap,
            ca_filepath=self.root_ca,
            on_connection_interrupted=self.on_connection_interrupted,
            on_connection_resumed=self.on_connection_resumed,
            client_id=self.client_id,
            clean_session=False,
            keep_alive_secs=6)
```

`AWSClient`-luokan `connect()`-metodissa kutsutaan MQTT-yhteysolion `connect()`-metodia ja odotetaan, että yhteydenmuodostus onnistuu. Tämä tapahtuu edellisen metodin palauttaman olion `result()`-metodilla. `disconnect()`-metodi rakennetaan vastaavalla tavalla. Viestin lähetystä varten luodaan `send_message(topic, message)`-metodi, jonka parametreista ensimmäinen on MQTT-aihe, johon viesti lähetetään ja toinen on viestin sisältö. Viestin lähetys toteutetaan MQTT-yhteysolion `publish(...)`-metodilla, jolle annetaan parametriksi aiheen ja viestin sisällön lisäksi QoS-taso. Tässä käytetään tasoa 0. Nämä metodit näyttävät tältä:

```
    def connect(self):
        print("Connecting to {} with client ID '{}'..."
            .format(self.endpoint, self.client_id))
        connect_future = self.mqtt_connection.connect()

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

```
    def disconnect(self):
        print("Disconnecting from AWS...")
        disconnect_future = self.mqtt_connection.disconnect()
        disconnect_future.result()
        print("Disconnected from AWS!")


    def send_message(self, topic, message):
        print("Publishing message to AWS in topic '{}': {}"
            .format(topic, message))
        self.mqtt_connection.publish(
            topic=topic,
            payload=message,
            qos=mqtt.QoS.AT_MOST_ONCE)
```

MQTT-aihe on muotoa `dt/weather/{location}/{client_id}` datankeruuohjelmassa. Tässä `dt` tarkoittaa ”data”, `{location}` on IoT-laitteen sijainti (esim. ”jyvaskyla”) ja `{client_id}` on IoT-laitteen tunniste (esim. ”RPi2”). Aiheen suunnittelussa on pyritty noudattamaan AWS IoT Coren MQTT-aiheen suunnitteluohjetta. Siinä suositellaan `dt/<application>/<context>/<thing-name>/<dt-type>`-muotoisia aiheita telemetriadatan siirtoon (“Designing MQTT Topics for AWS IoT Core” 2019).

Kun data on saatu kulkemaan AWS IoT Coreen, siirrytään luomaan DynamoDB-tietokantaa. Tietokannan ositusavaimeksi valitaan laitteen nimi, joka tallennetaan `device_id`-attribuuttiin. Järjestysavaimena käytetään arvojen mittausajanhetken POSIX-aikaleimaa millisekunneissa. Tämä tallennetaan tietoalkioiden `timestamp`-attribuuttiin. Kun avaimet valitaan näin, voidaan dataa hakea kyselyllä tietyltä laitteelta halutulta aikaväliltä ilman toissijaisia indeksejä. Tutkimuksen prototyypissä on vain yksi laite, minkä takia ositusavain saa aina saman arvon, mutta dataa tallennetaan ja sitä kysellään niin vähän, että käytännössä tämä ei haittaa. Lisäksi jos järjestelmään liitettäisiin useampia mittausasemia, jakautuisi kerätty data suhteellisen tasaisesti ositusavainten välille.

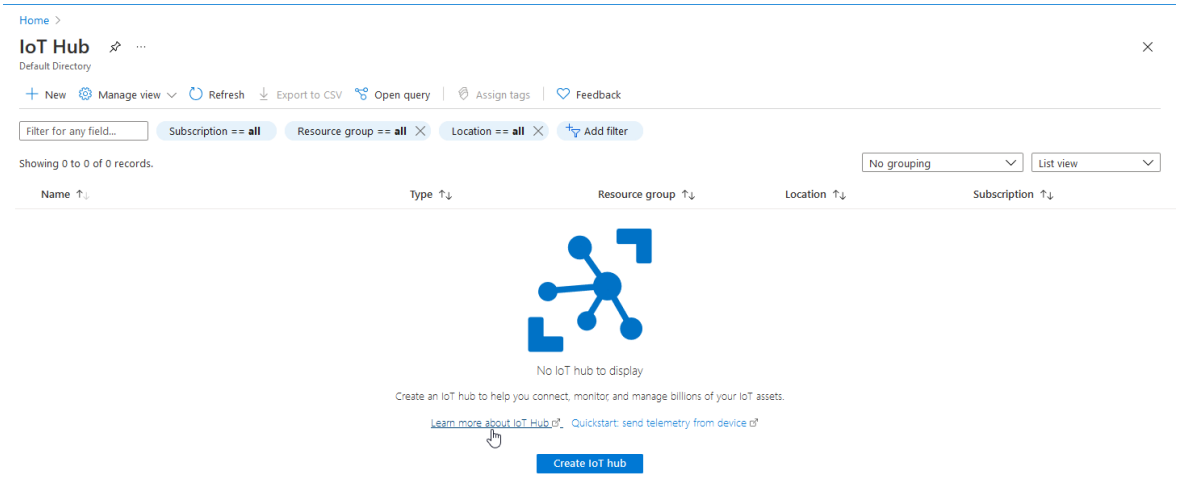
Datan vienti AWS IoT Coresta DynamoDB-tietokantaan tehdään AWS IoT -säännön avulla. Säännölle määritellään kyselylause (engl. *Rule query statement*) SQL-kieltä muistuttavalla syntaksilla. Kyselylauseeseen voi lisätä MQTT-aihesuodattimen, jonka mukaisista MQTT-

aiheista tulevat viestit käsitellään tässä säännössä. Kyselylauseella voi myös rajata JSON-muotoisesta MQTT-viestistä vain oleelliset tiedot säännön käsiteltäviksi. Kyselylauseeksi valitaan `SELECT * FROM 'dt/weather/#'`. Tämän lauseen aihesuodattimessa käytetään #-villikorttia, joka vastaa mielivaltaista merkkijonoa. Toisin sanoen suodatin rajaa käsiteltäviksi kaikki viestit MQTT-aiheisiin, joiden nimi alkaa `dt/weather/`. Kyselylauseen `SELECT`-osassa on vain $*$-merkki, mikä tarkoittaa, että viestin sisältö otetaan käsittelyyn sellaisenaan. IoT-säännön toiminnoksi valitaan *DynamoDBv2*. Tämän konfiguraationäkymästä valitaan DynamoDB-taulu, johon viestit tallennetaan sekä IAM-rooli sopivilla käyttöoikeuksilla. Roolin voi myös luoda automaattisesti samasta näkymästä. DynamoDBv2-toiminto vaatii, että MQTT-viesti on JSON-muotoinen ja sisältää juuritasolla sopivat avaimet ja arvot DynamoDB-taulun ositus- ja järjestysavaimille. Tässä tapauksessa viestissä tulee olla vähintään merkkijono `device_id`-avaimella ja lukuarvo `timestamp`-avaimella. Näin on esimerkiksi luvussa 4.2.2 esitetyssä JSON-muotoisessa esimerkkiviestissä. Kun datankeruuohjelman ajaa nyt Raspberry Pi:lla, siirtyy säädata IoT Coren kautta DynamoDB-tietokannan tauluun. Yksityiskohtaisempi kuvaus tässä luvussa läpikäydyistä vaiheista löytyy liitteestä A.3.

4.3.4 Microsoft Azure

Microsoft Azure -pilvialustaan tutustuminen aloitetaan Google-haulla ”azure iot”. Haun ensimmäinen tulos on Azuren IoT-ominaisuuksien esittelysivu. Sivun keskivaiheilla on eritelty Azuren IoT-tuotteita ja -palveluita linkkeinä kunkin palvelun omille esittelysivuille. Näistä ensimmäinen on linkki Azure IoT Hub -palveluun, jonka mainitaan kykenevän miljardien IoT-laitteiden yhdistämiseen ja hallintaan. Seuraamalla linkkiä päädytään IoT Hubin esittelysivulle, jonka ylälaidassa on ”Start free” -painike keskeisellä paikalla. Painike vie ilmaisen kokeilujakson esittelysivulle, jossa on toinen samanlainen painike. Tämä puolestaan ohjaa kirjautumisen tai tarvittaessa Microsoft-tilin ja Azure-tunnusten luonnin kautta Microsoft Azure -palvelun web-portaalin etusivulle. Web-portaali on graafinen käyttöliittymä Azuren resurssien hallintaan. Portaalin etusivulla ei välttämättä ole suoraa linkkiä IoT Hubin portaaliin, mutta yläreunan hakutoiminnolla tämä löytyy haulla ”iot”. IoT Hubin portaalisivun (ks. kuvio 8) keskellä on linkit yleiskuvan esittävälle *What is Azure IoT Hub*

-dokumentaatiosivulle sekä pika-aloitustutoriaaliin, jossa esitellään telemetriadatan keräämistä IoT-laitteella *Azure IoT device SDK*:n avulla. Seuraavissa tekstikappaleissa käsitellään näitä.



Kuvio 8. Microsoft Azure IoT Hub -palvelun web-portaalin etusivu kuvattuna 19.5.2021.

Azure IoT Hubin kehittäjädokumentaation yleiskuvan esittävällä “What Is Azure IoT Hub” (2021) -sivulla kerrotaan palvelun teknisistä kyvykkyyksistä esimerkiksi tietoliikenteen suojaamisessa, viestiliikenteen reitittämisessä ja integroinnissa muihin Azure-pilvialustan palveluihin. Sivulla listataan myös ohjelmointikielet, joille on saatavilla IoT-laitteiden yhdistämisessä käytettävä Azure IoT device SDK -paketti. Kirjoitushetkellä kieliin kuuluvat Python, Node.js, Java, C# ja C. Lisäksi sivulla mainitaan tuetut protokollat, jotka ovat MQTT, AMQP ja HTTPS. Kahta ensimmäistä voi tarvittaessa käyttää web-sokettien yli. SDK-paketit käyttävät kyseisiä protokollia sisäisesti, mutta kommunikointi IoT Hubiin onnistuu myös yleiskäyttöisillä protokollakirjastoilla.

IoT-laitteiden SDK-paketteihin pääsee tutustumaan IoT Hubin pika-aloitustutoriaalissa, jossa käydään läpi IoT-sovelluskehitystä Azure-pilvialustalla (“Send Device Telemetry to Azure IoT Hub Quickstart” 2021). Tutustuminen tehdään liittämällä IoT Hubiin yksinkertainen

SDK-pakettia hyödyntävä esimerkkisovellus, joka lähettää generoitua telemetriadataa sinne. Saman tutoriaalin voi suorittaa useammalla eri ohjelmointikielellä. Tämän tutoriaalin lisäksi kehittäjädokumentaatiosta löytyy useita muitakin tutoriaaleja, joissa mennään syvemmälle eri aihealueisiin. Tämän tutkimuksen kannalta yksi relevantti tutoriaali on "Tutorial - Configure Message Routing for Azure IoT Hub Using Azure CLI" (2021), jossa käsitellään viestien reititystä IoT Hubista eräisiin Azuren palveluihin. Tutoriaali on kaksiosainen ja kokonaisuudessaan melko suuri, koska siinä käydään läpi useita eri reititysmahdollisuuksia. Tarvittaessa osan näistä voi kuitenkin jättää väliin. Tutoriaalin reititysosuuden voi suorittaa käyttäen web-portaalia tai vaihtoehtoisesti esimerkiksi Azure CLI:tä, joka on komentorivipohjainen työkalu pilvialustan resurssien hallintaan. Tutoriaalissa IoT Hubiin saapuvat viestit reititetään kolmeen paikkaan. Osa viesteistä viedään tietovarastotilille (engl. *storage account*) lisättyyn BLOB-varastoon (so. engl. *Binary Large OBject*), osa viedään *Service Bus* -jonoon ja loput päätyvät oletusreitin ja *Azure Stream Analytics* -työkalun kautta *Power BI* -visualisointityökaluun. Tässä tutkimuksessa Power BI -työkalua ei kokeiltu.

BLOB-varastoon viestit voi viedä Apache Avro- ("Welcome to Apache Avro!" 2021) tai JSON-muodossa. Tietovaraston luonteen vuoksi tallennusmuodosta huolimatta datalle ei ole mahdollista tehdä suoraviivaisia kyselyjä, toisin kuin dokumenttipohjaisissa NoSQL-tietokannoissa. Varastoon luodaan minuutin välein tiedosto, johon kyseisen minuutin aikana saapuneet viestit tallennetaan sellaisenaan. Tuloksena voi olla esimerkiksi tekstitiedostoja, jotka sisältävät useita peräkkäin kirjoitettuja JSON-muotoisia viestejä. Jos viestejä on useita samassa tiedostossa, tämän sisältö ei vastaa täysin JSON-syntaksia. BLOB-varasto on tarkoitettu esimerkiksi lokitiedostojen tallennukseen sekä tiedon varmuuskopiontiin ja arkistointiin.

Tutoriaalissa viestejä viedään myös Service Bus -jonoon. Tämän avulla viestit voidaan kuljettaa yhdelle vastaanottajalle, joka voi sijaita pilvialustalla tai sen ulkopuolella. Tutoriaalissa vastaanottajana käytetään *Azure Logic Apps* -pilvisovellusta, jonka avulla viestit lähetetään määritettyyn sähköpostiosoitteeseen. Logic Apps -palvelu mahdollistaa esimerkiksi pilvialustan palveluiden integroinnin halutulla tavalla keskenään kirjoittamatta ohjelmakoodia ("Overview for Azure Logic Apps - Azure Logic Apps" 2021). IoT Hubista on mahdollista viedä viestejä myös Service Bus -aiheeseen, jossa vastaanottajia voi olla useita. Tämä vastaa

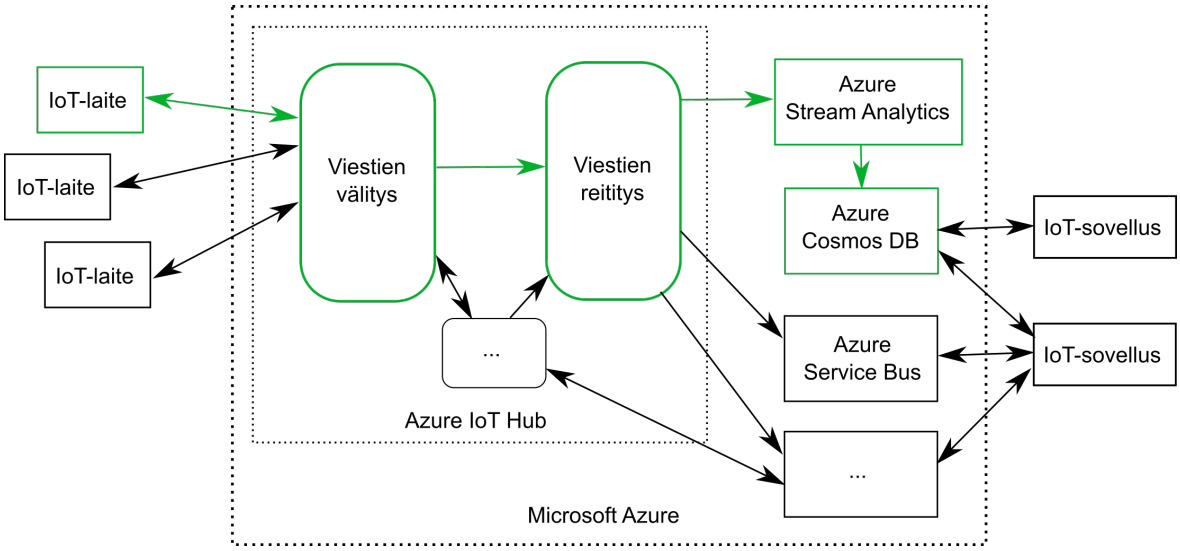
julkaise/tilaa-mallia, jossa Service Bus toimii viestinvälittäjänä.

Kolmanneksi tutoriaalissa käytetään IoT Hubin oletusreittiä, josta haetaan dataa Azure Stream Analytics -työkalulla. Tämä työkalu mahdollistaa suurtenkin datavirtojen analysoinnin ja prosessoinnin reaaliajassa. Tutoriaalissa työkalua käytetään vain datan vientiin Power BI -palveluun, mutta Stream Analytics mahdollistaa datan viennin myös useisiin muihin Azuren palveluihin. Yksi tällainen palvelu on *Azure Cosmos DB*, jota käytetään tässä tutkimuksessa.

Tutoriaalien lisäksi IoT Hubin kehittäjädokumentaatiossa kerrotaan laajasti sen ominaisuuksista, konsepteista ja teknisistä yksityiskohdista. Dokumentaatio ei käsittele konsepteja ainoastaan IoT Hubin osalta, vaan myös laajemmin IoT-teollisuuden näkökulmasta. Esimerkiksi "Concepts of Azure IoT Hub X.509 Security" (2021) -sivulla käsitellään autentikaatiota epäsymmetrisillä X.509 sertifikaateilla ja niiden hyötyjä mm. IoT-laitteiden massatuotannossa. Oletuksena IoT Hub tarjoaa IoT-laitteen rekisteröinnin yhteydessä symmetristä avainta laitteen autentikointiin. Tässä on etuna käyttöönoton helppous, mutta haittapuolena sama avain tulee olla tallennettuna laitteen lisäksi pilvipalveluun. Epäsymmetristen avainten tapauksessa pilvipalveluun voidaan tallentaa vain julkinen avain tai laitesertifikaatti. Epäsymmetrisen avainparin yksityinen avain pidetään ainoastaan laitteessa.

IoT-laitteet viestivät Azure IoT Hubin kanssa (ks. kuvio 9) käyttäen MQTT-, AMQP- tai HTTPS-protokollaa. Azure suosittelee dokumentaatiossaan ensisijaisesti IoT device SDK-pakettien käyttöä viestintään. Nämä käyttävät tuettuja protokollia sisäisesti kehittäjän valinnan mukaan sekä hoitavat salauksen TLS-protokollalla ja autentikaation valitulla menetelmällä. IoT Hubin kanssa on myös mahdollista kommunikoida suoraan tuetuilla protokollilla käyttämällä esimerkiksi yleiskäyttöistä protokollakirjastoa. IoT Hub ei kuitenkaan toteuta esimerkiksi täysiveristä MQTT-viestinvälittäjää, vaan viestintä on tarkkaan rajattua. MQTT-aihetta ei voi valita vapaasti, vaan sen tulee olla tietyssä muodossa. Esimerkiksi laitteelta pilveen suuntautuvassa viestinnässä MQTT-aihe määritellään muodossa `devices/{device_id}/messages/events/{property_bag}`. Tässä `{device_id}` on laitteelle IoT Hubissa määritelty tunniste ja `{property_bag}` on lista mahdollisista ylimääräisistä tunnisteista URL-enkoodatussa muodossa. Jos tunnistelistassa on sisältöä, sen tulee olla muodossa `prop1=value1&prop2=value2...`, missä kaikki `propX`:t ovat tunnisteiden avaimia ja `valueX`:t ovat näiden arvoja. Myös itse viestien formaatti on tar-

kasti määritetty, jotta toiminta eri protokollien välillä olisi mahdollisimman yhdenmukaista (“Understand Azure IoT Hub Message Format” 2021).



Kuvio 9. Microsoft Azure IoT Hubin toimintamalli mukaillen “Overview of Azure IoT Hub Message Enrichments” (2021) -sivun kuvaa sekä muuta IoT Hubin dokumentaatiosta kerättyä tietoa. Kuviossa on mukana keskeisimmät toiminnot, joita tarvitaan telemetriadataa keräävässä järjestelmässä. Tutkimuksen IoT-prototyypissä käytetyt osat on korostettu vihreällä.

Viestien reititys muihin Azuren palveluihin on toteutettu viestien kaksisuuntaisesta välityksestä erillisenä osiona, kuten kuviossa 9 on esitetty. Reitityksen avulla viestejä voidaan viedä esimerkiksi Azure Service Busiin tai Stream Analyticsiin, mutta vienti ei kuitenkaan onnistu suoraan Azure Cosmos DB -tietokantaan. Tähän täytyy sen sijaan käyttää Azuren muita palveluja, kuten Stream Analyticsiä. IoT Hubissa on myös muita toimintoja, kuten laitekaksoset (engl. *Device twins*), jotka ylläpitävät IoT-laitteiden tilaa pilvessä. Näitä ei kuitenkaan käsitellä tässä tutkimuksessa.

IoT-prototyypin telemetriadatan tallennukseen käytetään Azure Cosmos DB -tietokantaa, joka on mukana kuviossa 9. Cosmos DB on NoSQL-tietokanta, joka tukee useaa eri tietokantamallia ja API-rajapintaa (“Azure Cosmos DB Resource Model” 2021). Tuettuja rajapintoja ovat esimerkiksi *Cassandra API*, *Table API* ja *Azure Cosmos DB API for MongoDB*. Nämä mahdollistavat parhaimmillaan olemassaolevan ohjelmiston siirtämisen toisesta NoSQL-

tietokannasta Cosmos DB:hen ainoastaan tietokannan yhteysparametreja muuttamalla ja datan migroinnilla. Rajapinnat tarjoavat dokumenttipohjaisten tietokantojen lisäksi esimerkiksi avain-arvo- tai verkkotietokantoja. Erityisesti uusille projekteille suositeltu rajapinta on Cosmos DB:n natiivi SQL API, jolla esimerkiksi kyselyitä voi tehdä SQL:ää vastaavalla syntaksilla. Tätä rajapintaa käytetään tutkimuksen IoT-prototyypissä. Cosmos DB:n tietorakenne koostuu tietokannoista, säiliöistä (engl. *Container*), tietoalkioista (engl. *Item*) ja ominaisuuksista (engl. *Property*). Tietokannat sisältävät säiliöitä, jotka puolestaan sisältävät tietoalkioita, jotka koostuvat ominaisuuksista. Tietorakenteen osia kutsutaan eri termeillä riippuen käytetystä rajapinnasta. Esimerkiksi Cassandra API:a käytettäessä tietokannat ovat avaintiloja (engl. *Keyspace*), säiliöt ovat tauluja ja tietoalkiot ovat rivejä. Tässä käytetään ensin mainittuja termejä, jotka ovat samat kuin SQL API:ssa.

Kun tietokantaan lisätään säiliö, täytyy tälle määrittää osiointiavain (engl. *Partition key*), jonka perusteella tietoalkiot jaetaan loogisiin osioihin. Tämä mahdollistaa tietokannan skaalauksen niin suorituskyvyn kuin datamääränkin suhteen. Jokaiselle tietoalkiolle tulee määrittää yhtenä ominaisuutena myös tietoalkion ID (engl. *item ID*), joka yhdessä osiointiavaimen kanssa yksilöi tietoalkiot. ID:n voi myös määrittää osiointiavaimeksi, jolloin jokainen tietoalkio kuuluu erilliseen osioon. Cosmos DB indeksoi oletuksena säiliöiden sisällä kaikkien tietoalkioiden kaikki ominaisuudet (“Indexing in Azure Cosmos DB” 2021). Tämän ansiosta säiliöissä voi tehdä kyselyiden rajauksia mille tahansa tietoalkion ominaisuudelle ilman, että kehittäjän tarvitsisi määritellä toissijaisia indeksejä. Jos kyselyt sisältävät enemmän kuin yhden ominaisuuden esimerkiksi suodatus- tai järjestysehtona, on säiliöön kuitenkin hyvä määritellä yhdistelmäindeksi raskaampien hakujen välttämiseksi. Näitä ovat esimerkiksi jonkin indeksin tai koko säiliön täysi skannaus. Täysi skannaus tarkoittaa kaikkien tietoalkioiden lukua, mikä on kallis operaatio, jos dataa on paljon.

Telemetriaviestien vienti IoT Hubista Cosmos DB:hen onnistuu esimerkiksi Azure Stream Analyticsin avulla (ks. kuvio 9). Kuten aiemminkin tässä luvussa on mainittu, Stream Analytics on työkalu, joka kykenee käsittelemään ja analysoimaan suuriakin datavirtoja — tarvittaessa myös useista eri lähteistä (“Introduction to Azure Stream Analytics” 2021). Yksi tällainen lähde on IoT Hubin reitittäjän oletusreitti, josta Stream Analytics osaa lukea sinne saapuvat viestit. Työkalu pystyy lukemaan dataa myös mm. BLOB-varastosta ja Azure

Event Hubista, johon muista Azuren palveluista viedään tapahtumailmoituksia. Stream Analyticsin ulostuloiksi voidaan asettaa Cosmos DB SQL API -tietokannan lisäksi esimerkiksi Azure Power BI -visualisointityökalu, Event Hub tai Azure Functions -pilvifunktio. Stream Analyticsin data-analyysitoimintoja ei käsitellä tässä tutkimuksessa.

4.3.5 Raspberry Pi:n yhdistäminen Azureen

Tarkastellaan seuraavaksi tutkimuksen IoT-prototyypin näkökulmasta toimenpiteitä, jotka vaaditaan Raspberry Pi:n yhdistämiseksi Azure-pilvialustaan. Ensimmäiseksi Azureen luodaan uusi IoT Hub ja lisätään se uuteen resurssiryhmään. Resurssiryhmien avulla Azuressa voi ryhmitellä samoihin projekteihin tai muihin loogisiin kokonaisuuksiin liittyvät resurssit. IoT Hubille valitaan myös maantieteellinen alue, johon se perustetaan. Eri alueilla olevat resurssit pystyvät keskustelemaan keskenään, mutta tästä voi aiheutua lisäkustannuksia. IoT Hubin luonnin jälkeen Raspberry Pi rekisteröidään sinne uudeksi laitteeksi. Rekisteröinnin yhteydessä autentikaatioavaimeksi valitaan automaattisesti generoitu symmetrinen avain, koska se riittää prototyypin tarpeisiin. Rekisteröinnin jälkeen avataan laitteen tiedot ja kopioidaan generoiduista symmetrisistä avaimista ensisijainen avain (engl. *Primary Key*) talteen, sillä sitä tarvitaan Raspberry Pi:lla yhteyden muodostuksessa.

IoT Hubin luonnin ja Raspberry Pi:n rekisteröinnin jälkeen tämä voidaan yhdistää IoT Hubiin esimerkiksi Azure IoT Device SDK:n avulla. IoT-prototyypissä kyseisestä SDK-paketista käytetään Python-versiota. Aiemmin mainitun pika-aloitustutoriaalin (“Send Device Telemetry to Azure IoT Hub Quickstart” 2021) kautta löytyy SDK-paketin GitHub-sivu, jossa on esimerkkejä paketin käyttämisestä. SDK tukee Pythonin asynkronista I/O-rajapintaa, joka helpottaa esimerkiksi verkkoyhteyksistä riippuvan koodin kirjoittamista. SDK tukee myös synkronisia kutsuja, joita käyttämällä ohjelman pääsäie odottaa aina esimerkiksi yhteyden muodostuksen tai viestin lähetyksen onnistumista. Tämä on kuitenkin tarkoitettu yhteensopivaksi ensisijaisesti vanhan Python 2.7 -pohjaisen SDK:n kanssa, ja siksi GitHub-sivulla suositellaan asynkronisen API-rajapinnan käyttämistä. Tätä käytetään myös tässä tutkimuksessa.

Azure IoT Device SDK löytyy Pythonille `azure-iot-device`-nimisenä PyPI-pakettina.

Paketin asennuksen jälkeen SDK voidaan ottaa käyttöön datankeruuohjelmassa lisäämällä importit tarpeellisille `azure.iot.device`-kirjaston luokille. Tässä tarvitaan `Message`- ja `IoTHubDeviceClient`-luokkia. Näistä ensimmäisen avulla luodaan viesti, johon voi liittää mukautettuja ominaisuuksia. Näiden avulla esimerkiksi viestin reititys sujuu suoraviivaisemmin. `IoTHubDeviceClient`-luokan instanssi hoitaa puolestaan kommunikoinnin IoT Hubin kanssa. Seuraavaksi luodaan `AzureClient`-luokka, jonka konstruktorissa kyseinen luokkainstanssi alustetaan. Alustus tehdään `create_from_symmetric_key`-metodilla, jolle annetaan yhteysparametrit. Ensimmäinen parametri on aiemmin IoT Hubista kopioitu Raspberry Pi:n autentikaatioavain, toinen on IoT Hubin verkko-osoite ja kolmas on IoT Hubista löytyvä Raspberry Pi:n tunniste, joka annettiin rekisteröinnin yhteydessä. Alustuksen voi tehdä myös `create_from_connection_string`-metodilla, jolle samat parametrit viedään yhtenä yhteysmerkkijonona (engl. *connection string*). Tämän merkkijonon voi kopioida IoT Hubiin rekisteröidyn laitteen tiedoista samalta sivulta kuin ensisijaisen symmetrisen avaimenkin. Seuraavassa koodikatkelmassa on esitetty SDK-paketin importit, symmetrisen avaimen tuonti ympäristömuuttujana sekä `AzureClient`-luokan konstruktori. Epäolennaisia rivejä on jätetty pois kohdista, joissa on ”`# ...`”-kommentti.

```
# Python Device SDK for IoT Hub:
# https://github.com/Azure/azure-iot-sdk-python
from azure.iot.device.aio import IoTHubDeviceClient
from azure.iot.device import Message

# ...
AZURE_DEVICE_PRIMARY_KEY = os.getenv("AZURE_DEVICE_PRIMARY_KEY")

# ...
class AzureClient():
    """Class for all Azure Cloud specific functionality."""
    def __init__(self, args):
        # ...
        self.key = AZURE_DEVICE_PRIMARY_KEY
        self.hostname = self.azure_iot_hub_name + ".azure-devices.net"

        self.client = IoTHubDeviceClient.create_from_symmetric_key(
            self.key, self.hostname, self.device_id
```

```
        )
```

`IoTHubDeviceClient`-asiakasolion metodit ovat hyvin suoraviivaisia. `AzureClient`-luokan `connect()`- ja `disconnect()`-metodeissa kutsutaan asynkronisesti asiakasolion vastaavia metodeja. Viestin lähetys tehdään `send_message(custom_properties, json_message)`-metodissa. Tässä luodaan ensin `Message`-tyyppinen viesti metodin parametrina annetusta JSON-muotoisesta merkkijonosta sekä lisätään siihen annetut mukautetut ominaisuudet. Mukautetut ominaisuudet annetaan Pythonin dict-hakurakenteena (engl. *dictionary*). Datankeruuohjelmassa hakurakenne on `{"location": args.location}` -muotoinen. Tässä `args.location` on ohjelmalle argumenttina annettu Raspberry Pi:n maantieteellinen sijainti. Tämä voi olla esimerkiksi ”jyvaskyla”. Lopuksi viesti lähetetään asiakasolion `send_message()`-metodilla. Nämä metodit näytetään seuraavassa katkelmassa:

```
    async def connect(self):
        print("Connecting to host {} with device ID '{}'..."
            .format(self.hostname, self.device_id))
        await self.client.connect()
        print("Connected to Azure!")


    async def disconnect(self):
        print("Disconnecting from Azure...")
        await self.client.disconnect()
        print("Disconnected from Azure!")


    async def send_message(self, custom_properties, json_message):
        message = Message(
            json_message,
            content_encoding="utf-8",
            content_type="application/json"
        )
        message.custom_properties.update(custom_properties)

        print("Sending message to Azure: {}".format(message))
```

```
            await self.client.send_message(message)
```

Kun data välittyy Azure IoT Hubiin, siirrytään lisäämään Cosmos DB -tietokantaa. Ensin luodaan Cosmos DB -tietokantatili, jonka API-rajapinnaksi valitaan Core (SQL) API. Tietokantatili lisätään samaan resurssiryhmään ja samalle maantieteelliselle alueelle kuin IoT Hub. Luonnin jälkeen tilille lisätään uusi tietokanta ja säiliö. Säiliön osiointiavaimeksi määritetään `/device_id`, joka sisältää JSON-viestissä välitetyn IoT-laitteen tunnisteen. Viestissä tulee tämän lisäksi välittää `id`-kentässä tunniste, joka yhdessä `device_id`:n kanssa yksilöi viestin. Datankeruuohjelmassa viestin ID on merkkijonoksi muutettu aikaleima, joka on esitelty yhdessä muiden JSON-viestin kenttien kanssa luvussa 4.2.2.

Telemetriaviestit viedään IoT Hubista Cosmos DB -tietokantaan IoT Hubin reitittäjän oletusreitin ja Azure Stream Analyticsin kautta. Ensimmäiseksi lisätään IoT Hubiin reitti sisäänrakennettuun *events*-päätepisteeseen, joka toimii oletusreittinä. Reitille voi määrittää reitityskyselyn, jonka perusteella viesteistä valitaan reitille vain sellaiset, jotka kysely hyväksyy. Tässä käytetään esimerkin vuoksi kyselyä `IS_STRING(location)`, joka tarkistaa, löytyykö viestin mukautetuista ominaisuuksista ”location”-avaimella määritettyä merkkijonoa.

Kun oletusreitti on kyselyineen määritetty, luodaan Stream Analytics eräajo samalle maantieteelliselle alueelle ja samaan resurssiryhmään kuin muutkin resurssit. Eräajolle määritetään sisääntulo, ulostulo ja kysely. Sisääntuloksi asetetaan IoT Hubin oletusreittiin saapuvat viestit ja sen tunnisteeksi asetetaan `weatherinput`. Viestien tyypiksi asetetaan JSON ja osiointiavaimeksi voi määrittää esimerkiksi `device_id`. Tämä voi tehostaa viestivirran käsittelyä, jos eräajon käytössä on useita virtausyksiköitä (engl. *Streaming units*), jotka käsittelevät viestivirtoja. Eräajon ulostuloksi lisätään edellä luotu Cosmos DB -säiliö ja sen tunnisteeksi asetetaan `weatheroutput`. Eräajon kysely määritellään vaihtamalla oletuskyselyyn sisään- ja ulostuloksi edellä määritellyt. Viestien sisällöt säilytetään sellaisenaan, jolloin kyselystä tulee seuraava:

```
SELECT *
INTO [weatheroutput]
FROM [weatherinput]
```

Kun Stream Analytics eräajon käynnistää ja datankeruuohjelman ajaa, säädata siirtyy Rasp-

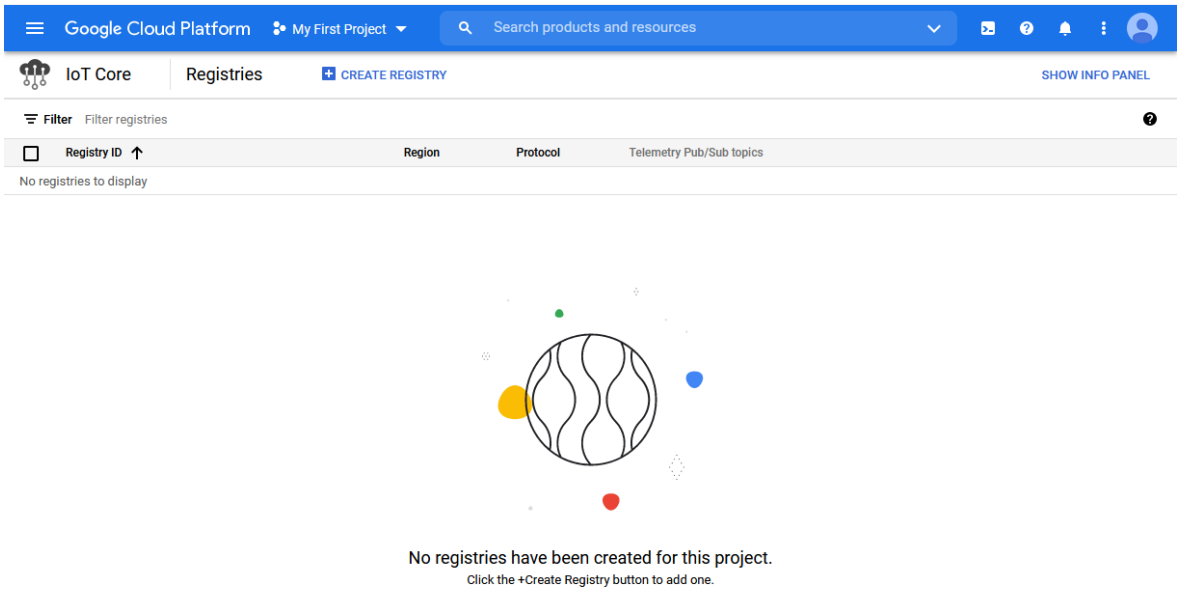
berry Pi:sta Cosmos DB -tietokannan säiliöön. Yksityiskohtaisempi kuvaus tässä läpikäydyistä vaiheista löytyy liitteestä A.4.

4.3.6 Google Cloud Platform

Tutkimuksen viimeinen pilvialusta on Google Cloud Platform (GCP), johon tutustuminen aloitetaan Google-haulla ”google cloud iot”. Haun ensimmäinen tulos on GCP:n IoT-ratkaisujen esittelysivu. Sivulla kerrotaan yleisellä tasolla, mitä GCP:llä voi tehdä IoT:n näkökulmasta, mutta suoria linkkejä työkalujen esittelysivuille ei ole. Työkaluista kuitenkin mainitaan esimerkiksi Cloud IoT Core, jolla IoT-laitteita ja niiden yhteyksiä hallitaan. Google-haun toinen tulos on Cloud IoT Coren esittelysivu, jossa kerrotaan esimerkiksi palvelun ominaisuuksista ja integraatiomahdollisuuksista muihin GCP:n palveluihin. Sivun ylälaidassa on painikkeet ja linkit ilmaisen kokeilujakson aktivoinnin lisäksi niin kehittäjädokumentaatioon kuin pika-aloitustutoriaaliinkin. Ilmaisen kokeilujakson aktivointipainike vie kirjautumisen ja kokeilujakson aktivoinnin kautta GCP:n web-konsoliin, josta pilvialustan resursseja voidaan hallita. Kehittäjädokumentaation linkki ohjaa puolestaan sivulle, joka tarjoaa yleiskuvan dokumentaation sisällöstä ryhmiteltyinä linkkilistoina, ja pika-aloitustutoriaalipainike vie IoT Coren käyttöä ja toimintaa esittelevään tutoriaaliin.

Tarkastellaan seuraavaksi pika-aloitustutoriaalia tarkemmin. Tutoriaalissa luodaan laiterekisteri Cloud IoT Coreen ja rekisteröidään sinne IoT-laite (“Quickstart | Cloud IoT Core Documentation | Google Cloud” 2021). Nämä tehdään käyttämällä web-konsolin käyttöliittymää, jonka etusivu näkyy kuviossa 10. Laitteen autentikaatiota varten luodaan epäsymmetrinen avainpari, ja laitteelle (tai sitä simuloivalle tietokoneelle) asennetaan esimerkkiohjelma, joka lähettää generoituja telemetriaviestejä IoT Coreen. Pilveen saapuneita viestejä luetaan GCP:n `gcloud`-komentorivityökalulla, jota voi käyttää myös web-konsolista löytyvän *Google Cloud Shell* -komentorivikäyttöliittymän kautta.

Pika-aloitustutoriaalin lisäksi kehittäjädokumentaatiossa on oppaita IoT Coren tarjoamien ominaisuuksien käyttämiseen, muutama sivu korkeamman tason konsepteista sekä paljon koodikatkelmia esimerkiksi IoT Coren resurssien hallintaan SDK-pakettien avulla. Oppaat ovat lyhyitä tutoriaalinomaisia selostuksia tietyn IoT Coren toiminnon käyttämisestä. IoT

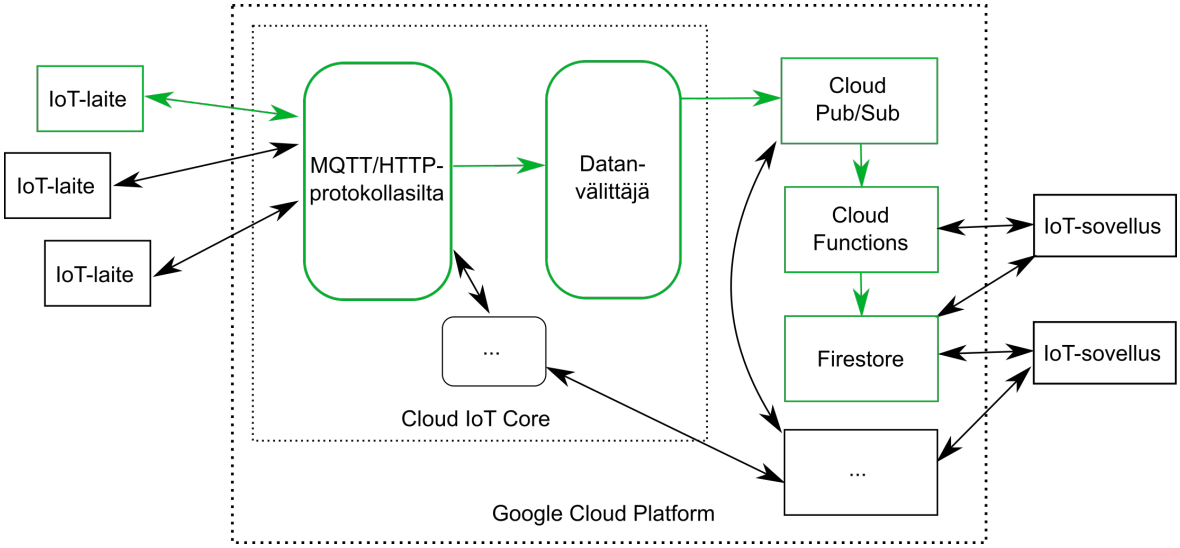


Kuvio 10. Google Cloud IoT Core -palvelun laiterekisterin web-käyttöliittymän etusivu kuvattuna 19.7.2021.

Coren dokumentaatio on tiivis, eikä sieltä löydy laajempia tutoriaaleja, joissa näytettäisiin, kuinka IoT Corea voidaan hyödyntää osana isompaa kokonaisuutta muiden GCP:n palveluiden kanssa. Dokumentaatiossa on kuitenkin paljon kehittäjälle tarpeellista tietoa. Tähän kuuluu esimerkiksi yksityiskohtaiset oppaat tuettujen protokollien — HTTP:n ja MQTT:n — käyttämisestä IoT-laitteiden ja IoT Coren välisessä kommunikaatiossa. Dokumentaatiosta selviää myös, että MQTT-liikenne on salattava TLS 1.2 -protokollalla ja autentikaatio tapahtuu allekirjoitetuilla JWT:illä (so. engl. *JSON Web Token*). JWT on tiivis formaatti esimerkiksi autentikaatiossa tarvittavien vaateiden kommunikointiin (Jones, Bradley ja Sakimura 2015). Vaateita voivat olla esimerkiksi pääsyoikeuden myöntämis- ja erääntymisajankohdat. Dokumentaatiossa on kerrottu koodiesimerkkien avulla, kuinka JWT luodaan ja allekirjoitetaan IoT-laitteella (“Using JSON Web Tokens (JWTs) | Cloud IoT Core Documentation” 2021).

Kehittäjädokumentaation konseptiosiossa esitellään Cloud IoT Coren toimintaperiaatetta esimerkiksi ydinkomponenttien osalta (“Cloud IoT Core Overview | Cloud IoT Core Documentation | Google Cloud” 2021). Kuviossa 11 näytetään telemetriadataa keräävien järjes-

telmien näkökulmasta keskeisimmät IoT Coren komponentit sekä muita GCP:n palveluita. IoT-laitteet kommunikoivat yleiskäyttöisillä MQTT- ja HTTP-asiakaskirjastoilla IoT Coren protokollasiltojen kanssa, sillä IoT Core ei tarjoa IoT-laitteille suunnattuja SDK-paketteja. IoT Coren MQTT-protokollasilta ei ole MQTT-standardin mukainen viestinvälittäjä, vaan se asettaa rajoitteita mm. MQTT-aiheiden ja client ID:n määrittelylle. MQTT-aiheen tulee olla esimerkiksi muodossa `/devices/{device_id}/events` telemetriaviestien julkaisussa. Tässä `{device_id}` on IoT-laitteen tunniste, joka on rekisteröity IoT Coreen.



Kuvio 11. Google Cloud IoT Core -palvelun toimintamalli mukaillen “Cloud IoT Core Overview | Cloud IoT Core Documentation | Google Cloud” (2021) -sivun kuvaa sekä muuta GCP:n dokumentaatiosta kerättyä tietoa. Kuviossa on mukana keskeisimmät toiminnot, joita tarvitaan telemetriadataa keräävässä järjestelmässä. Tutkimuksen IoT-prototyypissä käytetyt osat on korostettu vihreällä.

Protokollasillat välittävät laitteiden lähettämät telemetriaviestit datanvälittäjälle, joka puolestaan ohjaa ne *Cloud Pub/Sub* -palvelun aiheisiin (ks. kuvio 11). Cloud Pub/Sub on GCP:n palvelu, joka mahdollistaa asynkroonisen julkaise/tilaa-tyyppisen kommunikaation palveluiden välillä (“What Is Pub/Sub?” 2021). Pub/Subista viestit haetaan *Cloud Functions* -pilvifunktiolla, joka tallentaa ne *Firestore*-tietokantaan. Cloud Functions -palvelua ja Cloud Pub/Subia käsitellään myöhemmin tässä luvussa tarkemmin.

Firestore on dokumenttipohjainen NoSQL-tietokantapalvelu, jota suositellaan GCP:ssä esi-

merkiksi vanhemman *Datastore*n sijaan uusiin projekteihin (“Choosing between Native Mode and Datastore Mode | Firestore” 2021). Firestoressa tietorakenne on jaettu kokoelmiin (engl. *Collection*), jotka sisältävät dokumentteja, jotka puolestaan koostuvat avain-arvo-pareista (“Data Model | Firestore | Google Cloud” 2021). Jokaisella dokumentilla on nimi, joka toimii sen tunnisteena kokoelman sisällä. Toisin sanoen kokoelmassa ei voi olla kahta dokumenttia samalla nimellä. Dokumenttien sisältämät arvot voivat olla tietotyypiltään yksinkertaisten boolean-, aikaleima-, lukuarvo- ja merkkijonotyyppien lisäksi esimerkiksi taulukoita tai sisäkkäisiä kartoiksi (engl. *map*) kutsuttuja olioita. Kartta voi itsessään sisältää avain-arvo-pareja. Dokumentit voivat olla korkeintaan yhden megatavun kokoisia, mutta ne voivat sisältää myös alikokoelmia, joita ei lasketa kokorajoitukseen. Alikokoelmiin voi lisätä dokumentteja tavallisten kokoelmien tavoin, mikä mahdollistaa laajojenkin hierarkioiden luomisen.

Firestore indeksoi automaattisesti suuren osan dokumenttien kentistä yhden kentän indekseillä (engl. *single-field index*). Näiden lisäksi kokoelmiin voi määrittää yhdistelmäindeksejä (engl. *composite index*), jotka mahdollistavat monimutkaisempien kyselyiden tekemisen. Firestore takaa, että kyselyt suoritetaan tehokkaasti käyttämällä aina olemassa olevia indeksejä, eikä se suorita koskaan dokumenttikokoelmien täyttä skannausta (“Managing Indexes | Firestore | Google Cloud” 2021). Jos sopivaa indeksiä ei ole saatavilla, Firestore antaa virheilmoituksen ja linkin indeksin luomiseksi.

Telemetriaviestit kuljetetaan tässä tutkimuksessa Cloud IoT Coresta Firestore-tietokantaan Cloud Pub/Subin ja Cloud Functions -funktioiden kautta. Cloud Pub/Subissa on MQTT:n tavoin julkaisijoita, tilaajia ja aiheita (“What Is Pub/Sub?” 2021). Julkaisijat lähettävät viestejä aiheisiin ja tilaajat voivat tilata näitä vastaanottaakseen itselleen tarpeellisia viestejä. Tiivistäen tämä palvelu toimii GCP:n viestialustana, jolla voidaan esimerkiksi integroida palveluita toisiinsa tai muodostaa viestijonoja rinnakkaislaskentaa varten. Cloud IoT Core tukee kirjoitushetkellä vain Pub/Subia telemetriaviestien jatkotoimituksessa.

Cloud Functions on palvelimeton suoritusympäristö, joka mahdollistaa ohjelmalogiikan — pilvifunktioiden — suorituksen ennalta määrätyn tapahtuman käynnistämänä (“Cloud Functions Overview | Cloud Functions Documentation” 2021). Palvelimeton suoritusympäristö tarkoittaa tässä palveluna hallinnoitua ohjelmakoodin suoritusympäristöä, jossa kehittäjän

ei tarvitse ylläpitää esimerkiksi palvelimia tai ajoympäristöjä (vrt. PaaS, ks. luku 3.2.2). Tätä voidaan kutsua myös FaaS-malliksi (so. engl. *Functions as a Service*). Cloud Functions -palveluun kirjoitetun pilvifunktion voi käynnistää Pub/Subiin tulevan viestin seurauksena. Viestin sisältöä voi käsitellä funktiossa ja se voidaan tallentaa esimerkiksi Firestore-tietokantaan uutena dokumenttina.

4.3.7 Raspberry Pi:n yhdistäminen GCP:hen

Katsotaan seuraavaksi tutkimuksen IoT-prototyypin näkökulmasta, miten Raspberry Pi:n voi yhdistää Google Cloud Platform -pilvialustaan. Ensimmäiseksi Cloud IoT Coreen lisätään uusi rekisteri ja liitetään siihen uusi Cloud Pub/Sub -aihe, johon IoT Core myöhemmin reitittää viestit. Aiheen voi luoda rekisterin lisäyksen yhteydessä. Tässä aiheen tunnisteeksi annetaan `weather`, jolloin sen koko nimeksi tulee `projects/{project_id}/topics/weather`. Tässä `{project_id}` on GCP-projektin tunniste. GCP:ssä kaikki luodut resurssit, kuten Cloud IoT Core -rekisteri, kuuluvat projektiin, jolla on yksilöllinen tunniste. Raspberry Pi lisätään rekisteriin uutena laitteena ja sille generoidaan avainpari IoT Coren dokumentaation ohjeiden mukaisesti ("Creating Public/Private Key Pairs | Cloud IoT Core Documentation" 2021). IoT Core ei tarjoa automaattista avainten generointia web-konsolin käyttöliittymän kautta, vaan se pitää tehdä itse esimerkiksi `openssl`-komentorivityökalulla. Generoidun avainparin julkinen avain lisätään IoT Coreen rekisteröidylle laitteelle. Raspberry Pi:lle pitää lisäksi ladata Googlen juuri-CA-sertifikaattipaketti (engl. *root CA certification package*) viestiliikenteen TLS-salausta varten.

Raspberry Pi:n rekisteröinnin jälkeen se voidaan yhdistää IoT Coreen käyttämällä esimerkiksi "Eclipse Paho" (2021) MQTT-asiakaskirjastoa Python-ohjelmointikielelle. IoT Coren dokumentaatiosta löytyy "Publishing over the MQTT Bridge | Cloud IoT Core Documentation" (2021) -sivu, jolla on yksityiskohtaiset ohjeet kommunikaatioon MQTT:llä. Ohjeeseen upotetut koodikatkelmat on saatavilla usealla ohjelmointikielellä, joista yksi on Python. Koodikatkelmien yhteydessä on myös linkit esimerkkien GitHub-sivulle, josta löytyy yksityiskohtaisemmin muun muassa tarvittavat PyPI-kirjastopaketit. IoT-prototyypin datankeruuohjelmaa varten asennetaan kolme PyPI-pakettia. Paketit ovat `pyjwt`, `cryptography` ja `paho-mqtt`. Näistä kahta ensimmäistä käytetään apuna JWT:n muodostuksessa, ja vii-

meisen avulla hoidetaan MQTT-liikenne.

Kun PyPI-paketit on asennettu lisätään datankeruuohjelmaan tarvittavat importit. Sisällytettävät kirjastot ovat `paho.mqtt.client`, `ssl` ja `jwt`. Lisätään uusi `GCPClient`-niminen luokka GCP:n kommunikointilogiikalle. Luokan konstruktorissa alustetaan esimerkiksi MQTT-aihe ja -asiakastunniste. Tässä luodaan myös uusi MQTT-asiakasolio Paho-kirjaston avulla ja asetetaan sille IoT Coren vaatimat TLS-salausparametrit. Seuraavassa koodikatkelmassa on esitetty kirjastojen importit ja uuden luokan konstruktori. Epäolennaisia rivejä on jätetty pois kohdista, joissa on ”# ...”-kommentti.

```
import ssl
# API: https://eclipse.org/paho/clients/python/docs/
import paho.mqtt.client as paho_mqtt
import jwt

# ...
class GCPClient():
    """Class for all Google Cloud Platform specific functionality."""
    def __init__(self, args):
        # ...
        self.mqtt_topic = "/devices/{}/events".format(self.device_id)
        self.client_id =
           "projects/{}/locations/{}/registries/{}/devices/{}".format(
            self.project_id, self.cloud_region,
            self.registry_id, self.device_id
        )
        self.client = paho_mqtt.Client(client_id=self.client_id)

        # Enable SSL/TLS support.
        self.client.tls_set(
            ca_certs=self.ca_certs, tls_version=ssl.PROTOCOL_TLSv1_2)
```

MQTT-yhteyden muodostus tapahtuu `connect()`-metodissa. Tätä varten luodaan ensin JWT ja asetetaan se MQTT-asiakkaan käyttämäksi salasanaksi. IoT Core ei käytä käyttäjänimeä, joten sen sisällöllä ei ole merkitystä. Yhteyden muodostus tehdään MQTT-asiakasolion `connect(hostname, port)`-metodilla. Tämän lisäksi `loop_start()`-metodia kut-

sutaan samassa yhteydessä. Tämä käynnistää taustalle verkkoliikenteestä huolehtivan säikeen, mikä vapauttaa ohjelman pääsäikeen muuhun käyttöön. Yhteyden katkaisu hoidetaan luokan `disconnect()`-metodissa. Tässä kutsutaan MQTT-asiakkaan vastaavan metodin lisäksi `loop_stop()`-metodia, joka pysäyttää verkkoliikenteestä huolehtivan säikeen. Viestit lähetetään `GCPClient`-luokan `send_message(message)`-metodilla. Tämä tarkistaa aluksi, onko JWT vanhentunut ja tarvittaessa uusii sen muodostamalla yhteyden uudelleen. Lopuksi metodi julkaisee viestin MQTT-aiheeseen, joka määritettiin luokan konstruktorissa. Kuvaillut metodit on esitetty seuraavassa koodikatkelmassa.

```
    def connect(self):
        print("Connecting to host {}:{} with client ID '{}'..."
            .format(self.hostname, self.port, self.client_id))

        # With Google Cloud IoT Core, the username field is ignored,
        # and the password field is used to transmit a JWT to
        # authorize the device.
        self.client.username_pw_set(
            username="unused", password=self.create_jwt()
        )

        # Connect to the Google MQTT bridge.
        self.client.connect(self.hostname, self.port)
        self.client.loop_start()


    def disconnect(self):
        print("Disconnecting from GCP...")
        self.client.disconnect()
        self.client.loop_stop()


    def send_message(self, message):
        self.refresh_jwt_if_necessary()

        print("Sending message to GCP: {}".format(message))
        self.client.publish(self.mqtt_topic, message, qos=1)
```

Autentikaatioon tarvittava JWT luodaan `create_jwt()`-metodissa, jossa määritellään ensimmäiseksi IoT Coren vaatimat JWT-vaateet. JWT:n `iat`-vaateeseen asetetaan sen luontiajankohta sekunteina ajanhetkestä 1.1.1970 kello 00.00.00 UTC-aikavyöhykkeellä ("Using JSON Web Tokens (JWTs) | Cloud IoT Core Documentation" 2021). Erääntymisajankohta asetetaan `exp`-vaateeseen vastaavalla aikaleimalla, joka lasketaan datankeruuohjelmassa oletuksena 20 minuuttia tulevaisuuteen. Aikaleimojen muodostus onnistuu Pythonin *datetime*-kirjastolla. Viimeinen pakollinen vaade on `aud`, johon asetetaan GCP-projektin tunniste. Vaateet asetetaan Pythonin dict-hakurakenteeseen, joka annetaan parametrina `jwt`-kirjaston `encode(...)`-metodille. Tämä metodi tarvitsee myös aiemmin luodun avainparin yksityisen avaimen, jotta se voi allekirjoittaa JWT:n. Avain luetaan tiedostosta, jonka polku on annettu datankeruuohjelmalle argumenttina. JWT:n luonti on esitetty seuraavassa koodikatkelmassa.

```
    ### Internal methods ###

    # ...
    def create_jwt(self):
        """Creates a JWT (https://jwt.io) to establish an MQTT
        connection.
        """

        token = {
            # The time that the token was issued at
            "iat": datetime.utcnow(),
            # The time the token expires.
            "exp": datetime.utcnow() +
                timedelta(minutes=self.jwt_exp_minutes),
            # The audience field should always be set to the GCP
            # project id.
            "aud": self.project_id,
        }
        self.jwt_iat = token["iat"]

        # Read the private key file.
        with open(self.private_key_file, "r") as f:
            private_key = f.read()
```

```
            return jwt.encode(
                token, private_key, algorithm=self.algorithm)
```

Kun data kulkee Cloud IoT Coreen, siirrytään luomaan Firestore-tietokantaa. Ensimmäisellä Firestore-palvelun avauskerralla web-konsoli pyytää valitsemaan Firestoren toimintamoodiksi joko natiivin tai Datastore-moodin. Datastore-moodi jäljittelee vanhan Cloud Datastore -tietokannan toimintaa, kun taas natiivi moodi mahdollistaa kaikkien Firestoren ominaisuuksien käytön. Tässä valitaan natiivi moodi. Tietokannalle valitaan myös maantieteellinen sijainti. Samassa GCP-projektissa voi olla kerrallaan vain yksi Firestore-tietokanta, jonka toimintamoodia tai maantieteellistä sijaintia ei voi vaihtaa valinnan jälkeen. Kun tietokanta on luotu, sinne voi tallentaa suoraan dokumentteja haluttuihin kokoelmiin. Kokoelmia tai dokumentteja ei tarvitse erikseen luoda, jotta niihin voisi tallentaa dataa, vaan ne luodaan aina tarvittaessa automaattisesti.

Firestore-tietokannan aktivoinnin jälkeen siirrytään kirjoittamaan pilvifunktiota Cloud Functions -palveluun. Pilvifunktion käynnistimeksi asetetaan Cloud Pub/Sub -aiheeseen tulevat viestit. Aiheeksi valitaan sama `projects/{project_id}/topics/weather`-muotoinen aihe, joka luotiin IoT Core -rekisterin luonnin yhteydessä. Pilvifunktion ajoympäristöksi asetetaan tässä Python 3.9, joka on kirjoitushetkellä tuorein tuettu Python-versio. Funktion riippuvuuksiin lisätään `google-cloud-firestore` PyPI-paketti, jonka avulla voidaan tallentaa dataa Firestore-tietokantaan. Funktion ohjelmatiedoston ajettavaa funktiota muokataan siten, että Pub/Sub-palvelun kautta tulevan viestin sisältö dekoodataan Base64-formaatista ja tallennetaan Firestore-tietokantaan. Tallennus tehdään `weather`-nimiseen kokoelmaan uuteen dokumenttiin, jonka nimeksi asetetaan viestistä löytyvät `device_id`- ja `id`-arvot liimattuna yhdeksi merkkijonoksi. Näistä arvoista kerrotaan tarkemmin luvussa 4.2.2. Seuraavassa koodilistauksessa on pilvifunktion koko ohjelmakoodi.

```
import base64
import json
from google.cloud import firestore

def save_weather(event, context):
    """Triggered from a weather message on a Cloud Pub/Sub topic.
```

```
    Args:
        event (dict): Event payload.
        context (google.cloud.functions.Context): Metadata for the
            event.
    """
    pubsub_message = base64.b64decode(event['data']).decode('utf-8')
    print("Inserting Pub/Sub message to Firestore.", pubsub_message)
    weather = json.loads(pubsub_message)

    db = firestore.Client()
    doc_ref = db.collection(u'weather').document(
        str(weather["device_id"]) + str(weather["id"]))
    doc_ref.set(weather)
```

Kun pilvifunktion ottaa käyttöön ja ajaa tämän jälkeen datankeruuohjelman, säädata kulkeutuu Raspberry Pi:sta Firestore-tietokannan `weather`-kokoelmaan. Yksityiskohtaisempi kuvaus tämän luvun vaiheista löytyy liitteestä A.5.

5 Pilvialustojen vertailu

Tässä luvussa otetaan askel taaksepäin ja vertaillaan luvussa 4 läpikäytyjä julkisia pilvialustoja toisiinsa. Luvussa vastataan toiseen tutkimuskysymykseen: Kuinka keskeisimmät julkiset pilvialustat eroavat toisistaan käytännön kehitystyössä, kun kyseessä on IoT-dataa keräävä järjestelmä? Telemetriadataa keräävän IoT-laitteen liittäminen kuhunkin pilvialustaan onnistuu suhteellisen suoraviivaisesti. Eroja kuitenkin löytyy niin työkaluista kuin omaksuttavuudestakin esimerkiksi kehittäjädokumentaation osalta. Seuraavissa aliluvuissa käydään löytyneitä eroja ja yhtäläisyyksiä läpi. Luvussa 5.1 vertaillaan pilvialustojen omaksuttavuutta ensivaikutelman, tutoriaalien ja kehittäjädokumentaation osalta. Luvussa 5.2 puolestaan vertaillaan pilvialustojen tarjoamia työkaluja, joita tutkimuksen IoT-prototyypissä käytetään. Nämä voidaan jakaa IoT-laitteiden hallintatyökaluihin, tietokantoihin sekä nämä yhdistäviin dataväyläpalveluihin.

5.1 Omaksuttavuus

Pilvialustan hyvä omaksuttavuus auttaa erityisesti uusia pilvikehittäjiä oppimaan alustan käyttöä ja toimintamalleja. Hyvällä omaksuttavuudella tässä tarkoitetaan esimerkiksi käyttöliittymän selkeyttä, alustan tarjoamien palveluiden käyttöönoton suoraviivaisuutta sekä ohjeistusta, jonka avulla kehittäjää opetetaan käyttämään näitä palveluita. Tarkastellaan seuraavaksi tutkittavien pilvialustojen omaksuttavuutta ensivaikutelman, tutoriaalien ja dokumentaation osalta.

5.1.1 Ensivaikutelma ja tutoriaalit

Pilvialustoihin tutustuminen aloitettiin hakemalla niiden IoT-palveluita Google-hakukoneella. Kaikilla pilvialustoilla ensimmäinen hakutulos on IoT-ratkaisuista kertova sivu, jolla kerrotaan korkealla tasolla ja esimerkkienkin avulla, millaisia järjestelmiä alustalla on mahdollista rakentaa. Kaikilla alustoilla on myös erilliset sivut tutkimuksessa käytetyille IoT-palveluille. Näillä sivuilla kerrotaan teknisemmästä näkökulmasta palveluiden tarjoamista ominaisuuksista. IoT-ratkaisujen ja IoT-palveluiden esittelysivujen kautta syntyneissä ensi-

vaikutelmissa ei ole merkittäviä eroja. Vaihtelua esiintyy luonnollisesti esittelysivujen ulkoasussa ja esitystavassa.

Kaikki tutkitut pilvialustat tarjoavat IoT-palveluilleen pika-aloitustutoriaalit, joiden avulla alkuun pääseminen helpottuu. AWS:ssä pika-aloitustutoriaali on osa laajempaa käyttöönotto-ohjeistusta, jota käydään läpi luvussa 4.3.2. Ohjeistus on yksityiskohtainen ja se tarjoaa uudelle kehittäjälle myös korkeamman tason tietoa IoT-palvelun konsepteista ja toiminnasta. AWS:n pika-aloitustutoriaalissa IoT-laitteella sekä lähetetään viestejä pilveen että vastaanotetaan niitä pilvestä. Azure tarjoaa puolestaan useita pika-aloitustutoriaaleja, joissa tutustutaan IoT Hubin ominaisuuksiin erillisinä kokonaisuuksina. Tutkimuksessa käydään näistä läpi vain telemetrian lähetystä käsittelevä tutoriaali (ks. luku 4.3.4). Tämän lisäksi saatavilla olisi esimerkiksi tutoriaali, jossa tutustutaan IoT-laitteen ohjaukseen pilvestä käsin. GCP:n tarjonta pika-aloitustutoriaalien osalta on suppein. Se tarjoaa Cloud IoT Corelle vain yhden pika-aloitustutoriaalin (ks. luku 4.3.6), jossa käsitellään ainoastaan viestien lähetystä IoT-laitteelta pilveen, mutta ei toiseen suuntaan. Tästä huolimatta tutkimuksen aiheen kannalta kaikki pilvialustat tarjoavat lähes yhdenvertaiset pika-aloitustutoriaalit. Näin siksi, koska datan lähetystä IoT-laitteelta pilvialustalle käsitellään kaikilla alustoilla samankaltaisesti.

Pika-aloitustutoriaalien lisäksi kaikki alustat tarjoavat IoT-palveluidensa dokumentaatioissa laajempia tutoriaaleja tai muuta tutoriaalinomaista ohjeistusta. AWS:ltä löytyy useita laajempia tutoriaaleja, joissa ohjeistetaan esimerkiksi, kuinka IoT Coren voi liittää muihin AWS:n palveluihin. Yhdessä näistä tutoriaaleista dataa viedään suoraan DynamoDB-tietokantaan, mikä vastaa tutkimuksen tarpeita hyvin (“Store Device Data in a DynamoDB Table - AWS IoT Core” 2021). Myös Azure tarjoaa useita laajempia tutoriaaleja ja ohjeistusta eri aiheista. Tutkimuksen kannalta kiinnostava datan reititystä käsittelevä tutoriaali on laaja, ja siinä on useita esimerkkejä datan viemiseksi IoT Hubista muihin pilvialustan palveluihin (“Tutorial - Configure Message Routing for Azure IoT Hub Using Azure CLI” 2021). Suurin osa näistä esimerkeistä käyttää reitityksessä dataväyläpalveluita viestien kuljetukseen palveluiden välillä. Tutoriaali ei kuitenkaan tarjoa suoraa ratkaisua datan viemiseksi Cosmos DB -tietokantaan, jota tutkimuksessa käytetään. Tästä sekä dataväyläpalveluiden runsaudesta johtuen tutustumisprosessissa kului paljon aikaa eri palveluiden kokeilemiseen ja niihin tutustumiseen. Dataväylinä toimivia palveluita käsitellään tarkemmin luvussa 5.2.3. GCP ei

tarjoa Cloud IoT Coren dokumentaatiossa pika-aloitustutoriaalin lisäksi muita tutoriaaleja. Toisin sanoen sieltä puuttuu esimerkiksi aiempia vastaava tutoriaali, jossa käytäisiin läpi esimerkkien kautta, kuinka IoT Corea voi hyödyntää yhdessä muiden GCP:n palveluiden kanssa. Toisaalta kirjoitushetkellä Cloud IoT Coressa on niukasti toimintoja ja esimerkiksi viestit viedään aina Pub/Subiin, joten tällainen tutoriaali ei välttämättä toisi tarpeeksi lisäarvoa. Lisäksi IoT Coren dokumentaatiosta löytyy kattavat ohjeistukset esimerkiksi MQTT:n käyttämisestä asiakaskirjastoilla useilla eri ohjelmointikielillä. Tämän ja pika-aloitustutoriaalin avulla datan kuljetus Pub/Subiin asti onnistuu suoraviivaisesti. Tältä osin ohjeistus on tutkimuksen tarpeisiin riittävä, vaikka esimerkit datan viemiseksi Firestore-tietokantaan puuttuvatkin.

5.1.2 Dokumentaatio

Tarkastellaan seuraavaksi pilvialustojen IoT-palveluiden kehittäjädokumentaatioita hieman yleisemmin. Dokumentaatioiden yleiset painotukset eroavat alustojen välillä. GCP keskittyy Cloud IoT Coren dokumentaatiossa (“Cloud IoT Core Overview | Cloud IoT Core Documentation | Google Cloud” 2021) kuvaamaan pääasiassa tekniseltä kannalta, kuinka palvelua käytetään. Lisäksi joukosta löytyy muutamia sivuja, joilla palvelun käyttötarkoitusta, toimintaa ja hyviä käytänteitä kuvataan yleisemmällä tasolla. Azuren ja AWS:n IoT-palveluiden dokumentaatiot ovat huomattavasti edellistä laajempia. AWS:n IoT Coren dokumentaatiossa (“What Is AWS IoT? - AWS IoT Core” 2021) on runsaasti tutoriaaleja ja käytännön ohjeita koodikatkelmineen, mutta myös paljon yleisemmän tason konsepteja ja teoriaa sen toiminnasta yhdessä pilvialustan muiden palveluiden kanssa. Azuren IoT Hubin dokumentaatiossa (“Azure IoT Hub Documentation” 2021) on tutoriaalien ja käytännön ohjeiden sekä yleisemmän tason konseptien ja teorian lisäksi informaatiota IoT-laitteisiin ja -alaan yleisemmin liittyvistä käytännöistä. Esimerkiksi “Concepts of Azure IoT Hub X.509 Security” (2021) -sivulla kerrotaan, kuinka X.509 CA-sertifikaatteja voidaan hyödyntää teollisessa skaalassa IoT-laitteiden tietoturvallisessa valmistuksessa ja käyttöönotossa.

IoT-palveluiden dokumentaatioiden jäsennyksessä on myös eroja. GCP:n Cloud IoT Coren dokumentaatio on yleisilmeeltään selkeä. Se on jäsennetty neljään ryhmään — yleiset, koodiesimerkit, oppaat ja konseptit. Ryhmäjaon lisäksi selkeyteen vaikuttaa sivujen vähäinen

määrä. Myös Azuren IoT Hubin dokumentaatiossa ryhmittelyyn on panostettu. Ylimmän tason ryhmiä on dokumentaatiossa seitsemän — yleiskatsaus, pika-aloitus, tutoriaalit, konseptit, oppaat, työkalujen API-referenssit ja muut resurssit, kuten ulkopuoliset sivustot. Ryhmien sisällä sivut vaikuttavat välillä olevan satunnaisessa järjestyksessä, eikä niitä ole esimerkiksi aakkostettu tai muuten järjestetty loogiseen etenemisjärjestykseen. Tämä voi vaikeuttaa jonkin verran oikeiden sivujen löytymistä. Tähän voi vaikuttaa lisäksi sivujen suuri lukumäärä, vaikka sivuhierarkia on pääosin hyvin jäsennetty. AWS:n IoT Coren dokumentaatio on puolestaan jäsennetty edellisiä heikommin. Vaikka sivut on pääasiassa ryhmitelty aiheittain, osa näistä on kapea-alaisia, minkä takia dokumentaation sivuhierarkian korkeimmalle tasolle tulee suuri määrä sivuryhmiä. Tämä saattaa vaikeuttaa oikeiden sivujen löytämistä erityisesti uusien kehittäjien kohdalla. Esimerkiksi ”IoT-laitteiden hallinta” -sivuryhmä on samalla tasolla ”AWS IoT tutoriaalit” -ryhmän, ”Tietoturva”-ryhmän ja IoT-säännöistä kertovan sivuryhmän kanssa.

Kaikki tutkittavat pilvialustat tarjoavat laajoja dokumentaatioita kaikille muillekin tutkimuksessa käytetyille palveluille. Niihin ei kuitenkaan tutustuttu IoT-prototyyppiä tehdessä yhtä syvällisesti kuin IoT-palveluiden dokumentaatioihin, eikä niitä käydä tässä läpi, koska tutkimuksen pääpaino on IoT-järjestelmissä.

5.2 Työkalut

Tutustutaan seuraavaksi pilvialustojen tarjoamien työkalujen eroihin ja yhtäläisyyksiin. Työkaluilla tässä tarkoitetaan niin PaaS-mallisten pilvialustojen palveluita kuin IoT-laitteille suunnattuja SDK-paketteja, joiden avulla pilvialustojen palveluiden kanssa kommunikoidaan. Tässä keskitytään IoT-prototyyppiä tehdessä vastaan tulleisiin ja tutkimuskysymysten kannalta olennaisiin seikkoihin. Näitä ovat esimerkiksi Python-ohjelmointikielelle saatavilla olevat SDK-paketit, koska IoT-prototyypissä käytetään Pythonia. Toinen esimerkki tässä luvussa käsiteltävistä aiheista on MQTT-protokolla, jota IoT-prototyypissä myös käytetään. Katsotaan seuraavaksi eroja ensin IoT-laitteiden hallinnan ja SDK-pakettien osalta luvussa 5.2.1, sitten dokumenttipohjaisten NoSQL-tietokantojen osalta luvussa 5.2.2 ja viimeiseksi nämä yhdistävien dataväylinä toimiven palveluiden osalta luvussa 5.2.3.

5.2.1 IoT-laitteiden hallinta ja SDK:t

IoT-palvelun käyttöönotto lähtee jokaisella pilvialustalla liikkeelle IoT-laitteiden rekisteröinnistä. Tähän pilvialustoilla on hyvin samantyyppinen lähestymistapa. Kaikilta löytyy laiterekisteri, johon laitteet rekisteröidään. Lisäksi rekistereissä hallitaan esimerkiksi laitteiden autentikaatioavaimia. Autentikaatioon palataan myöhemmin tässä luvussa.

Kaikki tutkitut alustat tukevat MQTT-protokollaa viestintämenetelmänä IoT-laitteiden kanssa. Tarjottu tuki MQTT:n ominaisuuksille kuitenkin eroaa huomattavasti alustojen välillä. AWS IoT Coren MQTT-toteutus on lähes standardin mukainen yleiskäyttöinen MQTT-viestinvälittäjä (ks. MQTT luvussa 3.3.1). AWS tarjoaa esimerkiksi Pythonille IoT Device SDK -paketin, joka on yleiskäyttöinen MQTT-asiakaskirjasto. Tämän avulla kehittäjä voi käyttää suurinta osaa MQTT-standardin ominaisuuksista, kuten itse määriteltyjä rakenteisia MQTT-aiheita. IoT-laitteet voivat myös kommunikoida keskenään MQTT-viestinvälittäjän kautta protokollan julkaisu- ja tilausominaisuuksilla.

Azuren ja GCP:n lähestymistavat MQTT-tukeen poikkeavat AWS:stä paljon. Azuren IoT Hubin MQTT-toteutus ei ole yleiskäyttöinen MQTT-viestinvälittäjä, vaan se rajaa tarkkaan aiheiden formaatin ja viestin rakenteen. Azure tarjoaa IoT-laitteille myös SDK-paketin, mutta sen rajapinta ei paljasta kehittäjälle MQTT:n yksityiskohtia, kuten viestien julkaisussa ja tilauksessa tarvittavia MQTT-aiheita. Hyvänä puolena tässä on, että kehittäjän ei tarvitse huolehtia juuri mistään MQTT-protokollan vaatimista määrittelyistä. Toisaalta kehittäjä ei voi hyödyntää vapaasti MQTT-protokollan ominaisuuksia, kuten MQTT-aiheen rakennetta viestien julkaisussa ja tilaamisessa. Azuren IoT Hubin kanssa voi kommunikoida myös yleiskäyttöisellä MQTT-kirjastolla, mutta MQTT-aiheet ja viestien rakenne pitää määritellä Azuren vaatimalla tavalla. GCP:n Cloud IoT Corenkaan MQTT-toteutus ei ole yleiskäyttöinen viestinvälittäjä, eikä se salli esimerkiksi MQTT-asiakkaiden kommunikointia suoraan keskenään. Se rajaa myös MQTT-aiheiden formaatin tarkkaan. Viestien rakenne on kuitenkin vapaampi kuin Azuressa. GCP ei tarjoa SDK:ta IoT-laitteille, vaan ohjeistaa käyttämään kolmannen osapuolen yleiskäyttöisiä MQTT-asiakaskirjastoja. Mikään kolmesta pilvialustasta ei tarjoa IoT-palvelussaan täysin standardin mukaista MQTT-tukea. Esimerkiksi QoS-tasoa 2 ei voi käyttää niistä minkään kanssa.

Tietoturvaominaisuuksien osalta pilvialustojen IoT-palveluista löytyy sekä yhtäläisyyksiä että eroja. Kaikki IoT-palvelut tukevat viestiliikenteen salauksessa TLS v1.2 -protokollaa. AWS:ssä ja GCP:ssä sen käyttö on pakollista. Azuressa v1.2:n lisäksi voi käyttää toistaiseksi versioita 1.0 ja 1.1, mutta Azure suosittelee siirtymään v1.2:een, koska vanhempien versioiden tuki poistuu todennäköisesti lähitulevaisuudessa ("Azure IoT Hub TLS Support" 2021).

Autentikaatiossa eroa alustojen välillä on tietoliikenteen salausta enemmän. AWS vaatii MQTT-liikenteessä pääasiassa X.509-sertifikaatteihin pohjautuvaa autentikaatiota ("Client Authentication - AWS IoT Core" 2021). Jos MQTT:tä käyttää web-sokettien yli, käytetään autentikaatiomenetelmänä AWS Signature Version 4:ää. Myös Azure tukee X.509-sertifikaatteja IoT-laitteiden autentikoinnissa ("Control Access to IoT Hub Using SAS Tokens" 2021). Oletuksena kuitenkin IoT-laitteiden rekisteröinnin yhteydessä tarjotaan SAS-tunnisteita (so. engl. *Shared Access Signature*). SAS-tunnisteet allekirjoitetaan tutkimuksessa käsitellyssä yhden laitteen kommunikaatiossa IoT-laitteelle generoidulla yksilöllisellä symmetrisellä avaimella. Epäsymmetrisiin avaimiin, kuten X.509-sertifikaatteihin verrattuna symmetristen avainten salassapito on haastavampaa, sillä samaa avainta pitää säilyttää sekä laitteessa, että pilvialustan IoT-palvelussa. AWS:ssä ja Azuressa on lisäksi mahdollista implementoida tuki omalle autentikaatiomenetelmälle. Tästä voi alustojen mukaan olla hyötyä esimerkiksi vanhempien laitteiden migroinnissa pilvialustalle, ettei tarvitse generoida kaikille laitteille uusia avaimia. GCP:n Cloud IoT Core käyttää IoT-laitteiden autentikaatiomenetelmänä epäsymmetrisillä avaimilla allekirjoitettuja JWT-tunnisteita ("Device Security | Cloud IoT Core Documentation" 2021). Tässäkin julkinen avain voi olla muotoiltu X.509-sertifikaatiksi.

Kun laite on autentikaation kautta tunnistettu IoT-palveluun rekisteröidyksi laitteeksi, täytyy tarkistaa, onko laitteella valtuus (engl. *authorization*) suorittaa sen haluama toiminto. Tässä valtuutusta katsotaan IoT-laitteiden näkökulmasta, mutta sitä voisi lähestyä myös esimerkiksi IoT-palvelun osia hallinnoivien käyttäjien näkökulmasta. Käyttäjien valtuutusta ei käsitellä tässä tutkimuksessa. AWS IoT Core käyttää MQTT:llä yhdistävien IoT-laitteiden valtuutukseen IoT-käytäntöjä ("Authorization - AWS IoT Core" 2021). IoT-käytäntöjen avulla valtuudet voi määritellä yksityiskohtaisesti. IoT-laitteen voi esimerkiksi valtuuttaa julkaise-

maan viestejä vain tarkkaan määriteltyihin MQTT-aiheisiin tai suurempiin aihejoukkoihin. Sama pätee myös MQTT-aiheiden tilauksessa. Azuren ja GCP:n lähestymistapa poikkeaa AWS:stä. Kummankin alustan IoT-palveluissa IoT-laitteiden ja pilven välinen kommunikaatio voidaan joko sallia tai kieltää, mutta yksityiskohtaisempi valtuutus ei ole mahdollista. Toisaalta kummallakin alustalla MQTT-aiheiden määrittely on lähtökohtaisesti hyvin rajallista. Kukin IoT-laite voi julkaista viestejä vain laitteen omaan laitekohtaiseen MQTT-aiheeseen. Sama pätee pilvestä laitteelle suuntautuvassa viestinnässä, jossa laite voi tilata vain omaan laitekohtaiseen MQTT-aiheeseen tulevia viestejä.

Datan reititysvaihtoehdot IoT-palvelusta eteenpäin pilvialustan muihin palveluihin vaihtelevat alustojen välillä. GCP:n Cloud IoT Coressa suoria integraatiovaihtoehtoja on vähiten, ja kaikki viestit viedään Cloud IoT Coresta Cloud Pub/Sub -palvelun aiheisiin. Azuren IoT Hubissa on puolestaan suorat integraatiot Event Hubiin, Service Bus -jonoon ja -aiheeseen, BLOB-varastoon sekä IoT Hubin sisäänrakennettuun päätepisteeseen. BLOB-varastoa lukuunottamatta kaikki mainitut päätepisteet ovat Azuren yleiskäyttöisiä dataväyläpalveluita, joiden kautta viestejä voi välittää useisiin muihin palveluihin. Esimerkiksi Event Hubista ja IoT Hubin sisäänrakennetusta päätepisteestä voidaan lukea viestejä Azure Stream Analytics -palvelun avulla (ks. luvut 4.3.4 ja 4.3.5). AWS:n IoT Core tarjoaa tutkittavista pilvialustoista laajimman tarjonnan suoria integraatioita muihin pilvialustan palveluihin. Dataa voidaan viedä suoraan esimerkiksi Amazon DynamoDB -tietokantaan, AWS Lambda -funktioihin ja Amazon Simple Storage Service (S3) -tallennuspalveluihin.

5.2.2 Tietokannat

Suunnataan katse seuraavaksi pilvialustojen tarjoamiin dokumenttipohjaisiin NoSQL-tietokantoihin, joita tutkimuksen IoT-prototyypissä käytetään. Tässä vertailtavat tietokannat ovat Amazon DynamoDB AWS:ssä, Cosmos DB:n SQL API Azuressa ja Firestore GCP:ssä. Tässä luvussa Cosmos DB:n SQL API:in viitataan *Cosmos DB* -nimellä. Vertaillaan ensin tietokantojen käyttöönottoprosesseja. GCP:ssä käyttöönottoprosessi on yksinkertaisin. Firestoren ensimmäisellä käynnistyskerralla tarvitsee määrittää vain toimintamoodi natiivin ja Datastore-moodin välillä sekä valita maantieteellinen alue, johon tietokanta perustetaan. Kokoelmia tai dokumentteja ei tarvitse luoda erikseen, vaan ne luodaan automaattisesti aina

tarvittaessa. Azuressa puolestaan valitaan Cosmos DB:n tukemien API-rajapintojen välillä, kun luodaan tietokantatili. Lisäksi tietokannat ja säiliöt on luotava ennen tietoalkioiden tallennusta. Säiliölle pitää määrittää osiointiavaimet, jotka löytyvät tallennettavasta datasta. Myös AWS:n DynamoDB vaatii taulujen luonnin tietokantaan ennen tietoalkioiden tallennusta. Tauluille pitää määrittää osiointiavaimet sekä valinnaisesti järjestysavaimet, jotka löytyvät tallennettavasta datasta.

Tietokantapalveluiden ohjelmallisia kyselyrajapintoja ei tutkimuksen IoT-prototyypissä juuri käytetä, mutta käydään niitä tässä läpi lyhyesti. Kaikki tietokantapalvelut tarjoavat webkonsoleissaan graafiset käyttöliittymät, joilla tietokantoihin voi tehdä kyselyitä. Ne tarjoavat myös komentorivikäyttöliittymät (CLI) sekä SDK-paketit useille ohjelmointikielille. Näiden avulla tietokantojen käyttäminen onnistuu API-rajapintojen kautta. Azure Cosmos DB:n SQL API tukee suoria hakuja tietoalkioiden tunnisteiden perusteella sekä SQL-pohjaisia kyselyitä, joiden avulla voi hakea ja suodattaa dataa vapaammin. AWS:n DynamoDB tukee vastaavasti API-kutsujen kautta tehtäviä hakuja sekä SQL-kieltä jäljitteleviä PartiQL-kyselyitä, jotka vastaavat toiminnoiltaan Cosmos DB:n SQL-kyselyitä. GCP:n Firestoressa kyselyitä ei ole mahdollista tehdä SQL:llä, vaan ne tehdään erikoistuneiden API-rajapintojen avulla.

Datan indeksointi vaikuttaa hakunopeuksiin etenkin suurella datamäärällä. GCP:n Firestore ja Azuren Cosmos DB indeksoivat automaattisesti suuren osan avain-arvopareista ja ominaisuuksista yhden kentän indekseillä. Tämä mahdollistaa nopeat yksinkertaisia suodatuksia sisältävät kyselyt ilman lisämäärittelyitä. Molemmat tietokantapalvelut sallivat myös yhdistelmäindeksien luonnin monimutkaisemmille kyselyille. AWS:n DynamoDB ei luo toissijaisia indeksejä automaattisesti, vaan ne pitää luoda tarvittaessa itse myös yksinkertaisten suodatusten tehostamista varten.

Kaikissa tietokantapalveluissa piilee joitakin vaaranpaikkoja, jotka on hyvä huomioida. Näistä varoitetaan tietokantapalveluiden dokumentaatioissa, joista tässä käsiteltävät tiedot vaaranpaikoista on löydetty. Dokumentaatioiden kautta uudemmatkin kehittäjät voivat oppia huomioimaan ne. Ensimmäinen vaaranpaikka on huonosti valittu osiointiavain. AWS:n DynamoDB:ssä ja Azuren Cosmos DB:ssä osiointiavaimet pitää valita itse. Osiointiavaimeksi voi Cosmos DB:ssä valita myös tietoalkion ID:n, joka generoidaan tarvittaessa automaattisesti. Jos osiointiavaimen valitsee huonosti, suurilla datamäärillä loogisille osioille voi koh-

distua lukuja tai kirjoituksia epätasaisesti (“Best Practices for Designing and Using Partition Keys Effectively - Amazon DynamoDB” 2021; “Partitioning and Horizontal Scaling in Azure Cosmos DB” 2021). Tällöin suorituskyky voi kärsiä, koska fyysiset laitteistoresurssit varataan loogisen osioinnin mukaan, ja epätasaisella käytöllä jokin rinnakkaisista fyysisistä yksiköistä voi ylikuormittua. GCP:n Firestoressa dokumenteilla on tunniste, joka toimii hajautuksessa, kuten muiden tietokantojen osiointiavaimet. Tunnisteen voi myös valita itse, mutta vastaavasti huonosti valittuna, se voi johtaa suurilla datamäärillä kuumiin pisteisiin, jotka hidastavat kyselyitä (“Best Practices | Firestore” 2021).

Toinen vaaranpaikka liittyy tietokantakyselyihin ja tietokannan datan täyteen skannaukseen. Täysi skannaus tarkoittaa kaikkien tietokannan alkioiden lukemista indeksihakujen tekemisen sijaan. Tämä on kallis operaatio niin suorituskyvyn kuin siitä aiheutuvien kustannustenkin osalta erityisesti, jos dataa on paljon. AWS:n DynamoDB sallii täydet skannaukset, eikä luo indeksejä automaattisesti. DynamoDB:n käyttäjä voi valita, käyttääkö tietoalkioiden hakuun kyselyä, skannausta vai PartiQL-kyselyä. Kysely ja skannaus tekevät nimensä mukaisesti vain joko kyselyn tai skannauksen. Sen sijaan PartiQL-kysely tekee kyselyn tai skannauksen PartiQL-kyselyyn asetettujen `WHERE`-ehtojen perusteella. Jos ehdot jäävät kokonaan puuttumaan tai niissä ei määritetä tarkasti taulun tai toissijaisen indeksin osiointiavaimia, joista tietoa haetaan, suoritetaan täysi skannaus. Täydet skannaukset voi estää AWS:n IAM-käytännöillä (“PartiQL Select Statements for DynamoDB - Amazon DynamoDB” 2021). Azuren Cosmos DB sallii täydet skannaukset, mutta luo indeksejä automaattisesti. Näiden ansiosta yksinkertaisissa kyselyissä ei tehdä täyttä skannausta, ellei indeksointia muuteta tarkoituksella. Myös Cosmos DB:n SQL-kyselyt voivat johtaa täyteen skannaukseen. Näin tapahtuu esimerkiksi `WHERE`-ehdon puuttuessa. Jos ehto on mukana kyselyssä ja vähintään oletusindeksit ovat olemassa, täydet skannaukset ovat harvinaisia (“Indexing in Azure Cosmos DB” 2021). GCP:n Firestore ei salli täysiä skannauksia, vaan antaa virheilmoituksen ja linkin indeksin luomiseksi, jos indeksi puuttuu (“Managing Indexes | Firestore | Google Cloud” 2021).

5.2.3 Dataväylät

AWS:ssä erillistä dataväyläpalvelua ei tarvita, vaan ainoana dataväylänä toimii IoT Coren IoT-säännöt. Nämä tarjoavat suoran integraation DynamoDB-tietokantaan. Azuressa IoT Hubin reitittäjän oletusreitin lisäksi tarvitaan erillinen dataväyläpalvelu kuljettamaan IoT-laitteilta tuleva data tietokantaan. Tutkimuksen IoT-prototyypin kohdalla päädyttiin lopulta Stream Analytics -työkaluun, joka on tehokas ja skaalautuva datavirtojen käsittelypalvelu. IoT-prototyypin kehityksen aikana kokeiltiin myös Service Bus -jonoa yhdessä LogicApps-sovelluspalvelun kanssa ennen Stream Analyticsin löytämistä. Datan vienti Service Bus -jonon ja LogicApps-sovelluksen kautta on työläämpi toteuttaa, eikä välttämättä skaalautuisi yhtä hyvin kuin Stream Analytics -pohjainen ratkaisu.

GCP:n kohdalla datan vienti Cloud IoT Coresta Firestoreen onnistuu yhdistämällä kaksi dataväyläpalvelua. Cloud IoT Core reitittää kaikki viestit oletuksena Cloud Pub/Sub-palveluun. Tämä toimii IoT-prototyypissä ensimmäisenä dataväyläpalveluna. Toisena tällaisena palveluna toimii Cloud Functions, joka hakee viestit Pub/Subista ja tallentaa ne sellaisenaan Firestoreen. GCP:n Dataflow (“Dataflow Documentation | Google Cloud” 2021) voisi olla paremmin skaalautuva palvelu isojen datavirtojen käsittelyyn tai tallennukseen, mutta kirjoitushetkellä se ei tarjoa valmista datavirtamallia (engl. *streaming template*) Pub/Subin ja Firestoren integraation toteuttamiseen. Tämä saattaa johtua siitä, että Firestore on niin uusi, ettei kaikissa GCP:n palveluissa ole sille vielä kattavaa tukea. Dataflow’hun voi ohjelmoida myös omia datavirtamalleja, mutta tämä olisi dokumentaation ohjeistuksen (“Creating Classic Templates | Cloud Dataflow | Google Cloud” 2021) perusteella vaatinut enemmän palvelun käytön opettelua kuin tämän tutkimuksen puitteissa olisi mahdollista. Tutkimuksessa käytetty pilvifunktio toimii hyvin erityisesti IoT-prototyypin kokoluokan toteutuksissa. Funktion voi olettaa toimivan myös isommissa toteutuksissa, koska Cloud Functions on skaalautuva palvelu.

6 Pohdinta

Pohditaan seuraavaksi tutkimuksessa saatuja tuloksia ja tehdään niistä johtopäätöksiä. Aloitetaan käsittelemällä tutkimuksen aikana esiin nousseita huomioita. Tutkimuksen tarkoituksena on tehdä vertailua PaaS-mallisten pilvialustojen välillä, jotta sopivan alustan valitseminen olisi helpompaa. Pilvialustojen käytön opettelu vaatii uudelta kehittäjältä kuitenkin paljon aikaa hyvästä valinnasta huolimatta. Pilvialustat ovat IoT-palveluineen ja muine ominaisuuksineen niin suuria kokonaisuuksia, että niiden käyttö vaatii perehtymistä esimerkiksi alustojen ja palveluiden keskeisiin käsitteisiin ja toimintaperiaatteisiin. Tämä ei ole ihme, sillä ne on suunniteltu asiantuntijatyökaluiksi, joilla suurienkin järjestelmien rakentaminen on mahdollista. Pilvialustojen käytön opettelu- ja perehtymisprosessiin voi saada apua ja nopeutusta esimerkiksi reaaliaikaisista erikoistuneiden kouluttajien pitämistä koulutuksista. Jokaiselle tutkitulle alustalle on myös saatavilla suuri määrä interaktiivisia verkkokursseja. Käytön opettelussa itse kokeileminen on oleellinen osa, jotta alustojen palveluita oppisi käyttämään käytännössä. Tutkimuksen IoT-prototyypin kehityksen aikana huomattiin jokaisen pilvialustan kohdalla, että kaikkea ei ole tutoriaaleissa tai oppaissa suoraan kerrottu. Monet asiat pitää itse löytää esimerkiksi dokumentaatioita kahlaamalla ja kokeilemalla. Esimerkiksi Azuressa tai GCP:ssä ei kerrottu suoraan, miten telemetriaviestit saisi kuljetettua IoT Hubista Cosmos DB:hen tai Cloud IoT Coresta Firestoreen. Tähän sopivien dataväylien löytäminen vaati palvelutarjonnan tutkimista ja sopivilta näyttävien palveluiden kokeilemista.

Toisena huomiona Azuren kohdalla sopivan dataväylän löytäminen IoT Hubista Cosmos DB:hen tuotti haasteita. Tässä yhteydessä kokeiltiin useita eri tapoja datan vientiin. Yksi näistä vaihtoehdoista on viedä telemetriaviestit IoT Hubista Service Bus -jonoon, josta ne luetaan LogicApps-sovelluksella ja tallennetaan Cosmos DB -tietokantaan. Tätä rakennettaessa LogicApps-sovelluksen Cosmos DB -yhteyden määrittelyssä huomattiin käyttöliittymäbugi, joka vaikeutti määrittelyä. Lopulta tämä yhteys saatiin toimimaan, mutta myöhemmin huomattiin, että datan viennin voisi toteuttaa Stream Analyticsin avulla suoraviivaisemmin. Haaste oikean työkalun löytämiseksi erityisesti Azuren kohdalla kumpuaa osittain siitä, että tällä alustalla on joillain osa-alueilla useita osin päällekkäisiä palveluita saatavilla. Esi-

merkiksi datavirtaprosessointiin on Stream Analyticsin lisäksi muitakin palveluita (“Choosing a Stream Processing Technology - Azure Architecture Center” 2021). Tämä sama haaste voi ilmetä myös muilla tässä tutkimuksessa käsitellyillä pilvialustoilla, mutta ei välttämättä etsittäessä palvelua tutkimuksessa käsiteltyyn käyttötarkoitukseen. Oikean palvelun löytämiseen voi auttaa myös laajempi kokemus pilvialustan käytöstä.

Kolmanneksi tutkimuksessa huomattiin, että käsiteltyjä pilvialustoja kehitetään hyvin vauhdikkaasti. Esimerkiksi Azure ehti päivittää IoT Hubin pika-aloitustutoriaaleja IoT-prototyypin kehittämisen ja muistiinpanojen tekemisen jälkeen, mutta ennen tutkielman valmistumista. Tutkimuksen IoT-prototyypin kehityksen aikana suoritettiin vain vanhempi pika-aloitustutoriaali, jota luvussa 4.3.4 käsitellään. Suurimpana muutoksena uudessa tutoriaalissa käytetään uutta *Azure IoT Explorer* -käyttöliittymää IoT-laitteen rekisteröintiin ja sen lähettämien viestien tarkasteluun. Uudessa tutoriaalissa käytetään lisäksi *IoT Plug and Play* -toimintoa (“Introduction to IoT Plug and Play” 2021) hyödyntävää IoT-laitteen esimerkkiohjelmaa yksinkertaisemman viestejä lähettävän esimerkin sijaan. Toinen tutkimuksen aikana tullut uudistus on AWS:n Amazon DynamoDB:n uusi web-käyttöliittymä. Tämä valmistui ja otettiin AWS:ssä oletuksena käyttöön ilmeisesti kaikille kehittäjille tutkielman kirjoituksen aikana. Luvuissa 4.3.2 ja 4.3.3 sekä liitteessä A.3 käytetään DynamoDB:tä vanhan web-käyttöliittymän kautta. Tutkimuksessa käsiteltyyn toiminnallisuuteen ei kuitenkaan ole tullut muutoksia (vielä).

Tarkastellaan seuraavaksi tutkimuksen tulosten luotettavuutta ja yleistettävyyttä. Tutkimuksessa kaikkia pilvialustoja on pyritty kokeilemaan IoT-prototyypin kautta mahdollisimman yhdenmukaisesti. Kokeilun teki kuitenkin vain yksi henkilö — tutkija itse, joten tuloksiin on saattanut vaikuttaa tutkijan subjektiiviset kokemukset sekä aiempi tietotaito. Vaikka tutkija olikin aloittelija pilvialustojen käytössä, eikä tuntenut niitä tai tutkimuksessa käsiteltyjä palveluita juurikaan ennen tutkimusta, voisi laajemmasta otoksesta olla hyötyä. Tuloksiin on voinut vaikuttaa myös järjestys, jolla pilvialustoihin tutustuttiin, sillä tutkijan tietotaito pilvialustoista yleisesti kasvoi jokaisen pilvialustan kohdalla. Pilvialustoja kokeiltiin ja tutkittiin samassa järjestyksessä kuin niitä tässä tutkielmassa käsitellään — ensin AWS, sitten Azure ja viimeiseksi GCP.

Toinen tukimuksen tulosten luotettavuuteen ja yleistettävyyteen vaikuttava seikka on tutkit-

tu käyttötapaus. Tutkimuksessa keskityttiin vain yhteen käyttötapaukseen — telemetriadatan keräämiseen, minkä lisäksi käytössä oli vain yksi IoT-laite ja muutenkin pieni datamäärä. Tämä saattaa vaikuttaa yleistettävyyteen esimerkiksi muun tyyppisten käyttötapausten tai suurempien järjestelmien kohdalla. Toisaalta, vaikka tutkimuksessa pilvialustoille vietiin hyvin yksinkertaista dataa, prosessi kuvattiin tarkasti. Tämän ansiosta sama prosessi voitaisiin toistaa erityyppisellä datalla tai erisuuruisella datamäärällä suoraviivaisesti. Prosessia voisi muokata esimerkiksi niin, että sensoridatan sijaan kerättäisiin kuvadataa, jota vietäisiin pilvialustojen BLOB-varastoihin.

Mahdollisia jatkotutkimuskohteita löytyy tutkimuksen aihealueelta paljon. Tutkimuksessa rajoituttiin telemetriadatan keräämiseen ja tallentamiseen. Tästä luontevana jatkotutkimuskohteena voisi tarkastella esimerkiksi pilvialustojen eroja ja yhtäläisyyksiä datan visualisoinnissa tai käsittelyssä. Myös eroja työkaluissa, joilla dataa voi viedä web- tai mobiilisovelluksiin, voisi tarkastella. Toisaalta tällaisessa aiheessa tulosten yleistettävyys toisen liiketoiminta-alueen dataan tai muuten erityyppiseen dataan voisi muodostua haasteeksi. Jatkotutkimusta voisi tehdä myös pilvialustojen eroista IoT-ominaisuuksissa, joita tässä tutkimuksessa ei käsitelty. Aiheita voisivat olla esimerkiksi IoT-laitteiden ohjaus pilven kautta, IoT-laitteiden laiteohjelmistopäivitykset ja niiden asentaminen tai IoT-palveluiden reunalaskentatuki (engl. *Edge computing*). Jos aihealueen tutkimuksessa olisi käytettävissä suuri määrä resursseja tai useita suuria toimijoita, voisi jatkotutkimusta tehdä pilvialustojen IoT-ominaisuuksista suuressa skaalassa. Testailevaa tai vertailevaa tutkimusta voisi tehdä myös pilvialustojen tarjoamista IoT-laitteiden käyttöjärjestelmistä.

7 Yhteenveto

Tässä tutkimuksessa tutustuttiin kolmeen julkiseen pilvialustaan ja vertailtiin niitä telemetriadataa keräävien IoT-järjestelmien näkökulmasta. Vertaillut pilvialustat ovat Amazon Web Services (AWS), Microsoft Azure ja Google Cloud Platform (GCP). Vertailu tehtiin rakentamalla Raspberry Pi -tietokoneen päälle säädataa keräävä IoT-prototyyppi. Prototyypissä Raspberry Pi kerää dataa siihen liitetyllä sensorilla ja lähettää sen pilvialustoilla oleviin IoT-palveluihin. Näistä data kuljetetaan alustojen tarjoamiin NoSQL-tietokantoihin. Prototyypin kehitysprosessista saatujen kokemusten ja siinä huomattujen asioiden pohjalta pyrittiin vastaamaan kahteen tutkimuskysymykseen: Kuinka keskeisimpiä julkisia pilvialustoja voidaan hyödyntää IoT-datan keruussa, sekä kuinka nämä pilvialustat eroavat toisistaan käytännön kehitystyössä, kun kyseessä on IoT-dataa keräävä järjestelmä? Ensimmäiseen kysymykseen saatiin vastaus luvussa 4, jossa jokaisen tutkittavan pilvialustan kohdalla käytiin läpi, miten datan kerääminen ja tallentaminen pilvialustalle voidaan esimerkiksi tehdä. Kysymykseen eroista käytännön kehitystyössä vastattiin luvussa 5. Siinä käytiin läpi eroja niin pilvialustojen omaksuttavuudessa kuin prototyypin kehityksen aikana käytetyissä työkaluissakin.

Tutkimuksessa havaittuja eroja tiivistäen GCP tarjoaa esimerkiksi IoT-palvelulleen niukimman dokumentaation, mutta toisaalta Firestore-tietokannassa on kehittäjän työtä helpottavia indeksointitoimintoja. Azure tarjoaa suuren määrän monipuolisia työkaluja ja laajimman dokumentaation, mutta toisaalta vaatii paljon perehtymistä ja kokeilua esimerkiksi dataa välittävien palveluiden osalta. AWS tarjoaa puolestaan ominaisuuksiltaan monipuolisimman IoT-palvelun, mutta toisaalta DynamoDB:n indeksointitoiminnot ovat kilpailijoitaan alkeellisemmat.

Kokonaisuudessaan tutkimus onnistui hyvin. Suurimmat haasteet tulivat vastaan tutustuttaessa pilvialustoihin, kun merkittävää aiempaa kokemusta niistä ei ollut. Haasteiden ilmeneminen tässä oli toisaalta odotettavissa, koska aihetta oli tarkoituskin tutkia aloittelijan näkökulmasta. IoT-laitteita ja niiltä kerättyä dataa hyödynnetään entistä enemmän pilvipohjaisissa järjestelmissä. Julkiset pilvialustat tarjoavat tällaisten järjestelmien kehitykselle hyvän pohjan ja paljon erikoistuneita työkaluja, vaikka ne vaativatkin kehittäjältä perehtymistä.

Lähteet

“Amazon Simple Storage Service (S3)”. 2021. Amazon Web Services, Inc. Viitattu 6. huhtikuuta. https://aws.amazon.com/s3/.

“Amazon Web Services (AWS)”. 2021. Amazon Web Services, Inc. Viitattu 6. huhtikuuta. https://aws.amazon.com/.

“AWS IoT Core”. 2021. Amazon Web Services, Inc. Viitattu 3. helmikuuta. https://aws.amazon.com/iot-core/.

“Android Things”. 2021. Android Developers. Viitattu 20. heinäkuuta. https://developer.android.com/things?hl=fi.

“ARM Mbed Pelion”. 2021. Viitattu 10. helmikuuta. https://developer.pelion.com/docs/device-management/current/welcome/index.html.

“Authorization - AWS IoT Core”. 2021. Viitattu 15. syyskuuta. https://docs.aws.amazon.com/iot/latest/developerguide/iot-authorization.html.

“AWS IoT Rule Actions - AWS IoT Core”. 2021. Viitattu 7. elokuuta. https://docs.aws.amazon.com/iot/latest/developerguide/iot-rule-actions.html.

“Azure Cosmos DB Resource Model”. 2021. Viitattu 18. elokuuta. https://docs.microsoft.com/en-us/azure/cosmos-db/account-databases-containers-items.

“Azure Internet of Things (IoT) Technologies and Solutions”. 2021. Viitattu 30. heinäkuuta. https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-services-and-technologies.

“Azure IoT Hub”. 2021. Viitattu 10. helmikuuta. https://azure.microsoft.com/en-us/services/iot-hub/.

"Azure IoT Hub Communication Protocols and Ports". 2021. Viitattu 14. huhtikuuta. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-protocols.

"Azure IoT Hub Documentation". 2021. Viitattu 9. syyskuuta. https://docs.microsoft.com/en-us/azure/iot-hub/.

"Azure IoT Hub TLS Support". 2021. Viitattu 15. syyskuuta. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-tls-support.

"Azure RTOS". 2021. Viitattu 20. heinäkuuta. https://azure.microsoft.com/en-us/services/rtos/.

Bandyopadhyay, S., ja A. Bhattacharyya. 2013. "Lightweight Internet Protocols for Web Enablement of Sensors Using Constrained Gateway Devices". Teoksessa *2013 International Conference on Computing, Networking and Communications (ICNC),* 334–340. Tammikuu. doi:10.1109/ICCNC.2013.6504105.

Bastos, D. 2019. "Cloud for IoT — A Survey of Technologies and Security Features of Public Cloud IoT Solutions". Teoksessa *Living in the Internet of Things (IoT 2019),* 1–6. Toukokuu. doi:10.1049/cp.2019.0168.

Baun, Christian, Marcel Kunze, Jens Nimis ja Stefan Tai. 2011. *Cloud Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-20916-1 978-3-642-20917-8. doi:10.1007/978-3-642-20917-8.

Belshe, Mike, Martin Thomson ja Roberto Peon. 2015. "Hypertext Transfer Protocol Version 2 (HTTP/2)". Viitattu 23. huhtikuuta 2021. https://tools.ietf.org/html/rfc7540.

"Best Practices for Designing and Using Partition Keys Effectively - Amazon DynamoDB". 2021. Viitattu 19. syyskuuta. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-partition-key-design.html.

BME280 Data Sheet. 2020. https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf.

Bojan, V., I. Raducu, F. Pop, M. Mocanu ja V. Cristea. 2015. “Cloud-Based Service for Time Series Analysis and Visualisation in Farm Management System”. Teoksessa *2015 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP),* 425–432. Syyskuu. doi:10.1109/ICCP.2015.7312697.

Botta, Alessio, Walter de Donato, Valerio Persico ja Antonio Pescapé. 2016. “Integration of Cloud Computing and Internet of Things: A Survey”. *Future Generation Computer Systems* 56 (1. maaliskuuta): 684–700. ISSN: 0167-739X. doi:10.1016/j.future.2015.09.021.

Bziuk, W., C. V. Phung, J. Dizdarević ja A. Jukan. 2018. “On HTTP Performance in IoT Applications: An Analysis of Latency and Throughput”. Teoksessa *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO),* 0350–0355. Toukokuu. doi:10.23919/MIPRO.2018.8400067.

CERP-IoT. 2010. *Vision and Challenges for Realising the Internet of Things.* LU: Publications Office. Viitattu 13. maaliskuuta 2021. https://data.europa.eu/doi/10.2759/26127.

Chen, Yuang, ja Thomas Kunz. 2016. “Performance Evaluation of IoT Protocols under a Constrained Wireless Access Network”. Teoksessa *2016 International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT),* 1–7. Huhtikuu. doi:10.1109/MoWNet.2016.7496622.

“Choosing a Stream Processing Technology - Azure Architecture Center”. 2021. Viitattu 21. syyskuuta. https://docs.microsoft.com/en-us/azure/architecture/data-guide/technology-choices/stream-processing.

“Client Authentication - AWS IoT Core”. 2021. Viitattu 13. syyskuuta. https://docs.aws.amazon.com/iot/latest/developerguide/client-authentication.html.

“Cloud Functions Overview | Cloud Functions Documentation”. 2021. Viitattu 1. syyskuuta. https://cloud.google.com/functions/docs/concepts/overview?hl=fi.

“Cloud IoT Core Overview | Cloud IoT Core Documentation | Google Cloud”. 2021. Viitattu 30. elokuuta. https://cloud.google.com/iot/docs/concepts/overview?hl=fi.

“CoAP Receiver”. 2021. Viitattu 14. huhtikuuta. https://azuremarketplace.microsoft.com/en-ca/marketplace/apps/generic_de.generic-coap-receiver?tab=overview.

“Concepts of Azure IoT Hub X.509 Security”. 2021. Viitattu 17. elokuuta. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-x509ca-concept.

“Connect a Device to AWS IoT Core by Using the AWS IoT Device SDK - AWS IoT Core”. 2021. Viitattu 3. elokuuta. https://docs.aws.amazon.com/iot/latest/developerguide/sdk-tutorials.html.

“Connecting to AWS IoT Core Service Endpoints - AWS IoT Core”. 2021. Viitattu 3. lokakuuta. https://docs.aws.amazon.com/iot/latest/developerguide/iot-connect-service.html.

“Control Access to IoT Hub Using SAS Tokens”. 2021. Viitattu 13. syyskuuta. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-dev-guide-sas.

“Core Components of Amazon DynamoDB - Amazon DynamoDB”. 2021. Viitattu 6. elokuuta. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html.

“Creating Classic Templates | Cloud Dataflow | Google Cloud”. 2021. Viitattu 20. syyskuuta. https://cloud.google.com/dataflow/docs/guides/templates/creating-templates?hl=fi.

“Data Model | Firestore | Google Cloud”. 2021. Viitattu 31. elokuuta. https://cloud.google.com/firestore/docs/data-model?hl=fi.

“Dataflow Documentation | Google Cloud”. 2021. Viitattu 20. syyskuuta. https://cloud.google.com/dataflow/docs?hl=fi.

"Designing MQTT Topics for AWS IoT Core". 2019 (toukokuu): 32. https://d1.awsstatic.com/whitepapers/Designing_MQTT_Topics_for_AWS_IoT_Core.pdf.

"Device Communication Protocols - AWS IoT Core". 2021. Viitattu 2. elokuuta. https://docs.aws.amazon.com/iot/latest/developerguide/protocols.html.

"DigitalOcean". 2021. Viitattu 6. huhtikuuta. https://www.digitalocean.com/.

Dutta, S., S. S. L. Chukkapalli, M. Sulgekar, S. Krithivasan, P. K. Das ja A. Joshi. 2020. "Context Sensitive Access Control in Smart Home Environments". Teoksessa *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS),* 35–41. Toukokuu. doi:10.1109/BigDataSecurity-HPSC-IDS49724.2020.00018.

"Eclipse Paho". 2021. Viitattu 22. heinäkuuta. https://www.eclipse.org/paho/index.php?page=clients/python/docs/index.php.

Fielding, Roy Thomas. 2000. "Architectural Styles and the Design of Network-Based Software Architectures": 180. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.

Foster, Andrew. 2015. "Messaging Technologies for the Industrial Internet and the Internet of Things". *PrismTech Whitepaper* 21. https://www.smartindustry.com/whitepapers/2015/messaging-technologies-for-the-industrial-internet-and-the-internet-of-things/.

"FreeRTOS". 2021. FreeRTOS. Viitattu 20. heinäkuuta. https://www.freertos.org/RTOS.html.

"Getting Started with AWS IoT Core - AWS IoT Core". 2021. Viitattu 2. elokuuta. https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html.

"BME280 Sensor Driver". 2021. GitHub. Viitattu 22. heinäkuuta. https://github.com/rm-hull/bme280.

"MrCliff/Iot-Cloud-Connection-Proto". 2021. GitHub. Viitattu 4. heinäkuuta. https://github.com/MrCliff/iot-cloud-connection-proto.

"Best Practices | Firestore". 2021. Google Cloud. Viitattu 19. syyskuuta. https://cloud.google.com/firestore/docs/best-practices.

"Choosing between Native Mode and Datastore Mode | Firestore". 2021. Google Cloud. Viitattu 31. heinäkuuta. https://cloud.google.com/firestore/docs/firestore-or-datastore.

"Creating Public/Private Key Pairs | Cloud IoT Core Documentation". 2021. Google Cloud. Viitattu 4. syyskuuta. https://cloud.google.com/iot/docs/how-tos/credentials/keys?hl=fi.

"Device Security | Cloud IoT Core Documentation". 2021. Google Cloud. Viitattu 13. syyskuuta. https://cloud.google.com/iot/docs/concepts/device-security?hl=fi.

"Google Cloud IoT Core". 2021. Google Cloud. Viitattu 10. helmikuuta. https://cloud.google.com/iot-core.

"Google Cloud Platform (GCP)". 2021. Google Cloud. Viitattu 6. huhtikuuta. https://cloud.google.com/.

"Protocols | Cloud IoT Core Documentation". 2021. Google Cloud. Viitattu 14. huhtikuuta. https://cloud.google.com/iot/docs/concepts/protocols.

"Publishing over the MQTT Bridge | Cloud IoT Core Documentation". 2021. Google Cloud. Viitattu 2. syyskuuta. https://cloud.google.com/iot/docs/how-tos/mqtt-bridge?hl=fi.

"Google Cloud Courses and Training". 2021. Viitattu 30. heinäkuuta. https://cloud.google.com/training.

"Google Maps". 2021. Google Maps. Viitattu 31. maaliskuuta. https://www.google.com/maps.

"Google Scholar". 2021. Viitattu 3. helmikuuta. https://scholar.google.fi/.

"GPIO - Raspberry Pi Documentation". 2021. Viitattu 21. heinäkuuta. https://www.raspberrypi.org/documentation/usage/gpio/README.md.

Hardt, Dick. 2012. "The OAuth 2.0 Authorization Framework". Viitattu 20. huhtikuuta 2021. https://tools.ietf.org/html/rfc6749.

Hejazi, H., H. Rajab, T. Cinkler ja L. Lengyel. 2018. "Survey of Platforms for Massive IoT". Teoksessa *2018 IEEE International Conference on Future IoT Technologies (Future IoT),* 1–8. Tammikuu. doi:10.1109/FIOT.2018.8325598.

Hellbe, Anton, ja Gustaf Bohlin. 2018. "Evaluating IoT Cloud Platforms in the Context of Smart Buildings". Viitattu 24. helmikuuta 2021. http://muep.mau.se/handle/2043/28214.

Hevner, A.R., S.T. March, J. Park ja S. Ram. 2004. "Design Science in Information Systems Research". *MIS Quarterly: Management Information Systems* 28 (1): 75–105. doi:10.2307/25148625.

"How AWS IoT Works - AWS IoT Core". 2021. Viitattu 3. elokuuta. https://docs.aws.amazon.com/iot/latest/developerguide/aws-iot-how-it-works.html.

Hull, Jared. 2021. "RaspberryPi-FreeRTOS". GitHub. Viitattu 20. heinäkuuta. https://github.com/Forty-Tw0/RaspberryPi-FreeRTOS.

Iasio, A. De, A. Futno, L. Goglia ja E. Zimeo. 2019. "A Microservices Platform for Monitoring and Analysis of IoT Traffic Data in Smart Cities". Teoksessa *2019 IEEE International Conference on Big Data (Big Data),* 5223–5232. Joulukuu. doi:10.1109/BigData47090.2019.9006025.

"IBM Watson IoT". 2021. Viitattu 10. helmikuuta. https://internetofthings.ibmcloud.com/.

"IEEE Xplore". 2021. Viitattu 3. helmikuuta. https://ieeexplore.ieee.org.

"Indexing in Azure Cosmos DB". 2021. Viitattu 19. syyskuuta. https://docs.microsoft.com/en-us/azure/cosmos-db/index-overview.

"Introduction to Azure Stream Analytics". 2021. Viitattu 19. elokuuta. https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction.

"Introduction to IoT Plug and Play". 2021. Viitattu 21. syyskuuta. https://docs.microsoft.com/en-us/azure/iot-develop/overview-iot-plug-and-play.

"MQTT V3.1.1 ISO Standard (ISO/IEC 20922:2016)". 2016. ISO. Viitattu 9. huhtikuuta 2021. https://www.iso.org/standard/69466.html.

Jones, Michael, John Bradley ja Nat Sakimura. 2015. *JSON Web Token (JWT).* Request for Comments RFC 7519. Internet Engineering Task Force, toukokuu. doi:10.17487/RFC7519.

"JSON". 2021. Viitattu 23. heinäkuuta. https://www.json.org.

Kalmar, Edua Eszter, ja Attila Kertesz. 2017. "What Does I(o)T Cost?" Teoksessa *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion,* 19–24. ICPE '17 Companion. New York, NY, USA: Association for Computing Machinery, 18. huhtikuuta. ISBN: 978-1-4503-4899-7. doi:10.1145/3053600.3053601.

Kasanen, Eero, Kari Lukka ja Arto Siitonen. 1991. "Konstruktiivinen Tutkimusote Liiketaloustieteessä". *Liiketaloudellinen aikakauskirja* 40 (3): 301–329.

"Managing Indexes | Firestore | Google Cloud". 2021. Viitattu 1. syyskuuta. https://cloud.google.com/firestore/docs/query-data/indexing?hl=fi.

Mell, Peter, ja Timothy Grance. 2011. "The NIST Definition of Cloud Computing": 7. https://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf.

"Microsoft 365". 2021. Viitattu 31. maaliskuuta. https://www.microsoft.com/fi-fi/microsoft-365.

"Microsoft Azure". 2021. Viitattu 6. huhtikuuta. https://azure.microsoft.com/en-us/.

"Microsoft Azure Virtual Machines". 2021. Viitattu 6. huhtikuuta. https://azure.microsoft.com/en-us/services/virtual-machines/.

"MindMeister". 2021. Viitattu 31. maaliskuuta. https://www.mindmeister.com/folders.

Mishra, B., ja A. Kertesz. 2020. "The Use of MQTT in M2M and IoT Systems: A Survey". *IEEE Access* 8:201071–201086. ISSN: 2169-3536. doi:10.1109/ACCESS.2020.3035849.

Mogul, Jeffrey C., Jim Gettys, Tim Berners-Lee ja Henrik Frystyk. 1997. "Hypertext Transfer Protocol – HTTP/1.1". Viitattu 23. huhtikuuta 2021. https://tools.ietf.org/html/rfc2068.

Naik, N. 2017. "Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP". Teoksessa *2017 IEEE International Systems Engineering Symposium (ISSE),* 1–7. Lokakuu. doi:10.1109/SysEng.2017.8088251.

Nayak, Ameya. 2013. "Type of NOSQL Databases and Its Comparison with Relational Databases". *International Journal of Applied Information Systems* 5:4. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.402.3372&rep=rep1&type=pdf.

OASIS Standard. 2012. "OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0". Toimittanut Robert Godfrey, David Ingham ja Rafael Schloming. https://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf.

———. 2014. "MQTT Version 3.1.1". Toimittanut Andrew Banks ja Rahul Gupta. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

———. 2019. "MQTT Version 5.0". Toimittanut Andrew Banks, Ed Briggs, Ken Borgendale ja Rahul Gupta. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html.

Object Management Group. 2015. “Data Distribution Service (DDS)”: 180. https://www.omg.org/spec/DDS/1.4/PDF.

“Overview for Azure Logic Apps - Azure Logic Apps”. 2021. Viitattu 3. lokakuuta. https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-overview.

“Overview of Azure IoT Hub Message Enrichments”. 2021. Viitattu 18. elokuuta. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-message-enrichments-overview.

“PartiQL Select Statements for DynamoDB - Amazon DynamoDB”. 2021. Viitattu 19. syyskuuta. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-reference.select.html.

“Partitioning and Horizontal Scaling in Azure Cosmos DB”. 2021. Viitattu 19. syyskuuta. https://docs.microsoft.com/en-us/azure/cosmos-db/partitioning-overview.

Peter Loshin. 2003. *TCP/IP Clearly Explained.* Nide 4th ed. The Morgan Kaufmann Series in Networking. Amsterdam: Morgan Kaufmann. ISBN: 978-1-55860-782-8, viitattu 16. huhtikuuta 2021. http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=249308&site=ehost-live.

Pflanzner, T., ja A. Kertesz. 2016. “A Survey of IoT Cloud Providers”. Teoksessa *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO),* 730–735. Toukokuu. doi:10.1109/MIPRO.2016.7522237.

“Quickstart | Cloud IoT Core Documentation | Google Cloud”. 2021. Viitattu 30. elokuuta. https://cloud.google.com/iot/docs/quickstart?hl=fi.

“Raspberry Pi Foundation - About Us”. 2021. Raspberry Pi. Viitattu 17. maaliskuuta. https://www.raspberrypi.org/about/.

"Raspberry Pi Hardware - Raspberry Pi Documentation". 2021. Viitattu 19. maaliskuuta. https://www.raspberrypi.org/documentation/hardware/raspberrypi/README.md.

"Raspberry Pi OS - Raspberry Pi Documentation". 2021. Viitattu 19. maaliskuuta. https://www.raspberrypi.org/documentation/raspbian/.

Rescorla, Eric, ja Tim Dierks. 2008. "The Transport Layer Security (TLS) Protocol Version 1.2". Viitattu 20. huhtikuuta 2021. https://tools.ietf.org/html/rfc5246.

Saint-Andre, Peter. 2011. "Extensible Messaging and Presence Protocol (XMPP): Core". Viitattu 28. huhtikuuta 2021. https://tools.ietf.org/html/rfc6120.

Salminen, Ari. 2011. *Mikä Kirjallisuuskatsaus? : Johdatus Kirjallisuuskatsauksen Tyyppeihin Ja Hallintotieteellisiin Sovelluksiin.* Vaasan yliopisto. ISBN: 978-952-476-349-3, viitattu 19. helmikuuta 2021. https://osuva.uwasa.fi/handle/10024/7961.

"Scopus - Document Search". 2021. Viitattu 3. helmikuuta. https://www.scopus.com/search/form.uri.

"Send Device Telemetry to Azure IoT Hub Quickstart". 2021. Viitattu 21. elokuuta. https://docs.microsoft.com/en-us/azure/iot-develop/quickstart-send-telemetry-iot-hub.

Serrano, D., T. Baldassarre ja E. Stroulia. 2016. "Real-Time Traffic-Based Routing, Based on Open Data and Open-Source Software". Teoksessa *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT),* 661–665. Joulukuu. doi:10.1109/WF-IoT.2016.7845419.

Shelby, Zach, Klaus Hartke ja Carsten Bormann. 2014. "The Constrained Application Protocol (CoAP)". Viitattu 13. huhtikuuta 2021. https://tools.ietf.org/html/rfc7252.

Stanford-Clark, Andy, ja Hong Linh Truong. 2013. "MQTT For Sensor Networks (MQTT-SN) Protocol Specification": 28. https://www.oasis-open.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf.

"Store Device Data in a DynamoDB Table - AWS IoT Core". 2021. Viitattu 22. syyskuuta. https://docs.aws.amazon.com/iot/latest/developerguide/iot-ddb-rule.html.

Tanenbaum, Andrew S. 2015. *Modern Operating Systems.* Fourth edition. Boston: Pearson. ISBN: 978-0-13-359162-0.

Tulenkov, A., A. Parkhomenko ja A. Sokolyanskii. 2019. "Evaluation and Selection of IoT Service for Smart House System Big Data Processing". Teoksessa *2019 IEEE 14th International Conference on Computer Sciences and Information Technologies (CSIT),* 2:124–129. Syyskuu. doi:10.1109/STC-CSIT.2019.8929810.

"Tutorial - Configure Message Routing for Azure IoT Hub Using Azure CLI". 2021. Viitattu 15. elokuuta. https://docs.microsoft.com/en-us/azure/iot-hub/tutorial-routing.

"UM10204 I2C-Bus Specification and User Manual". 2014. 2014:64. https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

"Understand Azure IoT Hub Message Format". 2021. Viitattu 18. elokuuta. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-messages-construct.

"Using JSON Web Tokens (JWTs) | Cloud IoT Core Documentation". 2021. Viitattu 30. elokuuta. https://cloud.google.com/iot/docs/how-tos/credentials/jwts?hl=fi.

Uy, N. Q., ja V. H. Nam. 2019. "A Comparison of AMQP and MQTT Protocols for Internet of Things". Teoksessa *2019 6th NAFOSTED Conference on Information and Computer Science (NICS),* 292–297. Joulukuu. doi:10.1109/NICS48868.2019.9023812.

Wang, Heng, Daijin Xiong, Ping Wang ja Yuqiang Liu. 2017. "A Lightweight XMPP Publish/-Subscribe Scheme for Resource-Constrained IoT Devices". *IEEE Access* 5:16393–16405. ISSN: 2169-3536. doi:10.1109/ACCESS.2017.2742020.

Varvello, Matteo, Kyle Schomp, David Naylor, Jeremy Blackburn, Alessandro Finamore ja Konstantina Papagiannaki. 2016. “Is the Web HTTP/2 Yet?” Teoksessa *Passive and Active Measurement,* toimittanut Thomas Karagiannis ja Xenofontas Dimitropoulos, 218–232. Lecture Notes in Computer Science. Cham: Springer International Publishing. ISBN: 978-3-319-30505-9. doi:10.1007/978-3-319-30505-9_17.

“Welcome to Apache Avro!” 2021. Viitattu 16. elokuuta. https://avro.apache.org/.

“What Is Amazon DynamoDB? - Amazon DynamoDB”. 2021. Viitattu 4. elokuuta. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html.

“What Is AWS IoT? - AWS IoT Core”. 2021. Viitattu 9. syyskuuta. https://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html.

“What Is Azure IoT Hub”. 2021. Viitattu 3. lokakuuta. https://docs.microsoft.com/en-us/azure/iot-hub/about-iot-hub.

“What Is Pub/Sub? | Cloud Pub/Sub Documentation | Google Cloud”. 2021. Viitattu 1. syyskuuta. https://cloud.google.com/pubsub/docs/overview?hl=fi.

“Windows 10 IoT”. 2021. Viitattu 20. heinäkuuta. https://developer.microsoft.com/en-us/windows/iot/.

Winnie, Y., U. E ja D. M. Ajay. 2018. “Enhancing Data Security in IoT Healthcare Services Using Fog Computing”. Teoksessa *2018 International Conference on Recent Trends in Advance Computing (ICRTAC),* 200–205. Syyskuu. doi:10.1109/ICRTAC.2018.8679404.

Vinoski, Steve. 2006. “Advanced Message Queuing Protocol”. *IEEE Internet Computing* 10, numero 6 (marraskuu): 87–89. ISSN: 1941-0131. doi:10.1109/MIC.2006.116.

Yuyanto ja S. Liawatimena. 2018. “Implementation of Data Collecting Platform Over Distributed Sensors for Global Open Data for Agriculture and Nutrition”. Teoksessa *2018 6th International Conference on Cyber and IT Service Management (CITSM),* 1–7. Elokuu. doi:10.1109/CITSM.2018.8674057.

Liitteet

A Prototyypin yksityiskohtaiset vaiheet

Tässä listataan prototyypin kehittämisen aikana suoritetut yksityiskohtaiset vaiheet mukaan lukien yksittäiset komennot. Raspberry Pi -tietokoneesta käytetään nimitystä *RasPi* luettavuuden parantamiseksi.

A.1 Raspberry Pi:n asennus ja valmistelu

1. Asennetaan Raspberry Pi OS Lite (32-bit) -käyttöjärjestelmä RasPin virallisten verkkosivujen ohjeiden [1] mukaan MicroSD-muistikortille. Asennuksessa käytetään toiselle tietokoneelle asennettua Raspberry Pi Imager -ohjelmaa.
2. Lisätään muistikortin `/boot`-osiolle tyhjä `ssh`-niminen tiedosto. RasPin SSH-palvelin kytkeytyy tämän ansiosta automaattisesti päälle koneen käynnistyessä, eikä RasPiin tarvitse liittää näyttöä tai näppäimistöä.
3. Liitetään muistikortti RasPiin ja kytketään siihen verkkokaapeli.
4. Käynnistetään RasPi kytkemällä siihen virtakaapeli.
5. Selvitetään RasPille myönnetty IP-osoite verkon DHCP-palvelimelta. Kotiverkoissa tämä palvelin on yleensä reitittimellä.
6. Avataan saman verkon toiselta tietokoneelta SSH-asiakasohjelma (esimerkiksi *PuTTY* Windowsilla tai OpenSSH:n `ssh`-komento) ja yhdistetään RasPiin käyttämällä sen IP-osoitetta ja porttia 22.
7. Kirjaudutaan käyttämällä käyttäjänimeä *pi* ja salasanaa *raspberry*.
8. Vaihdetaan salasana vahvempaan `passwd`-komennolla.
9. Tehdään alkukonfigurointeja komennolla `sudo raspi-config`. Kaikki näistä eivät ole pakollisia.

    - Vaihdetaan RasPin verkkonimi valikosta 1-S4.
    - Kytketään I2C-väyläajuri päälle valikosta 3-P5.

1. `https://www.raspberrypi.org/documentation/installation/installing-images/README.md`

- Lisätään lokaali *fi_FI.UTF-8 UTF-8* valikosta 5-L1.
- Asetetaan aikavyöhykkeeksi Helsinki valikosta 5-L2.
- Laajennetaan tiedostojärjestelmä kattamaan kaiken tilan MicroSD-muistikortilla valikosta 6-A1.

10. Muutetaan *pi*-käyttäjän käyttäjänimeksi *joonatan* seuraavien vaiheiden mukaisesti.

    (a) Luodaan SSH-avainpari tietokoneella, jolla RasPiin on yhdistetty. Tämän voi tehdä esimerkiksi OpenSSH:n `ssh-keygen`-komennolla tai Windowsilla PuTTY:n *PuTTYgen*-työkalulla.

    (b) Lisätään avainparin julkinen avain RasPiin sekä *pi*- että *root*-käyttäjälle. Avainta ei ole pakko lisätä tässä *pi*-käyttäjälle, mutta lisätään se helpomman kirjautumisen vuoksi. Tämä tapahtuu kummallekin käyttäjälle seuraavilla komennoilla. (*root*-käyttäjäksi voi kirjautua, kun suorittaa *pi*-käyttäjänä komennon `sudo su`.)

        i. Jos `~/.ssh`-hakemistoa ei ole, lisätään se komennolla `mkdir ~/.ssh` ja asetetaan oikeat käyttöoikeudet komennolla `chmod 755 ~/.ssh`.

        ii. Jos tiedostoa `~/.ssh/authorized_keys` ei ole, lisätään se komennolla `touch ~/.ssh/authorized_keys` ja asetetaan oikeat käyttöoikeudet komennolla `chmod 644 ~/.ssh/authorized_keys`.

        iii. Avataan avaintiedosto komennolla `nano ~/.ssh/authorized_keys` tekstieditorissa ja kopioidaan toisella tietokoneella luodun avainparin julkinen avain tekstimuotoisena tähän tiedostoon. Tallennetaan ja suljetaan tiedosto.

    (c) Muokataan SSH-palvelimen asetuksia siten, että *root*-käyttäjän kirjautuminen sallitaan SSH-avaimella. Tämä tehdään avaamalla SSH-palvelimen asetustiedosto komennolla `sudo nano /etc/ssh/sshd_config` ja lisäämällä seuraavat rivit, jos niitä ei vielä ole.

        - `PermitRootLogin prohibit-password`
        - `PubkeyAuthentication yes`

    (d) Kirjaudutaan RasPille uudelleen sisään *root*-käyttäjällä SSH-avainparin avulla.

    (e) Muutetaan `usermod -l joonatan -d /home/joonatan -m pi` -ko-

mennolla *pi*-käyttäjän käyttäjänimeksi *joonatan*. Tässä `-l`-vaihde määrittää uuden käyttäjänimen, `-d`-vaihde määrittää uuden kotikansion ja `-m`-vaihde määrittää, että vanha kotikansio siirretään uudeksi kotikansioksi. Viimeisenä komennossa on muutettavan käyttäjän nimi.

(f) Muutetaan komennolla `groupmod -n joonatan pi` *pi*-käyttäjän ryhmän nimeksi *joonatan*.

(g) Käynnistetään RasPi uudelleen `reboot`-komennolla.

11. Kirjaudutaan RasPille *joonatan*-käyttäjällä. Konfiguroidaan ja kytketään palomuuri päälle seuraavien komentojen mukaisesti.

    (a) Asennetaan `ufw`-ohjelma helpottamaan palomuurin hallintaa. Tämä tehdään komennolla `sudo apt install ufw`.

    (b) Sallitaan SSH-yhteyden käyttämä TCP-liikenne porttiin 22 komennolla `sudo ufw allow 22/tcp`.

    (c) Kytketään palomuuri päälle komennolla `sudo ufw enable`. Palomuurin tilan voi tarkistaa komennolla `sudo ufw status verbose`.

12. Asennetaan muita hyödyllisiä ohjelmia mieltymysten mukaan. Esimerkiksi *nano*a monipuolisemman *emacs*-tekstieditorin voi asentaa komennolla `sudo apt install emacs`.

A.2 BME280-sensorin valmistelu ja datankeruuohjelman kirjoittaminen

1. Kytketään BME280-sensori RasPin I2C-väylään käyttäen tutkielman luvussa 4.1.2 ja kuviossa 5 kuvattuja kytkentöjä. Oikeiden GPIO-tappien etsimisessä käytettiin apuna RasPin GPIO-dokumentaatiota [2].
2. Kirjaudutaan RasPiin *joonatan*-käyttäjällä.
3. Tarkistetaan ja suoritetaan seuraavat esivaatimukset, jotka BME280-sensorin ajurikirjaston GitHub-sivulla [3] mainitaan.

    (a) Varmistetaan, että I2C-väylän kerneliajuri on kytketty päälle. Tämä tapahtuu varmistamalla, että komennoilla `dmesg | grep i2c` ja `lsmod | grep i2c`

2. `https://www.raspberrypi.org/documentation/usage/gpio/README.md`
3. `https://github.com/rm-hull/bme280#pre-requisites`

tulee vastaukseksi maininta `i2c`-ajurista. Jos vastausta ei tule, pitää ajuri käynnistää RasPin valmisteluvaihelistauksen A.1 kohdan 9 mukaisesti ja tämän jälkeen käynnistää RasPi uudestaan.

(b) Nostetaan I2C-väylän siirtonopeus nopeaan tilaan (400 kbit/s) varmistamalla, että tiedostossa `/boot/config.txt` on seuraava rivi ja käynnistämällä RasPi uudestaan:

- `dtparam=i2c_arm=on,i2c_baudrate=400000`

(c) Varmistetaan komennolla `sudo adduser joonatan i2c`, että *joonatan*-käyttäjä on *i2c*-ryhmässä.

(d) Asennetaan työkalu, jolla voi hakea BME280-sensorin I2C-osoitteen. Tämä tehdään `sudo apt install i2c-tools` -komennolla.

(e) Haetaan BME280-sensorin osoite komennolla `i2cdetect -y 1`, jossa 1 on I2C-väylän porttinumero. Vastauksena komento palauttaa taulukon käytössä olevista I2C-osoitteista heksadesimaalilukuina. Tässä tapauksessa käytetyn sensorin osoite on 0x76. `i2cdetect`-ohjelman `man`-sivun mukaan ohjelma voi sotkea I2C-väylän kommunikaatiota, jos siellä on muuta liikennettä komennon suorituksen aikaan.

4. Asennetaan `pipenv`-työkalu [4] suoraviivaisempaa riippuvuuksienhallintaa varten komennolla `sudo pip3 install pipenv`.
5. Luodaan hakemisto `~/gradu/proto` tutkielman prototyypin datankeruusovellusta varten ja vaihdetaan se työhakemistoksi.
6. Lisätään aiemmin mainittu BME280-sensorin ajurikirjasto datankeruuohjelman ensimmäiseksi riippuvuudeksi komennolla `pipenv install RPi.bme280`. Tämä komento luo työkansiolle myös Python-virtuaaliympäristön, jos sitä ei vielä ole ja asentaa kirjaston siihen.
7. Testataan sensorin ja ajurikirjaston toimintaa seuraavien vaiheiden mukaisesti.

   (a) Luodaan ja avataan uusi tiedosto tekstieditorissa. Tässä tiedostonimeksi asetetaan `bme280test.py`.

   (b) Kopioidaan esimerkkiohjelma BME280-sensorin ajurikirjaston GitHub-sivulta

4. `https://pipenv.pypa.io/en/latest/`

tiedostoon. Asetetaan I2C-väylän porttinumeroksi 1 ja I2C-osoitteeksi aiemmin haettu osoite, joka tässä on 0x76. Tallennetaan ja suljetaan tiedosto.

(c) Komennolla `pipenv run python3 bme280test.py` esimerkkiohjelma ajetaan virtuaaliympäristössä. Esimerkkiohjelma mittaa ympäristön lämpötilan, ilmankosteuden ja ilmanpaineen ja tulostaa lukemat näytölle.

8. Luodaan `git init` -komennolla git-varasto versionhallintaa varten. Lisätään varastoon `README.md` ja `.gitignore` tiedostot varaston kuvausta ja tarpeettomien tiedostojen sivuuttamista varten. Luodaan lisäksi etävarasto [5] GitHubiin, että kaikki muutokset ja lisäykset voidaan varmuuskopioida sinne välittömästi.
9. Luodaan tiedosto `weather.py` datankeruuohjelmaa varten ja aloitetaan ohjelman kirjoittaminen. Tästä kerrotaan tarkemmin tutkielman luvuissa 4.2.1 ja 4.2.2.
10. Tehdään tiedostosta suoritettava lisäämällä tiedoston alkuun rivi `#!/usr/bin/env python` ja lisäämällä tiedostolle suoritusoikeus `chmod +x weather.py`-komennolla.
11. Ajetaan ohjelma joko suoraan `pipenv run ./weather.py` -komennolla tai siirtymällä ensin virtuaaliympäristöön `pipenv shell` -komennolla ja ajamalla tämän jälkeen ohjelma `./weather.py`-komennolla.

A.3 Datan keruun implementointi AWS:llä

1. Luodaan AWS IoT Core -palveluun IoT-käytäntö seuraavien vaiheiden mukaisesti. Tässä oletetaan, että AWS-tunnukset on luotu aiemmin.

   (a) Siirrytään *Secure/Policies*-osioon [6] ja painetaan *Create*.

   (b) Kirjoitetaan käytännölle nimi (esim. ”Weather_Policy”) ja lisätään käytäntömääräykset (engl. *policy statement*) käyttäen IoT-käytäntöjen ohjeistuksia [7] apuna.

   (c) Ensimmäiseen käytäntömääräykseen lisätään toiminnoksi `iot:Connect`. Tämän valinnan perusteella käyttöliittymä täyttää resurssi-ARN:n lähes valmiiksi. Poistetaan ARN:n lopusta `replaceWithAClientId`-teksti ja lisätään sen ti-

5. `https://github.com/MrCliff/iot-cloud-connection-proto`
6. `https://console.aws.amazon.com/iot/home#policyhub`
7. `https://docs.aws.amazon.com/iot/latest/developerguide/iot-policies.html`

lalle `${iot:Connection.Thing.ThingName}`. Tämä sallii IoT-laitteiden käyttää vain IoT Coreen rekisteröityä laitenimeä `clientId`-tunnisteena yhteyttä muodostettaessa. RasPille tunnisteeksi ja laitteen nimeksi valitaan *RPi2*. Viimeiseksi asetetaan määräys sallivaksi *Allow*-valinnalla.

(d) Lisätään toinen käytäntömääräys vastaavasti toiminnolle `iot:Publish`. Tämänkin toimintovalinnan jälkeen resurssi-ARN täytetään lähes valmiiksi. Tällä kertaa `dt/weather/*/${iot:Connection.Thing.ThingName}`-aihemääritys lisätään `replaceWithATopic`-tekstin tilalle. Tässä pitää käyttää `*`-merkkiä villikorttina MQTT-aihesuodattimien villikorttien (+ ja #) sijaan, koska IoT Coren käytännöissä noudatetaan AWS:n IAM-käytäntöjen konventioita. Nyt käytäntömääritys sallii IoT-laitteen lähettää MQTT-viestejä vain aiheisiin, jotka ovat aihemäärityksen mukaisia. Tutkimuksen prototyypissä käytetty aihe `dt/weather/jyvaskyla/RPi2` on esimerkki tällaisesta. Viimeiseksi asetetaan tämäkin määräys sallivaksi *Allow*-valinnalla ja luodaan käytäntö painamalla *Create*.

2. Luodaan esineolio RasPi:lle.

(a) Siirrytään *Manage/Things*-osioon [8] ja painetaan *Create things*.

(b) Luodaan vain yksi laite valitsemalla *Create single thing* ja painamalla *Next*.

(c) Lisätään esineelle nimi ja jätetään muut valinnat oletuksille. Tässä nimeksi valitaan *RPi2*. Siirrytään seuraavaan painamalla *Next*.

(d) Annetaan AWS:n generoida sertifikaatti automaattisesti ja painetaan *Next*. Jos laite lisättäisiin esimerkiksi tuotantoympäristöön, olisi sille hyvä generoida sertifikaatti itse ja valita tässä oman sertifikaatin lisäys. Näin sertifikaatille voisi määritellä esimerkiksi sopivan erääntymisajan. Kirjoitushetkellä AWS:n luomissa sertifikaateissa erääntymisaika on 1.1.2050.

(e) Valitaan edellisessä vaiheessa luotu IoT-käytäntö yhdistettäväksi sertifikaattiin, ja luodaan esineolio painamalla *Create thing*.

(f) Ladataan aukeavasta ikkunasta kaikki sertifikaatti- ja avaintiedostot. Näitä tarvitaan myöhemmin laitteen yhdistämiseen, eikä avaimia voi ikkunan sulkemisen jälkeen ladata. Suljetaan ikkuna painamalla *Done*.

8. `https://console.aws.amazon.com/iot/home#thinghub`

3. Kirjaudutaan RasPiin *joonatan*-käyttäjällä, ja siirretään esineolion luonnin yhteydessä generoidut sertifikaatti- ja avaintiedostot RasPille.
4. Siirrytään hakemistoon `~/gradu/proto` ja asennetaan AWS IoT Coreen yhdistämistä varten tarvittavat Python-kirjastot `pipenv install awsiotsdk`-komennolla.
5. Lisätään tiedostossa `weather.py` olevaan datankeruuohjelmaan datan lähetystoiminnallisuus AWS-pilvialustalle. Tätä käydään ohjelmakoodin tasolla tarkemmin läpi luvussa 4.3.3.
6. Luodaan AWS:ään DynamoDB-tietokantataulu.
   (a) Avataan DynamoDB:n *Tables*-osio [9].
   (b) Luodaan taulu painamalla *Create table*.
   (c) Lisätään taululle nimi. Tässä käytetään nimeä *WeatherData*.
   (d) Lisätään taulun osiointiavaimen nimeksi *device_id* ja sen tyypiksi merkkijono (so. *String*).
   (e) Lisätään tauluun järjestysavain aktivoimalla valintaruutu *Add sort key*. Järjestysavaimen nimeksi asetetaan *timestamp* ja sen tyypiksi lukuarvo (so. *Number*).
   (f) Pidetään muuten oletusasetukset ja luodaan taulu painamalla *Create*.
7. Luodaan AWS IoT Coreen sääntö datan viennille tietokantaan.
   (a) Avataan AWS IoT Coren *Act/Rules*-osio [10] ja painetaan *Create*.
   (b) Lisätään IoT-säännölle nimi ja kuvaus. Tässä *weather_data_ddb* valitaan säännön nimeksi.
   (c) Asetetaan kyselylauseeksi `SELECT * FROM 'dt/weather/#'`. Tämän rakennetta ja toimintaa käsitellään tarkemmin luvussa 4.3.3.
   (d) Lisätään säännölle toiminto painamalla ensimmäistä *Add action* -painiketta.
   (e) Valitaan listauksesta *DynamoDBv2* ja painetaan *Configure action*.
   (f) Valitaan DynamoDB-taulujen listauksesta aiemmin luotu *WeatherData*-taulu. Lisäksi luodaan uusi IAM-rooli painamalla *Create Role* -painiketta ja antamalla sille nimi. Vaihtoehtoisesti listasta voi valita vanhan IAM-roolin ja päivittää se

9. `https://console.aws.amazon.com/dynamodb/home#tables`
10. `https://console.aws.amazon.com/iot/home#rulehub`

käyttöoikeuksiltaan sopivaksi painamalla *Update Role*. Hyväksytään lopuksi toiminto painamalla *Add action*.

(g) Tarvittaessa sääntöön voi lisätä tässä kohtaa myös muita vastaavia toimintoja sekä virhetoiminnon, joka suoritetaan virheen sattuessa. Lopuksi luodaan sääntö painamalla *Create rule*.

8. Lopuksi ajetaan datankeruuohjelma RasPilla ja annetaan sille komentoriviargumentteina tarvittavat tiedot. AWS:n osalta näitä ovat AWS-endpointin osoite, MQTT-yhteyden muodostuksessa käytettävä asiakastunniste (so. `clientId`) sekä avaintiedostojen sijainnit. Avaintiedostoista tässä tarvitaan yksityistä avainta, laitesertifikaattia sekä Amazonin juurisertifikaattia. Kun ohjelma muodostaa yhteyden ja lähettää dataa IoT Coreen onnistuneesti, voidaan siirtyä tarkastelemaan näkyykö data DynamoDB:ssä. Jos ei näy, käytetään MQTT-testiasiakasta [11] apuna vianselvityksessä.

A.4 Datan keruun implementointi Azurella

1. Luodaan IoT Hub ja resurssiryhmä Azureen seuraavien vaiheiden mukaisesti. Tässä oletetaan, että Azure-tunnukset on luotu aiemmin.

   (a) Siirrytään Azure-portaalissa IoT Hub -listaukseen [12] esimerkiksi yläreunan hakukentän kautta hakulauseella ”iot hub”.

   (b) Lisätään uusi IoT Hub painamalla *Create*.

   (c) Valitaan Azure-tilaus pudotusvalikosta. Luodaan tilauksen alle uusi resurssiryhmä painamalla resurssiryhmien pudotusvalikon alapuolelta *Create new* ja antamalla ryhmälle nimi. Tässä asetetaan nimeksi *gradu*.

   (d) Annetaan IoT Hubille nimi. Tässä asetetaan nimeksi *gradu-iot-hub*. Valitaan pudotusvalikosta IoT Hubille myös maantieteellinen alue, johon se perustetaan.

   (e) Siirrytään *Management*-välilehdelle, ja valitaan hinnoittelu- ja skaalaustasoksi *F1: Free tier*, joka vastaa toiminnoiltaan *standard tier* -tasoa, mutta pienemmillä resursseilla.

11. `https://console.aws.amazon.com/iot/home#test`
12. `https://portal.azure.com/#blade/HubsExtension/BrowseResource/resourceType/Microsoft.Devices%2FIotHubs`

(f) Jätetään muut valinnat oletuksille myös muilta välilehdiltä, painetaan *Review + create* ja sitten *Create*.

2. Kun IoT Hub on luotu ja käyttöönotto on valmis, avataan IoT Hub esimerkiksi painamalla *Go to resource*, ja rekisteröidään RasPi seuraavien vaiheiden mukaisesti.

   (a) Siirrytään IoT-laitteiden listaukseen vasemman reunan valikosta *Explorers / IoT devices*, ja lisätään uusi laite painamalla *Add Device*.
   (b) Annetaan laitteelle nimi. Tässä nimeksi annetaan *RPi2*.
   (c) Valitaan autentikaation tyypiksi symmetrinen avain, ja annetaan palvelun generoida avaimet automaattisesti valitsemalla *Auto-generate keys* -valintaruutu.
   (d) Jätetään muut valinnat oletuksille ja painetaan *Save*.
   (e) Avataan laitteen tiedot painamalla sen nimeä, ja kopioidaan ensisijainen avain leikepöydälle *Primary Key* -kentästä.

3. Konfiguroidaan IoT Hubin oletusreitti seuraavien vaiheiden mukaisesti.

   (a) Siirrytään reititysnäkymään IoT Hubin vasemman reunan *Messaging / Message routing* -valikosta. Lisätään uusi reitti painamalla *Add*.
   (b) Annetaan reitille nimi (esim. *weather-route*) ja valitaan sen päätepisteeksi *events*. Asetetaan datan lähteeksi laitteen telemetriaviestit (engl. *Device Telemetry Messages*). Vaihdetaan reitityskyselyksi `IS_STRING(location)`. Tätä käsitellään tarkemmin luvussa 4.3.5. Annetaan muiden asetusten olla oletuksilla ja viimeistellään lisäys painamalla *Save*.

4. Kirjaudutaan RasPiin *joonatan*-käyttäjällä, ja asetetaan IoT Hubista kopioitu avain `AZURE_DEVICE_PRIMARY_KEY` -nimisen ympäristömuuttujan arvoksi. Tämän voi tehdä esimerkiksi luomalla datankeruuohjelmalle käynnistysskriptin, jossa ympäristömuuttuja asetetaan ennen ohjelman ajamista.
5. Siirrytään hakemistoon `~/gradu/proto` ja asennetaan Azure IoT Hubin IoT device SDK -kirjasto Pythonille komennolla `pipenv install azure-iot-device`.
6. Lisätään `weather.py`-tiedoston datankeruuohjelmaan datan lähetys Azureen. Tätä käsitellään tarkemmin lähdekoodin tasolla luvussa 4.3.5.
7. Siirrytään luomaan Cosmos DB tietokanta seuraavien vaiheiden mukaisesti.

(a) Siirrytään Azure-portaalissa Cosmos DB -tietokantatilien listaukseen [13] esimerkiksi yläreunan hakukentän kautta hakulauseella ”cosmos db”.

(b) Lisätään uusi Cosmos DB -tietokantatili painamalla *Create*.

(c) Valitaan Core (SQL) API painamalla vastaavaa *Create*-painiketta.

(d) Valitaan pudotusvalikoista sama Azure-tilaus ja resurssiryhmä kuin IoT Hubin kohdalla. Kirjoitetaan tietokantatilille nimi (esim. *gradu-cosmos-db-account*), ja valitaan maantieteelliseksi alueeksi sama kuin IoT Hubilla.

(e) Jätetään muut valinnat oletuksille myös seuraavilla sivuilla, ja painetaan ensin *Review + create* ja sitten *Create*.

(f) Kun Cosmos DB -tietokantatili on luotu ja käyttöönotto on valmis, avataan se esimerkiksi painamalla *Go to resource*.

(g) Lisätään uusi tietokanta ja säiliö painamalla yläreunan *Add Container* -painiketta.

(h) Annetaan tietokannalle tunniste (esim. *weather-db*), ja jätetään muut tietokannan asetukset oletuksille.

(i) Annetaan säiliölle tunniste (esim. *weather*), ja asetetaan `/device_id` osiointiavaimeksi. Jätetään muut arvot oletuksille ja viimeistellään luonti painamalla *OK*.

8. Luodaan Stream Analytics -eräajo seuraavien vaiheiden mukaisesti.

(a) Siirrytään Azure-portaalissa Stream Analytics -eräajojen listaukseen [14] esimerkiksi yläreunan hakukentän kautta hakulauseella ”stream analytics job”.

(b) Lisätään uusi Stream Analytics -eräajo painamalla *Create*.

(c) Annetaan eräajolle nimi (esim. *WeatherJob*), ja valitaan pudotusvalikoista sille sama Azure-tilaus ja resurssiryhmä kuin IoT Hubin ja Cosmos DB:n kohdalla. Valitaan eräajon maantieteelliseksi alueeksi myös sama kuin muilla resursseilla.

(d) Valitaan sopiva määrä virtausyksiköitä viestivirtojen käsittelijöiksi. Kokeilua varten yksi riittää. Jätetään muut valinnat oletuksille, ja viimeistellään eräajon luonti painamalla *Create*.

13. `https://portal.azure.com/#blade/HubsExtension/BrowseResource/resourceType/Microsoft.DocumentDb%2FdatabaseAccounts`

14. `https://portal.azure.com/#blade/HubsExtension/BrowseResource/resourceType/Microsoft.StreamAnalytics%2FStreamingJobs`

(e) Kun Stream Analytics -eräajo on luotu ja käyttöönotto on valmis, avataan se esimerkiksi painamalla *Go to resource*.

(f) Lisätään eräajolle sisääntulo siirtymällä ensin vasemman reunan valikosta *Job topology / Inputs* -sivulle ja valitsemalla sitten näkymän yläreunasta *Add stream input / IoT Hub*.

(g) Lisätään sisääntulolle alias (esim. *weatherinput*), ja valitaan pudotusvalikoista Azure-tilaus ja aiemmin luotu IoT Hub. Valitaan seuraavaksi pudotusvalikosta jaetun käyttöoikeuskäytännön nimeksi (engl. *Shared access policy name*) *service*. Valitaan päätepisteeksi *Messaging*. Osiointiavaimeksi voi asettaa `device_id`, jota käytetään myös Cosmos DB -säiliön osiointiavaimena. Tämän asettaminen parantaa suorituskykyä, jos viestejä tulee paljon usealta laitteelta. IoT-prototyypissä eroa ei kuitenkaan huomaa. Jätetään lopuksi muut valinnat oletuksille ja viimeistellään sisääntulon lisääminen painamalla *Save*.

(h) Lisätään eräajolle ulostulo siirtymällä ensin vasemman reunan valikosta *Job topology / Inputs* -sivulle ja valitsemalla sitten yläreunasta *Add / Cosmos DB*.

(i) Lisätään ulostulolle alias (esim. *weatheroutput*), ja valitaan pudotusvalikoista Azure-tilaus ja aiemmin luotu Cosmos DB -tietokantatili. Valitaan alemmista pudotusvalikoista aiemmin luotu tietokanta ja säiliö. Jätetään muut valinnat oletuksille ja viimeistellään lisääminen painamalla *Save*.

(j) Muokataan eräajon kyselyä siirtymällä vasemman reunan valikosta *Job topology / Query* -sivulle. Vaihdetaan `YourOutputAlias`-tekstin tilalle `weatheroutput` ja `YourInputAlias`-tekstin tilalle `weatherinput`. Nyt kysely näyttää seuraavalta:

```
SELECT *
INTO [weatheroutput]
FROM [weatherinput]
```

Lopuksi tallennetaan kysely painamalla *Save query*.

(k) Ajetaan eräajo siirtymällä *Overview*-sivulle ja painamalla *Start*.

9. Kun resurssit on luotu ja Stream Analytics -eräajo on päällä, voi datankeruuohjelman ajaa RasPilla. Ajoa ennen IoT Hubista kopioitu symmetrinen autentikaatioavain tulee asettaa ympäristömuuttujaan, kuten edellä kerrottiin IoT Hubin luonnin jälkeen vai-

heessa 4. Datankeruuohjelmalle annetaan komentoriviargumentteina tarvittavat tiedot, joita Azuren osalta ovat IoT Hubin nimi sekä sinne rekisteröity RasPin laitetunniste (so. `device_id`). Kun ohjelma muodostaa yhteyden ja lähettää dataa IoT Hubiin onnistuneesti, voidaan katsoa näkyykö data Cosmos DB -säiliössä. Jos ei näy, debuggauksessa voidaan käyttää apuna IoT Hubin ja Stream Analyticsin etusivuilla näkyviä viestien määristä kertovia kuvaajia. Lisäksi IoT Hubin reititysnäkymässä voidaan testata reitityskyselyitä esimerkkidatalla, ja Stream Analytics -eräajon kyselyn muokkausnäkymässä sisääntuloja, ulostuloja ja itse kyselyä voidaan testata sisääntuloista tulevalla datalla.

A.5 Datan keruun implementointi GCP:llä

1. Luodaan Cloud IoT Core -rekisteri seuraavien vaiheiden mukaisesti. Tässä oletetaan, että GCP-tunnukset on luotu aiemmin.

   (a) Siirrytään GCP:n web-konsolissa Cloud IoT Core -palvelun rekisterilistaukseen [15] esimerkiksi yläreunan hakukentän kautta hakulauseella ”iot core”.
   (b) Aktivoidaan *Google Cloud IoT API* painamalla *Enable*.
   (c) Luodaan uusi IoT Core -rekisteri painamalla *Create registry*.
   (d) Annetaan rekisterille tunniste (esim. *gradu-registry*) ja valitaan maantieteellinen alue, johon rekisteri perustetaan.
   (e) Luodaan uusi Pub/Sub-aihe avaamalla aiheen valitsin ja painamalla *Create a topic*.
   (f) Annetaan aiheelle tunniste (esim. *weather*) ja luodaan aihe painamalla *Create topic*.
   (g) Poistetaan edistyneistä asetuksista valinta HTTP-protokollalta, koska sitä ei käytetä tutkimuksen IoT-prototyypissä.
   (h) Viimeistellään luonti painamalla *Create*.
2. Generoidaan avainpari RasPilla seuraavien vaiheiden mukaisesti.

   (a) Kirjaudutaan RasPiin *joonatan*-käyttäjällä ja siirrytään sopivaan hakemistoon,

15. `https://console.cloud.google.com/iot/registries`

johon avainparin voi generoida.

(b) Seurataan Cloud IoT Coren dokumentaation ohjeita avainparin generointiin [16]. Tässä generoidaan 2048-bittinen RSA-avain sekä itse allekirjoitettu X.509-sertifikaatti seuraavalla komennolla:

```
openssl req -x509 -nodes -newkey rsa:2048 -subj "/CN=unused" \
    -keyout rsa_private.pem -out rsa_cert.pem
```

Yksityinen avain tallennetaan tiedostoon `rsa_private.pem` ja julkinen sertifikaatti tiedostoon `rsa_cert.pem`.

(c) Lisäksi ladataan Googlen juuri-CA-sertifikaattipaketti [17] esimerkiksi samaan hakemistoon muiden avainten kanssa.

3. Lisätään RasPi IoT-laitteeksi IoT Core -rekisteriin seuraavien vaiheiden mukaisesti.

   (a) Avataan vaiheessa 1 luotu rekisteri, siirrytään *Devices*-välilehdelle ja lisätään uusi laite painamalla *Create a device*.

   (b) Asetetaan laitteelle tunniste. Tässä annetaan tunnisteeksi *RPi2*.

   (c) Kopioidaan vaiheessa 2 generoidun avainparin julkinen avain tai sertifikaatti *Authentication*-osion alla olevaan *Public key value* -tekstilaatikkoon. Tässä sertifikaatti kopioidaan RasPilta `rsa_cert.pem`-tiedostosta. Asetetaan myös avaimen formaatti oikeaksi *Public key format* -pudotusvalikosta. Tässä sertifikaatin formaatti on *RS256_X509*.

   (d) Viimeistellään IoT-laitteen lisäys painamalla *Create*.

4. Kirjaudutaan RasPiin *joonatan*-käyttäjällä ja siirrytään `~/gradu/proto`-hakemistoon. Asennetaan MQTT-yhteyden muodostuksessa tarvittavat kirjastot komennolla `pipenv install pyjwt cryptography paho-mqtt`.

5. Lisätään `weather.py`-tiedoston datankeruuohjelmaan datan lähetys Google Cloud Platform -alustalle. Tätä käsitellään tarkemmin lähdekoodin tasolla luvussa 4.3.7.

6. Luodaan Cloud Firestore -tietokanta seuraavien vaiheiden mukaisesti.

16. https://cloud.google.com/iot/docs/how-tos/credentials/keys

17. https://cloud.google.com/iot/docs/how-tos/mqtt-bridge#downloading_mqtt_server_certificates

(a) Siirrytään GCP:n web-konsolissa Firestore-palveluun [18] esimerkiksi yläreunan hakukentän kautta hakulauseella ”firestore”.

(b) Valitaan Firestoren toimintamoodiksi natiivi painamalla *Select Native mode*. Toimintamoodeja on selitetty tarkemmin luvussa 4.3.7.

(c) Valitaan Firestore-tietokannan maantieteellinen sijainti ja viimeistellään luonti painamalla *Create database*.

7. Luodaan Google Cloud Functions -pilvifunktio seuraavien vaiheiden mukaisesti.

(a) Siirrytään GCP:n web-konsolissa Cloud Functions -palveluun [19] esimerkiksi yläreunan hakukentän kautta hakulauseella ”functions”.

(b) Luodaan uusi pilvifunktio painamalla *Create function*.

(c) Annetaan funktiolle nimi (esim. *save-weather*) ja valitaan sille maantieteellinen sijainti.

(d) Asetetaan pilvifunktion käynnistimen (engl. *Trigger*) tyypiksi Cloud Pub/Sub ja valitaan Pub/Sub-aiheeksi vaiheessa 1 luotu aihe. Tallennetaan käynnistimen asetukset painamalla *Save*.

(e) Jätetään muut asetukset oletusarvoille, ja siirrytään seuraavalle sivulle painamalla *Next*.

(f) Asetetaan pilvifunktion ajoympäristöksi Python 3.9. Ajoympäristön valinnan jälkeen pilvifunktiolle luodaan automaattisesti ohjelmarunko. Tähän kuuluu ohjelmakoodin sisältävä `main.py`-tiedosto sekä `requirements.txt`-tiedosto, joka sisältää listauksen riippuvuuksista.

(g) Käyttöliittymä pyytää ajoympäristövalinnan jälkeen ottamaan *Cloud Build API* -rajapinnan käyttöön GCP-projektissa, jos se ei vielä ole käytössä. Otetaan tämä käyttöön painamalla *Enable API* ja aukeavalla sivulla *Enable*.

(h) Palataan pilvifunktion asetussivulle ja lisätään `requirements.txt`-tiedostoon rivi `google-cloud-firestore`. Tämä mahdollistaa Firestore-tietokannan käytön pilvifunktiosta.

(i) Muokataan `main.py`-tiedostoa luvussa 4.3.7 kuvatulla tavalla. Asetetaan ohjelman aloituspisteeksi `save_weather`-metodi, jota kutsutaan käynnistimen lau-

18. `https://console.cloud.google.com/firestore`
19. `https://console.cloud.google.com/functions`

etessa.

(j) Viimeistellään pilvifunktion luonti ottamalla se käyttöön *Deploy*-painikkeella.

8. Kun resurssit on luotu ja pilvifunktio on päällä, ajetaan datankeruuohjelma RasPilla. Ohjelmalle annetaan komentoriviargumentteina tarvittavat tiedot. GCP:n osalta näitä ovat GCP-projektin, Cloud IoT Core -rekisterin ja IoT-laitteen tunnisteet, IoT Core -rekisterin maantieteellinen sijainti sekä avaintiedostojen sijainnit ja yksityisen avaimen formaatti. Lisäksi tarvitaan isäntänimi ja portti, johon MQTT-yhteys avataan. Oletuksena isäntänimi on *mqtt.googleapis.com* ja portti on *8883*. Avaintiedostoista tarvitaan vaiheessa 2 generoidun avainparin yksityinen avain sekä samassa vaiheessa ladattu Googlen juuri-CA-sertifikaatti. Yksityisen avaimen formaatiksi valitaan joko *RS256* tai *ES256* sen mukaan, onko avaimen luontialgoritmina käytetty RSA:ta vai elliptisiä käyriä.
9. Kun datankeruuohjelma muodostaa yhteyden ja lähettää dataa Cloud IoT Coreen onnistuneesti, katsotaan näkyykö data Firestore-tietokannassa. Jos ei näy, voidaan debuggauksessa käyttää apuna IoT Coren IoT-laitteen tiedoissa näkyviä viimeisimmän toiminnan aikaleimoja ja *Monitoring*-välilehdellä olevia IoT-laitteiden aktiivisuudesta kertovia kuvaajia. Lisäksi Cloud Functions -palvelun pilvifunktiot keräävät lokitietoa, jota pääsee katsomaan pilvifunktion tiedoista *Logs*-välilehdeltä.