SAVONIA
ammattikorkeakoulu

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN JA LIIKENTEEN ALA

SISÄILMAN LAATUTIEDON MITTAAMINEN JA JAKAMINEN

TEKIJÄ Lauri Hyvärinen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä Lauri Hyvärinen	
Työn nimi Sisäilman laatutiedon mittaaminen ja jakaminen	
Päiväys 7. syyskuuta 2022	Sivumäärä/Liitteet 45/0
Toimeksiantaja Savonia-ammattikorkeakoulu	

Tiivistelmä:

Opinnäytetyön tavoitteena oli valmistaa IoT-järjestelmä, jota voidaan käyttää tietotekniikan tutkinto-ohjelman kurssien opetuksen tukena Savoniassa. Järjestelmä koostuu yhdestä fyysisestä laitteesta ja taustapalvelusta. Laite mittaa sisäilman laatuun vaikuttavia tekijöitä. Taustapalvelu tallentaa nämä tiedot tietokantaan ja muodostaa mittauksista tiedostoja Azuren Data Lake -palveluun.

Työssä perehdyttiin erilaisiin antureihin ja Azuren pilvipalveluihin. Fyysinen laitteisto koottiin Raspberry Pi:stä ja neljästä erillisestä anturista. Antureiden valinnassa piti ottaa huomioon niiden ominaisuudet ja hinnat sekä liitettävyys Raspberry Pi:hin. Azuressa taustapalvelu koostuu viidestä erillisestä pilvipalvelusta. Jokainen palvelu luotiin ja konfiguroitiin toimimaan yhdessä.

Lopputuloksena saatiin onnistunut järjestelmä: tiedonkulku antureilta Data Lake:een saakka toimii ilman virheitä. Data Lake:n hierarkisessa nimiavaruudessa voidaan seurata uusia tiedostoja reaaliajassa, mikä tarjoaa tasaisen tietovirran Big data -kurssin opetukseen. Järjestelmän tuottamia tiedostoja voidaan käyttää materiaalina Business Intelligence -kurssin opetuksessa. Rakennettua järjestelmää on myös helppo lähteä laajentamaan tulevaisuudessa.

Avainsanat

Azure, Big data, IoT, sisäilman laatu

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport	
Degree Programme Degree Programme in Information Technology	
Author Lauri Hyvärinen	
Title of Thesis Measuring and Sharing Indoor Air Quality Data	
Date 7. September 2022	Pages/Appendices 45/0
Client Organisation /Partners Savonia University of Applied Sciences	

Abstract

The aim of this thesis was to create an IoT system to complement the tuition provided in the Degree Programme in Information Technology at Savonia UAS. The system is composed of one physical device and a background service built in Azure. The device measures information about indoor air quality. The background service saves this information in a database and aggregates the information into files that are shared in Azure Data Lake.

The work was composed of researching different sensors and Azure cloud services. The physical equipment was built with Raspberry Pi and four different sensors. The sensors were chosen based on their features, costs and connectivity to the Raspberry Pi. The background service in Azure is composed of five different cloud services. Each service was created and configured to work seamlessly together.

As a result, the work is a success. Information flow from the sensors to the Data Lake works flawlessly. Files coming in to the Data Lake can be monitored in real-time, providing a datastream for Big data -course. The same files can also be used as material for Business Intelligence -course. In the future the system can be expanded upon, to allow new features and use cases to be added into the system.

Keywords

Azure, Big data, IoT, indoor air quality

SISÄLTÖ

KUVALUETTELO

1 JOHDANTO

Sisäilman laadun merkitys on kasvanut viime aikoina ja tietoisuus sen terveysvaikutuksista on lisääntynyt. Koronapandemian aikana viimeistään havahduttiin siihen, kuinka tärkeää kunnollinen sisäilma ja sen vaihtuvuus on terveydellemme (Yle, 2021). Sisäilman laatua voidaan kartoittaa esim. eri antureista kootulla IoT-laitteella (Internet of Things), joka voi toimia osana suurempaa IoT-järjestelmää.

IoT on ollut iso trendi jo jonkin aikaa, etenkin teollisuuden ja terveydenhuollon parissa. Erityisesti teollisuudessa IoT on otettu avosylin vastaan, mitä hyvin kuvaa IoT:n alakäsite IIoT (Industrial Internet of Things).

Opinnäytetyön tilaajana toimii Savonia-ammattikorkeakoulu. Opinnäytetyössä luodaan IoT-järjestelmä. Järjestelmään kuuluu yhden sisäilmaa mittaavan IoT-laitteen prototyypin suunnittelu ja rakentaminen sekä taustajärjestelmä mittaustiedon käsittelyyn ja jakamiseen. Prototyyppi rakennetaan antureista ja Raspberry Pi:stä. Taustajärjestelmä käsittelee ja tallentaa laitteen tuottamaa tietoa ja se rakennetaan käyttäen Azuren pilvipalveluita. Opinnäytetyötä voidaan käyttää esim. Big Data ja Business Intelligence (BI) -kurssien opetuksessa. Big Data -kurssilla opetellaan mm. käsittelemään reaaliaikaisia tietovirtoja. BI-kurssilla luodaan raportteja suuresta määrästä tietoa.

2 LYHENTEET

ABFS Azure Blob Filesystem on ajuri, jolla voidaan muodostaa yhteys Azuren Blob Storage:een. Se on suositeltu ja tehokkain tapa käyttää Blob Storage:a ohjelmallisesti.

API Application Programming Interface eli ohjelmointirajapinta. Rajapinta mahdollistaa eri ohjelmien tai laitteiden käytön toisesta ohjelmasta käsin.

Blob Blob eli Binary Large Object tarkoittaa yleisesti suurta määrää binääri-muotoista tietoa, kuten kuvat, videot ja äänitiedostot.

CI/CD Continious Integration (CI) ja Continuous Delivery (CD) ovat ohjelmistokehityksen työtapoja ja niitä palvelevia ohjelmia, joiden avulla ohjelmaan tehtävät muutokset integroidaan ohjelmaan, ohjelma testataan ja levitetään automaattisesti (Tomohito, 2017).

FaaS Functions-as-a-Service eli funktioiden suorittaminen palveluna. Palvelu tarjoaa funktioiden suorittamiseen alustan, jossa resurssien hallinnasta ja allokoinnista vastaa palveluntarjoaja.

GPIO GPIO eli General Purpose I/O tarkoittaa yleiskäyttöistä pinniä, joka voi joko vastaanottaa tai lähettää signaalia.

HDFS Hadoop Distributed File System on hajautettu tiedostojärjestelmä. Se soveltuu suurien tietomäärien käsittelyyn ja se on suunniteltu toimimaan edullisella laitteistolla.

I^2C Inter-Integrated Circuit on viestintäspesifikaatio, joka toimii kahdella signaalilla, SDA (Serial Data) ja SCL (Serial Clock). Protokollassa tieto lähetetään paketteina.

IAQ IAQ on lyhenne englannin kielen sanoista Indoor Air Quality eli sisäilman laatu.

IaaS Infrastructure-as-a-Service eli infrastruktuuri palveluna. Palvelu tarjoaa kuluttajalle erilaisia mahdollisuuksia verkkojen ja virtuaalikoneiden määrittämiseen ja ajamiseen. Kuluttajan ei tarvitse huolehtia fyysisistä ylläpitotöistä, kuten rikkoutuneiden prosessorien tai kovalevyjen korvaamisesta.

IoT Internet of Things eli ”laitteiden” tai ”asioiden” internet. Fyysisten laitteiden liittäminen internettiin.

IIoT Industrial Internet of Things, eli teollinen laitteiden internet. Erilaisten tuotantolaitteiden liittäminen verkkoon.

PaaS Platform-as-a-Service tarjoaa jonkun alustan pilvipalveluna. Tämän alustan päälle kuluttaja voi rakentaa oman ratkaisunsa. Kuluttajan ei tarvitse huolehtia fyysisistä laitteista eikä itse alustan ylläpidosta.

PM Particulate Matter eli pienhiukkaset kuvaavat pienten hiukkasten määrää ilmassa. PM2.5-hiukkasten halkaisija on alle 2.5 µm. Pienhiukkasilla on negatiivisia terveysvaikutuksia (Hengitysliitto, ei pvm).

PWM Pulse-Width Modulation eli pulssinleveysmodulaatio. Se on digitaalinen signaali, jolla on kaksi tilaa, ON ja OFF. Pulssin kesto pidetään vakiona, mutta ON ja OFF tilojen suhdetta voidaan muuttaa pulssin aikana.

SaaS Software-as-a-Service tarjoaa tietyn ohjelmiston palveluna. Palveluntarjoaja hoitaa kaiken ohjelmiston ylläpidon ja kuluttajan tarvitsee vain käyttää palvelua.

SBC SBC eli Single Board Computer tarkoittaa yhden piirilevyn tietokonetta. Kaikki tietokoneen komponentit ovat yhdellä piirilevyllä, jolloin tietokone mahtuu pieneen tilaan. Pienimmät SBC:t ovat noin luottokortin kokoisia pohjapinta-alaltaan.

SDK Software Development Kit eli ohjelmistokehityspaketti on paketti tai kokoelma paketteja erilaisista luokista ja funktioista. Niiden avulla ohjelmien kehitys on nopeampaa, kun kaikkea ei tarvitse tehdä itse. Esimerkiksi Azure SDK pythonille tarjoaa monia eri paketteja Azuren eri palveluiden käyttöön.

SPI Serial Programming Interface on viestintäspesifikaatio, joka toimii neljällä signaalilla. Iso ero I^2C:n kanssa on se, että SPI:ssä tieto lähetetään yhtenä jonona pakettien sijasta.

TLS TLS eli Transport Layer Security on eräänlainen suojattu yhteys. TLS on korkeasti kustomoitava protokolla. Tunnistautumiseen ja salaukseen voidaan käyttää erilaisia salausalgoritmeja, kuten RSA (tunnistautumiseen) ja AES (salaukseen). Algoritmien käytöstä sovitaan yhteyden alussa tapahtuvan kättelyn aikana (engl. ”TLS handshake”).

UART Universal Asynchronous Receiver Transmitter on sarjaliikennepiiri, joka muuntaa sarjamuotoista tietoa rinnakkaismuotoiseksi tiedoksi ja takaisin.

VOC Volatile Organic Compounds eli haihtuvat orgaaniset yhdisteet. VOC-yhdisteet voivat aiheuttaa mm. päänsärkyä sekä silmien ja limakalvojen ärsytysoireita (Hengitysliitto, ei pvm).

WASB Windows Azure Storage Blob (WASB) on ajuri, jota voidaan käyttää Azuren Blob Storage:een yhdistämisessä. Se on kuitenkin poistumassa käytöstä ja sen on korvannut ABFS-ajuri.

3 SAVONIA-AMMATTIKORKEAKOULU

Savonia-ammattikorkeakoululla on kolme kampusta Pohjois-Savossa: Kuopiossa, Iisalmessa ja Varkaudessa. Työntekijöitä Savonialla on noin 530 ja opiskelijoita yli 7000 (Savonia-ammattikorkeakoulu, ei pvm). Savonia tarjoaa laadukasta opetusta kuudella eri alalla, kuten sosiaali- ja terveys, liiketalous ja tekniikka. Savonian arvoihin kuuluvat yhteisöllisyys, rohkeus ja luotettavuus, ja kaikessa toiminnassa noudatetaan kestävän kehityksen periaatteita (Savonia-ammattikorkeakoulu, 2020). Vuonna 2020 Savonia oli ensimmäinen ammattikorkeakoulu, joka sai Great Place To Work -sertifikaatin kaudelle 2020-2021. Uusi sertifikaatti myönnettiin myös 2021-2022 kaudelle.

Kuva 1. Savonia-ammattikorkeakoulun värillinen logo (Savonia-ammattikorkeakoulu, ei pvm)

4 TEORIAOSA

4.1 Sisäilma

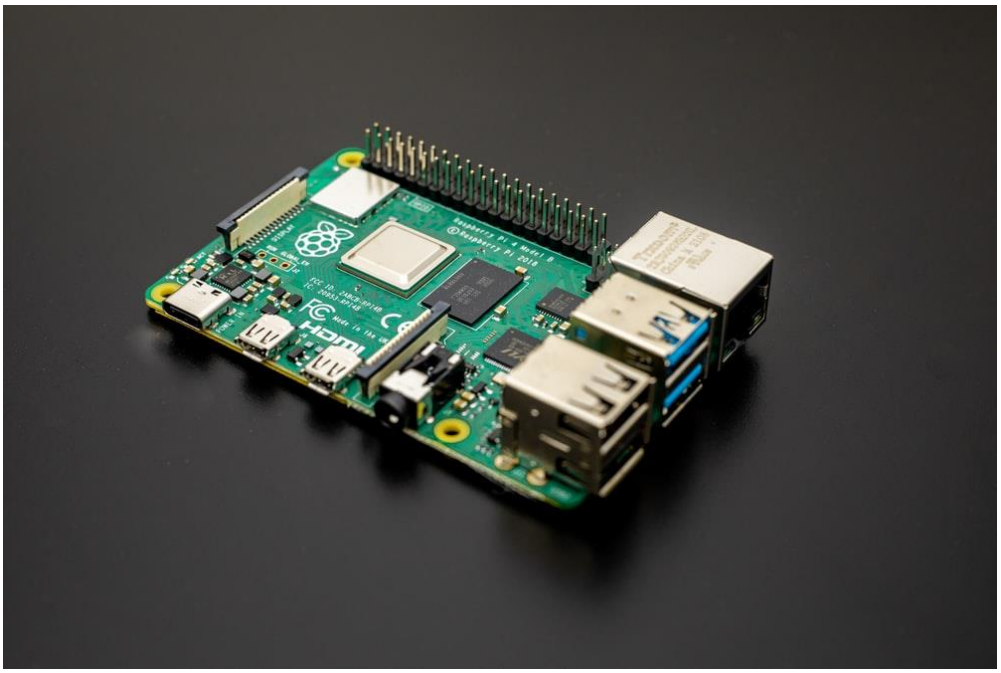
Sisäilmalla on tutkitusti vaikutusta ihmisiin, jotka sitä hengittävät. Huono sisäilma voi aiheuttaa mm. päänsärkyä, hengitysteiden ärsyyntymistä ja kuivia silmiä (Tran V. V., 2020). Viime aikoina erityisesti sisäilman kautta tarttuvat taudit ovat olleet esillä uutisissa ja tutkimuksissa koronapandemian takia. Hyvällä ilmanvaihdolla on mahdollista torjua Covid-19:n kaltaisia hengitysteitse leviäviä epidemioita (Yle, 2022).

Sisäilman laatua heikentävät erilaiset saasteet. Tärkeitä sisäilman laadun mittareita ovat mm. pienhiukkasten määrä (PM), CO_2 ja CO -pitoisuudet, VOC-yhdisteiden (haihtuvat orgaaniset yhdisteet) määrä ja typen oksidit (NO ja NO_2). Lisäksi sisäilman miellyttävyyteen vaikuttaa vahvasti lämpötila ja ilman kosteus (Nan Ma, 2021; Chau-Ren Jung, 2021).

4.2 Laitteet ja anturit

4.2.1 Raspberry Pi

Raspberry Pi on suosittu yhden piirilevyn tietokone (SBC, Single-Board Computer). Sen avulla on mahdollista luoda nopeita prototyyppejä rakennettavista laitteista ja testata niiden toimivuutta. Jos laitteita valmistettaisiin useampi, anturit voitaisiin liittää paljon halvempaan elektroniikkaan, kuten Arduino Nanoon. Tämän työn tarkoitukseen Raspberry Pi sopii kuitenkin erinomaisesti. Anturit voidaan liittää suoraan Raspberry Pi:n GPIO-pinneihin (General Purpose I/O). Kuvassa 2 on esillä Raspberry Pi 4 Model B, jollaista työssä käytettiin.



Kuva 2. Raspberry Pi 4 Model B (Ponnala, 2021)

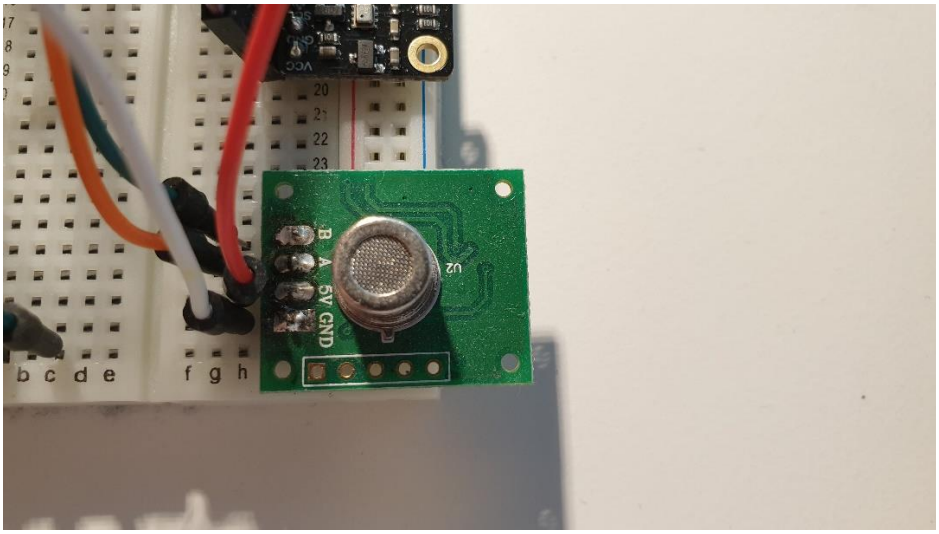
4.2.2 ZP01-MP503

ZP01-MP503 on anturi (Kuva 3), joka havainnoi ilmassa olevia VOC-yhdisteitä. Valmistajan dokumentaation (Zhengzhou Winsen Electronics Technology Co., Ltd., 2014, p. 3) mukaan anturi

havainnoi mm. asetonia, tolueenia, formaldehydia, metaania ja alkoholia. Anturi toimii 5V:n jännitteellä ja elinikä on yli 5 vuotta. Anturin virrankulutus on alle 60mA. Anturi ilmaisee VOC-yhdisteiden tason neliportaisella asteikolla (0–3) kahden 5V:n signaalin avulla. Asteikko näkyy alla olevassa taulukossa (Taulukko 1).

Taulukko 1. ZP01-MP503 anturin VOC-tasot (Zhengzhou Winsen Electronics Technology Co., Ltd., 2014, p. 4)

A	B	Porras	VOC-taso
0V	0V	0	Puhdas
0V	+5V	1	Kevyt
+5V	0V	2	Kohtalainen
+5V	+5V	3	Vakava

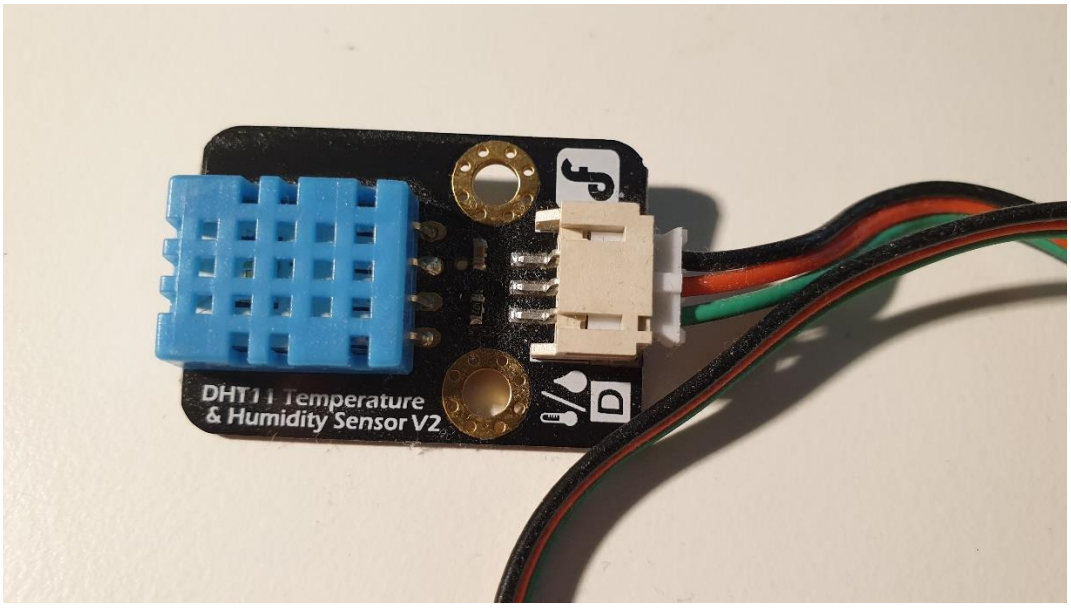


Kuva 3. ZP01-MP503 kytkettynä koekytkentäalustaan

4.2.3 DHT-11

DHT-11-anturi (Kuva 4) mittaa ilman lämpötilaa ja ilmankosteutta. Ilmankosteus voi aiheuttaa paljon ongelmia ihmisille ja rakennuksille, jos se on liian korkea tai matala. Matala ilmankosteus näkyy erityisesti ihmisissä ihon, silmien ja huulten kuivumisessa ja ärtymisessä. Korkea ilmankosteus taas useammin aiheuttaa ongelmia rakennuksiin, kuten kondensaatiota ikkunoissa ja muissa rakenteissa. Lämpötila ja ilmankosteus vaikuttavat paljon siihen, miten mukavaksi ihminen mieltää sisäilman.

Anturi toimii 5V:n jännitteellä ja kuluttaa hyvin vähän virtaa. Mitatessa virrankulutus on välillä 0,5mA – 2,5mA. Valmiustilassa anturin virrankulutus 100µA - 150µA. Anturin resoluutio on 1°C ja 1 %RH (suhteellinen kosteus) (Mouser, ei pvm, pp. 3-4).



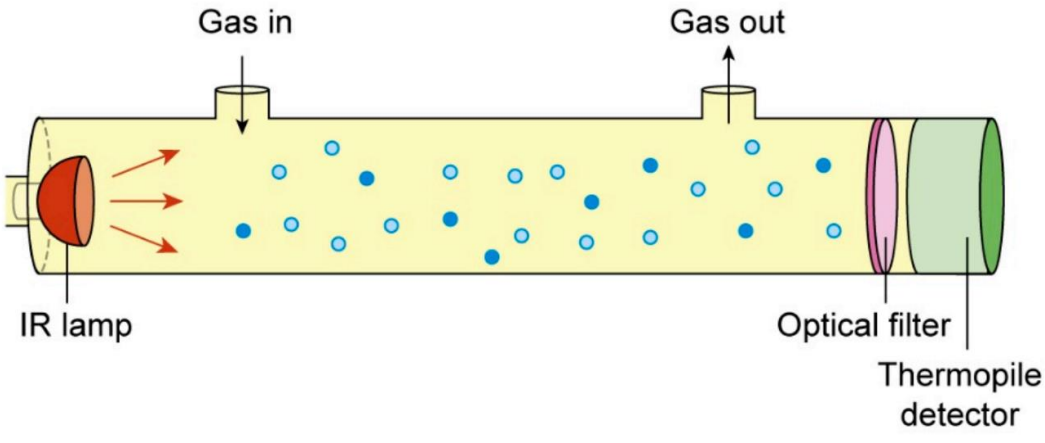
Kuva 4. DFRobotin valmistama DHT-11-anturi

4.2.4 MH-Z19B

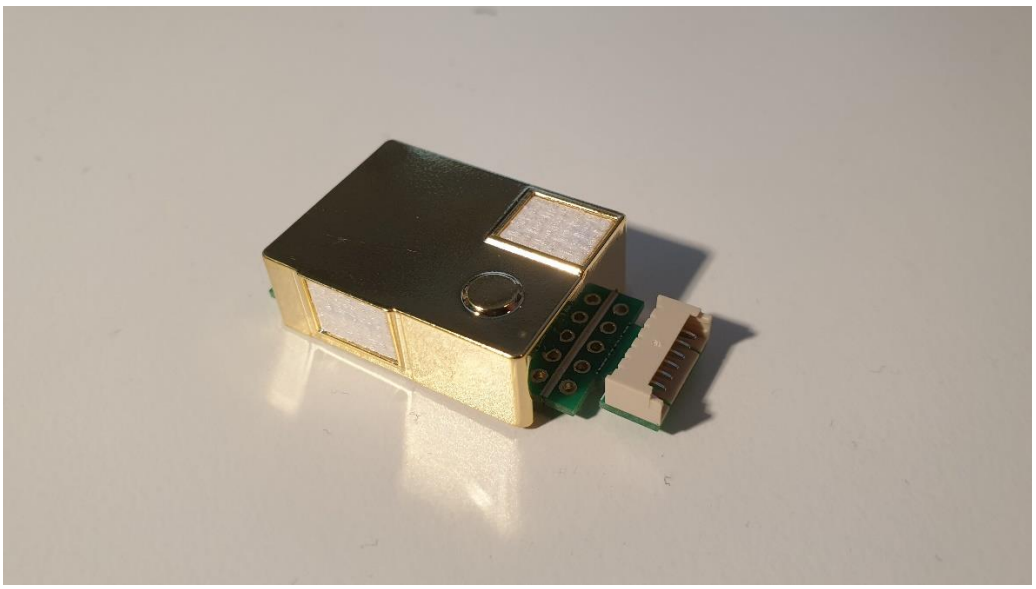
MH-Z19B (Kuva 6) on tarkka anturi hiilidioksidin mittaamiseen. Anturin toimintaperiaate on ”ei-dispersiivinen infrapuna” (engl. ”non-dispersive infrared”) eli NDIR. Anturissa on pieni kammio, jossa on ilmaa. Kammion toisessa päässä on infrapunalamppu, joka lähettää infrapunasäteilyä kohti kammion toista päätä. Kammiossa oleva hiilidioksidi absorboi infrapunasäteilyä ennen sen pääsyä kammion toiseen päähän, jolloin hiilidioksidin määrä voidaan arvioida läpi päässeen infrapunasäteilyn perusteella (Kuva 5).

Ihminen laskee ilmanlaatua tilassa oleskellessaan ensisijaisesti uloshengityksen mukana tulevan hiilidioksidin kautta. Siksi hiilidioksidi on tärkeä ottaa huomioon ilmanlaadun tarkkailussa niissä tiloissa, jotka ovat tarkoitettu ensisijaisesti ihmisille.

MH-Z19B toimii 5V:n jännitteellä ja kuluttaa alle 60mA virtaa. Anturin elinikä on yli 5 vuotta. Anturissa on kolme erilaista ulostuloa: analoginen, PWM ja UART (Universal Asynchronous Receiver Transmitter). Anturin tarkkuus on ± (50ppm + 3 % mitatusta arvosta) ja mitta-alue 0-5000 ppm (Zhengzhou Winsen Electronics Technology Co., Ltd, 2016, p. 4).



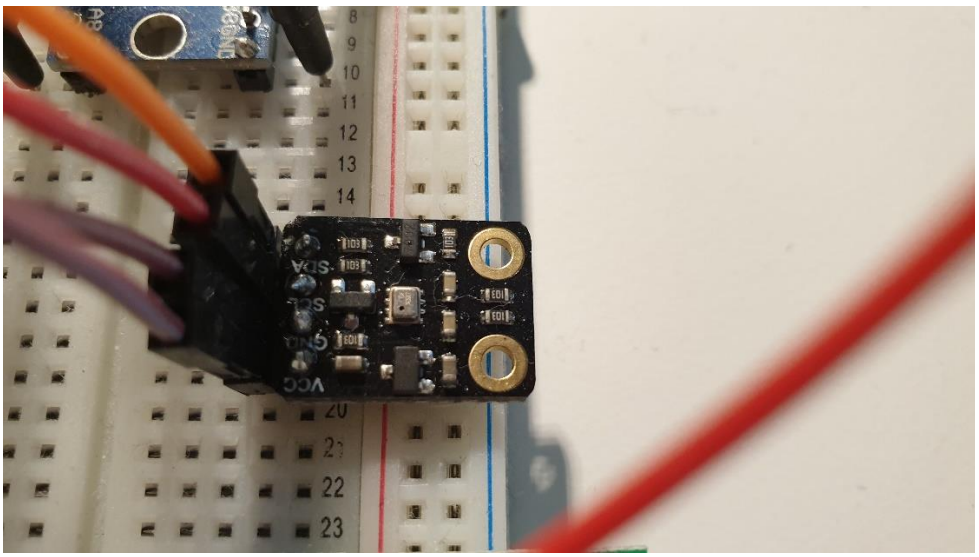
Kuva 5. NDIR:n toimintaperiaate (Mendes, 2015)



Kuva 6. MH-Z19B-anturi

4.2.5 BMP280

BMP280 (Kuva 7) on alunperin Bosch Sensortec:n suunnittelema anturi, joka mittaa ilmanpainetta ja lämpötilaa. Anturin ilmanpaineen mitta-alue on 300-1100 hPa, resoluutio 1,9 Pa ja tarkkuus ±1 hPa. Anturi toimii 3,3V jännitteellä ja kuluttaa virtaa vain 3,4 µA (0,1 µA lepotilassa). DFRobot:n valmistama anturi voidaan kytkeä Raspberry Pi:hin vain I^2C spesifikaatiolla, vaikka Bosch:n suunnittelema anturi tukee myös SPI-spesifikaatiota (DFRobot, ei pvm).



Kuva 7. BMP-280-anturi kytkettynä koekytkentäalustaan

4.3 Ohjelmointikielet ja kirjastot

4.3.1 C#

C# on Microsoftin kehittämä ohjelmointikieli, joka oli StackOverflow:n 2022 vuotuisen kehittäjille suunnatun kyselyn mukaan 8. käytetyin teknologia (Stack Overflow, ei pvm). Kieli on suunniteltu olemaan yksinkertainen, moderni ja yleiskäyttöinen ohjelmointikieli (ECMA International, 2001). C# on olio-ohjelmointiin tarkoitettu kieli. Sitä voidaan käyttää myös muiden ohjelmointiparadigmojen kanssa, kuten funktionaalinen ohjelmointi, jossa suuressa roolissa ovat lambda-lausekkeet.

C# on käännettävä ohjelmointikieli, mikä tarkoittaa, että kirjoitettu lähdekoodi kulkee erillisen kääntäjän läpi ennen kuin ohjelmaa voidaan suorittaa. .NET-alustan kääntäjän nimi on Roslyn, jonka lähdekoodi on avoimena GitHubissa (.NET Platform, ei pvm).

C# soveltuu monien eri sovellusten kehitykseen, kuten pelien, verkko-, työpöytä-, mobiili- ja IoT-sovellusten kehitykseen. Kieli tukee myös asynkronisia operaatioita, minkä takia se soveltuu hyvin hajautettujen järjestelmien kehittämiseen. C#:n syntaksi on hyvin samanlainen mm. C, C++ ja Java -kielien kanssa. Syntaksilla tarkoitetaan koodin kirjoittamiseen liittyviä kielioppi- ja tyylisääntöjä, kuten tarvitseeko jokainen rivi lopettaa tietyllä merkillä tai miten uusi muuttuja luodaan.

4.3.2 Python

Python on hyvin korkean tason yleiskäyttöinen kieli. Sen on suunnitellut Guido van Rossum ja se julkaistiin vuonna 1991. Python tukee montaa ohjelmointiparadigmaa, kuten olio-, funktionaalista ja proseduraalista ohjelmointia. Python on suosittu kieli mm. verkkosovellusten, koneoppimisen ja sulautettujen järjestelmien kehityksessä.

Toisin kuin C#, Python on tulkittu ja heikosti tyypitetty ohjelmointikieli. Myös syntaksissa on suuria eroja, kuten lohkojen erottelu sisennyksillä. Pythonin syntaksi on ohjelmointikieleksi hyvin lähellä englantia. Python on hyvin suosittu kielivalinta ensimmäisenä ohjelmointikielenä. Stack Overflow:n kehittäjäkyselyssä Python oli neljänneksi käytetyin teknologia (Stack Overflow, ei pvm).

4.4 Pilvipalvelut

Pilvipalvelut ovat verkon kautta käytettäviä palveluita. Verkolla voidaan tarkoittaa julkista internettiä, jolloin puhutaan julkisesta pilvestä (engl. ”public cloud”). Tämän vastakohtana on ns. yksityinen pilvi (engl. ”private cloud”), jolloin palvelut ovat tarjolla suljetussa, yksityisessä verkossa. Nämä mallit eivät kuitenkaan sulje toisiaan pois, vaan niitä voidaan yhdistää. Silloin puhutaan hybridipilvestä (engl. ”hybrid cloud”).

Suuria pilvipalvelujen tarjoajia on kolme: Amazonin AWS (Amazon Web Services), Microsoftin Azure ja Googlen GCP (Google Cloud Platform). Jokainen alusta tarjoaa palveluita IoT-ratkaisujen rakentamiseen, ja suunniteltu järjestelmä voitaisiin toteuttaa millä tahansa alustalla. GCP:n IoT-palvelu ”IoT Core” kuitenkin poistetaan käytöstä 15. elokuuta 2023 (TheRegister, 2022), eli uusien IoT-ratkaisujen rakentaminen palvelulla olisi huono idea. Työhön kuitenkin valittiin Azure, koska Savonia-ammattikorkeakoululla on käytössään VS Pro -tilauksia, joihin sisältyy Azure krediittejä 45€ / kk.

4.4.1 IoT Hub

IoT Hub on Azuressa oleva sovellusalusta-palvelu (PaaS) erilaisten IoT-ratkaisujen rakentamiseen. Sen vastuualueena on hallinnoida IoT-laitteita, toimia viestien vastaanottopisteenä ja välittää viestit eteenpäin muihin Azuren palveluihin, kuten Service Bus Queue:een tai suoraan tallennettavaksi Data Lake Storage:een. Tieto voidaan silloin tallentaa vain Apache AVRO tai JSON-formaatissa.

IoT Hub:n yksi ominaisuus on Device Twin. Device Twin on käytännössä .json-tiedosto, johon on tallennettu tietoa yhdestä fyysisestä laitteesta, joka on yhdistettynä IoT Hubiin. Digital Twin sisältää

mm. laitteen tunnisteet (engl. ”tags”), halutut ominaisuudet (engl. ”desired properties”) ja raportoidut ominaisuudet (engl. ”reported properties”).

IoT Hub mahdollistaa myös viestien ohjaamisen erilaisten reitityssääntöjen avulla (engl. ”routing query”), jonka avulla eri laitteiden ja erityyppiset viestit voidaan ohjata eri palveluihin Azuressa. Tämä helpottaa ison ja monipuolisen järjestelmän suunnittelua ja toteutusta, kun ominaisuudet voidaan suunnitella ja toteuttaa toisistaan erillisinä. Viesti voidaan lajitella sen lähettäneen laitteen tai itse viestin sisällön perusteella. Ajatellaan Digital Twin:iä, jonka tunnisteissa on tieto laitteen sijainnista, joka koostuu rakennuksesta ja kerroksesta (Kuva 8). Reitityssäännöllä

$twin.tags.sijainti.rakennus = 'Auditorio' AND $twin.tags.sijainti.kerros = 1

voitaisiin valita kaikki laitteet, joiden sijainniksi on merkitty auditorion ensimmäinen kerros.

```
"tags": {
    "sijainti": {
        "rakennus": "Auditorio",
        "kerros": 1
    }
}
```

Kuva 8. Tunniste-osio, jossa laite on sijoitettu auditorion ensimmäiseen kerrokseen

Laitteen yhdistämistä IoT Hub:iin varten Microsoft on luonut useita SDK:ita eri ohjelmointikielille, joiden avulla kehittäjien on nopeaa saada toimiva yhteys laitteen ja IoT Hub:n välille. Näitä ohjelmointikieliä ovat C, .NET (C#), Java, Python, Node (Javascript) ja Embedded C. Työssäni käytän Pythonille tehtyä SDK:ta, joka on ladattavissa ja asennettavissa pip -paketinhallintatyökalulla.

Viestintä laitteen ja IoT Hub:n välillä on mahdollista joko MQTT-, AMQP- tai HTTPS-protokollaa käyttäen. Oletuksena palvelu käyttää viestintään MQTT-protokollaa. Viestintäprotokollan valinnassa on huomioitava useita asioita, kuten tapahtuuko viestintää vain laitteelta pilveen, kuinka paljon laitteella on resursseja käytössään ja mitä rajoituksia käytettävä verkkoyhteys asettaa. HTTPS-protokollalla viestien lähetys pilvestä laitteelle on hyvin epätehokasta, koska palvelimella ei ole mahdollisuutta lähettää viestiä suoraan laitteelle, vaan laite joutuu itse kyselemään tätä tietoa tietyin väliajoin (suositus on n. 25 minuutta). MQTT on hyvin suosittu protokolla IoT-järjestelmissä mm. sen pienen viestikoon ja yksinkertaisen julkaise/tilaa-mallin (engl. ”publish/subscribe”) ansiosta.

4.4.2 Functions

Functions on Azuren tarjoama palvelu funktioiden suorittamiseen (FaaS, Functions-as-a-Service). Palvelua kutsutaan myös palvelittomaksi (engl. ”serverless”), koska funktioita suunnitellessa ja kehittäessä ei tarvitse murehtia ja ajatella palvelininfrastruktuurin rakentamisesta ja ylläpidosta. Palvelun käyttäjä ei siis näe tai pysty hallinnoimaan palvelimia, joilla funktiot suoritetaan.

Funktion suoritusta varten palvelussa on erilaisia laukaisimia (engl. ”trigger”), joista funktion suoritus voi alkaa. Yksi yleinen laukaisin on HTTP(S)-pyyntö. Muita laukaisimia on mm. ajastin, Blob ja

viestijono (engl. ”message queue”). Funktiot ovat yleensä tilattomia (engl. ”stateless”), eli useiden eri suorituskertojen välillä funktion tila ei säily. Functions-palvelussa on kuitenkin mahdollista luoda ”olio-funktioita” (engl. ”entity functions”). Olio-funktiot hallinnoivat ns. Durable Entity:jä, joihin voidaan tallentaa joitakin tila-tietoja eri funktioiden suoritusten välillä (Kuva 9). Muut funktiot voivat kutsua näitä olio-funktiota, käyttää olioihin luotuja operaatioita ja lukea niihin tallennettua tilaa (Kuva 10).

```
[JsonObject(MemberSerialization.OptIn)]
2 references
public class Laskuri
{
    [JsonProperty("arvo")]
    3 references
    public int NykyinenArvo { get; set; }

    0 references
    public void Summaa(int arvo) => this.NykyinenArvo += arvo;

    0 references
    public void Nollaa() => this.NykyinenArvo = 0;

    0 references
    public int Hae() => this.NykyinenArvo;

    [FunctionName(nameof(Laskuri))]
    0 references
    public static Task Run([EntityTrigger] IDurableEntityContext konteksti)
        => konteksti.DispatchAsync<Laskuri>();
}
```

Kuva 9. Laskuri-luokka Durable Entity:nä

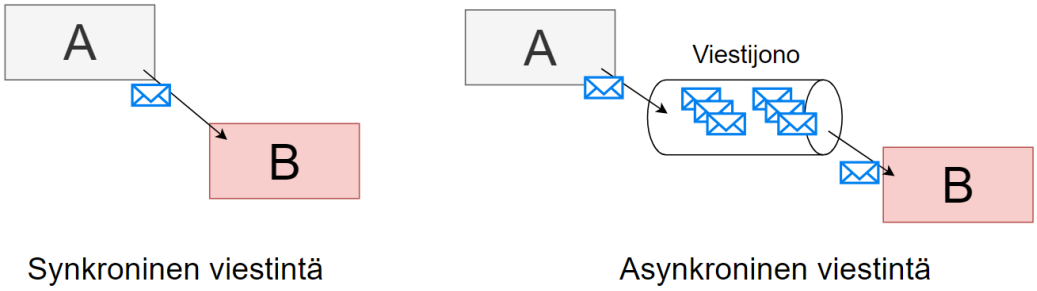
```
0 references
public static class Hae
{
    [FunctionName("hae")]
    0 references
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "get", Route = null)] HttpRequest req,
        [DurableClient] IDurableEntityClient client,
        ILogger log)
    {
        var olioId = new EntityId(nameof(Laskuri), "laskuri");
        EntityStateResponse<JObject> tila = await client.ReadEntityStateAsync<JObject>(olioId);

        return new OkObjectResult(tila.EntityState);
    }
}
0 references
public static class Summaa
{
    [FunctionName("summaa")]
    0 references
    public static async Task<IActionResult> Run(
        [HttpTrigger(AuthorizationLevel.Function, "post", Route = null)] HttpRequest req,
        [DurableClient] IDurableEntityClient client,
        ILogger log)
    {

        var sisältö = await new StreamReader(req.Body).ReadToEndAsync();
        var viesti = JsonConvert.DeserializeObject<Viesti>(sisältö);

        var olioTunnus = new EntityId(nameof(Laskuri), "laskuri");

        await client.SignalEntityAsync(olioTunnus, "Summaa", viesti.Arvo);

        return new OkResult();
    }
}

1 reference
public class Viesti
{
    1 reference
    public int Arvo { get; set; }
}
```

Kuva 10. Http-laukaisimella toimivat funktiot, jotka lukevat ja päivittävät Durable Entity:n tilaa

4.4.3 Service Bus

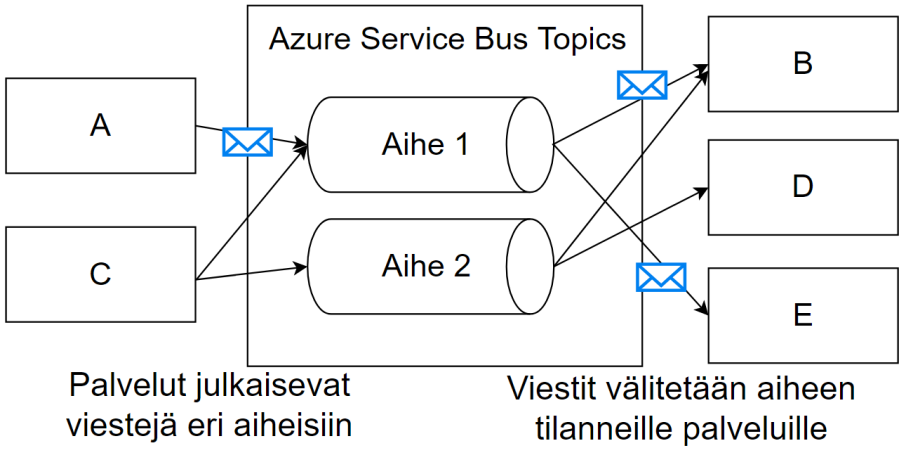
Azure Service Bus on yritystason viestinvälityspalvelu. Palvelussa voi välittää viestejä joko jonojen tai julkaise/tilaa-mallilla erilaisiin aiheisiin (engl. ”topic”). Jonoissa viestit välitetään suoraan kahden palvelun välillä, point-to-point-mallilla. Kun julkaise/tilaa -mallissa yksi palvelu julkaisee viestin kerran johonkin aiheeseen, välitetään viesti kaikille palveluille, jotka ovat tilanneet kyseisen aiheen.

Erillinen palvelu viestien välitykseen lisää viestien toimitusvarmuutta ja tuo joustoa viestien välitykseen. Se myös mahdollistaa helpon asynkronisen viestinnän kahden palvelun välillä. Kuva 11 vasemmalla puolella on esimerkki synkronisesta viestinnästä, jossa palvelu A viestii suoraan palvelun B kanssa. Tällöin kummankin palvelun tulee olla toiminnassa, kun A lähettää viestin. Oikealla puolella kuvaa on esimerkki asynkronisesta viestinnästä, jossa palvelu A lähettää viestin B:lle viestijonon kautta. Siten palvelun B ei tarvitse olla toiminnassa juuri silloin, kun palvelu A haluaa lähettää viestin.



Kuva 11. Kahden palvelun väliset suorat viestintämallit

Julkaise/tilaa-mallissa luodaan erilaisia aiheita. Palvelut julkaisevat viestejä tietyille aiheille ja toiset palvelut tilaavat näitä aiheita. Palvelu sopii hyvin useamman palvelun väliseen asynkroniseen viestintään. Malli on hyvin suosittu mikropalveluiden (engl. ”microservices”) kanssa. Malli on tehokas tapa usean eri palvelun asynkroniseen viestintään, mutta samalla se monimutkaistaa järjestelmää paljon. Kokonaisten prosessien hahmottaminen vaikeutuu, koska yhden viestin julkaisulla voi olla vaikutuksia useampaan eri palveluun (Kuva 12).



Kuva 12. Julkaise/tilaa-mallin viestintäkaavio. A julkaisee viestin aiheeseen 1, josta se välitetään aiheen tilanneille palveluille B ja E.

4.4.4 CosmosDB

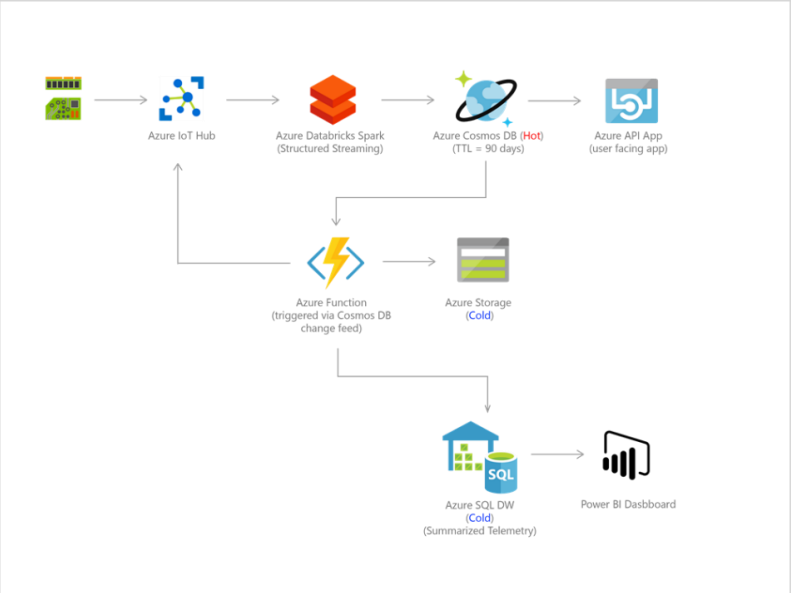
CosmosDB on Azuren tarjoama dokumenttitietokanta (engl. ”document database”). Dokumenttitietokannat ovat eräänlaisia NoSQL-tietokantoja, eli tallennetuilla tiedoilla ei ole tietokannan tasolla suhdetta toisiinsa. Usein tallennettava dokumentti on JSON-dokumentti. Dokumentit tallennetaan kokoelmiin, jotka vastaavat löyhästi ajateltuna SQL-tietokantojen tauluja. Kokoelman sisällä dokumentit ovat usein rakenteeltaan samanlaisia. Rakennetta ei kuitenkaan vahdita tietokannan tasolla, joten dokumenttien rakenteesta vastaa loppujen lopuksi tietokantaa käyttävä sovellus.

CosmosDB on erittäin vakaa ja nopea tietokanta. Tietokannalla on jopa 99,999%:n saatavuus, mikä tarkoittaa, että palvelu ei ole tavoitettavissa keskimäärin vain 26 sekunnin ajan jokaisena kuukautena (Microsoft Azure, 2022). Tietokanta myös skaalautuu hyvin suurille määrille tietoa, jota suuret IoT-järjestelmät voivat tuottaa. Skaalautuvuus pohjautuu konttien jakamiseen eri osiin (engl. ”partition”), joka tapahtuu jako-avaimen (engl. ”partition key”) perusteella. Konttien eri osat voivat tehdä

operaatioita samanaikaisesti. Operaatiot ovat myös nopeita: keskimääräinen kirjoitusoperaatio kestää vain n. 5 millisekuntia (Microsoft Azure, 2022).

CosmosDB tarjoaa viisi eri API:a, joiden kautta CosmosDB:tä voidaan käyttää. Lähtökohtaisesti suositellaan käytettäväksi SQL API:a, koska sen avulla voidaan käyttää uusimpia CosmosDB:n ominaisuuksia. Muut API:t (MongoDB, Cassandra, Gremlin ja Table) on luotu lähinnä sitä varten, että olemassa olevien sovellusten siirtäminen käyttämään CosmosDB:tä olisi mahdollisimman helppoa. Uuden sovelluksen luomisessa voi olla myös järkevää valita jokin muu kuin SQL API, jos sovelluksen kehittäjillä on jo syväosaamista jonkun muun API:n käytöstä.

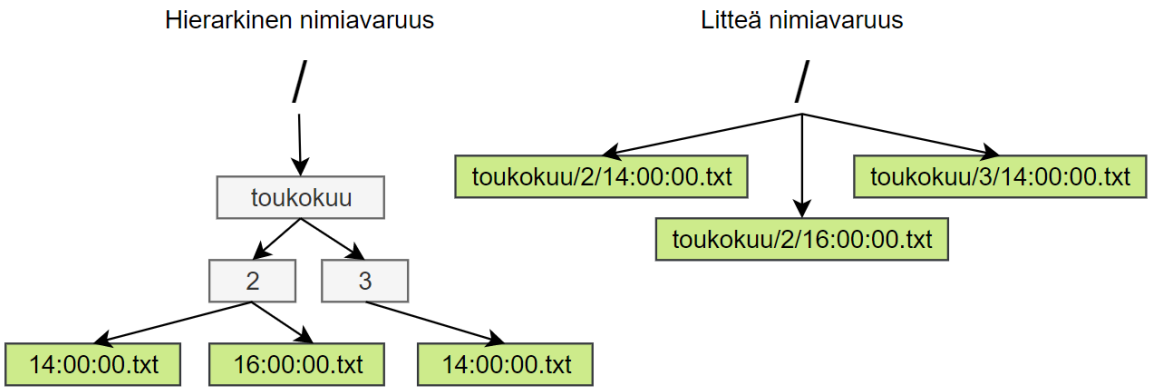
CosmosDB:n yksi tärkeä ominaisuus on muutosvirta (engl. ”change feed”). Muutosvirta ilmoittaa kaikista lisäyksistä ja päivityksistä, jotka tapahtuvat CosmosDB:ssä. Muutosvirtaa voidaan käyttää kahdella eri tapaa. Se voi toimia laukaisimena funktiolle tai muutosvirtaa voidaan lukea vapaaseen tahtiin. Useissa Azuren dokumentaation esimerkeissä hyödynnetään muutosvirtaa tiedon aggregointiin ja analysointiin (Kuva 13).



Kuva 13. Azuren esimerkki arkkitehtuuri IoT-ratkaisuille (Microsoft Azure, 2022)

4.4.5 Data Lake Storage Gen2

Data Lake Gen2 on Azure Blob Storagen ympärille rakennettu kokoelma toimintoja. Azure Blob Storage on Azuren tarjoama palvelu, jonne voidaan tallentaa suuria määriä rakenteetonta tietoa, kuten kuvia, videoita ja äänitiedostoja. Tärkein ominaisuus, jonka Data Lake Gen2 lisää Blob Storagen päälle, on hierarkinen nimiavaruus (engl. ”hierarchical namespace”). Blob Storagessa nimiavaruus on oletuksena ”litteä” (engl. ”flat namespace”), mikä tarkoittaa, että kaikki tiedostot ovat samalla tasolla tietorakenteessa. Hierarkinen nimiavaruus puolestaan muistuttaa tietorakenteeltaan enemmän puuta (Kuva 14).



Kuva 14. Kolme tiedostoa hierarkisessa ja litteässä nimiavaruudessa

Data Lake Gen2 on yhteensopiva Hadoop Distributed Filesystem:in (HDFS) kanssa. HDFS on tietojärjestelmä, joka on optimoitu suurten tietomäärien käsittelyyn. Aiemmin Data Lake:a käytettiin WASB-ajurilla (Windows Azure Storage Blob). Sen on korvannut uudempi, tehokkaampi ja paremmin optimoitu ABFS-ajuri (Azure Blob File System). ABFS-ajuri on hyvin pieni. Se kartoittaa HDFS:n tietojärjestelmän komennot Data Lake Storage:n REST-rajapinnan komentoihin.

4.4.6 Azure DevOps

DevOps on kokonaisvaltainen alusta projektien suunnitteluun, hallintaan ja yhteistyön tekemiseen ohjelmistokehityksessä. Palvelussa voidaan mm. säilyttää lähdekoodia Repos:ssa, suunnitella ja seurata työn etenemistä Boards:ssa ja luoda automatisoituja prosesseja CI/CD:n toteuttamiseen Pipeline:ssa. Lisäksi palvelussa voi ottaa käyttöön yhteisön luomia laajennuksia tai luoda omia laajennuksia. Visual Studio Marketplace:ssa on yli 1000 laajennusta, joiden avulla voi lisätä toiminnallisuuksia DevOps:iin. Työn aikana tallensin kirjoitetut lähdekoodit DevOps:n Repos:iin, jotta niiden luovuttaminen olisi helppoa työn päätyttyä.

4.5 Kehitysympäristö

4.5.1 Visual Studio Code

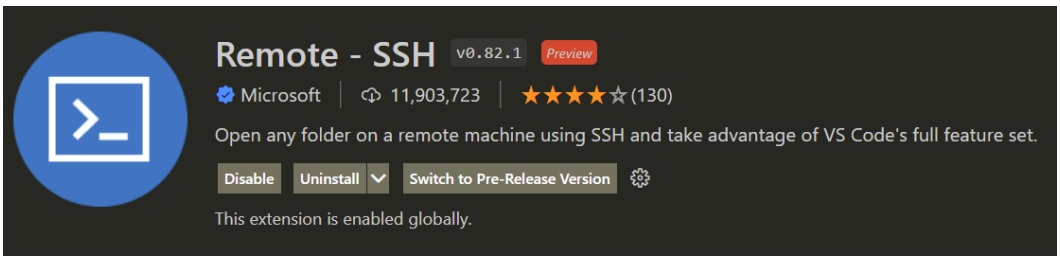
Visual Studio Code on Microsoftin tekemä koodieditori, joka on saatavilla yleisimmillä käyttöjärjestelmillä, kuten Windows, Linux ja Mac OS. Vuonna 2022 StackOverflow:n jokavuotisen kehittäjille suunnatun kyselyn mukaan se on suosituin koodieditori Stack Overflow:n käyttäjien keskuudessa (Stack Overflow, ei pvm). Visual Studio Code sisältää tärkeimmät koodieditorin ominaisuudet, kuten automaattinen koodintäydennys, syntaksin tarkistus, integroitu versionhallinta ja debuggeri. Debuggerilla tarkoitetaan erilaisia toimintoja, joiden avulla koodista voidaan etsiä vikoja.

Visual Studio Code (VS Code) on hyvä valinta editoriksi projektissa mm. sen monien lisäosien takia. Yksi Microsoftin tuottama lisäosa, Remote – SSH (Kuva 15), mahdollistaa etäyhteyden muodostamisen Raspberry Pi:lle SSH:n avulla. Tämä mahdollistaa Raspberry Pi:llä olevien tiedostojen muokkaamisen Code:lla etäyhteyden kautta. Muutoin Raspberry Pi:llä suoritettavat ohjelmat joutuisi joko

1. Luomaan ja muokkaamaan toisella tietokoneella ja sitten siirtämään jollain tapaa Pi:lle.

2. Muokkaamaan tiedostoja normaalin SSH:n avulla (jolloin käytössä olisi yksinkertaisempia tekstieditoreja, kuten Nano tai Vim).
3. Yhdistämään näytön, näppäimistön ja hiiren Pi:hin, ja muokata tiedostoja siten.

VS Code tukee myös montaa eri ohjelmointikieltä. Siksi sen avulla voidaan hyvin tehdä ohjelmistokehitystä projekteissa, joissa käytetään erilaisia ohjelmointikieliä. Projektissa käytin VS Code:a kuitenkin vain Python-koodin kirjoittamiseen, koska C#:n kirjoittaminen on minulle tutumpaa ja tehokkaampaa toisella koodieditorilla, Visual Studiolla (4.5.2).



Kuva 15. Remote – SSH -lisäosa Visual Studio Code:en

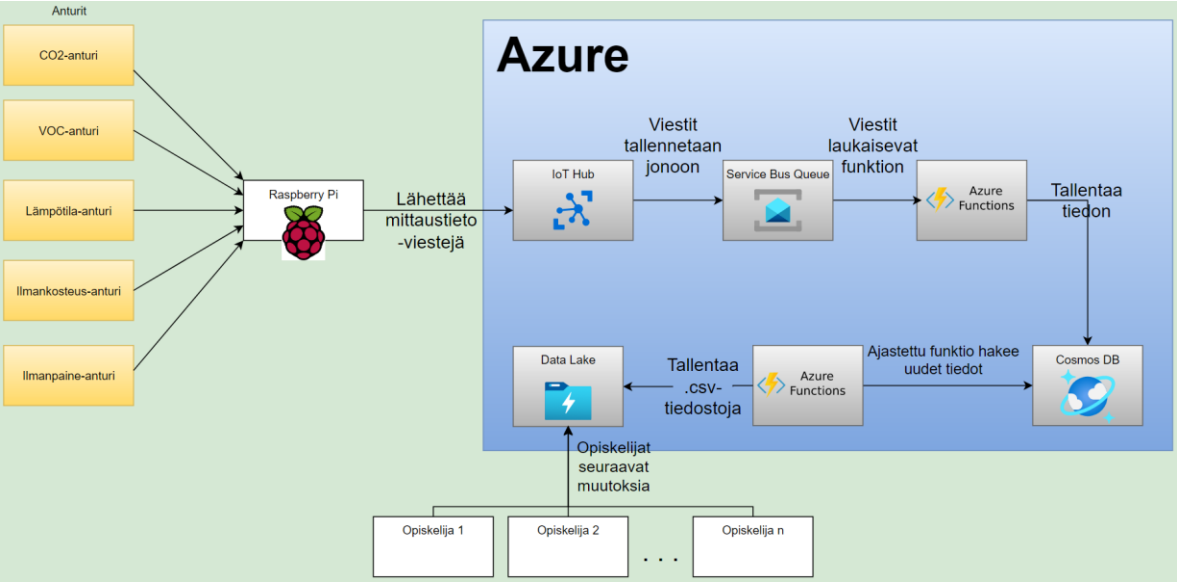
4.5.2 Visual Studio 2022

Visual Studio 2022 on toinen Microsoftin tarjoama koodieditori, jolla on pitkä historia. Ensimmäinen versio editorista julkaistiin jo 19. päivä maaliskuuta 1997 (Microsoft, 1997). Editorissa on paljon ominaisuuksia, jotka helpottavat muiden Microsoftin palveluiden ja tekniikoiden käyttöä. Esimerkiksi SQL Server Data Tools (SSDT) tuo monia tietokannan hallintajärjestelmän ominaisuuksia suoraan Visual Studioon. Siten tietokannan hallinta onnistuu suoraan Visual Studion kautta, eikä sitä varten välttämättä tarvitse käyttää toista ohjelmaa. Myös tietyt Azuren toiminnot ovat käytettävissä suoraan Visual Studion kautta, kuten Function App:in julkaisu.

Visual Studiosta on saatavilla kolme eri versiota, Community, Professional ja Enterprise. Community -versio on ilmaiseksi ladattavissa ja käytettävissä yksityishenkilöille. Professional-version käyttö maksaa $45/kk ja on tarkoitettu keskisuurten yritysten käyttöön. Enterprise-versiossa on ominaisuuksia, joita ei löydy kahdesta aiemmasta versiosta. Enterprise -versio maksaa $250/kk. JetBrains:in ”Developer Ecosystem Survey 2021” -kyselystä käy ilmi, että Visual Studio on suosituin editori C#-ohjelmointikieltä käyttävien keskuudessa (JetBrains, 2021).

5 KEHITYSTYÖ

Kehitystyö aloitettiin fyysisen laitteiston kokoamisella ja testaamisella, jonka jälkeen luotiin Raspberry Pi:llä suoritettava ohjelma. Seuraavaksi pilvipalvelut otettiin käyttöön tiedonkulun mukaisessa järjestyksessä. Kehitystyön viimeinen vaihe oli kehittää Azure Functions-palvelussa suoritettavat funktiot. Tiedon kulkeminen antureilta Data Lake:een asti näkyy kuvassa 16.



Kuva 16. Rakennettu järjestelmä kokonaisuudessaan

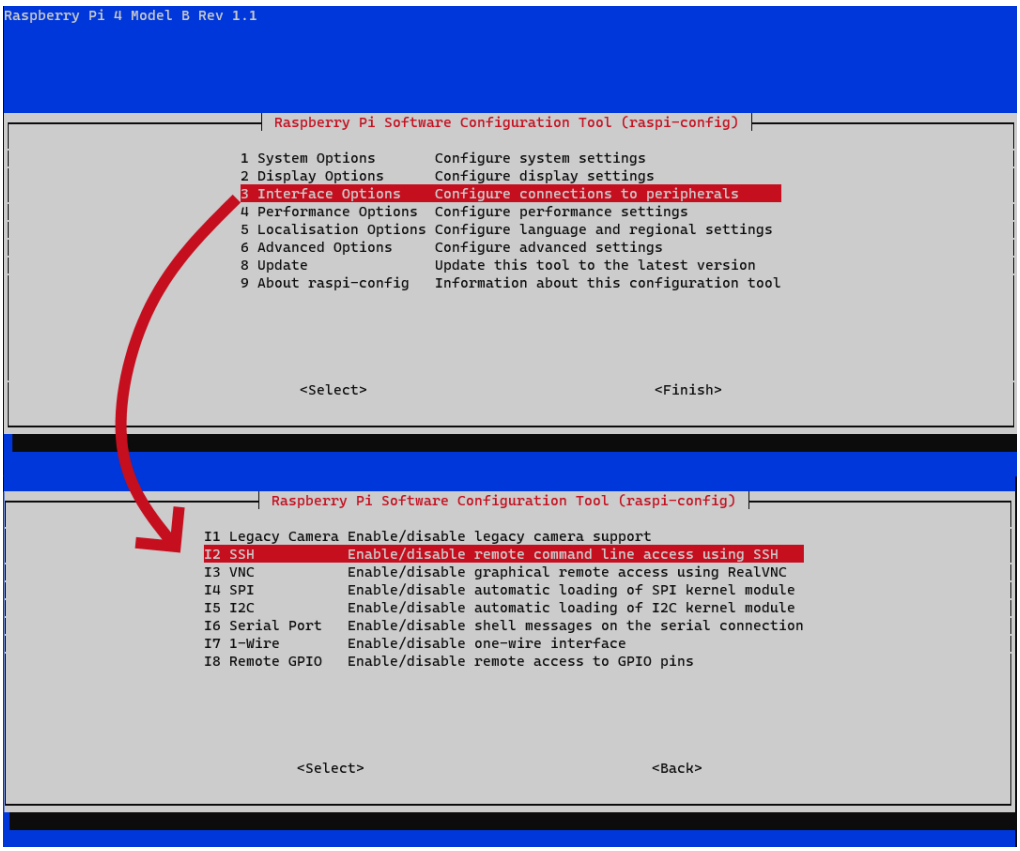
5.1 Raspberry Pi:n käyttöönotto

Raspberry Pi on kokonainen tietokone, joten se vaatii käyttöjärjestelmän. Työssä harkitsin kahta eri käyttöjärjestelmää, Windows for IoT Core:a ja Raspberry Pi OS:ää (nimi aiemmin Rasbian). Windows IoT Core tarjoaisi selkeän polun laitteiden massatuotantoon. Työn laajuuden takia päädyin kuitenkin käyttämään itselleni tähän tarkoitukseen tutumpaa käyttöjärjestelmää, Raspberry Pi OS:ää.

Raspberry Pi OS asennetaan ensiksi SD-muistikortille käyttäen Raspberry Pi Imager -ohjelmistoa, joka on ladattavissa Raspberry Pi:n verkkosivuilta (Raspberry Pi, ei pvm). Tämän jälkeen muistikortti laitetaan Raspberry Pi:hin. Seuraavaksi Raspberry Pi:hin voidaan kytkeä näyttö, näppäimistö ja hiiri. Lopuksi Raspberry Pi:hin voidaan yhdistää virtajohto.

Ensimmäisen käynnistyksen yhteydessä tarvitsee määrittää tiettyjä asetuksia, kuten sijainti, näppäimistön kieli ja asettelu, verkkoasetukset ja käyttäjän tiedot. Käyttöjärjestelmä ei enää tarjoa oletuksena käyttäjätunnusta ”pi”.

Käyttöönoton yhteydessä Raspberry Pi:lle sallittiin muodostaa SSH-yhteys. Ominaisuus sallitaan helposti terminaalin kautta. Komennolla ’sudo raspi-config’ aukeaa valikko, jonka kautta voidaan muuttaa tiettyjä Raspberry Pi:n asetuksia. Valikon kohdasta ”Interface Options” aukeaa alivalikko, jonka kohdasta ”I2 SSH” voidaan asettaa SSH-yhteyden salliminen päälle tai pois (Kuva 17). SSH:n avulla Raspberry Pi:hin saa yhteyden toiselta tietokoneelta, mikä helpottaa Respberry Pi:llä hoidettavia tehtäviä.



Kuva 17. SSH:n käyttöönotto

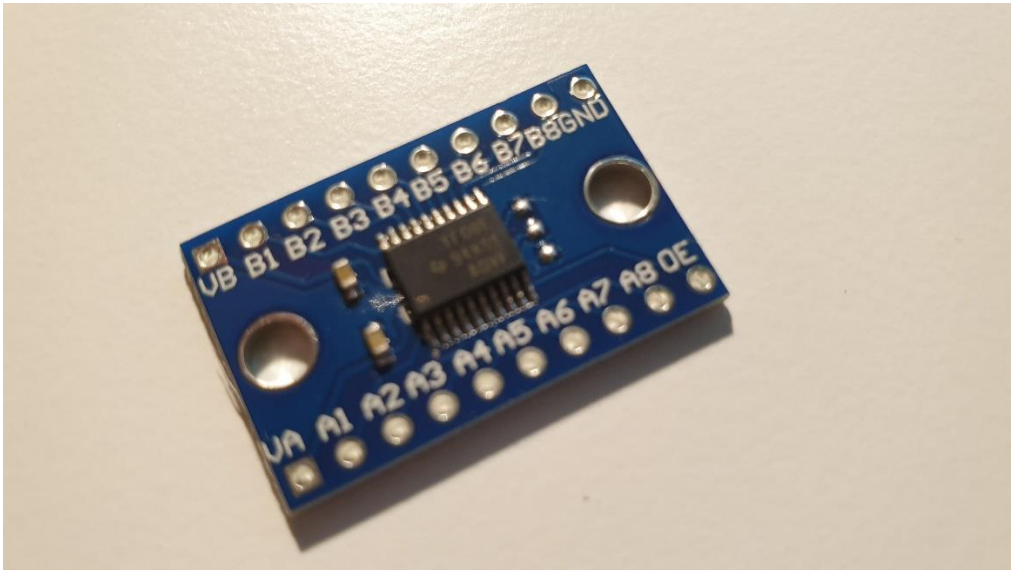
5.2 Anturien kytkeminen Raspberry Pi:hin

Seuraava vaihe kehitystyössä oli koota fyysinen laitteisto ja tehdä vaadittavat kytkennät. Kytkennät tehtiin koekytkentä alustaan. Siten laite on helppo purkaa, jos sen osat halutaan palauttaa muuhun opetuskäyttöön. Anturit toimivat joko 3,3V tai 5V jännitteellä, jotka saadaan Raspberry Pi:n GPIO-pinneistä. Koko laitteiston kytkentäkaavio on näkyvissä kuvassa 20.

5.2.1 ZP01-MP503

Anturin ulostulona toimii kaksi eri 5V signaalia, jotka vastaavat neljää eri VOC-tasoa. 5V jännite on kuitenkin liian korkea Raspberry Pi:n luettavaksi, minkä takia signaalit siirretään 3,3V tasoon ennen signaalin lukemista. Tätä varten työssä käytetään logiikkatasonsiirtäjää TXS0108E (Kuva 18).

Anturin 5V pinni yhdistetään 5V jännitteeseen ja GND maahan. Anturin ulostulot (A ja B) kytketään logiikkatasonsiirtäjän korkeamman jännitteen (5V) puolelle liittimiin B1 ja B2. Logiikkatasonsiirtäjän matalamman jännitten (3,3V) puolelta kohdat A1 ja A2 yhdistetään Raspberry Pi:n GPIO-pinneihin GPIO19 ja GPIO26 (Kuva 20). Laitteella suoritettava python-ohjelma lukee näitä pinnejä, ja muuntaa signaalit tiettyyn VOC-tasoon.

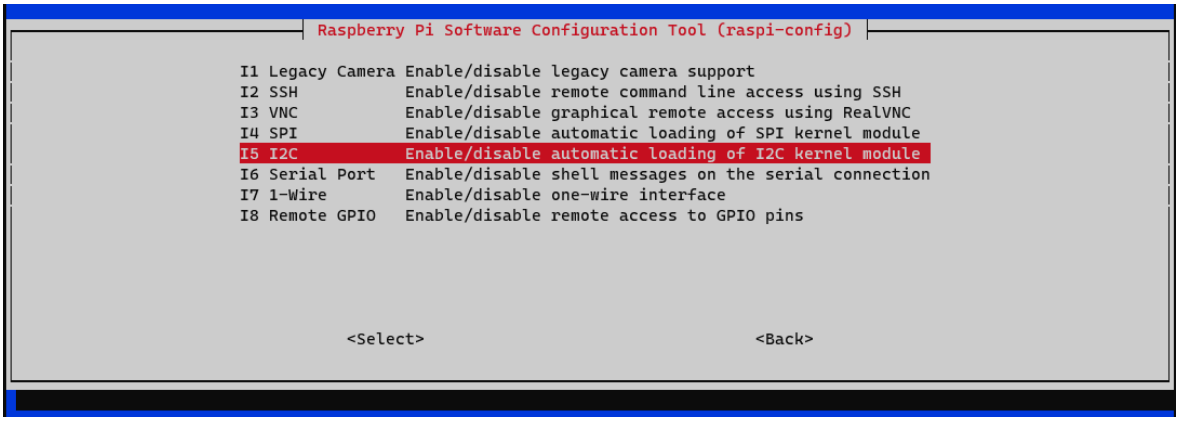


Kuva 18. TXS0108E-logiikkatasonsiirtäjä

5.2.2 BMP280

Anturin ulostulo on I²C-spesifikaation mukainen, joten kommunikaatio vaatii kaksi linjaa: SDA datalle ja SCL viestinnän nopeudensäätelyyn. Raspberry Pi:n pinnit GPIO02 ja GPIO03 sopivat I²C-viestintään. BMP280 anturin SDA yhdistetään GPIO02-pinniin ja SCL yhdistetään GPIO03-pinniin. BMP280 toimii 3,3V jännitteellä, joten VCC yhdistetään 3,3V jännitteeseen ja GND maahan.

I2C-viestintä pitää kytkeä erikseen päälle Raspberry Pi:ssä. Asetus on samassa alivalikossa, kuin SSH:n päälle tai pois kytkeminen (Kappale 5.1). Valikko avataan komennolla 'sudo raspi-config', josta valitaan kohta "Interface options" ja edelleen "I5 I2C" (Kuva 19).



Kuva 19. I²C:n päälle kytkeminen

5.2.3 DHT-11

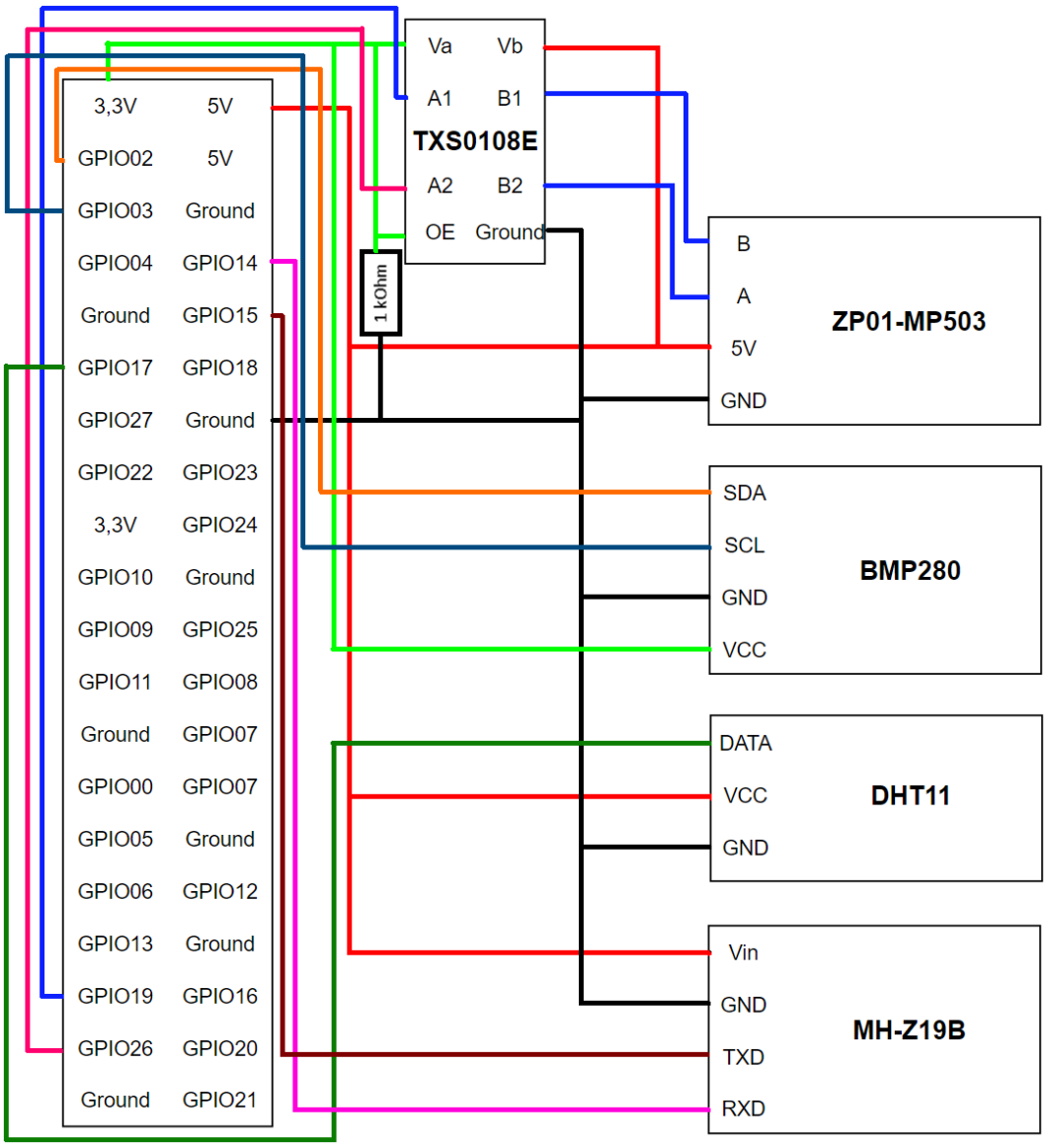
DHT-11 on hyvin suosittu ja yksinkertainen anturi käyttää. Se toimii 3-5,5V jännitteellä ja käyttää vain yhtä linjaa viestintään. Suositeltu jännite anturille on kuitenkin 5V, joten anturin VCC kytkettiin 5V:iin ja GND maahan. Data voidaan kytkeä mihin tahansa vapaaseen GPIO-pinniin. Tässä työssä se

yhdistettiin GPIO17-pinniin, joka myöhemmin annetaan parametriksi DHT-11-olion konstruktorille python-ohjelmassa.

5.2.4 MH-Z19B

MH-Z18B:ssä on kolme erillistä ulostuloa. Työssä päätin käyttää UART:ia, joka vaatii kaksi linjaa. Raspberry Pi:n GPIO14 (TXD) ja anturin RXD yhdistetään. Samoin GPIO15 (RXD) ja anturin TXD yhdistetään. Anturi toimii 5V jännitteellä, joten Vin kytkettiin 5V jännitteeseen ja GND maahan. Toisin kuin I^2C ja SSH, UART toimii pinneistä GPIO14 ja GPIO15 ilman erillistä asetuksen päälle asettamista.

Lopullinen kytkentäkaavio on alapuolella olevassa kuvassa (Kuva 20). Kuvan antureista ja TXS0108E-logiikkatasonsiirtäjästä on poistettu ylimääräiset liitännät kuvan yksinkertaistamiseksi.



Kuva 20. Anturien kytkentä Raspberry Pi:hin

5.3 Python-ohjelma tiedon mittaamiseen ja lähettämiseen

5.3.1 IoT Hub:iin yhdistäminen

Kehitystyön ohjelmointiosuus alkoi Raspberry Pi:llä suoritettavasta ohjelmasta. Ohjelman tehtävä on kerätä tiedot ilmanlaadusta eri antureilta, muodostaa tiedoista yksi viesti ja lähettää viesti Azureen, IoT Hub:iin.

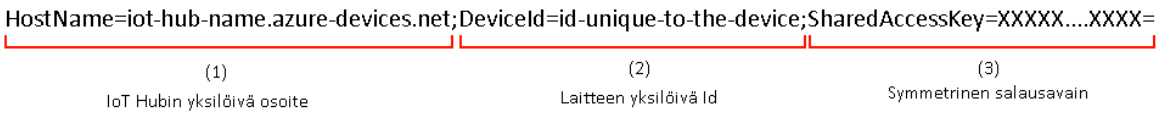
Ohjelma tukeutuu paljon avoimiin kirjastoihin, kuten Microsoftin julkaisemaan ja ylläpitämään kirjastoon ”azure-iot-device”. Kirjasto on osa Azuren SDK:ta, jonka kautta ohjelmat voivat vaivattomasti käyttää erilaisia Azuren palveluja. Kirjasto asennetaan suorittamalla komento ’pip install azure-iot-device’ terminaalissa.

Ohjelman pohja muodostettiin yhdistämällä Microsoftin kaksi eri esimerkkiohjelmaa, send_message.py ja receive_twin_desired_properties_patch.py (Azure Github, 2022). Ohjelmissa on esimerkkinä muodostettu yhteys IoT Hubiin, jonne lähetetään viesti ja näytetään miten Device Twin:in halutun ominaisuuden muutos voidaan käsitellä suoritettavassa ohjelmassa.

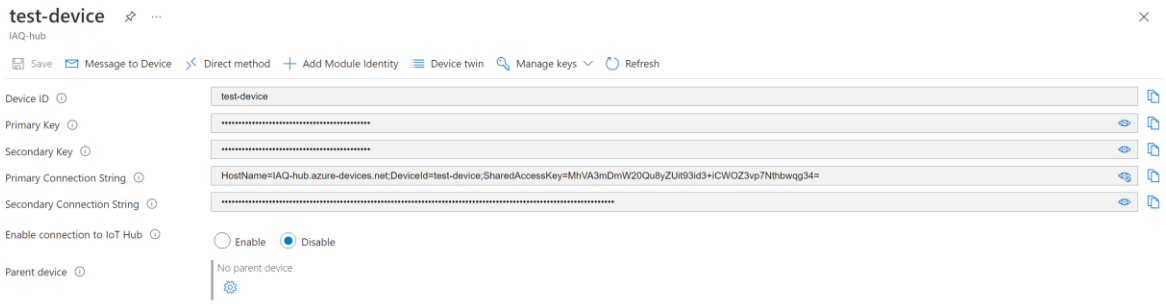
Azuren SDK:n tarjoamista luokista tärkein on IoTHubDeviceClient. Konstruktori ottaa vain yhden argumentin, yhteysmerkkijonon (engl. ”connection string”, Kuva 21). Se on merkkijono, joka sisältää kaikki tarvittavat tiedot yhteyden muodostamiseen IoT Hubiin. Alla olevassa kuvassa 22 on eriteltynä laitteen yhteysmerkkijonon kolme eri osaa. Laitteen yhteysmerkkijono saadaan IoT Hub:sta, laitteen tiedoista (Kuva 23). Yhteysmerkkijono on salattavaa tietoa, eikä käytössä olevia yhteysmerkkijonoja koskaan pidä näyttää tarpeettomille tahoille. Tässä työssä käytettyjen kuvien yhteysmerkkijonot ovat joko sensuroituja tai puhtaasti kuvaa varten luotuja esimerkkejä.

```
# Store connection string in an environment variable
conn_str = os.getenv("IOTHUB_DEVICE_CONNECTION_STRING")

# Create the client object from connection string
iot_device_client = IoTHubDeviceClient.create_from_connection_string(conn_str)
```

Kuva 21. IoTHubDeviceClient-olion luominen



Kuva 22. IoT Hub:n yhteysmerkkijonon rakenne



Kuva 23. Esimerkkilaitteen tiedot, mukaan lukien yhteysmerkkijono. Laite on poistettu IoT Hub:sta kuvan ottamisen jälkeen, joten yhteysmerkkijonoa ei voi enää käyttää mihinkään.

5.3.2 Tietojen lukeminen antureilta

Suurin osa anturien tietojen lukemisesta onnistuu suoraan asennettavien pakettien tarjoamien funktioiden avulla. Tohtori Takeyuki Ueda on luonut kirjaston 'mh-z19' (Takeyuki UEDA, 2022), jonka avulla ilman CO_2-pitoisuuden lukeminen on vain yksi funktiokutsu. Phillip Howard on julkaissut kirjaston 'bmp280' anturin BMP280 käyttöä varten (Howard, 2021). Adafruit on julkaissut kirjaston 'adafruit-circuitpython-dht', jonka avulla DHT11-anturin käyttö on yksinkertaista (Adafruit, 2022).

ZP01-MP503-anturin lukemista varten ohjelmassa piti lukea anturin kahden eri signaalin A ja B jännitteen taso. Jos signaalin jännite on korkea, tulkitaan signaalin arvoksi yksi, ja jos signaalin jännite on matala, tulkitaan sen arvo nollaksi. Nämä arvot muunnetaan kymmenkantaiseksi numeroksi 0-3 Taulukko 1 mukaan. Signaalien lukemisen avuksi työssä käytettiin Adafruitin CircuitPythonin 'digitalio' -moduulia. Moduuli on osa 'adafruit-blinka' pakettia, joka tarvitsee oikein toimiakseen toisen paketin, 'board'. Kaikki edellä mainitut paketit voidaan asentaa komennolla:

"pip install mh-z19 bmp280 adafruit-circuitpython-dht board adafruit-blinka"

```
# Define VOC sensor settings
VOC_A = digitalio.DigitalInOut(board.D26)
VOC_B = digitalio.DigitalInOut(board.D19)
VOC_A.direction = digitalio.Direction.INPUT
VOC_B.direction = digitalio.Direction.INPUT

# VOC level is read as two digit binary number, converting to decimal
# So:
# 00 -> 0
# 01 -> 1
# 10 -> 2
# 11 -> 3
def read_voc():
    total_value = 0
    aValue = VOC_A.value
    bValue = VOC_B.value
    if(aValue == 1):
        total_value+=2
    if(bValue == 1):
        total_value+=1
    return total_value
```

Kuva 24. ZP01-MP503-anturin määrittely ja VOC-tason lukeminen

Anturien arvojen ja ajan tallentaminen Pythonin sanakirjaan (engl. ”dictionary”) onnistuu nyt kuudessa rivissä (Kuva 25). Sanakirjan muuttaminen json-muotoiseksi viestiksi tapahtuu pythonin json-moduulin dumps()-funktiolla ja lähettäminen IoT Hub:iin IoTHubDeviceClient-luokan send_message() -funktiolla. (Kuva 26).

```
# Read values into a dictionary
values = {}
values.update(mh_z19.read())
values['temp'] = dht_11.temperature
values['humidity'] = dht_11.humidity
values['voc'] = read_voc()
values['pressure'] = bmp280.get_pressure()
values['time'] = str(datetime.now(timezone.utc))
```

Kuva 25. Arvojen kokoaminen sanakirjaan

```
# Send the values to IoT Hub
await iot_device_client.send_message(json.dumps(values))
```

Kuva 26. Sanakirjan muuntaminen json-formaattiin ja lähettäminen IoT Hub:iin

5.4 Ohjelman aloitus laitteen käynnistyksen yhteydessä

Laitteen käytön tulisi olla mahdollisimman yksinkertaista. Siksi Raspberry Pi:llä oleva ohjelma voidaan rekisteröidä käyttöjärjestelmässä taustapalveluksi, joka käynnistyy aina kun Raspberry Pi käynnistetään. Sitä varten muodostettiin ’iaq.service’ -tiedosto (Kuva 27), joka määrittää tarvittavat tiedot käynnistettävästä ohjelmasta. Tietoihin kuuluu esim. missä vaiheessa laitteen käynnistystä voidaan ohjelma käynnistää, mistä tiedosto löytyy ja miten se suoritetaan. Tiedosto tallennetaan polkuun ’/lib/systemd/system’.

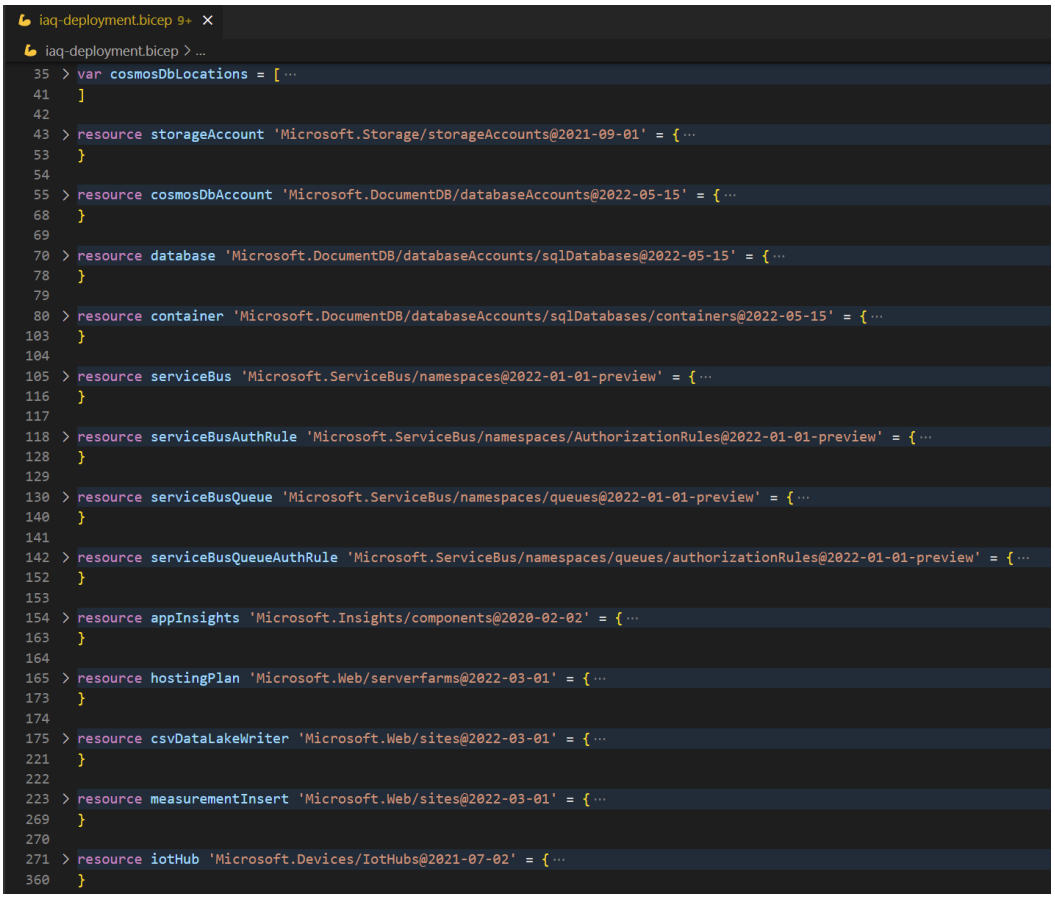
```
iaq@raspberrypi:/lib/systemd/system $ cat iaq.service
[Unit]
Description=Indoor Aiq Quality Service
After=multi-user.target

[Service]
Environment=IOTHUB_DEVICE_CONNECTION_STRING="[redacted]"
Type=idle
WorkingDirectory=/home/iaq/iaq-project-2022/
ExecStart=/usr/bin/python /home/iaq/iaq-project-2022/data_collection.py
Restart=on-failure
User=root

[Install]
WantedBy=multi-user.target
iaq@raspberrypi:/lib/systemd/system $
```

Kuva 27. iaq.service-tiedosto

5.5 Pilvipalveluiden käyttöönotto

Jokaisen pilvipalvelun käyttöönotto tehtiin graafisen verkkokäyttöliittymän kautta eli Azure Portalin kautta. Parempi käytäntö toistettavien ja dokumentoitavien käyttöönottojen kannalta on käyttää ns. IaaC (Infrastructure-as-a-Code) -menetelmiä. ARM eli Azure Resource Manager mahdollistaa käyttöönotettavien palvelujen käyttöönoton .json- tai .bicep-tiedostoilla. Siten käyttöönotto voidaan aina toistaa luotettavasti samalla tavalla.

Pilvipalveluista muodostettiin jälkikäteen iaq-deployment.bicep-tiedosto, jonka avulla pilvipalvelut voidaan luoda uudelleen nopeasti. Azure CLI:llä voidaan tiedoston avulla luoda palvelut komennolla 'az deployment create group create --resource-group <Azuren resurssiryhmä> --template-file iaq-deployment.bicep'.



Kuva 28. iaq-deployment.bicep-tiedoston resurssit

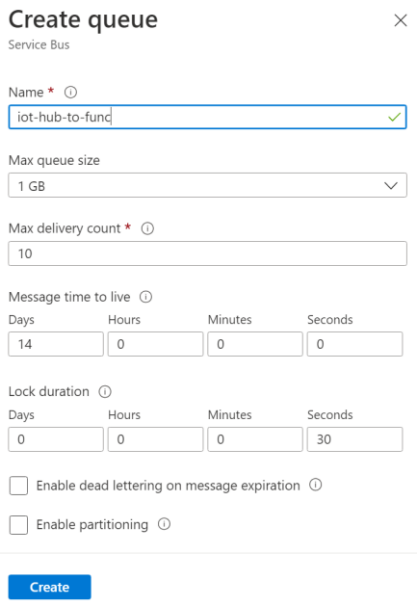
Pilvipalveluita varten sain Savonia-ammattikorkeakoululta Visual Studio Professional -tilauksen, jonka mukana tulee 45€ / kk arvosta Azure-krediittejä. Ensin Azuressa tilaukselle luotiin resurssiryhmä (engl. "resourse group"), jonka nimeksi annettiin "IAQ". Resurssiryhmä toimii eräänlaisena säiliönä, jonka avulla voi koota ja jäsennellä erilaisia kokonaisuuksia. Loput pilvipalvelut lisätään siis tähän resurssiryhmään.

5.5.1 Service Bus Queue

Service Bus Queue:n luominen on nopeaa ja yksinkertaista Azuren portaalin kautta. Service Bus:lle pitää määrittää nimi, fyysinen sijainti ja palvelun taso. Nimen tulee olla uniikki kaikkien Service Bus -palveluiden keskuudessa, joten kaikista yleisimmät nimet ovat jo käytössä.

Työtä varten palvelun tasoksi riittää hyvin edullisin taso (5 snt / miljoona operaatiota / kk). Jos yksi laite lähettäisi yhden viestin joka minuutti kuukauden ajan, olisi viestien kokonaismäärä kuukaudessa 43 200 kpl (60 kpl/h * 24 h/d * 30 d/kk). Keskimäärin jokainen viesti aiheuttaa kaksi operaatiota jonossa (kirjoitus ja luku), eli operaatioita kuukaudessa olisi 86 400 kpl (2 * 43 200 kpl). Jonon hinta kuukaudessa tällä käytöllä olisi alle sentin (5 snt * [43 200 operaatiota / 1 000 000 operaatiota] = 0,432 snt). Jonon käyttö lisää siis viestinnän joustavuutta ja luotettavuutta järjestelmässä lähes ilmaiseksi.

Service Bus voi sisältää useamman jonon. Työssä on kuitenkin tarve vain yhdelle jonolle IoT Hub:n ja funktion välillä. Jonon luominen vaatii muutaman tiedon, kuten nimen, maksimikoon ja kuinka kauan viestejä säilytetään (Kuva 29).



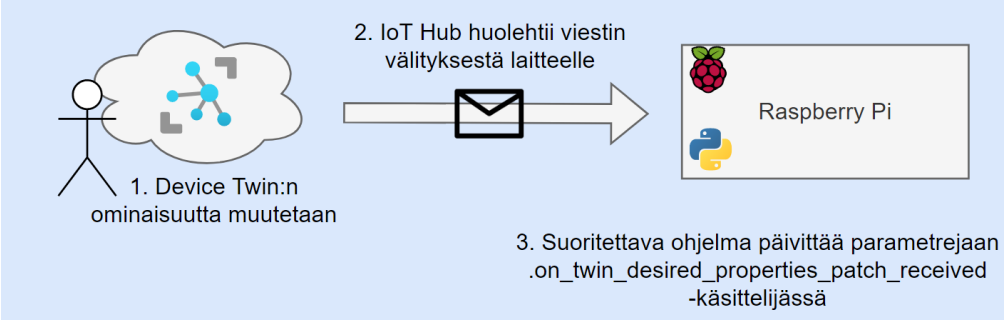
Kuva 29. Service Bus Queue:n luominen

5.5.2 IoT Hub

IoT Hub sisältää paljon erilaisia ominaisuuksia erilaisiin tarpeisiin. Työn kannalta tärkeitä ominaisuuksia ovat Cloud-to-Device-viestit ja Device Twin -ominaisuus. Nämä ominaisuudet yhdessä mahdollistavat tiedonkeräämisen parametrien muuttamisen ottamatta suoraan esim. SSH:lla etäyhteyttä Raspberry Pi:hin. Cloud-to-Device-viestit tulee kuitenkin erikseen käsitellä laitteella, joten ominaisuuden hyödyntäminen lisäsi monimutkaisuutta laitteella suoritettavaan ohjelmaan. Tiedonkeruun parametrien muokkaus tapahtuu kolmessa vaiheessa:

1. Azure Portalissa muokataan laitteen Device Twin:n ominaisuutta.
2. Azuresta lähtee viesti laitteelle (Cloud-to-Device).

3. Laitteella suoritettava ohjelma vastaanottaa viestin ja muuttaa toimintaansa.

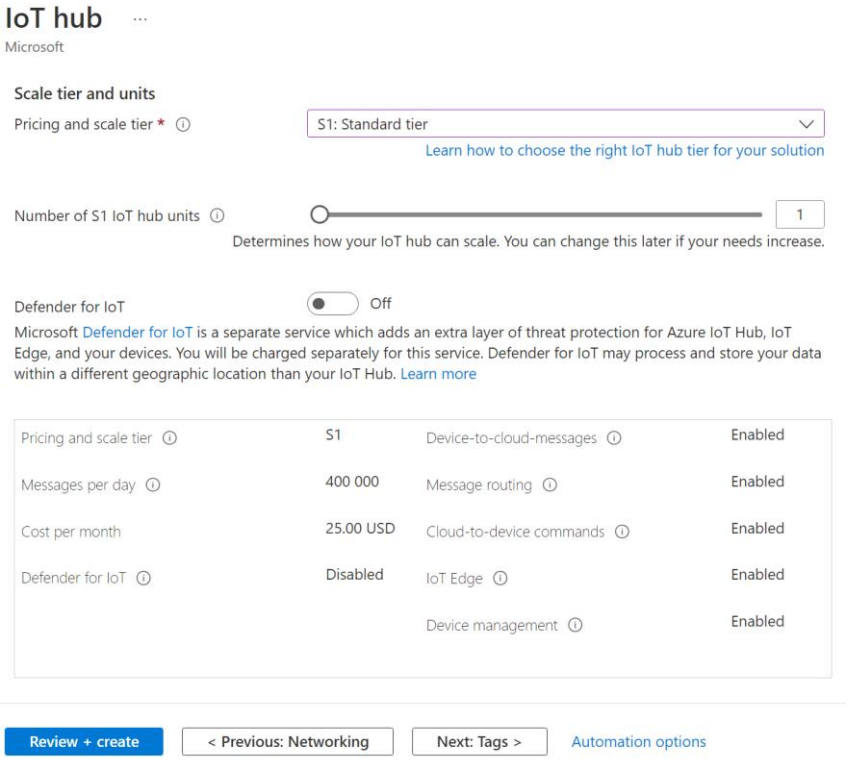


Kuva 30. Device Twin:n haluttujen ominaisuuksien muuttaminen

IoT Hub:n nimeksi asetettiin IAQ-hub. IoT Hub:lla on monta eri tasoa, joiden mukaan palvelussa on erilaisia toimintoja käytössä. IoT Hub:n tasoksi valittiin edullisin taso, joka sisältää Cloud-to-Device-ominaisuuden, eli S1 (Standard tier 1).

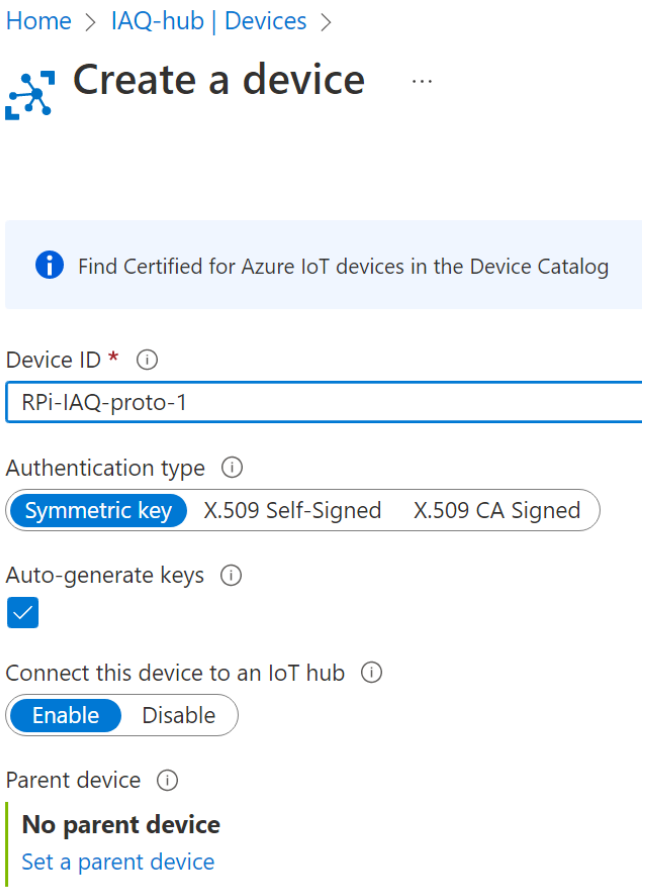
IoT Hub:n skaalautuvuuden määrittää se, kuinka monta ”yksikköä” otetaan käyttöön. Yksiköiden määrä ja palvelun taso määrittävät, kuinka monta viestiä IoT Hub voi käsitellä päivässä. Koska järjestelmään ollaan liittämässä vain yksi laite, riittää yksi yksikkö. Jos laitteita lisättäisiin myöhemmin lisää, voitaisiin myös yksiköitä lisätä jälkikäteen.

Yhden S1-tason yksikön IoT Hub maksaa kuukaudessa n. 24,7 €. IoT Hub käsittelee silloin 400 000 viestiä päivässä ja tärkeimmät IoT Hub:n ominaisuudet ovat käytettävissä, kuten Cloud-To-Device-viestit ja laitteiden hallinta.



Kuva 31. IoT Hub:n luominen

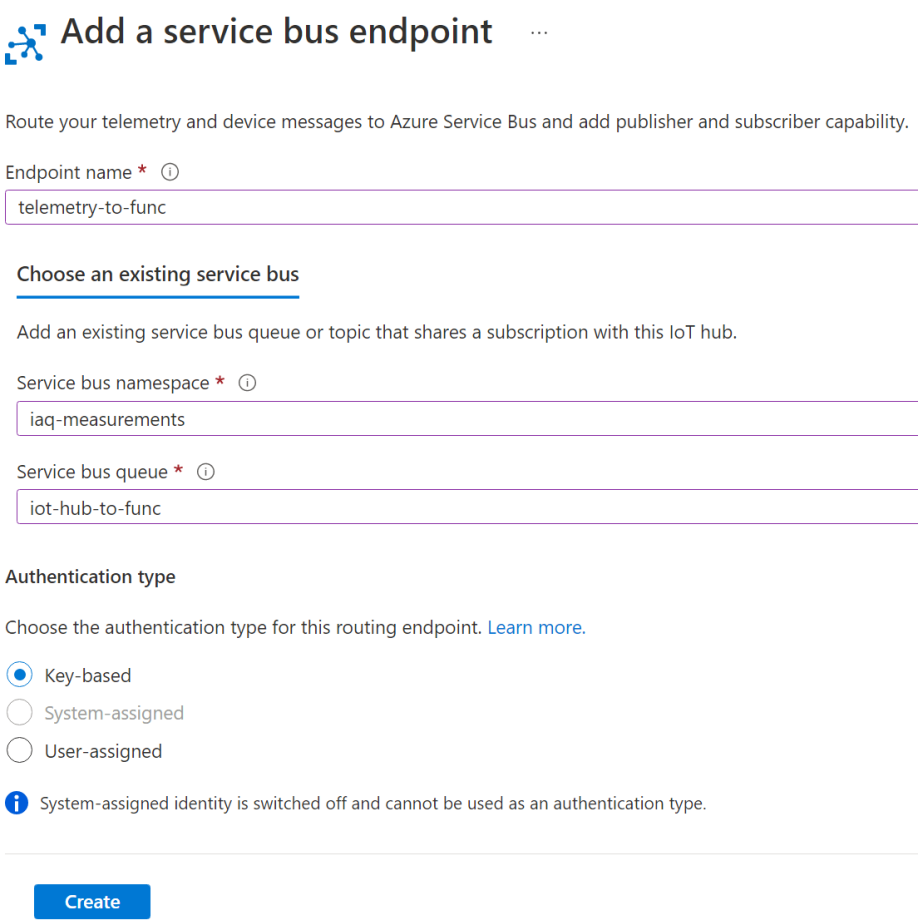
IoT Hub:n luomisen jälkeen laitteidenhallinnassa voidaan lisätä uusi laite IoT Hub:iin. Laitteen lisäämistä varten tarvitsee määrittää laitteen Id ja autentikoinnin tyyppi (symmetrinen salausavain tai X.509 sertifikaatti). Laitteelle annettiin kuvaava Id ”RPi-IAQ-proto-1” ja autentikoinnin tyypiksi valittiin symmetrinen salausavain (Kuva 32). Kun laite on luotu, voidaan laitteen yhteysmerkkijono hakea laitteen tiedoista. Tieto pitää lisätä fyysisellä laitteella suoritettavalle ohjelmalle, jotta se voi muodostaa yhteyden IoT Hub:iin.



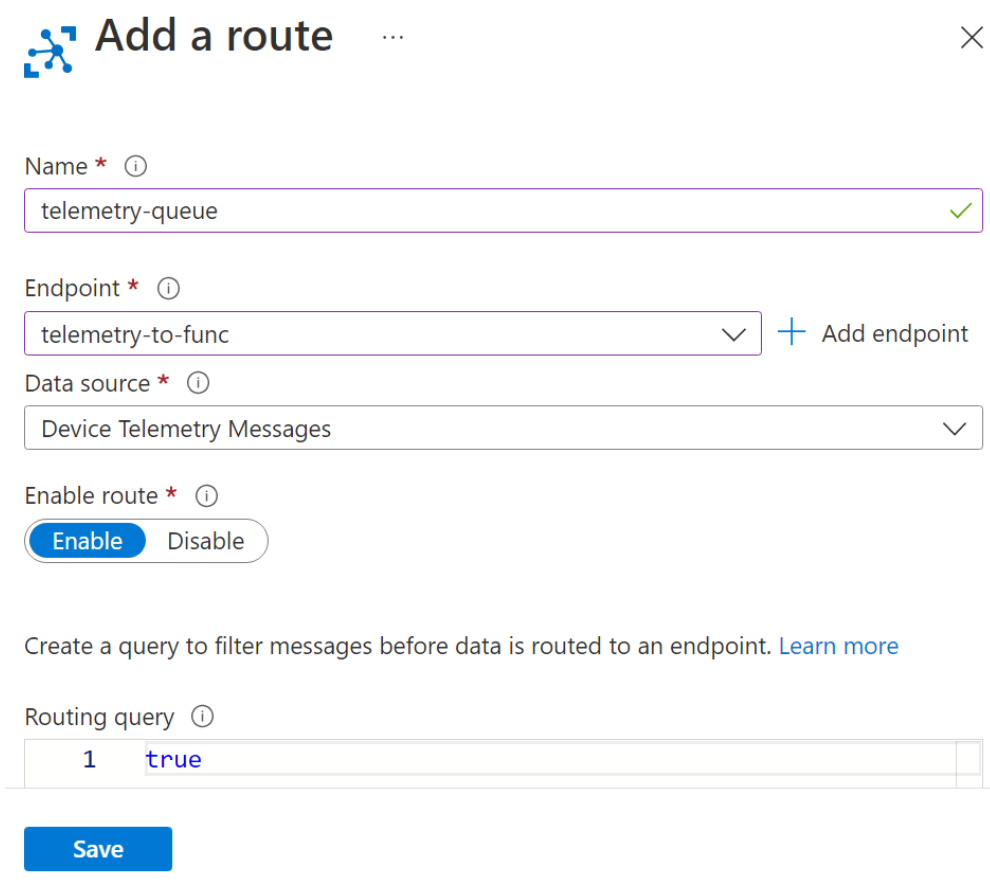
Kuva 32. Laitteen lisääminen IoT Hub:iin

IoT Hub voi ohjata viestejä erilaisiin päätepisteisiin, jotka käytännössä ovat muita pilvipalveluita Azuressa, kuten Service Bus Queue, Service Bus Topic, Blob Storage tai Event Hub. Ohjauksen määrittämistä varten IoT Hub:iin tulee ensiksi lisätä päätepiste (engl. ”endpoint”) ja sitten reitti (engl. ”route”) päätepisteeseen. Ohjattavia viestejä voidaan rajata reitityskyselyillä. Siten viestien ohjaaminen erilaisiin päätepisteisiin viestin ominaisuuksien perusteella on mahdollista.

Hub:in asetuksissa, viestien reitityksessä voidaan lisätä uusi päätepiste. Päätepisteeksi lisättiin aiemmin luotu jono (Kuva 33). Seuraavaksi IoT Hub:iin luotiin reitti, joka ohjaa viestit tähän päätepisteeseen. Reitin luomista varten pitää reitille antaa nimi ja päätepiste ja valita, millaiset viestit ohjataan päätepisteeseen. Työssä ei ole tarvetta tarkemmalle viestien ohjaukselle, joten reitityskyselyksi jätettiin ’true’, joka ohjaa kaikki valitun tyyppiset viestit päätepisteeseen (Kuva 34).



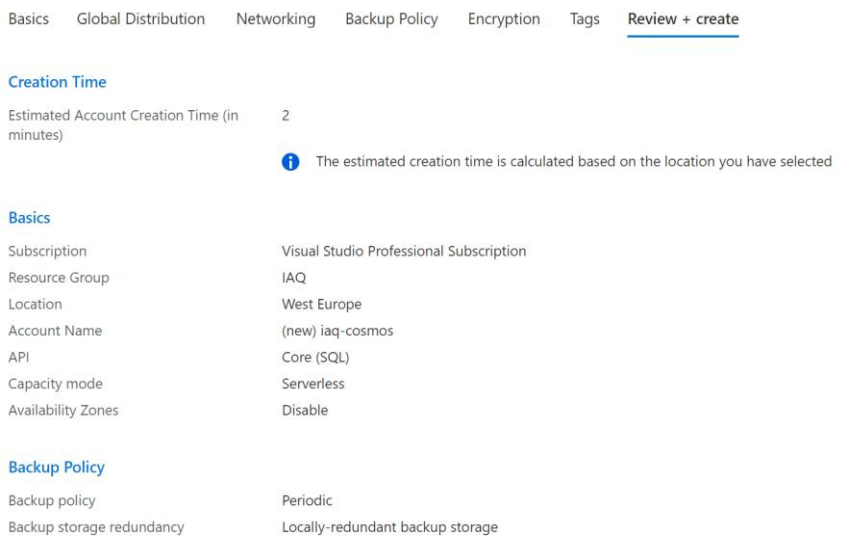
Kuva 33. Päätepisteen lisääminen IoT Hub:iin



Kuva 34. Reitin lisääminen IoT Hub:iin

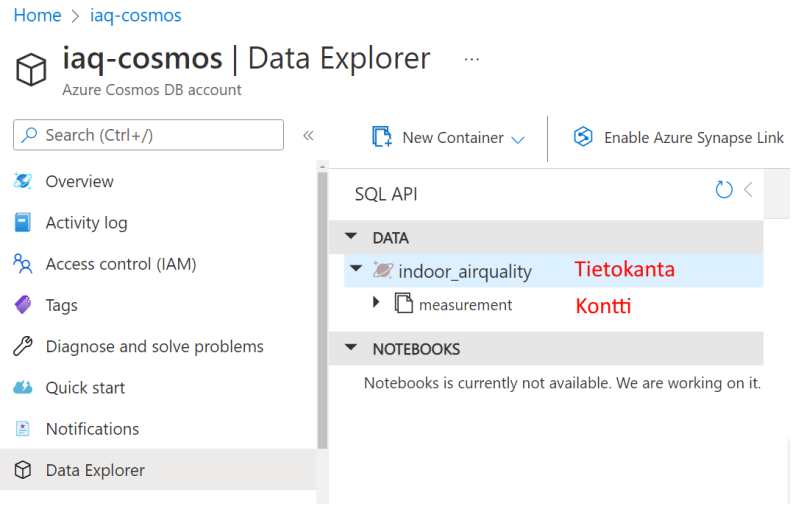
5.5.3 CosmosDB

CosmosDB-tietokannan luomisessa ensimmäinen vaihe oli valita käytettävä API. Valintaa ei voi vaihtaa myöhemmin. Työssä valitsin käytettäväksi Core API:n, koska se on suositeltu API uusien sovelluksien rakentamiseen eikä minulla ole merkittävää erityisosaamista muista API:sta. Seuraava iso päätös oli valita palvelun laskutuksen peruste varatun kapasiteetin ja palvelittoman väliltä. Palveliton sopii hyvin prototyyppien ja kehityksen ajaksi, jolloin tietokannan käyttö on satunnaisempaa. Varattu kapasitetti on usein halvempi valinta, jos tietokannan kuorma on tasainen ja tarkkaan arvioitavissa. Valitsin palvelittoman kapasiteetin, koska mahdollinen säästö kuorman arvioinnilla yhden laitteen tapauksessa ei ole merkittävä.



Kuva 35. CosmosDB-palvelun luominen

Luodussa CosmosDB-palvelussa ei vielä ole tietokantoja, vaan sellainen pitää erikseen luoda. Yhdessä CosmosDB-palvelussa voi olla useampi tietokanta-instanssi, ja yksi tietokanta voi sisältää useamman kontin (engl. ”container”). Kontti on kokoelma samankaltaisia dokumentteja. Työssä on kuitenkin tarve vain yhdelle tietokannalle ja kontille, jonne tallennetaan mittaustulokset. Tietokanta nimettiin ”indoor_airquality” ja kontin nimeksi annettiin ”measurements”.

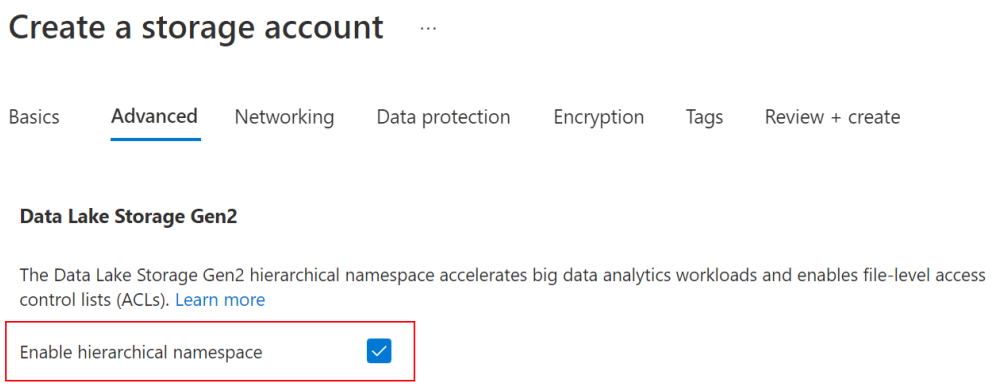


Kuva 36. Tietokanta ja kontti CosmosDB:ssä

Tärkeä valinta kontin luomisessa on jako-avaimen (engl. ”partition key”) valitseminen. Jako-avaimen pitää esiintyä jokaisessa kontin dokumentissa. Sen perusteella dokumentteja voidaan käsitellä samanaikaisesti kontin eri osissa. Konttien jakamisella osiin saavutetaan paras tietokannan suorituskyky. Työssä valitsin jako-avaimeksi laitteen Id:n. Valinta ei jaa tietokantaliikennettä yhden laitteen tapauksessa ollenkaan. Vasta jos laitteita olisi useampia, saavutettaisiin valitusta jako-avaimesta hyötyä. Järjestelmän prototyyppivaiheessa tietokannan suorituskyky ei kuitenkaan ole koetuksella. Jako-avaimen muuttaminen ei ole mahdollista kontin luomisen jälkeen, joten jako-avain valittiin järjestelmän mahdollista laajentamista silmällä pitäen.

5.5.4 Data Lake

Jaettavat .csv-tiedostot on tarkoitus tallentaa Azure Data Lake Gen2 -palveluun, joka luodaan Storage Account:na Portalissa. Storage Account:in luomisessa pitää asettaa hierarkinen nimiavaruus erikseen päälle, koska asetus ei ole oletuksena päällä (Kuva 37). Hierarkinen nimiavaruus parantaa erilaisten Big Data -sovellusten suorituskykyä, koska niiden toiminta usein vaatii väliaikaisia kansioita. Hierarkisessa nimiavaruudessa näiden väliaikaisten kansioiden poistaminen tai uudelleen nimeäminen on paljon tehokkaampaa kuin litteässä nimiavaruudessa (Microsoft Azure, 2021). Hierarkinen nimiavaruus on myös käyttäjille ja kehittäjille todennäköisesti tutumpi vaihtoehto, koska tietokoneiden tiedostojärjestelmät käyttävät hierarkista nimiavaruutta.

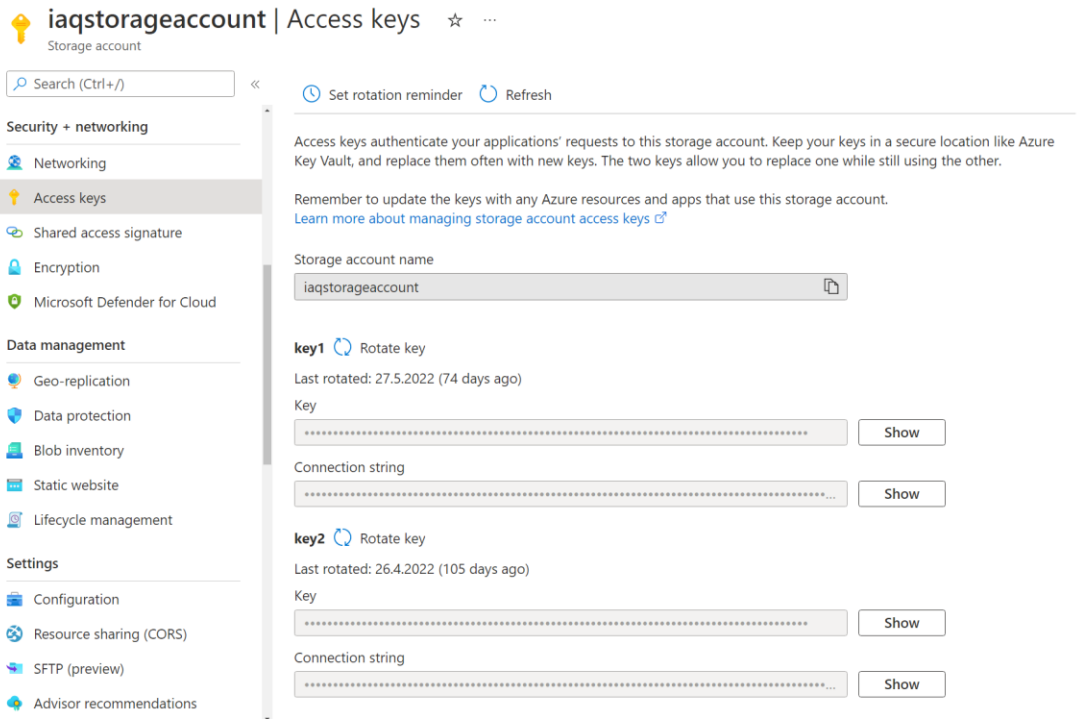


Kuva 37. Hierarkinen nimiavaruus Storage Account:n luomisessa

Toinen muutettava valinta koskee blob:ien oletuspääsytasoa (engl. ”access tier”). Valintoina oli joko kuuma (engl. ”hot”) tai kylmä (engl. ”cold”). Pääsytaso vaikuttaa tallennettavien blob:ien kustannuksiin ja siihen, kuinka nopeasti tieto on ladattavissa. Työssä päädyin valitsemaan oletuspääsytasoksi kylmän, koska tiedostojen lukukerrat ovat suhteellisen harvinaisia. Oppituntien aikana blob:ien lukukerrat näkevät kuitenkin hetkellisen piikin. Jos tästä piikistä aiheutuu ongelmia, pystytään blob:ien pääsytasoa muuttamaan myös jälkikäteen.

Palvelun luomisen jälkeen se on valmis käytettäväksi. Tunnistautuminen palveluun tapahtuu salasanan avulla (engl. ”access key”, Kuva 38). Storage Account:lla voi olla samaan aikaan kaksi erilaista, voimassa olevaa salasanaa. Tämä on tärkeää, jos salasanoja halutaan kierrättää siten, että muut Storage Account:ia käyttävät palvelut voivat pitää yhteyden Storage Account:iin kaiken aikaa.

Salasanojen kierrätys on ylläpidollinen toimi, joka parantaa tietoturvaa. Kierrätys kannattaa tehdä ajoittain tai jos on epäilys siitä, että salasana olisi vuotanut julkisuuteen.



Kuva 38. Storage Account:n käyttöön vaadittavat salausavaimet ja yhteysmerkkijonot

5.5.5 Functions

Järjestelmässä on kaksi eri funktiota. Tietovirrassa ensimmäinen funktio ottaa vastaan ilmanlaadun mittaus -viestejä jonolta (Service Bus Queue) ja tallentaa ne CosmosDB:ssä olevaan tietokantaan. Tietovirrassa jälkimmäinen funktio lukee CosmosDB:n muutosvirrasta funktion edellisen suorituksen jälkeen lisätyt mittaukset ja muodostaa niistä yhden .csv-tiedoston. Funktio lopuksi tallentaa tiedoston Data Lake:een.

Ensimmäisen funktion toiminta on erittäin yksinkertainen. Funktio ottaa vastaan viestin jonolta ja tallentaa sen tietokantaan. Sekä jonoa että tietokantaa varten on olemassa NuGet-paketit, joiden avulla palveluita voidaan käyttää. Jonoa varten funktioon lisätään paketti ”Microsoft.Azure.WebJobs.Extensions.ServiceBus” ja tietokantaa varten paketti ”Microsoft.Azure.WebJobs.Extensions.CosmosDB”. ServiceBus-paketti sisältää samalla ServiceBusTrigger-laukaisimen funktiolle. Jonolta tullut viesti deserialisoidaan json:sta olioksi. Tämän jälkeen tallennettavan dokumentin kenttiin asetetaan olion vastaavat arvot (Kuva 39).

```
[FunctionName("MeasurementInsert")]
0 references | Lauri Hyvärinen, 19 minutes ago | 1 author, 2 changes
public void Run([ServiceBusTrigger("iot-hub-to-func", Connection = "QueueConnectionString")]string measurementMessage,
    [CosmosDB(databaseName: "indoor_airquality", containerName: "measurement", Connection = "CosmosDBConnection")] out dynamic document,
    ILogger log)
{
    MeasurementDTO measurement = JsonConvert.DeserializeObject<MeasurementDTO>(measurementMessage);
    document = new
    {
        id = measurement.id,
        deviceId = measurement.deviceId,
        time = measurement.time.ToUniversalTime(),
        co2 = measurement.co2,
        pressure = measurement.pressure,
        temp = measurement.temp,
        humidity = measurement.humidity,
        voc = measurement.voc,
    };
}
```

Kuva 39. Funktio, joka tallentaa jonolta vastaanotetut viestit CosmosDB:een

Toinen funktio on myös toiminnaltaan melko yksinkertainen. Funktion laukaisimena toimii ajastin (engl. ”timer trigger”), jonka arvoa voidaan muuttaa palvelun konfiguraation kautta. Täten .csv-tiedostoja voidaan helposti muodostaa tiheämpään tahtiin oppituntien aikana, jolloin tiedostoja halutaan syntyvän useammin kuin normaalisti.

Funktion erikoisvaatimuksena on se, että sen pitää pystyä jatkamaan muutosvirran lukemista siitä kohdasta, johon edellisen funktion suoritus loppui. Tarkoitusta varten muutosvirrasta saa jatkamismerkin (engl. ”continuation token”). Jatkamismerkki voidaan tallentaa käyttäen Durable Entity:ä (Kuva 40). Kun funktio on lukenut kaikki muutokset muutosvirrasta, otetaan sen hetkinen jatkamismerkki ja kirjoitetaan se Durable Entity:n tilaan. Seuraavalla suorituskerralla voidaan hakea viimeksi tallennettu jatkamismerkki, jonka avulla muutosvirran lukemista jatketaan oikeasta kohtaa. Jos funktio ei löydä jatkamismerkkiä, esim. ensimmäisellä suorituskerralla, luetaan muutosvirtaa vain yksi päivä menneisyyteen (Kuva 41).

```
[JsonObject(MemberSerialization.OptIn)]

public class ContinuationTokenEntity
{
    [JsonProperty("continuationToken")]

    public string ContinuationToken { get; set; }


    public string Get() => this.ContinuationToken;

    public void Set(string token) => this.ContinuationToken = token;

    [FunctionName(nameof(ContinuationTokenEntity))]

    public static Task Run([EntityTrigger] IDurableEntityContext ctx) => ctx.DispatchAsync<ContinuationTokenEntity>();
}
```

Kuva 40. Jatkamismerkkiä varten luotu Durable Entity

```
// ContinuationToken is stored in a DurableEntity, that persists between function executions
var contTokenId = new EntityId(nameof(ContinuationTokenEntity), "continuationToken");
EntityStateResponse<ContinuationTokenEntity> continuationTokenResponse = await entityClient.ReadEntityStateAsync<ContinuationTokenEntity>(contTokenId);

if (continuationTokenResponse.EntityExists) // Found a continuation token, use it
{
    string continuationToken = continuationTokenResponse.EntityState.Get();
    feedIterator = measurementsContainer.GetChangeFeedIterator<MeasurementDTO>(
        ChangeFeedStartFrom.ContinuationToken(continuationToken),
        ChangeFeedMode.Incremental);
}
else // No continuation token found, go back one day in the change feed
{
    feedIterator = measurementsContainer.GetChangeFeedIterator<MeasurementDTO>(
        ChangeFeedStartFrom.Time(DateTime.UtcNow.AddDays(-1)),
        ChangeFeedMode.Incremental);
}
```

Kuva 41. Jatkamismerkin haku Durable Entity:stä ja muutosvirran luominen sen avulla

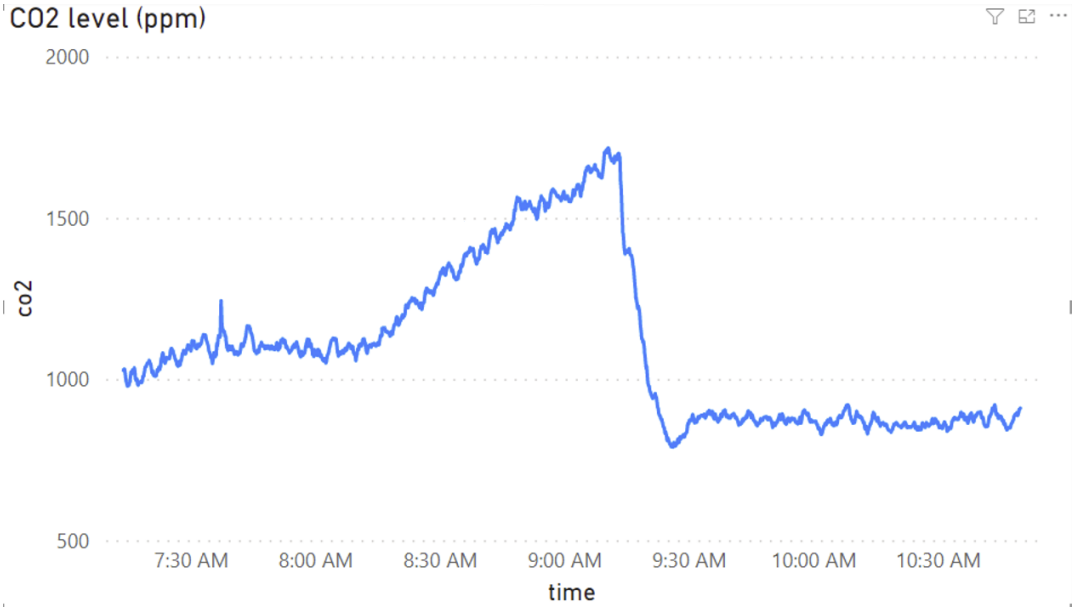
Funktiossa on käytössä neljä eri NuGet-pakettia: yksi CosmosDB:tä varten (”Microsoft.Azure.WebJobs.Extensions.CosmosDB”), toinen Data Lake:a varten (”Microsoft.Azure.WebJobs.Extensions.Storage.Blobs”) kolmas paketti Durable Entity:jä varten (”Microsoft.Azure.WebJobs.Extensions.DurableTask”) ja neljäs paketti on pakollinen Azure Functions-projekteissa (”Microsoft.NET.Sdk.Functions”) (Kuva 42).

```
<PackageReference Include="Microsoft.Azure.WebJobs.Extensions.CosmosDB" Version="4.0.0-preview3" />
<PackageReference Include="Microsoft.Azure.WebJobs.Extensions.DurableTask" Version="2.7.0" />
<PackageReference Include="Microsoft.Azure.WebJobs.Extensions.Storage.Blobs" Version="5.0.0" />
<PackageReference Include="Microsoft.NET.Sdk.Functions" Version="4.0.1" />
```

Kuva 42. Toisen funktion NuGet-paketit

6 LOPPUTULOS

Kuvassa 42 on esimerkki PowerBi-raportista, jossa näkyy työhuoneeni CO_2-pitoisuus. Kuvaajasta erottuu neljä eri vaihetta. Alussa CO_2-pitoisuus on melko tasainen, kun työhuoneen ovi on auki ja ikkuna kiinni. Oven sulkemisen jälkeen CO_2-pitoisuus alkaa kasvamaan suhteellisen tasaiseen tahtiin. Kolmannessa vaiheessa avasin huoneen oven ja ikkunan, jolloin huone tuulettui nopeasti, ja CO_2-pitoisuus laski äkkiä. Lopussa CO_2-pitoisuus on taas tasainen, kuitenkin alemmalla tasolla kuin alussa, jolloin huoneen ikkuna oli kiinni.



Kuva 43. Yksinkertainen PowerBI-raportti työhuoneen CO_2 -pitoisuudesta

Kokonaisuudessaan työ on onnistunut. Koko ketju antureilta Raspberry Pi:lle, Pi:ltä Azureen ja Azuren kaikkien eri palveluiden läpi toimii. Sain projektin aikana tutustuttua moniin eri Azuren tarjoamiin pilvipalveluihin, niiden tietoturvaan, toimintoihin ja kustannuksiin. Erityisesti Azuren eri palveluiden käyttö ja tiedonhaku niiden ympärillä on edistänyt ammatillista osaamistani merkittävästi. Tiedonhaun aikana oli helppoa huomata, miten nopeasti pilvipalvelut ovat kehittyneet ja kehittymässä. Artikkelien, blogien ja keskustelupalstojen kommenttien neuvot ja esimerkit saattoivat olla jo vanhentuneita, vaikka niiden julkaisusta olisi kulunut vain vuosi. Microsoft Azuren oma dokumentaatio ja esimerkit ovat kuitenkin laadukkaita ja ajantasaisia, joten suurimman osan ongelmista sain ratkaistua niiden avulla.

Työ soveltuu sellaisenaan osaksi useaa eri tietotekniikan tutkinto-ohjelman kurssia. Big data -kurssilla voidaan hyödyntää reaaliaikaisesti generoituvaa tietoa, .csv-tiedostoista voidaan tehdä raportteja Business Intelligence -kurssilla ja koko järjestelmä sopisi demoksi tai esimerkiksi IoT-ohjelmointi -kurssille.

Työn kustannukset pysyivät hyvin budjetissa. Kaikkien pilvipalveluiden kokonaiskustannukset ovat noin 30 €/kk, joka jää reilusti alle työtä varten annetun Visual Studio Professionalin Azure-krediittien (45 €/kk). Myös laitteiden hinnat jäivät kohtuullisiksi. Savonialla oli ennestään Raspberry Pi, jota voitiin työssä käyttää. Samoin DHT-11 anturi löytyi Savonialta valmiina. Työssä käytetyn laitteiston

kokonaishinta uutena työn kirjoitushetkellä olisi noin 170 €. Suurin osa kokonaishinnasta on Raspberry Pi:ssä. Raspberry Pi:n kyseisen mallin aloituspakkauksen hinta on nyt jopa 130 €.

Järjestelmän tietoturva on tyydyttävällä tasolla, joka täyttää vähintäänkin nykyaikaiset minimivaatimukset tietojärjestelmältä. Kaikkien palveluiden väliset yhteydet käyttävät TLS-suojausta (Transport Layer Security). Palveluihin tunnistautuminen tapahtuu joko symmetrisellä salausavaimella tai vahvalla salasanalla. Raspberry Pi:n käyttäminen vaatii käyttäjätunnuksella ja salasanalla kirjautumisen. Laitteen loppusijoituspaikalla on kuitenkin suuri merkitys tietoturvaan: laitteen pitäisi olla piilossa ja vaikeasti saavutettavassa paikassa. Heikoin kohta järjestelmän tietoturvassa on luultavasti nimenomaan laitteen fyysinen turvallisuus ja loppukäyttäjien toiminta. Esimerkiksi jos palveluiden väliseen viestintään käytettyjä yhteysmerkkijonoja tai salausavaimia näkyy opetuskäytössä opiskelijoille, pitää nämä välittömästi luoda uudelleen.

Työn aikana pääsin tutustumaan erilaisiin antureihin ja viestintäspesifikaatioihin. Sain myös perehdyttyä sekä sisäilman laatuun vaikuttaviin tekijöihin että sen terveysvaikutuksiin ihmisille. Aihe on ollut erityisen ajankohtainen jo kohta kolmen vuoden ajan, kun koronapandemia on vaivannut ihmiskuntaa. IoT on myös vahvasti kasvussa oleva ala ja olen onnellinen, että pääsin tekemään opinnäytetyöni siihen liittyen. Aion jatkaa IoT:n kehityksen seuraamista ja sen parissa työskentelyä.

7 JATKOKEHITYS

Työssä on monta eri osaa, joita voitaisiin jatkokehittää. IoT Hub:in voisi korvata IoT Central:illa, jonka avulla saataisiin valmis kojelauta laitteiden tarkkailuun, yhdistämiseen ja mahdollisten ilmoitusten lähettämiseen. Central tarjoaa valmiita pohjia erilaisten toimialojen perusteella, jotta käyttöönotto voisi olla nopeaa. Azure Sphere:n avulla puolestaan voitaisiin parantaa järjestelmän tietoturvaa. Azure Sphere:n avulla voidaan muodostaa salattu end-to-end-yhteys laitteen käyttöjärjestelmän tasolta Azuressa olevaan IoT-ratkaisuun (IoT Hub tai IoT Central).

Anturit on nyt kytketty hyppylangoilla koekytkentäalustaan. Antureita varten voitaisiin tulostaa piirilevy, jolloin anturit olisivat paljon vakaammin kiinnitettyinä alustaansa. Siten anturien kiinnittäminen piirilevyyn helpottaisi työn liikuteltavuutta huomattavasti. Hyvä puoli nykyisessä ratkaisussa kuitenkin on se, että anturit ovat helposti irrotettavissa muuhun käyttöön, jos tämä projektityö ”romutetaan” joskus. Uusien antureiden liittäminen työhön on myös vielä paljon helpompaa, kun käytössä on koekytkentäalusta ja hyppylangat.

Laite voisi toimia Report by Exception-mallilla, eli ilmoittaa tietyn suureen uusi, muuttunut arvo. Nyt kaikkien suureiden tiedot lähetetään joka sekunti, mikä osaltaan auttaa ratkaisun käyttöä opetuksessa, luomalla tasaisen ja ennustettavan tietovirran. Jos ratkaisun pääasiallinen tarkoitus olisi tarjota tietoa sisäilmasta ja laitteita olisi useampia, voitaisiin käytetyissä viesteissä, tietokannassa ja lähes kaikissa muissa Azuren palveluissa säästää rahaa käyttämällä Report by Exception -mallia. Ero kustannuksissa viestintämallien välillä kasvaa, kun laitteita lisättäisiin järjestelmään. Nykyinen malli toimii siksi, että laitteita on vain yksi ja sen on tarkoitus tuottaa tietoa nopeaan tahtiin Big Data -kurssin tarpeita varten.

CosmosDB voitaisiin korvata instanssilla, jonka laskutus perustuu varattuun kapasiteettiin. IoT laitteen käyttämä kapasiteetti on helppo laskea, koska sen pitäisi olla suhteellisen tasaista, joten varattu kapasiteetti voisi tulla halvemmaksi. Tätä kohtaa on hyvä miettiä edellisen kappaleen Report by Exception -mallin kanssa. Myös anturien suurinta mahdollista resoluutiota ei välttämättä tarvitsisi käyttää. Esim. 10 ppm CO_2 -pitoisuuden muutos saattaisi olla lähettämisen arvoinen tieto.

CosmosDB:hen tallennettuja tietoja voitaisiin myös hyödyntää enemmän esim. visualisoimalla erillisellä verkkosovelluksella. Nykyisen arkkitehtuurin tarkoitus on toimia helposti laajennettavana lähtökohtana, johon voidaan lisätä muita sovelluksia. Itsessään .csv-tiedostoja voitaisiin tietysti helposti muodostaa jo Raspberry Pi:llä, mutta sitä ratkaisua ei olisi kovin helppoa tai luontevaa jatkokehittää.

Työssä on myös mahdollisuus säästöihin. Kokonaisen tietokoneen (Raspberry Pi) voisi hyvin korvata pienemmällä ja halvemmalla mikrokontrollerilla, kuten Arduino Nano (noin 22 €) tai ESP8266 (noin 10 €). Samoin pilvipalveluissa on säästämisen varaa, erityisesti jos tietyistä IoT Hub:n ominaisuuksista voi luopua. IoT Hub:n osuus pilvipalveluiden laskusta on reilusti yli puolet, joka voitaisiin säästää käyttämällä IoT Hub:n ilmaista tasoa. IoT Hub:n ilmainen taso tarjoaa 8000 viestiä laitteelta pilveen päivässä. Tämä vastaa noin yhtä viestiä 30 sekunnin väliajoin. Ilmaiseen tasoon ei kuitenkaan kuulu kaikki työssä käytetyt ominaisuudet, kuten Digital Twin ja Cloud-to-Device-viestit.

LÄHTEET

.NET Platform. (ei pvm). Roslyn. Haettu 16. heinäkuuta 2022 osoitteesta https://github.com/dotnet/roslyn

Adafruit. (9. kesäkuuta 2022). *PyPi*. Haettu 9. elokuuta 2022 osoitteesta https://pypi.org/project/adafruit-circuitpython-dht/

Azure Github. (24. toukokuuta 2022). *Azure IoT device samples.* Haettu 04. elokuuta 2022 osoitteesta https://github.com/Azure/azure-iot-sdk-python/tree/main/azure-iot-device/samples/async-hub-scenarios

DFRobot. (ei pvm). *wiki.dfrobot.com*. Haettu 15. elokuuta 2022 osoitteesta https://wiki.dfrobot.com/BMP280_Digital_Pressure_Sensor_Module_SKU_SEN0372

ECMA International. (Joulukuu 2001). *ECMA-334 C# language specification.* Haettu 16. heinäkuuta 2022 osoitteesta https://www.ecma-international.org/publications-and-standards/standards/ecma-334/

Hengitysliitto. (ei pvm). *Sisäilman epäpuhtaudet ja hajut*. Haettu 22. toukokuuta 2022 osoitteesta Sisäilman epäpuhtaudet ja hajut: https://www.hengitysliitto.fi/kodin-sisailma-ja-kunnossapito/sisailman-laatu/sisailman-epapuhtaudet-ja-hajut/

Howard, P. (6. syyskuuta 2021). *PyPi BMP280*, 0.0.4. Haettu 6. elokuuta 2022 osoitteesta https://pypi.org/project/bmp280/

JetBrains. (16. kesäkuuta 2021). DevEcoSystem 2021. Haettu 2. elokuuta 2022 osoitteesta https://www.jetbrains.com/lp/devecosystem-2021/csharp/

Mendes, L. O. (13. toukokuuta 2015). NDIR Gas Sensor for Spatial Monitoring of Carbon Dioxide Concentrations in Naturally Ventilated Livestock Buildings. *Sensors, 15*(5), 4. Haettu 05. kesäkuuta 2022 osoitteesta https://www.mdpi.com/1424-8220/15/5/11239

Microsoft. (28. tammikuuta 1997). Microsoft Announces Visual Studio 97. Haettu 02. elokuuta 2022 osoitteesta https://news.microsoft.com/1997/01/28/microsoft-announces-visual-studio-97-a-comprehensive-suite-of-microsoft-visual-development-tools/

Microsoft Azure. (24. lokakuuta 2021). *Azure Data Lake Storage Gen2 hierarchical namespace*. Haettu 09. elokuuta 2022 osoitteesta https://docs.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-namespace

Microsoft Azure. (06. kesäkuuta 2022). *Common CosmosDB use cases*. Haettu 09. elokuuta 2022 osoitteesta https://docs.microsoft.com/en-us/azure/cosmos-db/use-cases

Microsoft Azure. (06. kesäkuuta 2022). *Consistency levels in Azure Cosmos DB*. Haettu 28. heinäkuuta 2022 osoitteesta https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels#consistency-levels-and-latency

Microsoft Azure. (06. kesäkuuta 2022). *Welcome to Azure Cosmos DB*. Haettu 28. heinäkuuta 2022 osoitteesta Azure: https://docs.microsoft.com/en-us/azure/cosmos-db/introduction#mission-critical-ready

Mouser. (ei pvm). *Mouser.com.* Haettu 15. elokuuta 2022 osoitteesta https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf

Ponnala, J. (13. syyskuuta 2021). *Unsplash.* Haettu 19. maaliskuuta 2022 osoitteesta Unsplash: https://unsplash.com/photos/jvHymbpto1E

Raspberry Pi. (ei pvm). Software. Haettu 03. elokuuta 2022 osoitteesta https://www.raspberrypi.com/software/

Savonia-ammattikorkeakoulu. (Tammikuu 2020). *Savonian strategia 2021-2024.* Haettu 11. elokuuta 2022 osoitteesta Savonia.fi: https:https://www.savonia.fi/app/uploads/2020/12/Savonian-strategia-2021-2024-tiivistelma-1.pdf

Savonia-ammattikorkeakoulu. (ei pvm). *Savonian viestintä.* Haettu 11. elokuuta 2022 osoitteesta Savonia.fi: https://www.savonia.fi/tutustu-savoniaan/organisaatio-ja-johtaminen/savonian-viestinta/

Savonia-ammattikorkeakoulu. (ei pvm). *Tutustu Savoniaan*. Haettu 11. elokuuta 2022 osoitteesta Savonia.fi: https://www.savonia.fi/tutustu-savoniaan/

Stack Overflow. (ei pvm). 2022 Developer Survey. Haettu 16. heinäkuuta 2022 osoitteesta https://survey.stackoverflow.co/2022/#technology-most-popular-technologies

Takeyuki UEDA. (25. helmikuuta 2022). *pypi mh-z19*. Haettu 8. elokuuta 2022 osoitteesta https://pypi.org/project/mh-z19/

TheRegister. (19. elokuun 2022). *TheRegister.com*. Haettu 24. elokuun 2022 osoitteesta https://www.theregister.com/2022/08/19/google_iot_core_axed/

Tomohito, K. (2017). Ohjelmistokonttien CI/CD-palveluiden määrittely ja asennus Kubernetes-ohjelmistokonttiklusteriin. Haettu 1. elokuuta 2022 osoitteesta https://urn.fi/URN:NBN:fi:amk-201705239747

Tran V. V., P. D.-C. (2020). Indoor Air Pollution, Related Human Diseases, and Recent Trends in the Control and Improvement of Indoor Air Quality. *International Journal of Environmental Research and Public Health*, 1. Haettu 11. heinäkuuta 2022 osoitteesta https://www.mdpi.com/1660-4601/17/8/2927

Yle. (10. heinäkuuta 2021). Korona herätti ilmanvaihdon puutteisiin - "hengitystiesairauksien torjunta unohdettu". Suomi. Haettu 16. helmikuuta 2022 osoitteesta https://yle.fi/uutiset/3-12016640

Yle. (10. heinäkuuta 2022). Huippututkijat: paremmalla ilmanvaihdolla olisi mahdollista torjua hengitysteitse leviäviä epidemioita. Haettu 11. heinäkuuta 2022 osoitteesta https://yle.fi/uutiset/3-12518229

Zhengzhou Winsen Electronics Technology Co., Ltd. (21. tammikuuta 2016). *winsen-sensor.com.* Haettu 15. elokuuta 2022 osoitteesta https://www.winsen-sensor.com/d/files/infrared-gas-sensor/mh-z19b-co2-ver1_0.pdf

Zhengzhou Winsen Electronics Technology Co., Ltd. (01. marraskuuta 2014). *Air Quality (VOC) Sensor.* Haettu 17. maaliskuuta 2022 osoitteesta Winsen-Sensor Web Site: https://www.winsen-sensor.com/sensors/voc-sensor/zp01-mp503.html