Pilvipalvelu IoT-alustana

CASE: IoT-ratkaisu Microsoft Azurella

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Visamäki, Tietojenkäsittelyn koulutusohjelma

Kevät, 2019

Mikko Heikkilä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Tietojenkäsittelyn koulutusohjelma
Visamäki

Tekijä Mikko Heikkilä **Vuosi** 2019

Työn nimi Pilvipalvelu IoT-alustana
Case: IoT-ratkaisu Microsoft Azurella

Työn ohjaaja/t Lasse Seppänen, Hämeen Ammattikorkeakoulu

TIIVISTELMÄ

Esineiden internet eli IoT on yksi nousevista teknologiatrendeistä, ja sen avulla povataan useille toimialoille kustannushyötyjä ja kokonaan uusia liiketoimintamalleja. Esineiden internetissä on kysymys tietojenkäsittelyyn liittyvästä ilmiöstä ja toimintatavasta, jonka avulla kerätään systemaattisesti tietoa sensorien avulla fyysisen maailman ilmiöistä muuntamalla ne digitaaliseen muotoon.

Esineiden internet liittyy voimakkaasti pilvipalveluihin. Pilvipalvelut tarjoavat alustan, mihin sensoreilla kerättävä tieto ohjataan jatkokäsittelyä varten. Alusta on keskeinen työkalu esineiden internetissä. Sen ominaisuuksilla on suuri merkitys, miten tietoa voidaan hyödyntää, ja IoT-järjestelmää hallita.

Azure on Microsoftin pilvipalvelu, joka tarjoaa kehittyneen IoT-alustan. Se toteuttaa käytännössä kaikki hyvälle alustalle asetetut vaatimukset. Tässä työssä on rakennettu Azuren avulla IoT-toteutus paikkatiedon keräämistä varten.

Työn tavoitteena on uuden oppiminen ja tilaajayrityksen tietoisuuden lisääminen koskien esineiden internettiä. Työ vastaa kysymyksiin, miten Azuren avulla on mahdollista toteuttaa IoT-ratkaisu ja miten Azure täyttää alustoille asetetut vaatimukset, sekä millaisia vaatimuksia hyvälle IoT-alustalle voidaan asettaa. IoT-toteutuksessa on ratkaistu ongelma liittyen paikkatietoon. Työn tilaajana on YIT Oyj.

Avainsanat esineiden internet, IoT, Azure, pilvipalvelut, Python

Sivut 61 sivua, joista liitteitä 4 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Visamäki

Author Mikko Heikkilä **Year** 2019

Subject Cloud Service as an IoT Platform
Case: IoT Solution using Microsoft Azure

Supervisors Lasse Seppänen, Häme University of Applied Sciences

ABSTRACT

Internet of Things is one off the main technology trends in computer science today. Hence there are great expectations when it comes to cost efficiency and information-based management. It is also predicted that it is possible to create new business models with it. With Internet of Things, companies can systematically collect data from physical world's phenomena with the help of different kinds of sensors.

Internet of Things is related to cloud services because they provide IoT-specific platform which can be used for collecting telemetry data. IoT-platform is the main tool of the IoT ecosystem. That is the reason why it is so important to choose IoT platform based on the correct requirements. There are eight main requirements of the developed IoT platform. Platform's features play a key role when it comes to analyzing and taking advantages of the collected data and controlling IoT devices.

The main goals of the thesis are to learn about IoT platform, programming the IoT applications and getting familiar with platform requirements. It also creates knowledge for the company. It answers questions such as what are the main characters of the IoT platform and how does Microsoft's Azure fulfill the requirements. It also determines how specific IoT solution can be created with Azure. The mandator of the thesis is YIT Ltd.

Keywords Internet of Things, IoT, Azure, cloud services, Python

Pages 61 pages including appendices 4 pages

KÄSITELUETTELO

Alusta
Alustalla tarkoitetaan tässä yhteydessä pilvipalveluun rakennettua arkkitehtuuria, jota käytetään esineiden internetin sovellusten rakentamiseen ja hallintaan.

Analytiikka
Analytiikka on datan analysointia erilaisten työkalujen avulla. Analytiikan tavoitteena on havaita datasta korrelaatioita. Analytiikkaa käytetään johtamisen apuvälineenä.

API -rajapinta
API-rajapinnalla tarkoitetaan sovellusten välistä kerrosta, jota hyödyntämällä eri sovellukset voivat käyttää tietojen kyselyihin toisiltaan. Rajapinnat ovat tärkeässä roolissa, kun tietoa siirretään eri ohjelmistojen välillä.

Big data
Big data viittaa suuriin tietomääriin. Se mitä voidaan pitää big datana, on aina kuitenkin kontekstiriippuvaista.

Docker
Docker on ohjelmistokonttien rakentamiseen ja hyödyntämiseen rakennettu ohjelmisto. Sen avulla lähdekoodi tai tietokanta, jne. voidaan pakata ohjelmistokonttiin, joka on ympäristöriippumaton. Tavoitteena on käyttää samaa lähdekoodia yhä uudelleen.

IoT
IoT:lla tarkoitetaan esineiden internettiä. Esineiden internet tarkoittaa esineiden liittämistä osaksi internettiä erilaisten sensorien avulla.

Microsoft Azure
Azure on Microsoftin pilvipalvelu. Se sisältää myös esineiden internettiä varten tehtyjä työkaluja.

NoSQL-tietokanta
NoSQL-tietokanta on rakenteeton tai puolirakenteellinen tietokanta. Rakenteettomuutensa vuoksi siihen tallentavan tiedon ei tarvitse noudattaa ennalta määritettyä skeemaa.

Pilvipalvelu
Pilvipalvelu on Internetin yli käytettävä palvelu, jossa resurssit, kuten virtuaalitietokone tai tietokanta, sijaitsevat.

Reunalaskenta
Reunalaskennalla viitataan pilvipalvelun ”reunalla” sijaitseviin, laskentaa suorittaviin laitteisiin. Tällöin laskenta tapahtuu pilvipalvelun sijasta muualla.

Sensori

Sensori muuttaa fyysisen maailman ilmiön digitaaliseen muotoon. Sensoreita käytetään tietojen keräämiseen. Esimerkkejä sensorista ovat kiihtyvyys- ja kosteusanturi.

SQL-tietokanta

SQL-tietokanta on perinteinen, rakenteellinen tietokanta, joka noudattaa sille asetettua skeemaa, ja jota tietokantaan tallennettavan tiedon on noudatettava. SQL-tietokannat ovat todella suosittuja ja yleisesti käytössä olevia.

SISÄLLYS

1 JOHDANTO

Esineiden internet (Internet of Things, myöhemmin IoT) leviää kovaa vauhtia yritysten käyttöön toimialasta riippumatta. IoT:lla tarkoitetaan järjestelmiä, jotka keräävät sensoriensa avulla tietoa, ja jotka ovat yhteydessä Internettiin. IoT-järjestelmän kautta on mahdollista etäohjata ja hallita laitteita. Laitteet voivat myös kommunikoida keskenään. Laitteita voidaan kytkeä niin kuluttajatuotteisiin kuin teollisuuslaitoksiinkin. Asia sinänsä ei ole aivan uusi. Sensoreita on hyödynnetty jo kauan aikaa tiedon keruussa. Uutta on se, miten tietoa kerätään ja käsitellään. Usein puhutaan lähes samasta asiasta eri termein. Esineiden internet ja teollinen internet ovat molemmat usein esiintyviä termejä. Tässä työssä niitä ei ole eroteltu toisistaan vaan puhutaan pelkästään esineiden internetistä, vaikka niissä pieniä eroja onkin.

Sensorimäärän kasvaessa toimintoja olisi kyettävä automatisoimaan mahdollisimman tuottavasti, ja sensorien tuottamat tietovirrat tulisi kerätä talteen mahdollisimman systemaattisesti. Kasvavien tietomäärien hallitsemisessa työkalujen valinta, ja tavoitteiden määrittäminen ovat kriittisessä roolissa. Omien konesalipalveluiden sijaan yhä useampi yritys perustaa toimintansa erilaisten pilvipalveluiden varaan. Ne tarjoavat pelkän tiedonvarastoinnin lisäksi lukuisia työkaluja tiedon käsittelyyn, laitehallintaan ja tiedon analysointiin. Useilla pilvipalveluntarjoajalla onkin esineiden internetiä varten räätälöidyt työkalut.

Tässä työssä tutkitaan IoT-alustojen ominaisuuksia ja rakennetaan Microsoft Azuren IoT-arkkitehtuuria hyödyntävä toteutus, jonka tarkoituksena on paikkatiedon kerääminen ja käsittely Azuren IoT-komponenttien avulla ja tietojen esittäminen sovelluksen avulla. Työ vastaa siihen, miten Azuren avulla on mahdollista toteuttaa IoT-ratkaisu, miten Azure täyttää alustoille asetetut vaatimukset, sekä millaisia vaatimuksia hyvälle IoT-alustalle voidaan asettaa.

2 PILVIPALVELUT

Tässä luvussa käsitellään pilvipalveluja yleisellä tasolla. Aluksi määritellään ja luokitellaan pilvipalvelut, ja miksi niitä käytetään, ja mitä heikkouksia ja turvallisuusriskejä niissä on. Työssä tutustutaan muihin aihealueeseen liittyviin teknologioihin, kuten big dataan ja Docker container-teknologiaan.

Pilvipalvelut voidaan yleisesti ottaen määrittää verkon yli jaettaviksi ja käytettäviksi, joko fyysisiksi tai virtuaalisiksi resursseiksi. Pilvipalveluiden määräävinä piirteinä voidaan pitää elastisuutta ja skaalautuvuutta, sekä mahdollisuutta jakaa resursseja tarpeen mukaan. Palvelun pitää olla mitattavissa ja sitä tulee voida käyttää verkon yli itsepalveluna. Lisäksi pilvipalvelun tulee olla monen käyttäjän käytettävissä yhtäaikaisesti. (Murugesan & Bojanova 2016, s. 4-5)

Pilvipalvelun resurssien tulee olla asiakkaan käytettävissä verkon yli oikeastaan mistä tahansa. Palvelun tulee skaalautua, kun palvelun käyttäjällä on tarve hankkia lisäresursseja. Resurssien jakamisella tarkoitetaan sitä, että useampi käyttäjä voi käyttää samaa resurssia, esim. palvelinta kulloisenkin tarpeen mukaan. Tällöin palvelun on skaalauduttava usealle asiakkaalle. Vastaavasti resurssin tulee skaalautua siten, että asiakas voi tarvittaessa skaalata omaa käytössä olevaa palveluaan lisääntyneestä tarpeesta johtuen. Pilvipalvelun käyttö laskutetaan usein käytön mukaan, joten palvelun käytön on oltava mitattavissa. (Murugesan & Bojanova 2016, s. 5-6)

Pilvipalveluiden etuina voidaan pitää matalampia hankinta ja ylläpitokustannuksia verrattuna perinteisiin, paikallisiin, yrityksien tiloissa sijaitseviin konesaleihin. Yrityksen on myös helpompi suunnata resursseja tarkemmin tiettyjen palveluiden käyttöön, ja kasvattaa hetkellisesti laskentatehoa. Pilvipalveluiden käytön hintaan kuuluu palveluntarjoajan toimesta resurssien ylläpito, eikä käyttäjän tarvitse huolehtia myöskään palveluiden tietoturvasta. Pilvipalvelun tietoturva on usein parempi kuin mitä yksittäiset yritykset pystyisivät tarjoamaan omissa konesaleissaan. Pilvipalvelut helpottavat myös tiedostojen ja sovellusten jakamista työtä tekevien tiimin jäsenten kesken. (Murugesan & Bojanova 2016, s. 10)

2.1 Pilvipalveluiden luokittelu

Pilvipalvelut voidaan jakaa muutamiin eri luokkiin sen mukaan, miten ne on teknisesti toteutettu. Yleisimpiä luokkia ovat Platform as a Service (PaaS), Infrastructure as a Service (IaaS), Software as a Service (SaaS), Public cloud, Private cloud ja Hybrid cloud.

2.1.1 Infrastructure as a Service (IaaS)

IaaS-mallissa asiakas saa käyttöönsä virtuaalisen konesalin, josta hänelle lohkotaan palveluja. Niihin asiakas perustaa omat virtuaalikoneensa, ja asentaa käyttöjärjestelmät ja tarvittavat resurssit. IaaS-malli tarjoaa käyttäjälle kaikista eniten liikkuma- ja säätövaraa. Samalla se myös vaatii asiakkaalta eniten omaa osaamista. (Salo 2014, s.97)

2.1.2 Platform as a Service (PaaS)

PaaS-tyyppisessä pilvipalvelussa palveluntarjoaja antaa asiakkaalle käyttöön virtuaalisen palvelinympäristön, josta asiakkaalle lohkotaan palveluja käyttöön. Asiakas käyttää palvelua jonkinlaisen käyttöliittymän läpi, ja asiakkaan sovellukset hyödyntävät API-rajapintoja kommunikointiin pilven kanssa. (Heino 2010, s.51)

2.1.3 Software as a Service (SaaS)

SaaS-mallissa käyttäjä ostaa pelkän ohjelmiston käyttöönsä, jota käytetään yleensä selaimen välityksellä. Palveluntarjoaja vastaa käytännössä kaikesta muusta, kuten esim. tietoturvasta, päivittämisestä ja ylläpidosta. SaaS-palvelu on hyvin yleinen nykyään. (Salo 2014, s.98)

2.1.4 Public cloud

Public cloud-termi tarkoittaa julkisessa käytössä olevaa, internet-yhteyden päässä olevaa palvelua, josta eri asiakkaat voivat ostaa resursseja käyttöönsä. Asiakas saa käyttöönsä muiden asiakkaiden kanssa monikäyttäjäympäristön, jossa usea eri asiakas käyttää samoja resursseja. Palvelu laskutetaan käytön mukaan aikaperusteisesti, tai jonkin muun käytetyn resurssin, kuten tallennuskapasiteetin tai tietoliikenteen määrän mukaan. (Heino 2010 s. 54-55)

2.1.5 Private cloud

Private cloud on pilvipalvelutekniikka, jossa pilvipalvelua käytetään LAN-lähiverkon yli yrityksen tai julkisyhteisön toimesta. Sama pilvipalvelu ei ole muiden saatavilla. Palvelun käyttäjä tai käyttäjät maksavat itse kaikki kustannukset ja investoinnit, joita palvelu vaatii ja vastaavat itse kaikista ylläpitoon liittyvistä asioista. (Heino 2010 s. 55)

2.1.6 Hybrid cloud

Hybrid cloud on yhdistelmä yksityistä ja julkista pilvipalvelua. Siinä yksityinen pilvipalvelu on yhdistetty julkisen pilvipalvelun kanssa tietoverkon vä-

lityksellä. Tällaisen ratkaisun avulla julkisesta pilvipalvelusta voidaan tarpeen mukaan hankkia lisäresursseja johonkin haluttuun tehtävään. (Heino 2010, s. 56)

2.2 Pilvipalveluiden rajoitukset ja huolenaiheet

Pilvipalveluiden käytössä on jonkin verran haasteita, kuten tarve nopealle Internet-yhteydelle. Hidas yhteys vaikuttaa vasteaikaan eli latenssiin, varsinkin kun pilvipalvelusta siirretään, tai haetaan suuria tietomääriä. Pilvipalveluiden yhteydessä voidaan myös pohtia riskiä tietojen joutumisesta vääriin käsiin, koska tiedot ovat tällöin jonkun toisen tahon palvelimella, jota ei voi täysin kontrolloida. Yrityksille saattaa tuottaa huolta ajatus siitä, että yrityksen tiedot ovat oman palomuurin ulkopuolella. Myös tietojen yksityisyys huolestuttaa. Pilvipalveluiden käyttöön liittyy paljon epäilyjä heikommasta tietoturvasta. Ajatellaan, että pilvipalvelun tarjoajilla ei ole halua suojata tietoja yhtä hyvin kuin palvelua käyttävä yritys sen itse tekisi. Usein näille epäilyille ei kuitenkaan ole syytä, vaan pilvipalvelun tarjoajat pystyvät antamaan paljon kehittyneempää tietoturvaa, kuin mitä asiakasyritys itse pystyisi. Tämä pätee erityisesti pienyritysten kohdalla. (Murugesan & Bojanova 2016, s. 10-11)

Taulukossa 1 on lueteltu Cloud Security Alliancen mukaan suurimmat pilvipalveluihin liittyvät uhkatekijät.

Taulukko 1. Pilvipalveluiden suurimmat uhkatekijät vuonna 2017.

Cloud Security Alliancen mukaan vuonna 2017 suurimmat uhkatekijät pilvipalveluihin liittyen ovat:	
1	Tietomurrot
2	Heikko käyttöoikeuksien hallinta
3	Rajapintojen tietoturvariskit
4	Sovellusten haavoittuvuus
5	Tilien kaappaukset
6	Sisäiset uhkatekijät
7	Kohdistetut hyökkäykset
8	Tietojen katoaminen
9	Riittämätön due diligence- prosessi
10	Vääränlainen ja vahingollinen pilvipalvelun käyttö
11	Palvelunestohyökkäykset
12	Jaetun teknologian haavoittuvuudet

2.3 Pilvipalveluiden tietoturva ja keskeiset riskitekijät

Pilvipalveluita käytetään nykyään yleisesti, ja suurimpia palveluntarjoajia pidetään luotettavina ja osaavina tahoina. Ne ovat ottaneet tietoturvaan

liittyvät asiat hyvin huomioon luodessaan palveluja. Siitä huolimatta organisaatioissa herää usein huoli siitä, etteivät tiedot ole enää omassa konesalissa, vaan jossain toisessa maantieteellisessä paikassa, ja vielä samalla virtuaalisella palvelimella kuin jonkin toisen yrityksenkin tiedot. Pilvipalveluissa käytetään useita eri suojaustekniikoita, kuten tietoliikenne- ja palvelintekniikassa yleisestikin. (Heino 2010, s. 92-93)

Pilvipalvelu suojataan palomuureilla, jotka kontrolloivat sisään ja ulos tulevaa liikennettä niiden sääntöjen mukaan, jotka niille on asetettu. Palomuuri on pilvipalvelun tarjoajan järjestämä ja hallinnoima kokonaisuus. Pilvipalveluissa käytetään yleisesti tunkeilijoiden havaitsemisjärjestelmää, josta käytetään termiä Intrusion Detection System (IDS) tai Intrusion Detection and Prevention System (IDPS). Tällaiset järjestelmät osaavat reagoida hyökkäystilanteisiin katkaisemalla ne yhteydet, joista hyökkäys havaitaan. (Heino 2010, s. 93)

Palveluun siirrettävät tiedot salataan kryptaamalla ne siten, että mahdollisessa tietomurtotilanteessa niitä ei pitäisi pystyä lukemaan. Menettely vaatii siihen sopivan ohjelmiston ja menetelmän, jolla avaimet pidetään salassa. Pilvipalveluiden palvelimiin käytetään niin sanottua koventamista, jossa niistä karsitaan niin paljon ylimääräistä pois kuin mahdollista. Tämä pienentää hyökkäysvektorin kokoa, eli vähentää niitä paikkoja, joita pitkin mahdollinen tunkeutuja voisi päästä käsiksi järjestelmään. Tarpeettomat palvelut käyttävät usein UDP/TCP-portteja, joita pitkin mahdollinen hyökkääjä saattaisi päästä tietoihin käsiksi. Koventaminen kuuluu pilvipalvelun tarjoajan tehtäviin ja se tehdään joko käsin tai hyödyntämällä tarkoitukseen sopivia työkaluja. Pilvipalveluiden käyttämät tekniikat ovat yleisiä muissakin verkkopalveluissa ja ne ovat olleet käytössä kauan. Pankkipalvelut ovat hyvä esimerkki siitä, miten kriittinen palvelu voidaan tuottaa turvallisesti, ja niin, että ihmiset voivat luottaa siihen. (Heino 2010, s. 93-94)

Immo Salon (2015) mukaan keskeisimmät pilvipalvelun riskitekijät ovat dataan, käyttäjänhallintaan, suorituskykyyn, hallintaan, sopimusehtoihin, tekniseen toteutukseen, palveluntarjoajiin ja säännöksiin liittyvät huolet. Tässä listauksessa olevien huolien katsotaan olevan niitä, jotka yleensä hidastavat tai estävät pilvipalveluiden käyttöönottoa. (Salo 2015, s.103-104)

Dataa koskevat huolet liittyvät yrityksen hallussa olevan tiedon arvokkuuteen, ja siihen kuinka paljon vahinkoa seuraisi siitä, että tiedot päätyvät vääriin käsiin. Yrityksillä on käsissään usein liiketoiminnan kannalta todella arvokasta tietoa, jonka ei haluta joutuvan varsinkaan kilpailijoiden käsiin. Siksi sen siirtäminen pilveen aiheuttaa usein pelkoa ja huolta. Osa tiedosta on lisäksi sellaista, ettei sitä saa säilyttää EU:n ulkopuolella. Esimerkkinä tästä ovat henkilötiedot. Toinen arkaluontoinen tietojoukko on tuotekehitykseen ja yrityksen avaintuotteisiin liittyvät tiedot. Ne voivat olla niin salaisia, ettei niitä ole syytä siirtää organisaation ulkopuolelle. (Salo 2015, s.105)

Käyttäjiin liittyvillä huolilla tarkoitetaan käyttäjien inhimillisiä heikkouksia. Huolimaton käyttäjä saattaa esimerkiksi käyttää heikkoa salasanaa, jolloin sen selville saaminen on osaavalle hyökkääjälle helppoa. Salasanoja voidaan myös urkkia sosiaalisen hakkeroinnin keinoin. Tällaisessa hyökkäyksessä järjestelmään yritetään päästä käsiksi sitä käyttävien ihmisten välityksellä. Tavoitteena on saada selville salasanoja, tai saada hyökkäyksen kohde asentamaan jokin haittaohjelma murron kohteena olevaan järjestelmään. (Salo 2015, s.108)

Palveluntarjoajaan liittyvillä huolilla tarkoitetaan palveluiden saavutettavuuteen liittyviä asioita. Pilvipalveluiden resurssit sijaitsevat etäällä käyttökohteesta, ja palvelut toimivat tietoverkon yli. Tietoverkon tai laitteiston ongelmat estävät palvelun käytön. Etäisyydet fyysiseen sijaintiin voivat olla tuhansia kilometrejä. (Salo 2015, s.108-109)

Hallintaan liittyvillä huolilla tarkoitetaan sitä, etteivät asiakkaat voi täysin hallita koko järjestelmää tai fyysisiä laitteita, vaan palveluntarjoaja antaa asiakkaan käyttöön tietyt välineet ja rajapinnat, joilla on tultava toimeen. Nämä huolet liittyvät myös tekniseen toteutukseen. Asiakkaille ei useinkaan paljasteta taustalla toimivaa arkkitehtuuria. Suorituskykyyn liittyvillä huolilla tarkoitetaan pelkoa siitä, ettei palvelun suorituskyky ole riittävä. (Salo 2015, s.110)

Palveluntarjoajan kanssa tehdyt sopimukset on tehtävä sitä tarkemmin, mitä kriittisemmästä tiedosta on kysymys. Erityisesti asiakkuuksiin, tuotekehitykseen, talouteen ja kirjanpitoon liittyvät tiedot, ja niiden saatavuus ja pysyvyys ovat yrityksille elintärkeitä. Sopimukset eivät saa olla löyhiä, koska ne johtavat silloin useimmiten ongelmiin. Sopimussakot ovat toistaiseksi olleet varsin pieniä, jos palvelua ei saavuteta. Kriittisellä hetkellä toimimaton järjestelmä saattaa kuitenkin aiheuttaa palvelua käyttävälle yritykselle merkittäviä tappioita. (Salo 2015, s.106-110)

Lainsäädäntö ja palvelut voivat kehittyä eri tahtiin, jolloin esimerkiksi lakimuutokset voivat pakottaa yrityksiä muuttamaan tapaa, jolla ne palveluita käyttävät. Hyvä esimerkki tästä on EU:n tietosuoja-asetus, jolla on vaikutusta myös pilvipalveluihin tallennettavan tiedon kannalta. (Salo 2015, s.111)

2.4 **Pilvipalvelut Big Datan näkökulmasta**

Big Data on termi, josta puhutaan nykyisin kaikkialla. Suoraan suomennettuna se tarkoittaa suurta tietomäärää. Big Datalla tarkoitetaan lyhyesti sanottuna suuria tietomääriä, jotka voivat olla rakenteellisia, rakenteettomia tai puoliksi rakenteellisia. Tietolähteitä on useita. Niitä voivat olla esimerkiksi sensoreista tulevat tietovirrat, avoimen tiedon lähteet, sovellusten keräämä tieto ja Internetin eri tietolähteet. Tiedon määrä on nykyisin niin valtava, ettei kaiken tiedon varastointi ole edes kannattavaa tai mahdollista. Pitää osata poimia oleellisin tieto. Tiedon määrä tulee lähivuosina

edelleen kasvamaan merkittävästi. Tämä tarjoaa organisaatioille mahdollisuuksia kehittää liiketoimintaa. Se myös mahdollistaa tarkemman ennustettavuuden ja tiedolla johtamisen. (Salo 2015, s. 163)

Pilvipalveluihin on kehitetty työkaluja, jotka sopivat hyvin Big Datan vaatimuksiin. Suosittu big data-työkalu Hadoop on saatavilla pilvipalveluna. Esimerkiksi Azure-pilvipalvelussa Hadoop-työkalua voi käyttää osana HDInsight-palvelua (Microsoft 2019). Hadoop on tarjolla monissa pilvipalveluissa nimellä Elastic MapReduce. (Salo 2015, s. 163)

3 ESINEIDEN INTERNET

Tässä luvussa käydään esineiden internetin teoriaa. Luku vastaa kysymyksiin mitä esineiden internet on ja miksi se on olemassa. Luvussa käsitellään referenssiarkkitehtuuria ja sitä, mitä tekemistä esineiden internetillä ja pilvipalveluilla on keskenään.

Esineiden internetillä tarkoitetaan sitä kokonaisuutta, jonka avulla erilaisia sensoreita, laitteita ja koneita hallitaan, liitetään toisiinsa ja Internettiin tiedon keräämiseksi niiden avulla. Esineet voivat kommunikoida keskenään ja/tai olla vuorovaikutuksessa fyysisen maailman kanssa. Esineiden internet on yläkäsite kaikelle tälle. Se pitää sisällään lukuisia erilaisia teknologioita. Esineiden Internetin tunnusomaisia piirteitä ovat myös tiedon keräys prosessointia ja analysointia varten. Esineiden internetiä voidaan hyödyntää lukuisissa erilaisissa sovelluksissa kuten esimerkiksi rakennusautomaatiossa, älykkäissä kaupungeissa, teollisuudessa ja maataloudessa. Teollinen internet (Industrial Internet of Things, IIoT) on esineiden internetin käyttämistä teollisuudessa. Sovellusalueita ovat valmistava teollisuus, logistiikka, öljy ja kaasu ja muu energiasektori, kaivosala ja ilmailuala. (Girani, Ferrari, Picone & Veltri 2019)

Älykkäillä koneilla tarkoitetaan laitteita, jotka pystyvät reagoimaan keräämäänsä tietoon, ja toimimaan tietojen pohjalta algoritmien avulla. Laitteet voivat käyttää apunaan myös muita lähteitä kuin vain omia sensoreitaan, ja ne voidaan liittää osaksi suurempaa sensoriverkostoa. Laitteiden analytiikka sijaitsee usein pilvipalvelussa, jossa ovat myös suuret tietomäärät ja älykkäät algoritmit. (Salo s. 18-20, 2014)

Tässä opinnäytetyössä ei erotella teollista internetiä ja esineiden internetiä toisistaan, vaan molemmista puhutaan esineiden internettinä selvyyden vuoksi. Työn kannalta sillä ei ole suurtakaan merkitystä kumpaa termiä käytetään, koska työn tavoitteiden näkökulmasta tarkasteltuna niillä ei ole merkittävää eroa vaikkakin joitakin eroavaisuuksia on. Samaa alustaa voidaan käyttää molemmissa sovelluskohteissa. Teollinen internet asettaa usein hieman erilaisia vaatimuksia johtuen fyysisestä ympäristöstä, protokollista ja laitteistosta.

3.1 Esineiden internetin arkkitehtuurimalli

Cisco määritteli vuonna 2014 esineiden internetin referenssimallin seitsemään kerrokseen, jotka on esitetty taulukossa 2.

Taulukko 2. Referenssimallin seitsemän kerrosta (Cisco 2014).

Taso 7	Prosessikerros (Collaboration & Processes)
Taso 6	Sovelluskerros (Application)
Taso 5	Erottelukerros (Data Abstraction)

Taso 4	Tiedon varastointikerros (Data Accumulation)
Taso 3	Reunalaskentakerros (Edge (Fog) Computing)
Taso 2	Yhteyskerros (Connectivity)
Taso 1	Laitekerros (Physical device & Controllers)

Laitekerros käsittää fyysiset laitteet ja ohjaimet. Niihin kuuluvat koko ajan laajeneva joukko erilaisia laitteita, jotka on suunniteltu tiettyä tarkoitusta varten. Niiden kokoluokalle, sijainnille tai muodolle ei ole asetettu mitään rajoitteita. Laitteiden ominaispiirteitä ovat analogisen tiedon muuttaminen digitaaliseen muotoon tarvittaessa, tiedon keruu ja niiden hallinnoinnin mahdollisuus internetin yli. (Cisco 2014)

Yhteyskerroksen tehtävänä on luotettava ajanmukainen tiedonsiirto laitekerroksen laitteiden kesken, verkossa itä-länsi-suunnassa ja yhteyskerroksen verkkojen ja reunalaskentakerroksen välillä. Yhteyskerroksessa voidaan myös tehdä verkkomuunnoksia tarpeen mukaan. Yhteyskerroksen päätehtävät ovat kommunikointi laitekerroksen kanssa, luotettava tiedonsiirto, tiedonsiirtoprotokollien toteutus, reititys, protokollamuunnokset, verkkotason tietoturva ja tietoverkkoanalytiikka. (Cisco 2014)

Reunalaskentakerros sisältää tiedon suodatus-, puhdistus- ja koontitoimintoja, pakettien sisällön tutkimuksia, tietoverkon ja tietotason analytiikkaa, kynnysarvojen tarkkailua ja tapahtumien luontia. Tason toiminnallisuus keskittyy pohjoiseteläsuuntaiseen kommunikaatioon tasojen välillä. Reunalaskentakerros sisältää reunalaskennalle suunnattuja analytiikan tehtäviä, joiden avulla kerros voi ohjata jonkin laitteen toimintaa. Pakettien prosessoinnin yhteydessä paketeille voidaan suorittaa toimenpiteitä, kuten arvioidaan kriteerien perusteella, kuuluuko tieto käsitellä tasolla 4. Lisäksi tietoa voidaan muotoilla uudelleen tai leikata, kuten vähentää tai koota ja purkaa suojattua tietoa. Tiedon sisältöä arvioidaan sen perusteella, sisältääkö tieto raja-arvoja tai hälytyksiä, joiden perusteella tietoa voidaan kohdistaa eri määränpäihin. (Cisco 2014)

Tiedon varastointikerros jakaa tiedon eri varastoihin määritelmien mukaisesti. Varastointikerros huolehtii tiedon muunnoksista. Se muuntaa ”liikkeessä olevan” tiedon ”pysyväksi” tiedoksi, ja tietoverkkopakettien sisällön tietokantatyyppiseksi tiedoksi. Varastointikerros suorittaa lisäksi muunnoksen tapahtumaperusteisesta kyselyperusteiseksi ja vähentää tiedon määrää tarvittaessa suodattamalla ja valikoivan tallennuksen avulla. Kerros mahdollistaa tiedon käyttämisen erilaisille tietoa hyödyntäville sovelluksille, jotka eivät tarvitse reaaliaikaista tietoa. (Cisco 2014)

Tiedon erottelukerros sisältää funktioita, joiden avulla tietoa voidaan renderöidä ja tallentaa tavoilla, jotka mahdollistavat yksinkertaisempien ja suorituskykyisempien sovellusten kehittämisen. Eri tiedonlähteistä tulevaa tietoa ei ole perusteltua sijoittaa samaan tietovarastoon. Liian suuri tiedon määrä samassa paikassa johtaa siihen, että tiedon liikuttaminen tietokan-

taan saattaa viedä liikaa prosessointitehoa. Sen myötä prosessi on eroteltava tiedon luomisprosessista. Laitteiden maantieteellinen sijainti saattaa erota siitä, mille prosessointi on optimoitu. Tasot 3 ja 4 saattavat erotella jatkuvan tietovirran tapahtumakeskeisestä tiedosta. Näihin tyyppeihin on parempi soveltaa erilaista tiedon tallennusvaihtoehtoa. Prosessointitarpeet saattavat olla erilaisia eri tietokokoelmille. (Cisco 2014)

Sovelluskerros käsittää ne ohjelmat ja toiminnot, jotka hyödyntävät tason 5 pysyvää tietoa. Sovelluksia ei määritellä tarkasti vaan ne voivat olla hyvinkin monenlaisia kokonaisuuksia tiedon esittämisestä laitteita ohjaaviin sovelluksiin. Sovellukset voivat yhdistellä tietoa eri lähteistä. Osa tiedoista voi tulla laitteiden ulkopuolisista lähteistä. (Cisco 2014)

Prosessikerros kuvaa järjestelmää käyttävien ihmisten välistä yhteistyötä ja sovellusten hyödyntämistä liiketoiminnassa. Järjestelmän tarkoitus ei ole palvella järjestelmää itseään, vaan auttaa liiketoimintaa päätöksenteossa ja tiedon hyödyntämisessä.

Tämä Ciscon vuonna 2014 tekemä hahmotelma referenssimallista on yksi mahdollinen tapa rakentaa IoT-arkkitehtuuri. Alalla ei ole standardia, joka määrittelisi miten järjestelmä tulisi rakentaa. Siihen miten järjestelmä kannattaa toteuttaa, vaikuttaa myös mahdollisten palveluntarjoajien tarjonta. (Cisco 2014)

IoT-arkkitehtuuri voidaan jakaa referenssimallin lisäksi kahteen osaan: keskitettyyn tai hajautettuun malliin. Keskitetyllä arkkitehtuurimallilla tarkoitetaan järjestelmää, jossa on yksi taustajärjestelmä. Se huolehtii laitteiden ohjauksesta, tiedontallentamisesta, analytiikasta, koneoppimisesta, jne. Järjestelmä tarjoaa päätepisteet, jotka toimivat datan tuottajan ja kuluttajan roolissa. Hyvä esimerkki tästä on pilvipalvelu, joka sisältää edellä mainitut toiminnot. Keskitetyllä arkkitehtuurilla on kuitenkin rajoitteensa. Kun halutaan siirtyä kohti kehittyneempää ratkaisua, keskitetty arkkitehtuuri ei enää riitä, vaan se pitää laajentaa hajautetuksi arkkitehtuuriksi. Hajautettu arkkitehtuuri mahdollistaa mm. sensorien keskinäisen kommunikaation, hajautetun auditoinnin ja datan jakamisen ilman, että sitä ohjaa keskitetty taustajärjestelmä. Hajautetun arkkitehtuurin taustalla toimii edelleen keskitetty ratkaisu, mutta sen rooli on muuttunut. Hajautetun arkkitehtuurin eduiksi voidaan lukea tietoliikenteen ja datan prosessoinnin viiveen pieneneminen ja resurssien tehokkaampi käyttö. (Collin & Saarelainen 2016)

3.2 **Alusta - Pilvipalvelut esineiden internetin näkökulmasta**

Pilvipalvelut tarjoavat esineiden internettiin uudenlaisen näkökulman, joka mahdollistaa sensoreiden keräämän tiedon nopean analysoinnin pilvipalvelussa algoritmien toimesta. Pilvipalveluiden lähes loputon laskentateho, skaalautuvuus ja elastisuus mahdollistavat kehittyneen analytiikan, jonka avulla sensorien keräämien tietojen analysointi saadaan lähes reaaliaikaiseksi. Vasteeseen vaikuttaa lähinnä tietoverkkoyhteyden nopeus.

Pilvipalvelut mahdollistavat sensoreilla varustettujen koneiden oppimisen ympäristöstään kerätyn tiedon ansiosta. (Salo 2014, s.18-19)

Esineiden internetin alusta koostuu eri tasoista, joita ovat sensorit, tietoliikenne, tietovarasto, analytiikka, sovellus ja digitaaliset palvelut. Pilvipalvelut pystyvät tarjoamaan nämä teknologiat tai työkalut niiden yhteen liittämiseksi. Tällaista palvelua kutsutaan alustaksi. Varsinaiset sensorit eivät kuitenkaan kuulu alustaan, mutta alustan toimittaja tarjoaa usein ohjelmistoja sensorien hallintaan. Jokainen alustan taso on oleellinen tiedon hyödyntämisen ja jatkojalostuksen kannalta. (Collin & Saarelainen 2016)

Alustan tehtävät voivat olla moninaiset ja niiden välillä voi olla suuriakin eroja. Alusta toimii kokoavana tekijänä järjestelmän osien välillä ja muodostaa IoT-arkkitehtuurin selkärangan. Se huolehtii infrastruktuurista ja mahdollistaa osien keskinäisen liitettävyyden. Ilman alustaa moni asia on tehtävä itse, kuten sensorien ja palvelimien välisen kommunikaation järjestäminen, tietoturvasta huolehtiminen ja datavirtojen hallinta. Alusta huolehtii monista näistä asioista automaattisesti. Taulukossa 3 on esitelty alustojen ominaisuuksia. (Collin & Saarelainen 2016)

Taulukko 3. IoT-alustan ominaisuudet (Collin & Saarelainen 2016)

IoT-alustan ominaisuuksia	
1	alustan ja sensoriverkon välinen liitettävyys
2	alustaan liitettyjen laitteiden hallinta ja sääntöjen muodostaminen
3	datan kokoaminen ja prosessointi
4	tallennuspaikan järjestäminen
5	datan analysointi, raportointi ja visualisointi
6	sovelluskehityksen työkalut
7	rajapinnat sovelluskehitykselle ja ulkoisiin tietojärjestelmiin
8	tietoturvasta, pääsynhallinnasta ja tunnistamisesta huolehtiminen

Alustojen laajuudessa ja kypsyydessä voi olla paljonkin keskinäisiä eroja. Pienin mahdollinen alustaratkaisu saattaa kattaa pelkästään sensorien liitettävyyden ja hallinnan ja se pystyy käsittelemään vain pienen sensorijoukon tietovirtaa. Laajemmassa alustassa on usein mukana kehitysalusta, tietoturvan ja integroinnin perusominaisuudet ja tuki muutamille protokollille ja ohjelmointikielille. Sillä on myös kyky käsitellä rajatun sensorimäärän datavirtaa. Kehittynyt alusta tarjoaa sen sijaan kyvyn hallita laajaa sensorijoukkoa ja niiden tuottamaa datavirtaa ja mahdollistaa erilaiset integraatiot muihin järjestelmiin ja rajapintoihin, sekä kykenee tarjoamaan erilaisia tietoverkkoratkaisuja ja analytiikan työkaluja ja julkaisuvaihtoehtoja. Loppuun asti viety alusta tarjoaa lisäksi työkaluja ja tallennuspaikan Big Datalle ja se sisältää monipuolisen kehitysalustan. (Collin & Saarelainen 2016)

Kehittyneemmälle IoT-alustalle voidaan asettaa kahdeksan vaatimusta, jotka ovat liitettävyys ja normalisointi, laitehallinta, prosessoinnin ja toimintojen hallinta, datan visualisointi, analytiikka, lisätyökalut, ulkoiset rajapinnat ja tietokantapalvelut. (IoT Analytics 2015)

Liitettävyys ja normalisointi on oleellinen osa kehittyneitä IoT-alustoja. Niiden avulla eri protokollien ja tietoformaattien hallinta yhtenäistyy, koska kerros nivoo eri protokollat ja tietoformaatit yhdeksi rajapinnaksi. Sen avulla mahdollistetaan luetun datan oikeellisuus ja laitteiden vuorovaikutus. Tällöin tieto on yhdessä paikassa ja yhdessä muodossa. Se mahdollistaa laitehallintakerroksen toiminnan. Kehittyneemmät laitteet tarjoavat usein rajapinnan, jonka kautta yhteys luodaan alustaan. Usein niin kutsuttu agenttiohjelma asennetaan laitteelle, jotta vakaa ja turvallinen yhteys on mahdollista luoda laitteen ja alustan välille. (IoT Analytics 2015)

Laitehallinnan rooli IoT-alustassa on huolehtia siitä, että laitteet ovat käynnissä ja ne toimivat halutulla tavalla. Lisäksi tehtäviin kuuluu ohjelmien päivitys ja niiden käynnissä pysymisen varmistaminen, laitteiden etäkonfigurointi ja vianetsintä. Laitehallinnan roolin merkitys kasvaa laitteiden määrän kasvun myötä, jolloin massatoimintojen suorittaminen ja automaatio ovat oleellisia manuaalisten toimintojen vähentämiseksi ja kustannusten hallitsemiseksi. (IoT Analytics 2015)

Esineiden internet asettaa tietovarastot uudenlaisten haasteiden eteen. Sensorien tuottama data voi olla keskenään hyvin erilaista, eri vauhtista, eri määräistä ja se voi sisältää virheellisyyksiä. Lisäksi tietovaraston pitäisi skaalautua suurille tietomäärille ja mahdollisesti maantieteellisesti, ja sen pitäisi rakenteellisen tiedon lisäksi vastaanottaa rakenteetonta tietoa. (IoT Analytics 2015)

Prosessointi ja toimintojen hallinta mahdollistaa tapahtumaperusteisten toimintojen suorittaminen. Tieto voi olla peräisin sensorien tuottamasta datasta, tai se voi olla peräisin muista lähteistä, kuten tietokannoista. Esimerkkinä tapahtumaan perustuvasta toiminnosta voi olla etäisyyden kasvaminen riittävän suureksi jostain pisteestä, ja sen seurauksena tapahtuvasta toiminnosta, kuten vaikka sähköjen pois kytkeytyminen. Tällöin kysymyksessä on tyypillinen, ”jos tämä tapahtuu, niin sitten tapahtuu näin” -toiminnallisuus (IFTTT). (IoT Analytics 2015)

Analytiikan tehtävä on havaita ja oppia toistuvia kaavoja tuotetun datan perusteella. Tästä johtuen analytiikka liittyy läheisesti koneoppimiseen ja algoritmeihin. Alusta sisältää yleensä oman analytiikkamoottorin, joka osaa etsiä datasta syy-seuraussuhteita. Algoritmit on usein tuotettu valmiiksi alustan puolesta eikä käyttäjän ole välttämätöntä tuottaa niitä itse. Esimerkkinä analytiikasta voi olla älykoti, jossa alustan analytiikka voi oppia säätämään kodin lämpötilan ja valaistuksen käyttäjälle sopivaksi ottamalla huomioon käyttäjän aikaisemmat itse tekemät säädöt, ja liittämällä ne yhteen ulkoilman lämpötilan kanssa. (IoT Analytics 2015)

Tiedon visualisointi ja graafinen esittäminen ovat edelleen tärkeässä roolissa, kun dataa halutaan esittää ihmisille. Graafinen esitystapa auttaa ihmisiä havaitsemaan kaavoja ja erilaisia kehityksen suuntaviivoja. Visualisointia voidaan tehdä erilaisilla kaavioilla, sekä 2- ja 3- ulotteisilla malleilla. Kehittyneet alustat sisältävät yleensä visualisointityökalut ainakin jollain tasolla. (IoT Analytics 2015)

Kehittyneet alustat tarjoavat usein myös työkaluja kehittäjille, joiden avulla voidaan testata ja luoda prototyyppejä erilaisista ratkaisuista. Lisäksi tarjolla on myös työkaluja alustan hallintaan ja tietoturvaan liittyen sekä mahdollisia raportointityökaluja. (IoT Analytics 2015)

Alustat toimivat harvoin eristettyinä yrityksissä. Sen sijaan on usein tarpeellista yhdistää niiden keräämä tieto yhteen muiden yritysten keräämien tietojen kanssa, kuten ERP-järjestelmän, johtamisjärjestelmien, valmistusjärjestelmien ja muun IT-ekosysteemin kanssa. Integraatioiden kannalta ulkoiset rajapinnat ja yhdyskäytävät ovat avainasemassa. Hyvät rajapinnat säästävät paljon aikaa ja rahaa. (IoT Analytics 2015)

Tässä luvussa käsitellyt kahdeksan elementtiä ovat avainasemassa, kun arvioidaan, onko yrityksen markkinoima IoT-alusta oikeastaan IoT-alusta vai jotain ihan muuta. Välillä termejä venytetään ja käytetään väärin. Sen vuoksi kriittinen arviointi ja ominaisuuksien selvittäminen on tärkeää. Yritykset, jotka tarjoavat vain tallennustilaa pilvessä, asiakkuuden hallintajärjestelmän, tietoturvaa tai yhteyksien hallintajärjestelmän, markkinoivat ratkaisua IoT-alustana, vaikka se ei sitä todellisuudessa ole. (IoT Analytics 2015)

3.3 **Rajapinnat**

Rajapinnoilla (API) on keskeinen rooli pilvipalveluiden käytössä. Ne mahdollistavat pilvipalveluiden ja sovellusten välisen kommunikaation. Rajapintojen avulla sovellukset kutsuvat pilvestä resursseja tai toimintoja. Client-server-arkkitehtuurin yleistymisen myötä yhteisten rajapintojen rooli on kasvattanut merkitystään. Niiden ansiosta eri valmistajien teknologiat voivat keskustella keskenään. Se taas mahdollistaa sen, että kaikkea ei ole pakko ostaa samalta valmistajalta. Tämä vähentää palveluiden hankkimiseen kohdistuvaa riskiä. (Heino 2010, s. 76)

Monet pilvipalvelut käyttävät hyödykseen REST-rajapintaa. REST muodostuu sanoista ”Representational state transfer”. REST on malli, jota voidaan käyttää hajautettujen järjestelmien verkkopohjaisten sovellusten rakentamiseen. (Heino 2010, s.77)

3.4 **Yhteysprotokollat**

Protokollien valintaan kannattaa kiinnittää huomiota, jotta data liikkuu niin kuin sen on haluttu liikkuvan. Protokollien kirjo on laaja ja niiden ominaisuudet vaihtelevat keskenään. Toisissa on keskitetty suureen nopeuteen, kun taas toisissa on panostettu tietoturvaan ja varmuuteen. Osa protokollista on omiaan laitteen ja pilven välillä tapahtuvassa liikenteessä, kun taas osa on tehty laitteiden välillä tapahtuvaan viestintään. Osa yrityksen järjestelmistä on vanhoja ja tukee vain tiettyjä protokollia. Se on otettava huomioon, kun liitetään IoT-alustaa osaksi vanhoja järjestelmiä. Myös protokollien kehitysaste on huomioitava. Vanhentuneiden protokollien tilalle saattaa tulla korvaajia. Tämä voi aiheuttaa ongelmia, jos on valinnut vanhan protokollan käyttöön. (Collin & Saarelainen 2016)

CoAP on sovelluskerroksen protokolla, joka on varteenotettava esineiden internetin kannalta, koska se on suunniteltu käytettäväksi koneelta koneelle (M2M). Se sopii hyvin antureille, kytkimille, venttiileille ja muille vastaaville komponenteille, joita on tarpeen valvoa ja hallita etänä. CoAP muistuttaa HTTP-protokollaa, ja se käyttää samanlaista asiakas-palvelinmallia. Metodit ovat HTTP:stä tutut GET, PUT, POST ja DELETE. Ero protokollien välillä on mm. siinä, että CoAP:ssa viestien vaihto on asynkronista eikä se perustu aiemmin luotuun yhteyteen. Tästä syystä se soveltuu hyvin monikanavaiseen viestintään, joka ei ole reaaliaikaista. CoAP välittää viestejä UDP-kerroksessa. Siinä on myös sisäänrakennettu DTLS-pohjainen tietoturva. Kaiken kaikkiaan protokolla on joustava, ja sitä voidaan käyttää vertaisverkon tavoin laitteiden väliseen kommunikaatioon, tai myös laitteen ja pilvipalvelun välillä välityspalvelimen avulla. (Collin & Saarelainen 2016)

DDS eli Data Distribution Service on erittäin nopea koneiden väliseen viestintään tarkoitettu protokolla, joka kykenee välittämään useille laitteille samanaikaisesti jopa miljoonia viestejä sekunneissa. Protokolla tarjoaa monipuoliset mahdollisuudet suodattaa dataa ja määritellä datan kohteet. Sen vahvuuksia ovat yhteyksien laadunhallinta, toimintavarmuus ja monilähetys erittäin nopeasti, jopa mikrosekunneissa. (Collin & Saarelainen 2016)

AMQT on binäärinen viestintäprotokolla, joka on lyhenne sanoista Advanced Message Queuing Protocol. Se kehitettiin vuonna 2003 finanssialalla. Siitä tuli ISO/IEC-standardi vuonna 2014. Protokollan vahvuus on sen luotettavuudessa, yksinkertaisuudessa ja tietoturvassa. Se sopii hyvin kriittisiin kohteisiin, joissa on tärkeää, että jokainen viesti saapuu varmasti perille. Siksi se on yleinen pankkitoiminnoissa eri puolilla maailmaa. Myös Google ja Nasa käyttävät sitä. AMQT perustuu aihepohjaiseen julkaisija-tilaaja-malliin sekä viestijonoihin. Se kykenee pienten viestien osalta nopeatahtiseen viestintään. (Collin & Saarelainen 2016)

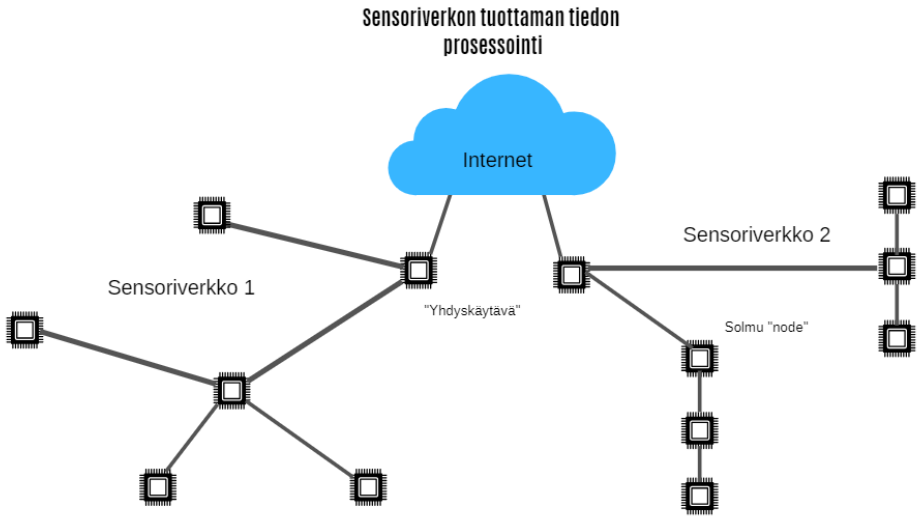
MQTT on yleinen protokolla esineiden internetin ja pilvipalveluiden välisessä tietoliikenteessä. MQTT tulee sanoista Message Queue Telemetry Transport, ja se on suunniteltu nimenomaan telemetriatietojen keräämiseen, etämittaukseen ja valvontaan. MQTT toimii erityisesti suuren sensorijoukon ja pilvipalvelun välillä tiedonsiirtoprotokollana. MQTT vaatii, että sekä viestin lähettäjä, että vastaanottaja ovat toiminnassa, jotta viesti voi mennä perille. Protokollan ei ole mahdollista käyttää viestijonoja, joten mikäli vastaanottaja ei ole vastaanottavassa tilassa, on vaarana viestin katoaminen matkalle. MQTT:n yleisenä heikkoutena voidaan pitää heikkoa suojausta. Protokollan perusversiossa ei ole suojausta ollenkaan, mutta liikenteen voi suojata joko VPN-tunnelilla tai TSL/SLL-salauksella, koska MQTT liikennöi TCP-protokollan avulla. TSL/SLL-salauksen käyttö kuluttaa enemmän energiaa, joten se voi koitua haasteeksi sensoreilla, joilla on vähäinen energian kulutus. Tällöin voidaan hyödyntää vain paketin viestisisältöön perustuvaa rajausta koko IP-paketin sijaan. (Collin & Saarelainen 2016)

OPC UA on teollisuuden automaatiosta tuttu protokolla ja arkkitehtuuri. Se mahdollistaa koneelta koneelle-liikenteen ja järjestelmien ristiin toimivuuden. Se sisältää sekä binaarisen, että web-protokollan (SOAP) ja myös mahdollistaa niiden yhtäaikaisen toiminnan. SOAP-protokolla toimii kuljetuskerroksena koodin ollessa binaaria. Tämä tekee siitä resurssitehokkaan vaihtoehdon sulautetuille järjestelmille. Sen reitittäminen on helppoa tietoliikenteen käyttäessä vapaasti valittavaa TCP-porttia, joka helpottaa liikenteen tunnelointia ja pääsyä palomuureista. Web-protokolla käyttää HTTP(S)-portteja. OPC UA on turvallinen sen sisältämän vahvan tietoturvan ansiosta. (Collin & Saarelainen 2016)

SNMP on protokolla, jota käytetään jonkin tietyn objektin valvontaan. Lyhenne SNMP tulee sanoista ”Simple Network Management Protocol”. Protokollan avulla voidaan mm. tarkkailla laitteiden tilaa ja toimintaa. SNMP-protokolla käyttää tiedonhakuun OID-tunnuksia, jotka ovat uniikkeja. OID-tunnusten avulla voidaan ohjelmallisesti hakea jonkin objektin tila. (SNMP Labs 2018)

3.5 **Wireless Sensor Network – WSN**

Wireless Sensor Network on sensorien keskenään muodostama langaton verkko, jonka avulla siihen kytketyt sensorit voivat kommunikoida keskenään. Sensoriverkko muodostuu solmuista (node), jotka liittyvät toisiin solmuihin, ja lopulta yhdyskäytävään, jonka kautta verkko on yhteydessä internettiin. Sensoriverkon jokainen solmu voi olla myös suoraan yhteydessä yhdyskäytävään. Sensoriverkon periaate on esitetty seuraavan sivun kuvassa 1. Solmujen, jotka välittävät muiden sensorien tietoja eteenpäin, tulee huolehtia tietojen erottelusta. Niillä on myös mahdollisuus muokata ja siistiä solmujen lähettämää tietoa, jolloin yhdyskäytävältä lähtevä tietomäärä on pienempi. (Dargie & Poellabauer 2010, s. 7-8)



Kuva 1. Sensoriverkko.

3.6 **Kapillaariverkot**

Ericsson on tutkinut vuodesta 2010 lähtien kapillaariverkkoja, jotka on suunniteltu sensorien väliseksi langattomaksi verkoksi. Teknologia perustuu mobiiliverkkoihin. Yhden kapillaariverkon sensorit ovat yhteydessä yhdyskäytävään, joka muodostaa mobiiliverkon yli yhteyden etätietovarastoon. Kapillaariverkon luvataan tuovan helpotusta sensorien päivityksiin ja tietoturvaan. Sensorit voivat olla myös suoraan yhteydessä mobiiliverkkoon, jolloin niillä on oltava SIM-kortti. Kapillaariverkkoon yhteydessä olevat sensorit kommunikoivat yhdyskäytävän kanssa lyhyen kantaman radiotekniikan, kuten Bluetoothin tai ZigBeen, avulla. Yhdyskäytävä luo jokaiselle sensorille mobiiliverkon kaltaisen palvelun, johon kuuluu esimerkiksi yksilöllinen tunniste, joka vastaa mobiililaitteen SIM-korttia. Kapillaariverkot parantavat sensorien tietoturvaa tuomalla mobiiliverkkojen tapaisen autentikoinnin ja varmenteen sensoreille. Kapillaariverkot hyödyntävät 3GPP-standardia, ja lisäksi liikenne voidaan salata TLS-protokollan avulla. (Collin & Saarelainen 2016)

Kapillaariverkon luvataan parantavan myös sensorien akun kestoa niin sanotun välimuistilaitteen avulla. Sen avulla sensorit voivat tallentaa mittausdataa ilman, että niiden on oltava yhteydessä mobiiliverkkoon langattoman yhteyden avulla. Paikallinen välimuisti mahdollistaa sensorien ajoittaisen sammutuksen, jolloin laitteen virrankulutus on pienempää ja laite lähettää viestejä vain silloin, kun se on tarpeen. Paikallinen välimuistilaite auttaa myös samaan verkkoon kuuluvien sensorien päivityksessä, ja se mahdollistaa myös hajautetun laskennan. Hajautettu laskenta on usein järkevää, koska sensorien keräämillä tiedoilla on merkitystä vain paikallisesti. Sen avulla voidaan myös tehdä tietojen esiprosessointia, jolloin sensorien ja verkon resursseja säästyy. Se myös vähentää tarpeettoman ja itseään

toistavan datan keräämistä ja tallentamista pilveen. Saman kapillaariverkon sensorit voivat olla yhteydessä myös toisiinsa, jolloin ne voivat muuttaa toimintaansa toisiltaan keräämien tietojen avulla. (Collin & Saarelainen 2016)

3.7 **Sensorit**

Esineiden internetin tarkoituksena on kerätä dataa reaalimaailman ilmiöistä, ja muuttaa se sähköiseen muotoon, ja siirtää pilvipalveluun myöhempää hyödyntämistä varten. Laite, joka vastaa tiedon keräämisestä, ja muuntamisesta sähköiseen muotoon on nimeltään sensori. Sensoreita on paljon erilaisia ja eri käyttötarkoitukseen sopivia. Taulukossa 4 on lueteltu erilaisia sensoriluokkia esimerkkeineen. Sensorit muodostavat datan alkulähteen, jolta muu alusta komponentteineen saa tietonsa. Aiemmin sensorit rakennettiin yhtä tiettyä tarkoitusta varten, ja ne tehtiin usein osaksi jotain toista laitetta, esimerkiksi piirikorttia. Niillä oli sulautettu ohjelmisto, joka oli kehitetty pelkästään kyseistä käyttötarkoitusta varten. Näkyään kehitys vaikuttaa kulkevan kahteen suuntaan. Toisaalta on tarkoituksena eriyttää laite ja äly toisistaan, jolloin ohjelma voi olla täysin pilvessä ajettava. Toisaalta taas reunalaskennan rooli korostuu kriittisten järjestelmien tapauksessa. Voisi sanoa, että käyttötarkoitus ja järjestelmän vaatimukset ovat ratkaisevassa roolissa valintaa tehtäessä. Usein päädytään edellä mainittujen yhdistelmään. (Collin & Saarelainen 2016)

Taulukko 4. Sensoriluokat esimerkkeineen (Dargie & Poellabauer 2010, s. 5).

Sensoriluokka	Esimerkki
Lämpötila-anturi	Termistori, lämpösähköpari
Paineanturi	Painemittari, barometri, ionisaatiomittari
Optinen anturi	Valodiodi, valotransistori, infrapunasensori, CCD sensori
Akustinen anturi	Pietsosähköinen resonaattori, mikrofoni
Mekaaninen anturi	Jännitysanturi, kosketussensori, kapasitiivinen kalvo, pietsoresistiivinen solu
Liikeanturi	Kiihtyvyysanturi, gyroskooppi, kuvasensori
Virtausanturi	Tuulisensorit
Sijaintianturi	GPS-anturi, ultraäänisensori, infrapunasensori, deklinaattori
Elektromagneettinen anturi	Hall-komponentti, magnetometri
Kemiallinen anturi	pH-anturi, elektrokemiallinen anturi, infrapuna kaasuanturi
Kosteusanturi	Kapasitiivinen ja resistiivinen anturi, kosteusmittari, mikrosysteemiin perustuva kosteusanturi
Säteilymittari	ionisaation havaitsija, Geiger–Müller-ilmaisin

Suurin osa mitattavista ilmiöistä ja asioista on luonteeltaan ei sähköisiä. Ei-sähköisiä ilmiöitä ovat esimerkiksi tuulen voimakkuus, lämpötila, kosteusprosentti, pH-arvo ja sijainti. Sensorin tehtävänä on muuntaa informaatio sähköiseen muotoon, jolloin tietoa voidaan käsitellä ja siitä voidaan tehdä johtopäätöksiä. Usein sensori tarvitsee avukseen joko A/D-muuntimen tai I/O-tiedonkeruuyksikön, joka muuntaa sensorin analogisen tiedon digitaaliseksi. Anturin oma ohjausyksikkö mahdollistaa peräkkäisten mittausten keskiarvojen laskemisen, ja mittausten tarkentamisen erilaisilla korjauskertoimilla. Sensorit voidaan edelleen jakaa aktiivisiin ja passiivisiin. Aktiiviset välittävät energiaa ympäristöönsä ja odottavat vastetta, kun taas passiiviset pelkästään vastaanottavat informaatiota. Suurin osa sensoreita on passiivisia. Sensorit toimivat yleensä muiden laitteiden yhteydessä, jolloin ne on tavallisesti yhdistetty mikroprosessoriin, virtalähteeseen ja modeemiin, joka on liitetty verkkoon. Laitteissa on usein oma muisti paikallista käsittelyä ja lyhytaikaista tallennusta varten. (Collin & Saarelainen 2016)

Sensorien luotettavuus on tärkeä huomioon otettava asia. Sensorien pitkäaikaiseen toimintavarmuuteen vaikuttaa mm. fyysinen toimintaympäristö, laitteen laatutaso ja mittausalgoritmin luotettavuus. Fyysinen toimintaympäristö voi kuluttaa laitetta, joka voi johtaa mittaustulosten epätarkkuuteen. Varsinkin teollisessa ympäristössä laitteeseen voi kohdistua monenlaista ulkoista energiaa. Laitteen laatutaso ja sopivuus ympäristöön on tärkeää ottaa huomioon, kun valitaan käytettävää laitetta. Laite voi usein vaatia myös mekaanista puhdistamista, jotta se voi toimia sille asetettujen vaatimusten mukaisesti. Alustojen tarjoamien hallintatyökalujen ansiosta laitteen asetusten muuttaminen ja tarkasteleminen onnistuu etänä verkon yli. Sen ansiosta mm. algoritmeja voidaan muuttaa suhteellisen nopeasti ilman, että laitteen luona on käytävä. (Collin & Saarelainen 2016)

Sensorien virransyöttö on yleensä järjestetty pariston avulla. Siitä syystä virrankulutus on tärkeässä roolissa. Mikäli on mahdollista, kannattaa sensori ohjelmoida lähettämään viestejä vain tietyin väliajoin. Virrankulutuksen näkökulmasta on myös syytä kiinnittää huomiota lähetettävän datan määrään. Anturien virrankulutus on usein todella pieni, jopa muutamia mikroampeereja. On myös olemassa itsestään energiaa keräämiä sensoreja. Energian keräystavat voivat perustua lämpötila eroon, liikkeeseen tai sensorissa voi olla kiinni pieni aurinkokenno. Sensorien käyttämä langaton radiopiiri on suurin energian kuluttaja, ja verkkoteknologioiden kehityksellä on tärkeä rooli energian kulutuksen pienentämisessä. Paristokäyttöisyydelle vaihtoehtona on kehitteillä oleva tekniikka Power over Wi-Fi (PoWiFi), joka mahdollistaisi langattoman virransyötön sensoreille. Kiinteässä Ethernet-verkossa olevalle sensorille taas voidaan syöttää virtaa Power over Ethernet-tekniikalla. (Collin & Saarelainen 2016)

Pienet anturit ovat usein toivottuja esineiden internetin sovelluksissa. Tähän tarpeeseen vastaavat hyvin MEMS-anturit (Microelectromechanical-systems) eli mikrosysteemit, joiden koko voi olla 1-100 mikrometriä. Mikrosysteemistä koostuvan laitteen koko on tavallisesti 20 mikrometristä yhteen millimetriin. Teollisen internetin markkina-alueella rakennusautomaatio on suosituin mikrosysteemien käyttökohde. Seuraavana tulee kaluston seuranta ja sähköverkot. (Collin & Saarelainen 2016)

3.8 Tiedon tallentamisen haasteet esineiden internetin osalta

Esineiden internet tuo mukanaan joukon uusia vaatimuksia tietojen varastoinnin suhteen. Kaksi keskeisintä huomioitavaa asiaa tiedon varastointia suunniteltaessa ovatkin tietokannan tyyppi ja integraatiot. Sensoreilta tuleva data on usein rakenteetonta, ja sen määrä kasvaa todella kovaa vauhtia. Lisäksi eri tavoilla rakentunutta dataa on kyettävä yhdistämään toisiinsa. Nämä vaatimukset yksinään luovat perinteisille tietokannoille, kuten MySQL- ja SQL-palvelimille haasteita. Ongelman ratkaisemiseksi on kehitetty rakenteettomia ja puolirakenteellisia NoSQL-tietokantoja. Ne sallivat erityyppisen datan keräämisen yhteen tietokantaan. NoSQL- tietokantaan on mahdollista tallentaa mittausdataa, valokuvia, asiakirjoja, video- ja äänimateriaalia. Analytiikan välineet vaativat usein datan keräämistä samaan paikkaan, jotta kerättyjen tietojen osalta voidaan tehdä johtopäätelmiä. Sensoreilta kertyvä tieto ei usein noudata perinteisiä yritys- ja toimialarajoja. Tämä näkökulma on tärkeää ottaa huomioon jo tietovaraston suunnitteluvaiheessa. (Collin & Saarelainen 2016)

Edellä mainittujen skaalautuvuuden ja rakenteellisuuden lisäksi tietokantaa valittaessa tulee ottaa huomioon tietokantaan kasaantuvat tiedon nopeus. Osa tiedoista saattaa vaatia nopeaa käsittelyä, kun taas osa analysoidaan vasta myöhemmin. Sen takia tiedot voidaan lajitella niin sanottuihin kylmiin ja kuumiin polkuihin. Tallennettua tietoa tulisi myös voida tiivistää erilaisilla algoritmeilla, jotta tallennustilaa saadaan pienennettyä. NoSQL-tietokannan rakenteettomuus mahdollistaa sen, ettei tietojen tallennusvaiheessa ole pakko määrittää tietokannan rakennetta. Sillä taas on merkitystä tietojen myöhemmän analysoinnin kannalta. Suosittuja NoSQL-tietokantoja ovat MongoDB ja Apache Cassandra. Apache Cassandra sopii hyvin erityisesti teollisen- ja esineiden internetin vaatimuksiin sen laajasarakkeisuuden, suorituskyvyn ja skaalautuvuuden vuoksi. Kaikesta huolimatta SQL-tietokanta voi olla hyvä vaihtoehto, kun taulujen rakennetta ei tarvitse muuttaa, ja skaalautuvuus ja indeksointi on mietitty alusta alkaen huolellisesti. Mikäli taulujen rakenteeseen tulee jatkuvasti muutoksia, on NoSQL-tietokanta parempi vaihtoehto. (Collin & Saarelainen 2016)

Pilvipalvelu on usein edullisin vaihtoehto kustannusnäkökulmasta katsottuna. Aina pilvipalvelun käyttö ei kuitenkaan tule kysymykseen. Tällaisia tilanteita ovat henkilötietoja sisältävien tietojen tallentaminen sekä tilanteet, joissa pilvipalvelun latenssi on liian suuri tai verkkoyhteyden häiriöt estävät datan välittömän perille pääsyn. Tällaisissa tilanteissa tulee harkita

paikallisia tallennuspaikkoja ja turvautumista reunalaskentaan. (Collin & Saarelainen 2016)

Sensoridatan päälle on mahdollista integroida myös muita olemassa olevia tietolähteitä, kuten toiminnan- ja tuotannonohjausjärjestelmän sisältämät datat. Integrointi on mahdollista tehdä erilaisten rajapintojen avulla. Usein tällaiset tiedot ovat aivan erilaisia kuin sensorien tuottama data. Järjestelmien sisältämä data on suunniteltu eri tarkoitukseen kuin sensoridata. (Collin & Saarelainen 2016)

3.9 **Reunalaskenta**

Paikoitellen sensorien keräämää dataa halutaan analysoida jo ennen kuin se on viety pilvipalveluun saakka. Pilvipalveluiden ominaisuuteen kuuluu tiedonsiirrosta johtuva viive, ja se voi jossain tapauksissa aiheutua ongelmaksi. Tällöin sensorien tuottama data on analysoitava lähellä sen alkulähdettä. Tästä käytetään englannin kielessä nimitystä ”edge computing” tai ”fog computing”. Suomen kielen vakiintuneita nimityksiä ovat reunalaskenta, lähilaskenta, usvalaskenta tai hajautettu laskenta. Käytännössä reunalaskennan toteutuksesta huolehtii dataa keräävän tietokoneen oma prosessori tai gateway-laite. Laskentaa voidaan toteuttaa myös GPU-laskentana, jossa grafiikkaprosessori hoitaa laskennan. Valittava laite pitää määrittää sen mukaan, millainen laskentaprosessi on kyseessä. Teollisuusympäristö, sydämen toimintaa mittaava anturi tai esimerkiksi lentokone, antavat aivan erilaiset mahdollisuudet reunalaskennan hyödyntämiseen. Laitteen teho määrittää jonkin verran sitä millaista laskentaa voidaan suorittaa. Toisaalta voidaan ajatella, että myös laskentatarve voi ohjata laitevalintaa. (Collin & Saarelainen 2016)

Esineiden internetin yleistyminen lisää reunalaskennan tarvetta. Reunalaskenta mahdollistaa dataa keräävien laitteiden ohjaamisen kerätyn datan avulla lähes reaaliaikaisesti. Tällöin analytiikan tulosten hyödyntäminen ei ole riippuvainen tietoverkkojen aiheuttamasta viiveestä tai verkkojen käyttökatkoksista. Jossain tapauksissa tällainen viive saattaa aiheuttaa hengenvaarallisen tilanteen. Tällaisia tapauksia ovat esimerkiksi terveydenhuollon sovellukset tai liikenteen sovellukset. (Collin & Saarelainen 2016)

Reunalaskenta vähentää myös tallennustilan tarvetta, kun kaikkea tietoa ei siirretä pilveen. Pilvessä olevan tallennustilan ajatellaan usein olevan loputonta. Sensorimäärän kasvaessa myös niiden tuottama data lisääntyy. Tällöin myös kustannukset kasvavat. Toinen reunalaskennan tuoma etu on tietoliikenteen kuormituksen väheneminen. Kun osaa tiedosta ei tarvitse siirtää pilveen pelkästään analyysiä varten, kaistan kuormitus vähenee. Pilveen ei kannata viedä kaikkea laitteen tilasta kertovaa dataa sellaisenaan, koska siitä ei ole välttämättä mitään hyötyä. Jos laite tuottaa samana pysyvää dataa millisekunnin välein, niin on syytä pohtia, onko sillä merkitystä analyysin kannalta vai voisiko pilveen tallentaa dataa vain sekunnin välein

millisekunnin sijaan? Tällöin säästetään tietoliikennekaistassa ja tallennustilassa, mutta analytiikan kannalta tarpeellinen tieto on olemassa. Reunalaskennalle tieto voi kuitenkin olla merkityksellistä poikkeamien havainnoinnin kannalta. (Collin & Saarelainen 2016)

3.10 **Analytiikka**

Analytiikka on sensorien ja muun alustan talteen keräämän datan analysointia. Onnistunut analytiikka vaatii IT-osaston ja liiketoiminnan yhteistyötä, jotta datasta saadaan olennaiset tulokset esiin. Analytiikassa tietoa kerätään päätöksenteon tueksi, joten datalta pitää osata kysyä oikeat kysymykset. Olennaista analytiikassa on, että se tuottaa liiketoiminnalle lisäarvoa ja sen tavoitteena on mahdollistaa organisaation automatisoitu tiedolla johtaminen. (Collin & Saarelainen 2016)

Haastavinta analytiikan kannalta on se, miten datalta saadaan kysyttyä oikeita kysymyksiä. Tällöin tulee tuntea tarkasti liiketoiminta, jotta tiedetään mistä ja millaista tietoa tulee kerätä. Se vaatii yhteistyötä liiketoiminnan kanssa, koska analytiikan osaajilla ei ole välttämättä tietoa liiketoiminnasta, ja vastaavasti liiketoiminnalla ei ole analytiikan osaamista. On tärkeää ymmärtää liiketoimintaa, jotta kausaliteettien kanssa ei tule väärinkäsityksiä. Analytiikka on usein kokeilemista. Kokeilemalla voidaan löytää merkittäviäkin korrelaatioita, jotka poikkeavat ennakko-odotuksista. (Collin & Saarelainen 2016)

Analytiikan menetelmiä ovat mm. koneoppiminen, neuroverkkolaskenta ja muistinvarainen analytiikka. Analytiikan menetelmät myös kehittyvät jatkuvasti. Koneoppimisen avulla koneet oppivat kirjaimellisesti tekemään itsenäisiä päätöksiä niille syötetyn datan perusteella. Koneoppiminen voidaan jakaa kolmeen osaan, jotka ovat ohjattu oppiminen, vahvistusoppiminen ja ohjaamaton oppiminen. Ohjatussa oppimisessa kone oppii tunnistamaan oikean ja väärän sille ennalta syötetyn datan mukaisesti ja tekemään päätelmiä sen perusteella. Vahvisteoppimisessa kone oppii saamansa palautteen perusteella, eikä ennalta syötettyä aineistoa ole ollenkaan. Ohjaamaton oppiminen on selkeästi vaikein näistä menetelmistä. Siinä kone oppii tunnistamaan suuresta datamäärästä rakenteita ilman, että datan sisällöstä tiedetään mitään. Ohjaamattoman oppimisen menetelmiä ovat itseorganisoituvat kartat, klusterointi ja neuroverkot. (Collin & Saarelainen 2016)

Analytiikan tarpeet voivat olla välillä todella nopeita. Tällöin joudutaan turvautumaan datavirrassa suoritettavaan analytiikkaan. Se tarkoittaa käytännössä sitä, että sensorilta pilveen saapunutta dataan analysoidaan lennossa ennen kuin se on saapunut tietokantaan asti. Virtausanalyysi on käytössä useissa IoT-alustoissa. (Collin & Saarelainen 2016)

3.11 **IoT-sovellukset**

Sovellusta voidaan pitää esineiden internetin eräänlaisena lopputuotteena. Sen kehittämisen tulisi olla mahdollisimman liiketoimintalähtöistä, koska usein liiketoiminta on se, joka osaa parhaiten kertoa mitä datasta halutaan hyötyä. Sovellukset ottavat usein yhteyttä tietokantoihin ohjelmointirajapintojen kautta, jotka on avattu sitä varten. REST-rajapinta on usein hyvä vaihtoehto sen suosion vuoksi. SaaS-tyyppinen web-sovellus on usein näppärä, koska sitä ei tarvitse erikseen asentaa päätelaitteelle, ja päivittäminen on usein helpompaa. Kunhan sovellus on responsiivinen niin se mukautuu hyvin erilaisiin näyttökokoihin, ja sitä voi käyttää älypuhelimellakin. Web-sovellus on myös tietoturvamielessä parempi, koska se ei tallenna päätelaitteelle taustajärjestelmän dataa. (Collin & Saarelainen 2016)

Sovelluksen käyttöliittymän suunniteluun kannattaa panostaa, koska vasta käyttöliittymä auttaa liiketoimintaa saamaan hyötyjä irti kerätystä datasta. Ilman selkeää visuaalista esitystä data ei aukea kuin pienelle joukolle. Esitettyyn dataan on voitava porautua, ja raporttien olisi hyvä olla mahdollisimman reaaliaikaisia. Myös erilaisten hälytysten asettamisen tulisi olla mahdollista. (Collin & Saarelainen 2016)

3.12 **Digitaaliset palvelut**

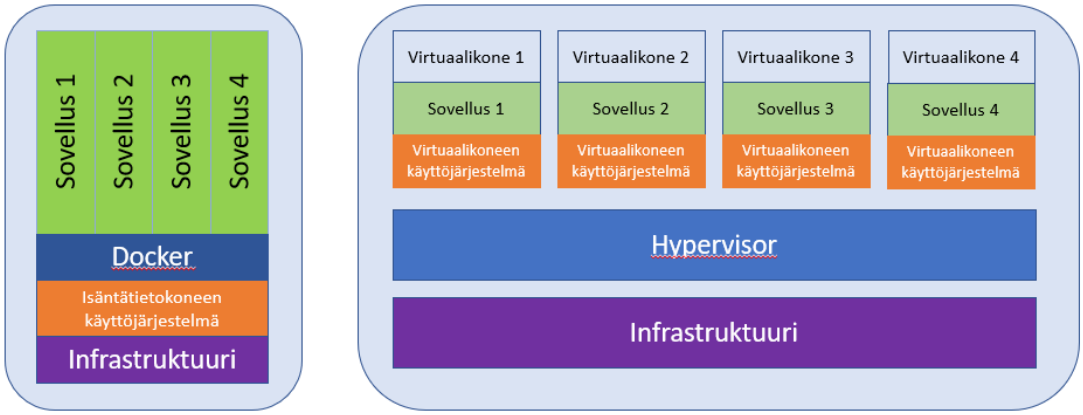
Digitaalisten palveluiden ansiosta yritysten on mahdollista tuottaa kerätystä datasta liiketoimintahyötyä. Palveluiden avulla IoT-ratkaisut on mahdollista tuotteistaa. Palvelu on korkeinta liiketoiminta-arvoa tuottava osa esineiden internetissä. Sen suunnittelussa mukana olevilta henkilöillä pitää olla liiketoimintaymmärrystä, jotta palvelu tuottaa parhaan mahdollisen arvon yritykselle.

3.13 **Docker Container**

Container-teknologiasta puhuttaessa tarkoitetaan ohjelmistokontteihin (container) pakattua ohjelmakoodia, joka sisältää ohjelmakoodin, runtime-ohjelmiston, järjestelmätyökalut, järjestelmäkirjastot ja asetukset. Yhteen konttiin pakataan aina yhteen tiettyyn käyttötarkoitukseen tarkoitettu ohjelma, jota voidaan käyttää useassa eri laitteessa ilman, että konttiin tarvitsee tehdä muutoksia. Kontit ovat kevyitä ohjelmia, jotka toimivat aina samalla tavalla oli ympäristö millainen tahansa. Kontti erottaa ohjelman muusta ympäristöstä. Ohjelmat tallennetaan Docker-kuvina (Docker image), ja niistä tulee kontteja siinä vaiheessa, kun niitä ajetaan Docker Runtimella. (Docker 2019)

Docker Runtime on ohjelma, joka mahdollistaa laitteeseen ympäristön, joiden avulla kontit voivat käyttää sen laitteen ominaisuuksia, minne se on

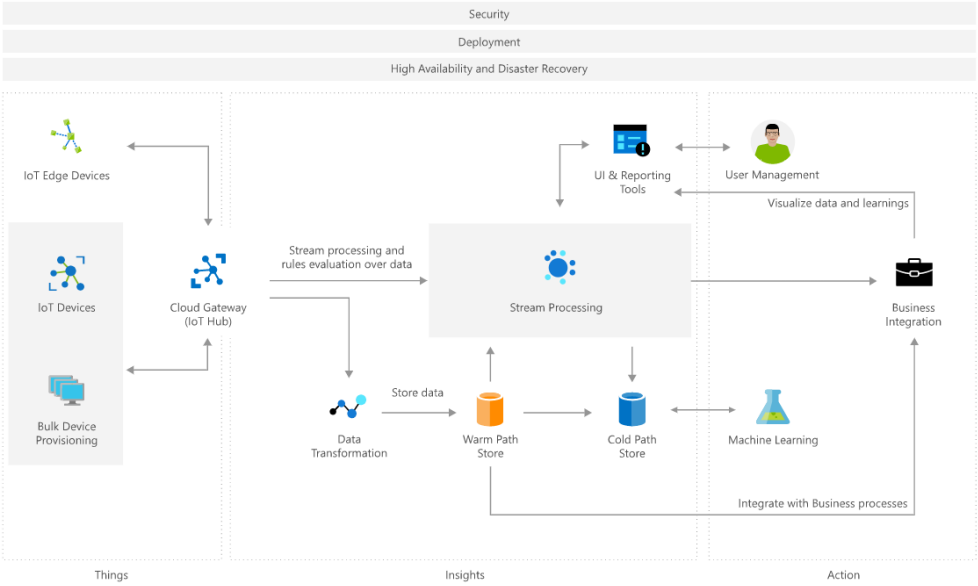
asennettu. Docker Runtime mahdollistaa laitteen käyttöjärjestelmän ja ytimen (kernel) laitteeseen asennettujen konttien käyttöön. Docker Runtimea voisi verrata virtualisoinnissa käytettyyn hypervisor-ohjelmaan, joka mahdollistaa virtuaalikoneille laitteen fyysisten muistien ja prosessorin käytön. Virtualisointiin verrattuna kontit ovat paljon kevyempiä kuin virtuaalikoneet. Kontit myös käynnistyvät nopeammin ja ne ovat kooltaan pienempiä. Docker-teknologian ja virtualisoinnin eroavaisuudet on ilmoitettu tarkemmin kuvassa 2. (Foulds 2019, s.299-302)



Kuva 2. Docker-teknologian ja virtualisoinnin eroavaisuudet (Docker 2019).

4 MICROSOFT AZUREN IOT-ARKKITEHTUURI

Tässä luvussa käydään läpi Microsoft Azure pilvipalvelun IoT-arkkitehtuuri ja sen pääkomponentit. Tutkimus keskittyy erityisesti niihin komponentteihin, joista osaa on hyödynnetty tämän opinnäytetyön toiminnallisessa osassa. Azuren IoT-arkkitehtuuri sisältää IoT-alustan kaikki tasot, mukaan lukien laitteiden yhdyskäytävän, käyttäjänhallinnan, tietovirran prosessointityökalun, tallennustilan, analytiikkavälineet ja esitystyökalut. Kuvan 3 mukaan Azuren IoT-alustan voi jakaa karkeasti kolmeen osaan. Ne ovat Things (dataa lähettävät laitteet), Insights (tiedon käsittely pilvipalvelussa) ja Action (kerätyn tiedon hyödyntäminen ja analysointi). (Microsoft 2019a)



Kuva 3. Azuren IoT-komponenttien arkkitehtuurimalli. (Microsoft 2019a)

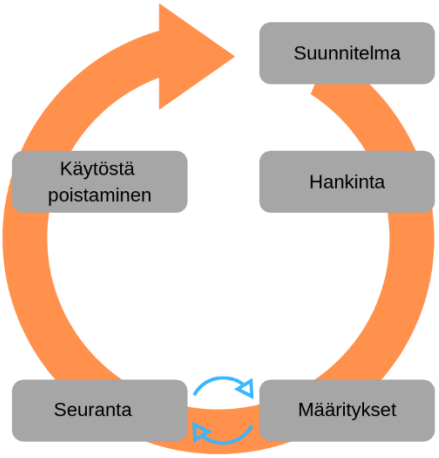
Things käsittää laitteet ja palvelut, jotka lähettävät tietoa pilveen IoT Hub -yhdyskäytävän kautta. Azure mahdollistaa myös reunalaskentatoiminnot IoT Edge -resurssin kautta ilman, että tietoa on ensin lähetettävä pilvipalveluun käsittelyä varten. Insights käsittää palvelut, joille Things tuottaa tietoa käsittelyä varten. IoT Hub on yhdyskäytävä, jonka kautta reunalla toimivat ja muut laitteet kommunikoivat pilven resurssien ja komponenttien kanssa. Insights käsittää tietovirran prosessointiin käytettävät työkalut ja tietokannat. Action osa koostuu käyttäjänhallinnan työkaluista, liiketoimintasovelluksista ja koneoppimisen työkaluista. Azuren IoT-arkkitehtuuri on pilvinatiivi-, mikropalvelu- ja serverless-pohjainen. Se tukee hybrid cloud-mallia ja mahdollistaa reunalaskennan käytön. (Microsoft 2019a)

4.1 **Azure IoT Hub**

IoT Hub on yksi Azuren keskeisimmistä palveluista IoT-laitteiden hallinnassa. Se on pilvessä oleva hallintapalvelu, joka toimii yhdyskäytävänä sovellusten ja IoT-laitteiden välillä. Hub tarkoittaa suomeksi keskitintä. Se kuvaa hyvin kyseisen komponentin toimintaa, sillä se keskittää ja kerää laitteilta tulevan liikenteen. IoT Hub mahdollistaa kaksisuuntaisen kommunikaation laitteiden ja pilvipalvelun välillä, ja se tukee useita viestintäkaavioita, kuten telemetriatietojen lähetystä laiteesta pilveen, tiedoston lataamista laitteesta ja pyyntö-vastaus-metodia laitteiden kontrollointia varten. Yhtä IoT Hubia on mahdollista käyttää useiden erilaisten laitteiden toimesta samanaikaisesti. Laitteiden hallinnan pääpiirteet voi jakaa neljään osaan, jotka ovat skaalautuvuus ja automaatio, avoimuus ja yhteensopivuus, asiayhteystietoisuus ja palvelu monissa rooleissa. (Microsoft 2018c)

Skaalautuvuudella ja automaatiolla tarkoitetaan suurenkin laitemäärän helppoa hallittavuutta, ja sen automatisointia niin pitkälle kuin mahdollista. Avoimuudella ja yhteensopivuudella tarkoitetaan yhteensopivuutta laajoihin, erilaisiin laitteiden ekosysteemeihin. Palvelun on kyettävä toimimaan erilaisten laitetyyppien kanssa huolimatta siitä millaisia ne ovat. Asiayhteystietoisuudella tarkoitetaan palvelun dynaamisuutta ja toiminnan jatkuvuuden varmistamista. Ympäristö, jossa laitteet ovat, on dynaaminen ja siinä tapahtuu usein muutoksia. Tästä huolimatta ylläpito ei saa vaikuttaa vaarantavasti prosesseihin. Palvelun saatavuus tulee mahdollistaa monille eri roolien toimijoille, jotta tiedonkulku ja palvelun ylläpidettävyys voidaan taata. (Microsoft 2018e)

Laitteiden elämänkaari voidaan yleisesti jakaa viiteen eri vaiheeseen, jotka ovat suunnitelma (plan), toimitus (provision), konfiguraatio (configure), valvonta (monitor) ja vetäytyminen (retire). Suunnitteluvaiheessa operaattori luo metatietoon perustuvan skeeman (schema) jonka avulla tiedot ovat haettavissa ja joiden avulla suurta laitejoukkoa voidaan hallita. Metatietojen tallentamiseen käytetään Device Twin-palvelua, josta kerrotaan tarkemmin luvussa 4.2. Toimitusvaiheessa laitteen tiedot viedään Identity Registry-palveluun, jotta laitteet voivat luoda yhteyden IoT Hubiin. Konfiguraatiovaiheessa laitteiden ominaisuuksia ja päivityksiä ylläpidetään Device Twinin tietoja muuttamalla. Valvontavaihe liittyy läheisesti konfiguraatioon ja usein niitä tehdään rinnakkain. Siinä tarkastellaan käynnissä olevien laitteiden toimintoja, hälytyksiä ja yleistä tilaa. Vetäytymisvaiheessa laitetta pitää päivittää, tai sen palvelut poistetaan käytöstä. Device Twinin avulla laitteen tiedot ovat mahdollista tallentaa, jos laite on korvattava toisella. Tiedot voi myös arkistoida. Kuvassa 4 laitteiden elinkaari on kuvattu jatkuvana kiertokulkumaisena prosessina. Konfiguraatio ja valvonta seuraavat usein toisiaan prosessien ylläpitoaikana. (Microsoft 2018e)

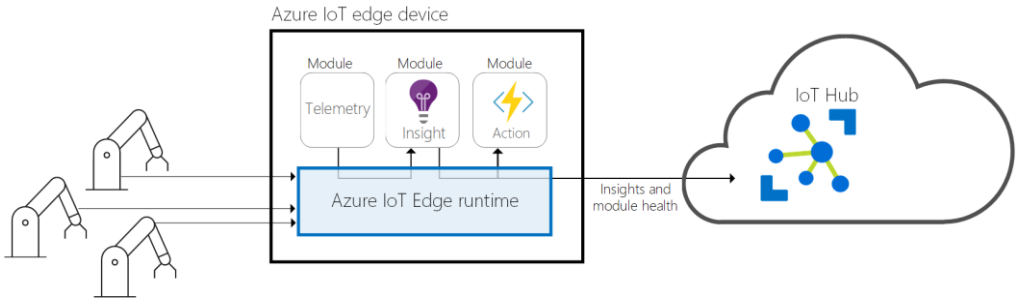


Kuva 4. Laitteen elinkaari (Microsoft 2018e).

IoT Hub tarjoaa REST-rajapinnan kommunikointiin laitteiden ja sovellusten välillä. Rajapinnan käyttäminen on mahdollista kaikilla teknologioilla, jotka voivat lähettää ja vastaanottaa HTTP- ja HTTPS-viestejä. Rajapinta mahdollistaa Digital Twinin hallinnan, laiteidentiteettien hallinnan ja töiden hallinnan ja aikatauluttamisen. Rajapinnan mahdollistavat pilven ja laitteen välisen, kaksisuuntaisen kommunikaation. (Microsoft 2018d)

4.2 Azure IoT Edge ja Edge Runtime

IoT Edge-laitteista puhuttaessa tarkoitetaan laitteita, joihin on asennettu Edge Runtime-ohjelmisto. Edge Runtime on kokoelma ohjelmia, joiden avulla laitteen on mahdollista kommunikoida pilven ja muiden liitettyjen laitteiden kanssa ja suorittaa siihen asennettuja ohjelmia. Edge Runtimen toimintoihin kuuluu työsuoritteiden asennus ja päivitys laitteeseen, turvallisuusstandardien ylläpito laitteessa, ja moduulien päällä pysymisen varmistaminen aina laitteen ollessa päällä. Lisäksi se raportoi moduulien tilan IoT Hubille, jotta niitä voidaan ohjata etänä. Sen tehtäviin lukeutuu myös kommunikoinnin helpottaminen Edgen- ja muiden laitteiden välillä, kuten myös Edge-moduulien välillä, sekä IoT Edgen ja IoT Hubin välillä. Kuvassa 5 runtime-ohjelman toiminta on kuvattuna. (Microsoft 2018b)

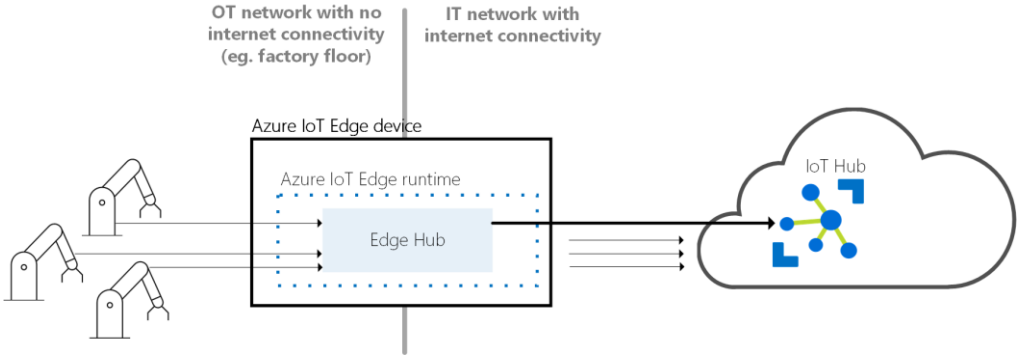


Kuva 5. IoT Edge-arkkitehtuuri (Microsoft 2018).

Edge Runtimen päätehtävät voidaan jakaa kahteen kategoriaan: kommunikointiin ja moduulien hallintaan. Edge Runtime muodostuu kahdesta moduulista, jotka hoitavat näiden tehtävien suorittamisen. Ne ovat Edge Hub ja Edge Agent. Edge Hub on vastuussa kommunikoinnista, ja Edge Agent huolehtii moduulien sijoittamisesta ja tarkkailusta. Molemmat moduuleja ovat moduuleja siinä missä mikä tahansa toinenkin Edge-moduuli. (Microsoft 2018b)

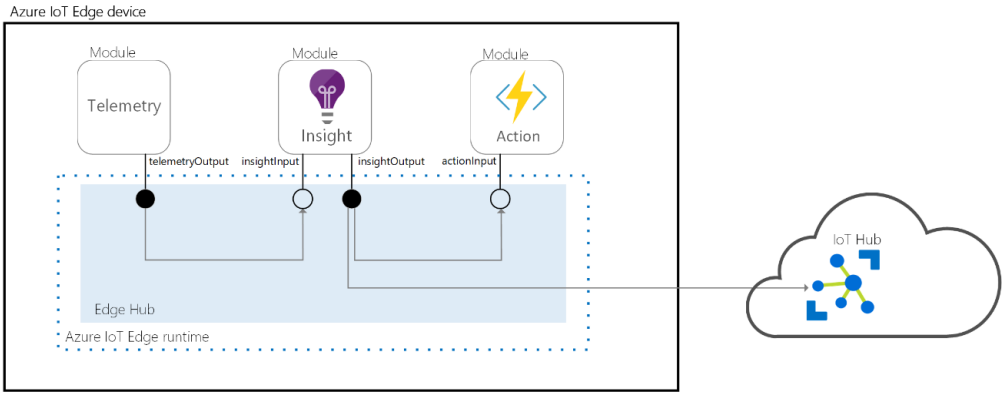
Edge Hub toimii paikallisen välityspalvelimen tavoin IoT Hubille, välittäen samat protokollan päätepisteet kuin IoT Hub. Tästä johtuen asiakkaat (clients) voivat yhdistyä Edge Runtimeen kuten IoT Hubiin. Edge Hub tukee MQTT ja AMQP-protokollia, mutta ei HTTP-protokollaa. Edge Hub delegoi osan tehtävistä IoT Hubille. Tästä on esimerkkinä pyyntöjen todentaminen, kun siihen liitetty laite yrittää ensimmäisen kerran ottaa yhteyttä. Kun yhteys on perustettu laitteen ja Edge Runtimen välille, Edge Hub säilyttää yhteystiedot automaattisesti tulevia yhteydenottoja varten ilman, että se joka kerta varmistaa niitä IoT Hubilta. Tästä syystä Edge Runtimen pitää olla yhteydessä joka kerta kun laitetta todennetaan. (Microsoft 2018b)

Kuvassa 6 Edge Hub yhdistää siihen liitettyjen laitteiden ja moduulien yhteyksiä yhdeksi fyysiseksi yhteydeksi sen ja IoT Hubin välillä pienentääkseen tiedonsiirron määrää. Asiakkaille tämän näkyy kuitenkin niin, että heillä kaikilla on oma yhteys IoT Hubiin, vaikka he käyttävätkin yhtä yhteyttä. (Microsoft 2018b)



Kuva 6. Kuvaus IoT Hubin, Edge Hubin ja sen asiakkaiden välisestä tiedonsiirrosta (Microsoft 2018b).

Edge Hub voi päätellä onko se kulloinkin yhteydessä IoT Hubiin. Yhteyden ollessa suljettu, se tallentaa viestit ja Device Twinin päivitykset paikallisesti. Kun yhteys on palautunut, Edge Hub synkronoi tiedot IoT Hubin kanssa. Tiedon paikalliseen tallentamiseen käytettävän välimuistin tallennuspaikka on määritetty Edge Hubin Module Twinin ominaisuuksissa. Välimuistin koko on riippuvainen vain tallennuskapasiteetista. Kuvassa 7 IoT Edge Hub ohjaa moduulien välistä kommunikaatiota, ja pitää moduulit erillään. Kehittäjän tulee määritellä mistä moduuli saa dataa, ja minne se lähettää dataa. (Microsoft 2018b)



Kuva 7. Moduulien välinen kommunikaatio Edge Hubin sisällä (Microsoft 2018b).

IoT Edge Agent on yksi niistä moduuleista, jotka muodostavat Edge Runtimen. Sen vastuualueisiin kuuluu moduulien toteutus, käynnissä pysymisen varmistaminen, ja moduulien tilan raportointi IoT Hubille. IoT Edge Security Daemon huolehtii laitteen käynnistyessä IoT Edge Agentin käynnistyksestä. Edge Agent noutaa käynnistystietonsa IoT Hubilta. IoT Hubista se saa JSON-tiedoston, jossa on kerrottu konfiguraatiotiedot, kuten se mitkä moduulit sen tulee käynnistää. (Microsoft 2018b)

Identiteettirekisteri on osa jokaista IoT Hubia. Jotta laite tai moduuli voidaan yhdistää IoT Hubiin, on niiden tiedot oltava tallennettuna identiteettirekisteriin. Jokainen rekisteriin tallennettu instanssi (laite tai moduuli) tulee lisäksi olla todennettu IoT Hubiin perustuvilla tunnistetiedoilla. Rekisteriä tulee käyttää, kun instansseja toimitetaan IoT Hubiin, tai instanssien pääsyä yhteyspisteisiin halutaan hallinnoida. Identiteettirekisterin toimintaan kuuluu instanssien identiteetin luonti, päivitys, poistaminen, hakeminen ID:n perusteella, listaustoiminto, tuonti ja vienti Azuren tietovarastoon. (Microsoft 2018h)

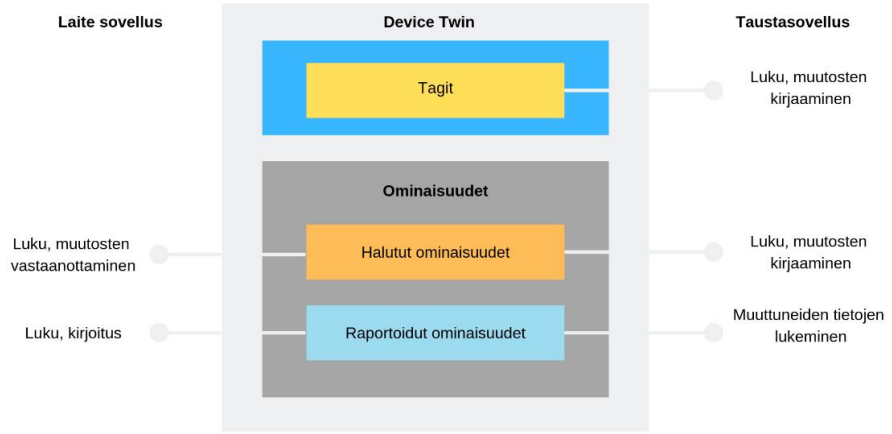
Identiteettirekisteri ei sisällä sovellusten metatietoja, eikä se tuo ekspressiivisiä hakuja, ja siihen voi päästä käsiksi kuten hakemistoon deviceid:n tai moduleid:n avulla. Rekisteriä ei tulisi myöskään käyttää korkeaa suoritustehoa vaativiin tehtäviin, vaan pelkästään laitehallintaan. (Microsoft 2018h)

4.3 Azure Device Twin

Device Twin on JSON-tiedosto, jossa säilytetään laitteiden tilaa koskeva tieto, kuten metatieto, konfiguraatio ja ehdot. Device Twinia säilytetään ja ylläpidetään IoT Hubissa. Tiedot ovat olemassa jokaisesta IoT Hubiin liitetystä laitteesta. Device Twinia käytetään pääasiassa laitekohtaisten metatietojen tallentamiseen, laitteiden tilaa koskevien tietojen raportointiin, työtehtävien synkronointiin laitteen ja pilven sovellusten välillä, ja laitetta

koskevien tietojen hakemiseen, kuten metatiedot, konfiguraatio ja tila. Device Twin sisältää seuraavat tiedot: tagit, halutut ominaisuudet, raportoidut ominaisuudet ja laitteet identiteettiä koskevat ominaisuudet (kuva 8). (Microsoft 2018f)

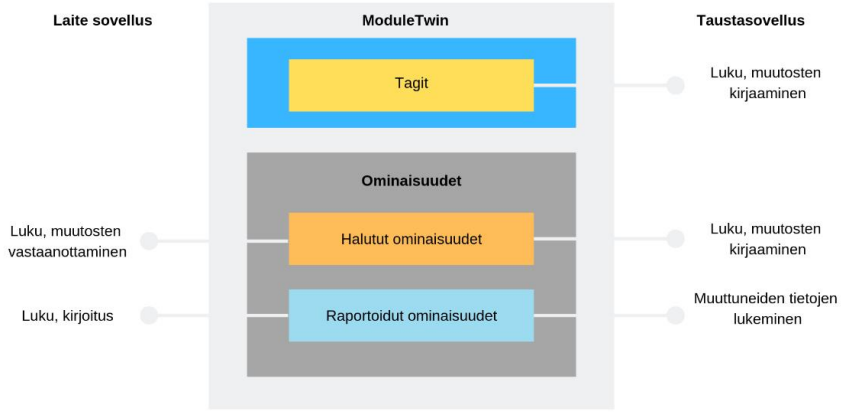
Tagit ovat tietoja laiteesta, kuten sijainti rakennuksessa. Halutut ominaisuuksia ja raportoituja ominaisuuksia käytetään rinnakkain laitteen synkronointiin ja päivitykseen. Niiden avulla laitteen toimintaa voidaan ohjata ja säätää haluttuun suuntaan, ja saada selville nykyiset ominaisuudet. Laitteen identiteettiä koskevat tiedot pitävät sisällään mm. laite ID:n, tilatiedot ja yhteystiedot. (Microsoft 2018f)



Kuva 8. Device Twinin sisältö (Microsoft 2018f).

4.4 **Azure Module Twin**

Device Twinin tapaan jokaisesta moduulista säilytetään moduuleja koskevat tiedot JSON-tiedostossa, josta käytetään nimeä Module Twin. Se koostuu tageista, halutuista ja raportoiduista ominaisuuksista ja moduulin identiteettiä koskevista ominaisuuksista (kuva 9). (Microsoft 2018g)

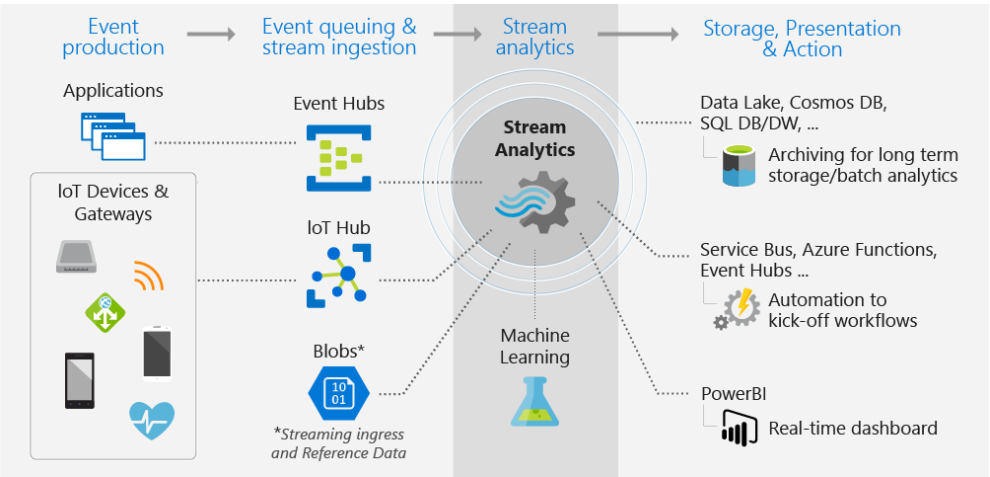


Kuva 9. Module Twinin sisältö (Microsoft 2018g).

Yhteen IoT Hubiin yhdistetyn laiteidentiteetin taakse voi liittää 20 moduuli-identiteettiä, joista IoT Hub ylläpitää tietoja. Jokaisesta moduulista, jotka on luotu laitteeseen, voidaan luoda oma itsenäinen yhteys IoT Hubiin. Tämä mahdollistaa moduulien yksilöllisemmän hallinnan, joka vähentää vikaherkkyyttä ja helpottaa ylläpitoa. Jokaista laitetta voidaan ylläpitää yksilöllisemmin, kun niitä pääsee käsittelemään moduulitasolla. (Microsoft 2018g)

4.5 Azure Stream Analytics

Azure Stream Analytics on tapahtumien ja tietovirtojen käsittelyyn käytettävä komponentti, joka mahdollistaa IoT Hubilta tulevien syötteiden käsittelyn. Tietoa voidaan reitittää myös muista lähteistä, kuten nettisivuilta, sosiaalisen median syötteestä, sovelluksista ja niin edelleen. (Microsoft 2018k)



Kuva 10. Stream Analyticsin rooli osana Azuren tiedon prosessointia. (Microsoft 2018k).

Stream Analyticsiin kirjataan töitä (jobs), joiden avulla määritellään mistä tieto kerätään, ja minne tietoa syötetään. Stream Analytics toimii eräänlaisena kokoojana ja reitittäjänä, kuten kuvasta 10 käy ilmi. Siellä on mahdollista SQL-perusteisen kielen, T-SQL:n avulla suodattaa, ja/tai jalostaa tietoa ennen sen eteenpäin lähettämistä. Eri lähteistä tulevia tietovirtoja on mahdollista yhdistää yhdeksi eteenpäin lähetettäväksi virraksi. Tietovirran mukana tulevia paranmetrien arvoja voidaan yhdistää ja nimetä uudelleen. Stream Analytics on serverless-tyyppinen, PaaS-palvelu (Platform as a Service), joten kehittäjän ei tarvitse hankkia erikseen palvelinta, joka palvelua ajaa. Palvelu on mahdollista perustaa myös IoT Edge-laitteelle, jolloin on mahdollista toteuttaa hybridiarkkitehtuuri prosessointia varten. Reunaratkaisu voi olla tarpeen tehtäville, joissa vaaditaan mahdollisimman pieni latenssi eli viive. (Microsoft 2018k)

Stream Analyticsin käsiteltyä tiedon, se on mahdollista ohjata eteenpäin, joko erilaisiin tallennuskohteisiin, tai siitä voidaan tehdä reaaliaikaisia visualisointeja Power Bi-ohjelman avulla, tai se voidaan ohjata eteenpäin Azure Functionille. Azure Functionin avulla voidaan luoda ulkoisia rajapintoja sovellusten käyttöön. Tieto voidaan ohjata myös koneoppimistyökalulle. (Microsoft 2018k)

4.6 **Azure Databricks**

Stream Analyticsin palvelun ohella Azure tarjoaa muitakin tietovirran käsittelytyökaluja. Databricks on datan käsittelyyn tarkoitettu työkalu, joka tarjoaa ratkaisuja erityisesti vaativampien ja monimutkaisimpien analyyttisten ongelmien ratkaisemiseen. Databricks on rakennettu Apache Spark-analytiikka-alustan päälle, ja se tarjoaa sitä kautta Sparkin ominaisuudet Databricksin käyttöön. Stream Analyticsiin verrattuna Databricksin ohjelmointikielivalikoima on laajempi, koska se tukee Pythonia, R-kieltä, Scalaa ja Javaa. IoT-ratkaisujen näkökulmasta tärkeää on Databricksin ominaisuus ottaa vastaan IoT Hubilta lähettyä tietoa. (Microsoft 2019e)

4.7 **Azure SignalR**

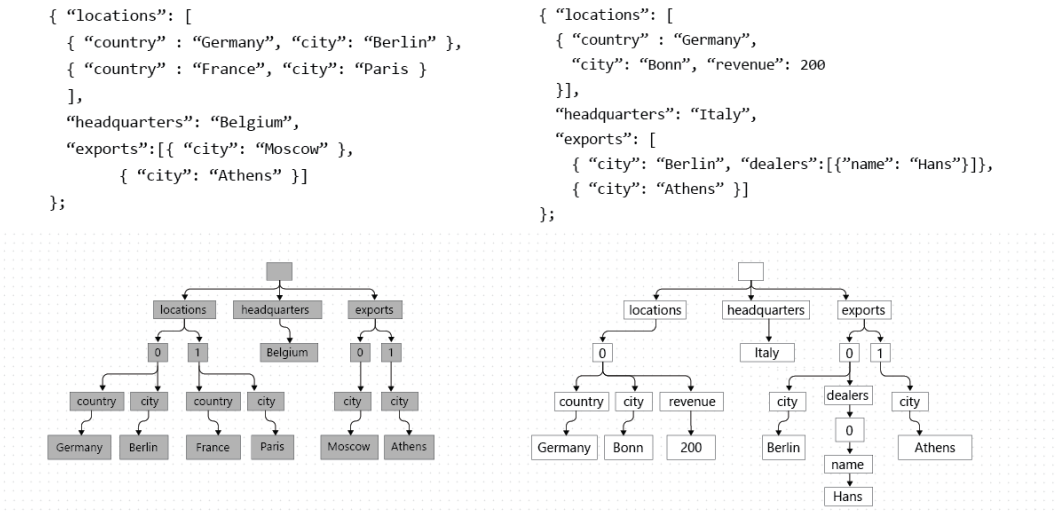
SignalR on Azuren sisäänrakennettu palvelu, jonka avulla on mahdollista lisätä reaaliaikaisia www-toiminnallisuuksia käyttämällä HTTP-protokollaa. SignalR käyttää push-toiminnallisuutta, jonka avulla asiakkaiden (clients) ei tarvitse ottaa yhteyttä palveluun, vaan viesti lähtee niille, jotka ovat palvelun tilanneet. SignalR on toimiva erityisesti palveluissa, joita on serverin puolesta päivitettävä usein, kuten esimerkiksi peli-, äänestys- tai kartta- ja paikkatietosovellukset, sekä usein päivittyvät myyntiraportit. Microsoft (2018j)

WebSocketit, Server-Sent Event (SSE) ja Long Polling ovat SignalR:n käyttämiä tekniikoita. Käytettävä tekniikka valikoituu automaattisesti sen perusteella mitä palvelin- ja asiakasohjelmat tukevat. Azuressa SignalR-palvelua voidaan käyttää kolmella eri tavalla, jotka ovat ASP.NET Core SignalR App, osana Azure Functionia tai REST-rajapinnan kautta. Microsoft (2018j)

4.8 **Azure Cosmos DB**

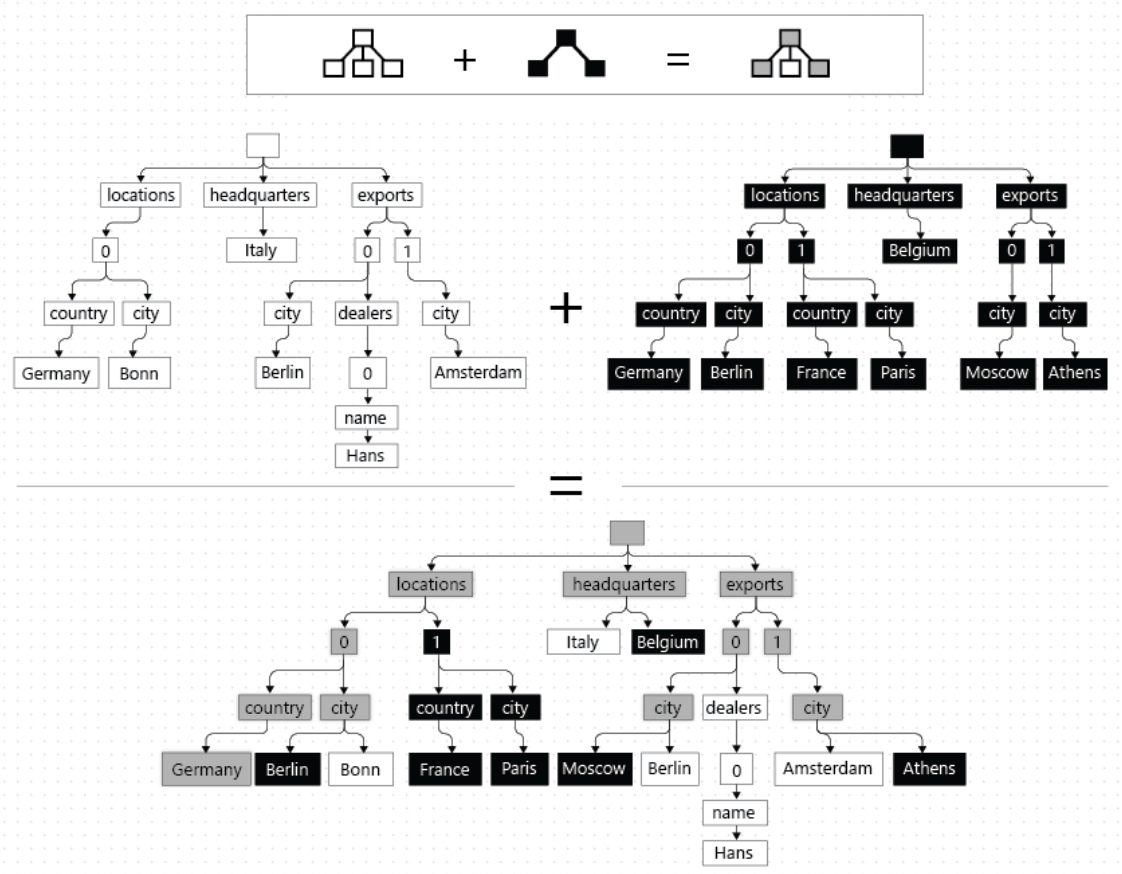
Azure Cosmos DB on skaalautuva ja maailmanlaajuisesti hajautettu tietokantapalvelu, joka on tehty vastaamaan matalan viiveen ja korkean saavutettavuuden tietokantakyselyihin. Cosmos DB mahdollistaa tietokannan ajamisen usealla palvelimella yhtä aikaa, jolloin viive muodostuu mahdollisimman pieneksi. Microsoft takaa palvelulle 99,999% saatavuuden maailmanlaajuisesti ja alle 10 millisekunnin viiveen tietokantahakuihin ja kirjoituksiin. Cosmos DB on suunniteltu toimivaksi erityisesti sellaisten tietojen tallentamisessa, joilla on oltava mahdollisimman pieni latenssi. Pieni latenssi varmistetaan tiedon hajauttamisella useaan maantieteelliseen alueeseen samanaikaisesti. Palvelu tarjoaa monimalliin perustuvat rajapinnat (multi-model API), jonka vaihtoehdot ovat SQL, MongoDB, Cassandra, Tables tai Gremlin. Oletuksena Cosmos DB indeksoi kaikki tiedot automaattisesti, eikä kehittäjän ole tarpeen käyttää sekundaarisia indeksejä. Palvelu on myös skeemariippumaton (schema-agnostic). (Microsoft 2018i)

Cosmos DB osittaa tietoja eri ryhmiin erikseen määritetyn avaimen avulla, joka määritetään tietokokoelman luomisen yhteydessä. Ositusavaimen (partition key) avulla Cosmos DB jakaa saman ositusavaimen omaavat tiedot samoihin loogisiin tietosäiliöihin (container). Ositusavain määritetään tietosäiliön luonnin yhteydessä. Kuvassa 11 on esitetty JSON-dokumentin pohjalta luotu puukaavio, jossa palvelu luo automaattiset indeksinumerot taulukkotyyppisesti syötetyille tiedoille. Cosmos DB:n käyttämä käänteistiedosto (inverted index) mahdollistaa nopeampien tietokantahakujen suorittamisen. (Microsoft 2018i)



Kuva 11. Esimerkki Cosmos DB-tietokannan indeksoinnista. Palvelu luo automaattisesti indeksit taulukkomuotoisille syötteille (Microsoft 2018i).

Hakemistopuu (index tree) on dokumentti, joka muodostetaan yhden tietosäiliön sisällä olevista puukaavioista. Se päivittyy sen mukaan, kun kokoelmaan lisätään uutta tietoa. Kuvassa 12 on esitetty hakemistopuu, joka on luotu kahdesta puukaaviosta.



Kuva 12. Hakemistopuun muodostuminen (Microsoft 2018i).

4.9 **Azure Data Lake Store**

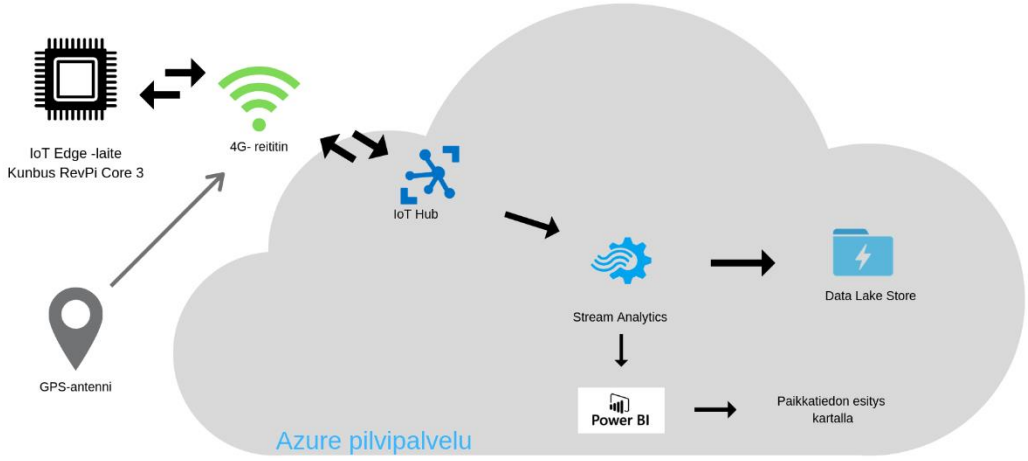
Azure Data Lake Store eli datajärvi on tiedon pitkäaikaiseen tallentamiseen tarkoitettu tallennusvaihtoehto Blob Storagen ohella. Sen avulla voidaan tallentaa eri muodoissa olevaa tietoa, ja se tarjoaa ympäristön tiedonkäsittelijöille ja koneoppimisen hyödyntämiseen. Data Lake mahdollistaa petatavun kokoisten tiedostojen käsittelyn, eikä tietokanta aseta minkäänlaisia rajoitteita tietokannan, tai siellä olevien tiedostojen koolle tai säilytysajalle. Data Lake on yhteensopiva Hadoop-klusterin kanssa, ja se tarjoaa WebHDFS-yhteensopivan REST-rajapinnan sovellusten käyttöön. Data Lake on myös skeemariippumaton, jolloin etukäteen ei ole tarpeen määrittää millaista tietoa varastoidaan. (Microsoft 2019d)

4.10 **Azure Blob Storage**

Azure Blob Storage on tiedon tallennus vaihtoehto Data Lakelle. Se soveltuu pysyvämmän tiedon tallennuspaikaksi, tiedon raportointilähteeksi ja tietoarkistoksi. Sen avulla on mahdollista tallentaa myös rakenteetonta tietoa. Tuotetta valittaessa valitaan myös taso, jolla tietoa haetaan tietokannasta. Blob Storage on skaalautuva palvelu, jota voidaan laajentaa tarpeen vaatiessa. Blob Storagen käyttö vaatii voimassa olevan SQL-palvelimen. Blob Storage soveltuu erityisen hyvin, kun haetaan kuvia tai dokumentteja suoraan selaimeen, tallennetaan tiedostoja hajautetusti, video- ja äänimateriaalien suoratoistoihin, loki tiedostojen kirjoittamiseen, tieto varmuuskopiointiin ja analyysipalveluissa käytettävälle tiedolle. Blob Storageen voidaan ottaa HTTP- ja HTTPS-yhteys REST-rajapinnan avulla, Azuren sisäänrakennetulla Power Shellillä tai ohjelmakirjastoilla, jotka ovat ohjelmointikielikohtaisia. (Microsoft 2019c)

5 CASE: IOT-RATKAISUN RAKENTAMINEN MICROSOFT AZUREN IOT-ARKKITEHTUURIN AVULLA

Tämän case-tapauksen tavoitteena on rakentaa IoT-sovellus hyödyntämällä Microsoft Azure pilvipalvelua. Kyseessä on ”päästä päähän”-toteutus, jossa haetaan sensorilta paikkatieto, joka viedään pilvipalveluun prosessointia ja tallentamista varten. Tieto esitetään visuaalisesti Power Bi-sovelluksen avulla. Kuvassa 13 on esitetty havainnollistava järjestelmän kuvaus. Siitä käy hyvin ilmi mitkä ovat varsinaisia pilvipalvelun resursseja ja mitä muita työkaluja järjestelmän rakentamiseen on tarvittu. Ratkaisu noudattaa keskitettyä arkkitehtuurimallia.



Kuva 13. IoT-toteutuksen periaatekuvaus.

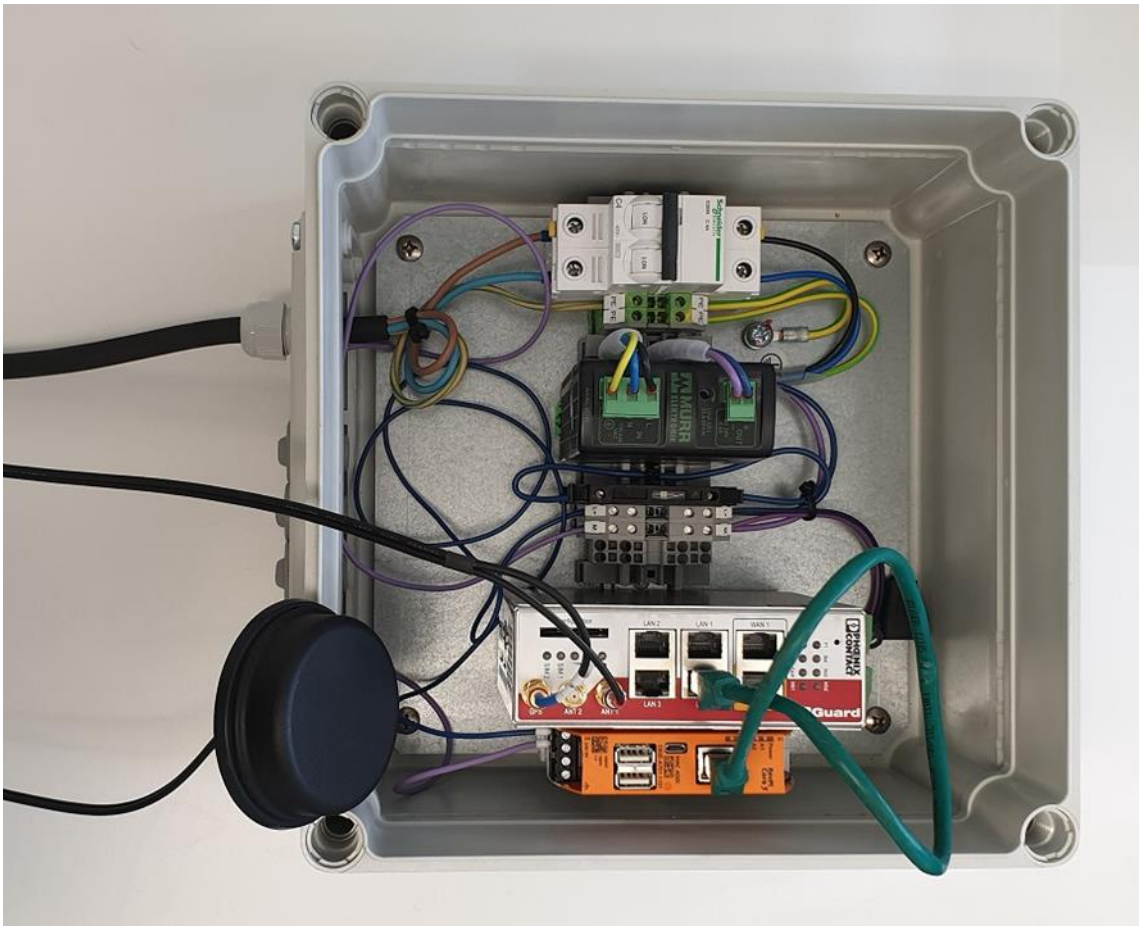
5.1 Azuren resurssien perustaminen

Ensimmäinen askel sovelluksen rakentamisessa on resurssien perustaminen Azure-pilvipalveluun. Perustettavia resursseja ovat IoT Hub, Stream Analytics Job, Container Registry ja Data Lake Store. Tiedon graafista esittämistä varten tulee olla voimassa oleva Power Bi-tili. Tässä työssä ei käydä läpi Azure-tilin perustamista, vaan se on oletuksena olemassa.

Resurssit voidaan perustaa Azure-portaalin tai komentorivityökalun avulla. Tässä työssä käytetyt resurssit perustettiin käyttämällä Azure-portaalia. Resurssit perustetaan valitsemalla Azure-portaalista ”Create a resource”-painike. Sen takaa avautuu valikko, jossa Azuren resurssit on listattu ryhmittäin. Resurssia voi myös hakea hakutoiminnon avulla. Luotaessa resursseja valitaan oikea Resource Group, jonne resurssit liitetään. Jos sellaista ei ole niin sen voi myös perustaa tässä vaiheessa. Olennaista on valita oikea Resource Group ja tilaus, jonne resurssit perustetaan.

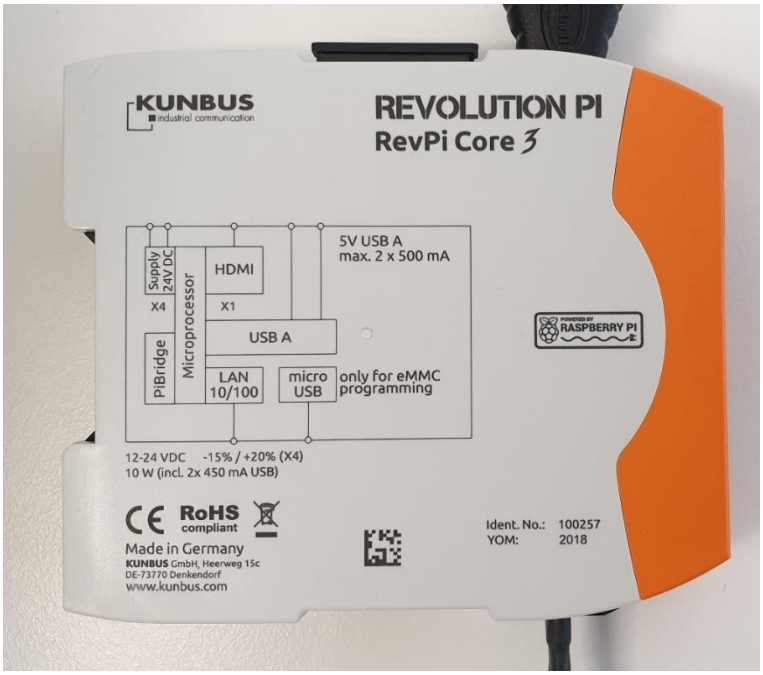
5.2 **Reuna ja sen laitteet ja ohjelmat**

Kuvassa 14 on esitetty järjestelmän rakentamisessa käytetyt laitteet. Kuvassa alimmaisena on RevPi Core 3-tietokone ja sen yläpuolella on reititin, sekä virtalähde ja muuntaja. Muuntajan tarve vaihtelee tapauskohtaisesti riippuen siitä, minne kokoonpano on asennettu, ja käytettävästä jännitteestä. Kyseinen kokoonpano on rakennettu verkkovirran muuntamiseksi sopivalle jännitteelle. Kuvassa vasemmalla alhaalla näkyvä musta kiekko on GPS-antenni. Se on kiinnitetty suoraan reitittimeen.



Kuva 14. Kuvassa on esitetty kokoonpano, jota työssä käytettiin.

Kuvassa 15 on Edge-laitteena käytetty Kunbuksen valmistama RevPi Core 3-tietokone. Se on rakennettu Raspberry Pi:n ympärille, ja se on suunniteltu teollisuuskäyttöön.



Kuva 15. Kunbus RevPi Core 3- tietokone.

Valmistajan mukaan RevPi Core 3:n ominaisuuksiin kuuluu neljä ytiminen Broadcom BCM2837-prosessori, 1,2 GHz:n kellotaajuus, 1 gigatavun RAM-muisti, 4 gigatavun eMMC flash-muisti, 4kV/8kV sähköstaattinen suojaus, 10,8 V-28,8 V käyttöjännite ja käyttölämpötila välillä -40 ^{o}C - +55 ^{o}C. Tietokoneessa on myös elektromagneettinen suojaus ympäröivien laitteiden aiheuttamilta vaikutuksilta. Valmistajan mukaan esimerkiksi hitsauskoneen vaikutukset on tältä osin simuloitu. (Kunbus 2019)

Työssä käytettiin Phoenix Contact mGuard RS4000 4G-reititintä. Se on teollisuuskäyttöön suunniteltu reititin, joka mahdollistaa 4G-internetyhteyden Edge-laitteella. Laite on muun muassa rakennettu kestämään lämpötilanvaihteluita, ja valmistaja lupaa sen toimivan -40 ^{o}C – 60 ^{o}C lämpötilassa. Siinä on sisäänrakennettu GPS-vastaanotin ja SNMP-palvelin. (Phoenix Contact 2019)

Reitittimen SNMP-palvelin tulee ottaa käyttöön selainpohjaisen hallintapaneelin kautta. Laite tarjoaa lisäksi VPN-yhteensopiva, joka helpottaa huomattavasti laitteen etähallintaa, koska tällöin ei ole tarvetta ulkoiselle IP-osoitteelle, vaan laitteeseen pääsee käsiksi SSH- ja VPN-yhteyksien avulla. Tarve etäyhteydelle voi syntyä tilanteesta jossa esim. IoT Edge-palvelu on käynnistettävä, asennettava uudelleen, tai laitteen asetuksiin on tehtävä sellaisia muutoksia, joiden tekeminen ei ole mahdollista IoT Hubin kautta.

5.2.1 Edge-laitteen ohjelmistot

Jotta tietokonetta voidaan pitää Edge-laitteena tässä yhteydessä, tulee siihen asentaa tarkoituksenmukaiset ohjelmistot, joita ovat tässä tapauksessa Linux Debian ”Jessie” käyttöjärjestelmä, Moby-ohjelmisto, Python-runtime, Edge-runtime ja snmp-tietojen hakemiseen suunniteltu Edge-moduuli, sekä pysnmp-kirjasto. Edge-moduuli on kustomoitu ohjelma, joka on tehty tiettyä käyttötarkoitusta varten. Moduuli asennetaan IoT Hubin kautta, kun se on ensin ladattu Azuren Container Registryyn, josta sen voi IoT Hubin avulla hakea. Moby-ohjelma on Dockerin kehittämä ohjelma, jota käytetään Docker konttien orkestrointiin. Paikkatiedon keräämistä varten on ohjelmoitu moduuli, joka hakee paikkatiedot SNMP-protokollan avulla reitittimen sisäänrakennetusta GPS-vastaanottimesta. Edge Runtime asennetaan kirjautumalla laitteelle SSH-yhteyden avulla, ja käyttäen Linuxin komentorivityökalua. Asennuskomennot on esitetty taulukossa 5.

Taulukko 5. Ohjelmistojen asennuskomennot (Microsoft 2019b).

Asennettava ohjelma	Asennuskomennot
Moby Runtime	# Download and install the moby-engine curl -L https://aka.ms/moby-engine-armhf-latest -o moby_engine.deb && sudo dpkg -i ./moby_engine.deb # Download and install the moby-cli curl -L https://aka.ms/moby-cli-armhf-latest -o moby_cli.deb && sudo dpkg -i ./moby_cli.deb # Run apt-get fix sudo apt-get install -f
IoT Edge Security Daemon	# Download and install the standard libiothsm implementation curl -L https://aka.ms/libiothsm-std-linux-armhf-latest -o libiothsm-std.deb && sudo dpkg -i ./libiothsm-std.deb # Download and install the IoT Edge Security Daemon curl -L https://aka.ms/iotedged-linux-armhf-latest -o iotedge.deb && sudo dpkg -i ./iotedge.deb # Run apt-get fix sudo apt-get install -f
Python-runtime	sudo apt-get install python3 python3-pip
pysnmp-kirjasto	pip install pysnmp

5.2.2 Asetukset laitteessa

Kirjaudutaan root-käyttäjänä Edge-laitteeseen SSH-yhteyden ja Bash-komentorivityökalun avulla. Siirrytään tiedoston /etc/iotedge/congif.yaml kohtaan ”provisioning”. Tiedostoon kirjaudutaan root-käyttäjänä ja Nano-kirjoittimen avulla komennolla ”sudo nano /etc/iotedge/config.yaml”. Tiedosto sisältää IoT Edge-asetukset, kuten sen mihin IoT Hubiin laite on yhdistetty ja mikä on IoT Hubin Connection String. Connection String sisältää IoT Hubin nimen, jonne laite on kytketty, laitetunnuksen (device_id) ja jaetun kirjautumisavaimen (shared_access_key). Reitittimen perus-, verkkoasetukset tulee tarkastaa ja muuttaa tarvittaessa ympäristöön sopivaksi.

5.3 **Moduulien ohjelmointi**

Luotaessa uutta Edge-projektia, määritetään Azuren Docker Image Repository, jonne ohjelma luodaan. Edge-projektin ohjelmointiin on tässä työssä käytetty Microsoftin Visual Studio Code-ohjelmointiympäristöä (myöhemmin VS Code). Kun VS Codea käytetään Edge-projektin ohjelmointiin, on siihen asennettava joukko laajennuksia (Extensions). Ne ovat Azure IoT Edge, Azure IoT Hub Toolkit, Python ja Docker. Python asennetaan tässä tapauksessa, koska projekti ohjelmoidaan Python-kielen avulla. Toisella kielellä ohjelmoitaessa on vastaavasti käytettävä kyseisen kielen laajennusta. Edge-sovellusten kehittäminen on mahdollista seuraavilla ohjelmointikielillä: C, C#, Node.js, Java ja Python. Docker laajennusta tarvitaan konttien kääntämiseen lähdekoodista. Azure IoT Edge sisältää toiminnallisuudet, joiden avulla Edge-projekti voidaan luoda, ohjelmoida ja julkaista laitteille. Laitehallinta ja julkaiseminen edellyttävät lisäksi laajennusta Azure IoT Hub Toolkitille, koska laajennus sisältää toiminnallisuudet IoT Hubin hallintaan, ja moduulien luontiin ja hallintaan. Azuren lisäosilla on lisäksi mahdollista simuloida laitteen toimintaa rakennetun moduulin avulla, mutta opinnäytetyön kirjoittamisen aikana on raportoitu, ettei ominaisuus toimi Python-kielen kanssa. Laajennukset voi ladata suoraan VS Codesta reittiä View -> Extensions tai Microsoftin Marketplacesta.

Ohjelmointiprojekti perustetaan seuraavasti: Mennään VS Coden valikkoon View ja valitaan Command Palette…. Kirjoitetaan avautuneeseen komentorivityökaluun komento ”Azure IoT Edge: New IoT Edge Solution.” Ohjelma pyytää avaamaan kansion, jonne projekti tallennetaan paikallisesti. Tämän jälkeen projektille annetaan nimi, valitaan ohjelmointikieli ja syötetään moduulille nimi. Sen jälkeen valitaan Docker Image Repository, jonne käännetyt Docker Imaget julkaistaan. Sitä varten Azuressa pitää olla Container Registry-palvelu perustettuna. Osoite on muotoa <containerregistryname.azurecr.io/modulename>. Tämän jälkeen VS Code perustaa projektin ja pyytää asettamaan env-tiedostoon Container Registryn käyttäjätunnuksen ja salasanan.

Kun VS Code on luonut ohjelman, sisältää se valmiina main.py-moduulin, jossa on valmiina metodeja viestien välittämiseksi IoT Hubiin. Moduuliin

on sisällytetty (import) kirjastomoduuli iothub_client, josta on haettu metodit IoTHubModuleClient, IoTHubClientError, IoTHubTransportProvider, IoTHubMessage, IoTHubMessageDispositionResult ja IoTHubError. Moduulissa on myös asetettu valmiiksi yhteysprotokolla, joka on työtä kirjoitettaessa MQTT.

Sijaintitiedon hakemiseen käytetään pysnmp-kirjastoa, jonka avulla sijaintitiedot saadaan haettua reitittimestä. Hakemiseen tarvitaan OID-yksilöintitunnus, jonka arvoa kysytään SNMP-protokollan avulla. Sen avulla voidaan hakea reitittimeltä koordinaatit nykyiseltä sijainnilta. Alla on snmpget-metodi, jota käytetään SNMP-kyselyyn (JCutrer 2015).

```
def snmpget(self, oid):
        cmdGen = cmdgen.CommandGenerator()
        errorIndication, errorStatus, errorIndex, varBinds =
cmdGen.getCmd(
           cmdgen.CommunityData(SNMP_COMMUNITY),
           cmdgen.UdpTransportTarget((SNMP_HOST, SNMP_PORT)),
           oid
        )

        # Check for errors and print out results
        if errorIndication:
           print(errorIndication)
        else:
           if errorStatus:
              print('%s at %s' % (
                 errorStatus.prettyPrint(),
                 errorIndex and varBinds[int(errorIndex)-1] or '?'
                 )
                 )
           else:
              for name, val in varBinds:
                 #print('3: ','%s = %s' % (name.prettyPrint(), val.prettyPrint()))
                 return val
```

Metodi hakee sille annetun OID-tunnuksen perusteella reitittimeltä sitä vastaavan arvon ja palauttaa sen. Metodi vaatii toimiakseen parametrit SNMP_COMMUNITY, SNMP_HOST ja SNMP_PORT, jotka on määriteltävä. SNMP_HOST on sen laitteen IP-osoite, jonne kysely osoitetaan. SNMP_PORT on SNMP-protokollan käyttämä portti. SNMP_COMMUNITY vastaa salasanaa tai käyttäjä ID:tä. snmpget-metodia kutsutaan get_coordinates-metodin sisältä ja sille annetaan parametreinä OID-tunnus. Ennen haettujen arvojen palautuksia tiedot muunnetaan GEO-JSON muotoon, jossa laitteelle annetaan myös yksilöivä nimi.

```
def get_coordinates(self):
    latitude = self.snmpget(LATITUDE)
```

```
        longitude = self.snmpget(LONGITUDE)
        geo_format = GEO_JSON % (longitude, latitude)
        return geo_format

GEO_JSON = {
  "type":"Feature",
  "geometry":{
    "type":"Point",
    "coordinates":{
      "longitude":%.5f,
      "latitude":%.5f
    }
  },
  "properties":{
    "name": " "
  }
}
```

snmp_request-metodin tehtävänä on kutsua get_coordinates-metodia ja muotilla metodikutsun vastaus viestiksi, joka voidaan lähettää IoT Hubille. Metodi sisältää myös metodikutsun self.client.send_event_async-metodille, joka huolehtii viestin lähettämisestä.

```
def snmp_request():
   location = None

   while True:
      if SNMP is not None:
         current_location = SNMP.get_coordinates()
         if current_location is not location:
            try:
               message = IoTHubMessage(current_location)

               # Send the message.
               print( "Sending message: %s" % message.get_string() )
               self.client.send_event_async(message,      self.send_confirmation_callback, None)
               location = current_location

            except IoTHubError as iothub_error:
               print ( "Unexpected error %s from IoTHub" % iothub_error )
               return
```

Lähdekoodin metodit on esitetty liitteessä 1.

5.4 **Tiedonkulku Azuren resurssien välillä**

Tässä case-tutkimuksessa on rakennettu tilanne, jossa IoT Hubiin tuleva tieto ohjataan Stream Analyticsista käsin Data Lake tallennuspaikkaan,

sekä suoraan Stream Analyticsista Power Bi:hin, josta se esitetään kartan avulla. Stream Analytics on järjestelmän keskiössä, ja sen avulla hallitaan tietovirtoja (input ja output).

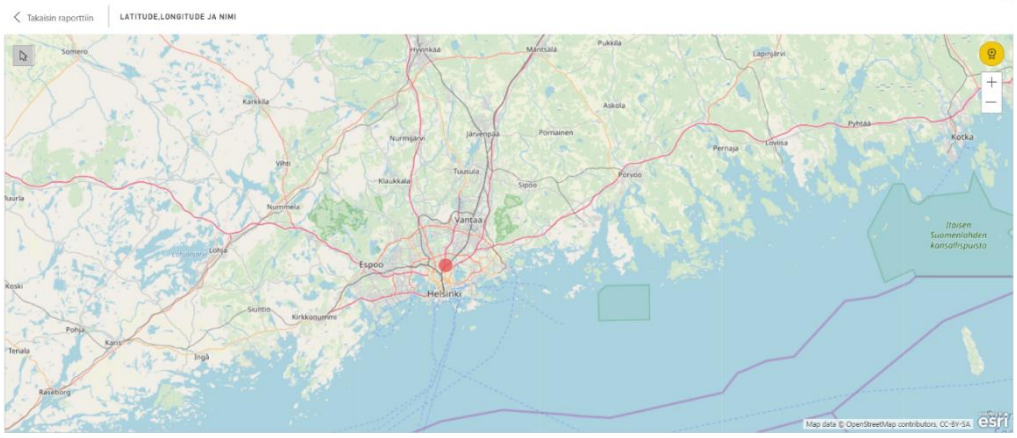
Stream Analytics (myöhemmin SA) on IoT Hubin kanssa IoT-putken keskeisin resurssi. Se reitittää ja ohjaa dataa IoT Hubin ja muiden komponenttien välillä. Kuten kuvassa 13 esitettiin, SA poimii tiedon IoT Hubista, ja ohjaa sen uudelleen Power Bi:lle ja Datalake tietovarastoon. SA mahdollistaa myös tiedon muokkauksen, ja eri tietolähteiden yhdistämisen. Tietovirtaa voidaan myös muokata Transact-SQL-ohjelmointikielen (myöhemmin T-SQL) avulla, joka on toteutus SQL-kielestä. Monet SA:n keskeisistä toiminnoista, kuten tietolähteiden yhdistäminen, sarakkeiden uudelleen nimeäminen, tietovirran reitittäminen ja valinnat tapahtuvat T-SQL-kielen avulla. Tässä case-esimerkissä T-SQL-kieltä käytettiin tiedon poimimiseen IoT Hubista, ja sen uudelleen ohjaamiseen tietovarastoon ja sovellukselle. IoT Hubilta tuleva tietovirta on GeoJSON-muodossa, ja jotta se voidaan esittää Power Bi:n ArcGIS Maps for Power Bi-palvelussa, on sitä ensi muokattava sopivaan muotoon. Karttapalvelu vaatii pituus- ja leveyskoordinaatin olevan samassa sarakkeessa, kun taas GeoJSON-muodossa ne on sijoitettu eri sarakkeisiin. Yhdistäminen tehdään T-SQL-kielen avulla Query-välilehdellä.

```
SELECT
    concat(Geometry.Coordinates.latitude,', ', Geometry.Coor-
    dinates.longitude) AS location,
    Properties.Name AS name,
    System.Timestamp AS time
 INTO
    <output source> #Power Bi
 FROM
    <input source> #IoT Hub
SELECT
    Geometry.Coordinates.latitude AS latitude,
    Geometry.Coordinates.longitude AS longitude,
    Properties.Name AS name,
    System.Timestamp AS time
 INTO
    <output source> #Data Lake Store
 FROM
    <input source> #IoT Hub
```

5.5 **Sovellus**

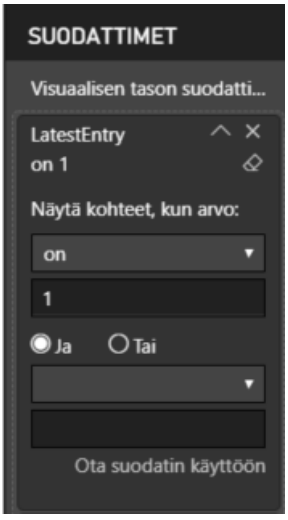
Sijainnin esittäminen on toteutettu ArcGIS Maps for Power Bi -karttapalvelun avulla. Power Bi on Microsoftin tarjoama työkalu tiedon graafista esittämistä varten. Azuren Stream Analytics-palvelussa hallitaan Output-pisteitä, joihin Stream Analytics syöttää valittuja tietoja. Power Bi:n voi määrittää Outputiksi, jolloin tietovirta tulee näkyviin tietomallina Power

Bi:ssä. Power Bi tukee myös suoratoistoa, jolloin sijaintitieto voidaan esittää kartalla reaaliaikaisesti. Tiedon muokkaaminen tapahtuu Power Bi Desktop -ohjelman avulla, josta se julkaistaan valittuun työtilaan raporttina. Henkilöt, joilla on oikeudet kyseiseen työtilaan, voivat nähdä raportin Power Bi:n web-versiossa. Kuvassa 16 on esitetty valmis karttanäkymä, jossa laitteen sijainti näkyy pisteenä.



Kuva 16. Power Bi:n karttanäkymä.

Power Bi ei automaattisesti näytä pelkästään viimeisintä sijaintia kartalla, vaan palveluun tulevaa tietoa on ensin muokattava mallintamalla. Ilman mallintamista kartalla näkyvät kohteen kaikki eri sijaintipisteet, jotka paikannin on lähettänyt. Mallintaminen on toiminto, jossa tietosyötteen Power Bi:lle tulevia tietoja voidaan suodattaa Data Analysis Expression (DAX)-kielen avulla. Laitteen viimeisin sijainti saadaan näkyviin seuraavalla kaavalla: *LatestEntry = IF(MAX('coordinates'[time]) = CALCULATE(MAX('coordinates'[time]); ALLEXCEPT(coordinates; coordinates[name])); 1; 0).* Lisäksi suodattimen arvoksi tulee asettaa yksi, koska kaava luo uuden sarakkeen tietolähteen sarakkeiden rinnalle ja asettaa ehdot täyttävän rivin arvoksi numeron yksi. Kuvassa 17 on esitetty arvon asettaminen suodattimeen.



Kuva 17. Suodattimen arvon asettaminen Power Bi:ssä.

5.6 **Tietojen varastointi**

Tässä IoT-ratkaisussa päädyttiin siihen, että tiedot tallennetaan Data Lake-tietokantaan, koska ei ole tarvetta tiedon nopealle jatkokäsittelylle. Stream Analytics palvelusta tieto voidaan suoraan ohjata Power Bi:n karttasovellukseen. Tietovarastoon tallennettaville parametreille annetaan nimet, jotta ne voidaan tunnistaa jälkikäteen. Tallennettavia tietoja ovat korkeus- ja leveysasteet, laitteen nimi ja kellonaika.

6 POHDINTAA

Tässä on kerrottu työn aikana esiin tulleista huomioitavista asioista ja esitetty omaa pohdintaa työn aihepiiriin liittyen.

6.1 Ohjelmointikielen valinta

SNMP-moduuli oli aluksi tarkoitus ohjelmoida C#-kielellä. Siinä ongelmaksi muodostui SNMP-kirjastomoduulien yhteensopimattomuus Edge-projektien kanssa. Projektityyppi ei tunnistanut käytettävää kirjastomoduulia. Vastaava kirjastomoduuli toimi kuitenkin moitteettomasti Console App-ohjelmassa. Tästä syystä ohjelmointikieleksi vaihtui Python. Pythonille oli tarjolla vastaavia moduulikirjastoja, ja ne osoittautuivat yhteensopiviksi Edge-projektin kanssa.

6.2 Saman ohjelmakoodin käyttäminen eri laitteissa

Käytettävää ohjelmaa olisi mahdollisuuksien mukaan kyettävä monistamaan ja käyttämään useassa eri laitteessa ilman, että ohjelmaan on tarvetta tehdä muutoksia. Tämän toteuttamiseksi on otettava huomioon muutamia tekijöitä. SNMP-protokollan käyttö perustuu yksilöivään OID-tunnukseen. Jotta samaa ohjelmakoodia voidaan käyttää useammassa laitteessa, on edellä mainittu OID-tunnus saatava selville kyselemällä sitä laitteesta sen selvittämiseksi. Tällainen ominaisuus on rakennettava ohjelmakoodiin, jotta saman koodin käyttö muissakin laitteissa olisi mahdollista. Muussa tapauksessa kontit pitää ainakin tältä osin ohjelmoida laitekohtaisesti. Mikäli käytettävä reititinmalli on sama, niin silloin OID-tunnuksen vaihtuminen ei pitäisi olla ongelmana. Kontteihin pakattava ohjelmakoodi tuo mukanaan monia etuja, kuten sen, että koodi voidaan hakea Azuren Identity Registrystä, ja käyttää monissa laitteissa ilman uudelleen ohjelmoinnin tarvetta. Laitteet pitää ensin rekisteröidä IoT Hubiin, ja laitteilla on oltava Docker-runtime tai jokin muu vastaava ohjelma asennettuna, jotta kontteja voidaan käyttää.

6.3 Tietokantojen käyttö

Käytettävä tietokanta riippuu hyvin pitkälti siitä mihin tietoa halutaan hyödyntää, ja miten nopeasti sen on oltava saatavilla. Jos tieto on saatava käsittelyyn pienellä viiveellä, niin silloin tietokannan on kyettävä nopeaan tiedonkäsittelyyn. Kerättävien tietojen rakenteet vaikuttavat oleellisesti siihen millaista tietokantaa on käytettävä. SQL-tietokantoja käytettäessä datan pitää olla tarkkaan etukäteen määritettyä. Esimerkiksi sarakkeiden tietotyypit pitää tietää ennalta. NoSQL-tyyppisessä tietokannassa, kuten Data Lake- tai Cosmos DB-tietokannoissa, tietokantoihin syötettävien tietojen ei tarvitse olla etukäteen määritettyjä. Se mahdollistaa tietokantojen skaa-

lautuvuuden tarpeen vaatiessa. Se on esineiden internetin ja big datan näkökulmasta etu, koska aina ei voida olla varma mitä kaikkea kantaan halutaan tallentaa. Tarpeisiin voi myös tulla muutoksia tuotteen elinkaaren aikana.

Myös tietokantojen nopeuksissa on eroja. Cosmos DB sopii hyvin sellaisen datan käsittelyyn, jossa data on saatava nopeasti hyödynnettäväksi. Sitä voidaan käyttää sovellusten tietovarastona. Vastaavasti Data Lake sopii paremmin tietojen pidempi aikaiseen tallentamiseen. Cosmos DB on myös mahdollista hajauttaa maantieteellisesti, jolloin sen käyttö on nopeampaa globaaleissa sovelluksissa.

6.4 **Pohdintaa IoT-alustojen ominaisuuksista**

IoT-alustan rooli IoT-ratkaisujen toteutettavuuden ja yhteensopivuuden näkökulmasta on todella tärkeä. Informaatioteknologia on ala, joka muuttuu nopealla syklillä teknologian kehityksen seurauksena. Muutosnopeudesta johtuen on tärkeää, että valitun alustan toimittajana on organisaatio, joka jatkuvasti kehittää alustaa ja ottaa uusia teknologioita käyttöön. Itse nostaisin alustan kehitystyön yhdeksi hyvän IoT-alustan vaatimukseksi muiden vaatimusten rinnalle. Kehitystyö ei kuitenkaan ole tekninen ominaisuus, vaan siinä ennemminkin arvioidaan alustan toimittajaa yrityksenä.

7 YHTEENVETO

Tässä työssä tutkittiin IoT-alustojen ominaisuuksia, ja rakennettiin Microsoft Azuren IoT-arkkitehtuuria hyödyntävä toteutus, jonka tarkoituksena on paikkatiedon kerääminen ja käsittely Azuren IoT-komponenttien avulla. Työ pyrki vastaamaan siihen, miten Azuren avulla on mahdollista toteuttaa IoT-ratkaisu, ja miten Azure täyttää alustoille asetetut vaatimukset. Työssä selvitettiin myös, millaisia vaatimuksia hyvälle IoT-alustalle voidaan asettaa.

Työssä saatiin selville mitä ominaisuuksia hyvältä IoT-alustalta voidaan edellyttää. Kriteerien avulla alustan valinta selkeytyy, ja eri alustojen vertailu helpottuu. Kuten aiemmin mainittiin, niin IoT-alustoja voi olla monenlaisia, ja yksinkertaisiakin alustoja markkinoidaan IoT-alustoina. Alustan valinnassa voidaan päätyä virheelliseen valintaan, ellei ensin tunneta alustojen kriteerejä ja ominaisuuksia tarpeeksi hyvin.

Kappaleessa 3.2 määritettiin kriteerit edistyneelle IoT-alustalle. Ne ovat liitettävyys ja normalisointi, laitehallinta, prosessoinnin ja toimintojen hallinta, datan visualisointi, analytiikka, lisätyökalut, ulkoiset rajapinnat ja tietokantapalvelut. Näiden kriteerien valossa Microsoft Azurea voidaan pitää edistyneenä IoT-alustana. Azure täyttää kaikki kehittyneeltä alustalta vaadittavat kriteerit, ja se sisältää kattavan valikoiman erilaisia työkaluja tiedon analysointiin ja käsittelyyn. Azure sisältää yhdyskäytävät laitteiden liitettävyydelle, ja kehittyneen laitehallinnan, jonka avulla onnistuu suurenkin laitemäärän hallitseminen. Azure tarjoaa myös DevOps-ympäristön ja CI/CD-putken nopeuttamaan kehitystyötä. Analytiikan näkökulmasta Azure sisältää tarpeellisia työkaluja tietojen analysointiin. Työkaluista mainittakoon Azure Machine Learning, Azure Databricks ja HDInsight. Azure mahdollistaa koneoppimisen suorittamisen myös reunalaitteissa, jolloin tulokset ovat nopeammin käytettävissä. Azure tarjoaa myös ulkoisia rajapintoja, joiden avulla integraatiot ulkoisten sovellusten kanssa ovat mahdollisia. Tietojen esittäminen on mahdollista Power Bi-sovelluksen avulla. Azuressa on myös sisäänrakennettu Azure Maps-sovellus, jonka avulla voidaan rakentaa karttapohjaisia palveluja.

Työn yhtenä tavoitteena oli rakentaa Azuren avulla IoT-toteutus. Työn case-tutkimuksen tarkoituksena oli paikkatietosovelluksen rakentaminen. Siinä onnistuttiin ja työn tavoite täyttyi siltäkin osin. Azure tarjoaa riittävän kattavan valikoiman työkaluja sovelluksen tekemiseksi.

Työ oli opettavainen, ja se tarjosi mahdollisuuden oppia esineiden internetistä, pilvipalveluista, ja niiden avulla toteutettavista sovelluksista. IoT-projekti on monitahoinen. Siinä on otettava huomioon monia eri asioita. Projektin toteutukseen vaikuttaa myös fyysinen ympäristö, minne järjestelmä rakennetaan. On aivan eri asia toteuttaa projekti louheen murskauslaitokseen tai kaivinkoneeseen, kuin asuinrakennuksen sisätiloihin. Ympäristö

asettaa fyysisille laitteille aivan erilaiset vaatimukset. Murskaimeen ja siinä oleviin sensoreihin, jotka ovat ulkotiloissa 15 asteen pakkasessa, kohdistuu erilaiset kestävyysvaatimukset kuin huoneenlämmössä oleviin laitteisiin. Puhumattakaan laitteista, jotka ovat kemiallisesti aggressiivisessa ympäristössä. Tietojenkäsittelyn näkökulmasta IoT-sovellusten eroavaisuuksia muodostuu esimerkiksi käytettävistä protokollista, tietovarastoista, ja siitä minkälaista analytiikkaa halutaan hyödynnettävän. Myös käytettävissä arkkitehtuureissa on eroja. Kun halutaan laajentaa IoT-sovellusta, tulee keskitetyn arkkitehtuurin rajat nopeasti vastaan. Silloin joudutaan laajentamaan ratkaisua hajautetun arkkitehtuurin suuntaan.

LÄHTEET

Cisco (2014). The Internet of Things Reference Model. Haettu: 12.2.2019 osoitteesta http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf

Cloud Security Alliance (2017). The Treacherous 12 Top Threats to Cloud Computing + Industry Insights. Haettu 6.2.2019 osoitteesta https://downloads.cloudsecurityalliance.org/assets/research/top-threats/treacherous-12-top-threats.pdf

Collin J. & Saarelainen A. (2016). Teollinen internet. Alma Talent Oy.

Dargie w. & Poellabauer C. (2010). Fundamentals of Wireless Sensor Networks: Theory and Practice. Chichester, Iso-Britannia: John Wiley & Sons.

Docker (2019). What is a Container? A standardized unit of software. Haettu 27.2.2019 osoitteesta: https://www.docker.com/resources/what-container

Foulds I. Learn Azure in a Month of Lunches. Haettu 7.5.2019 osoitteesta https://azure.microsoft.com/mediahandler/files/resourcefiles/learn-azure-in-a-month-of-lunches/Learn_Azure_in_a_Month_of_Lunches.pdf

Girani S., Ferrari G., Picone M. & Veltri L. (2019). Internet of Things – Architecture, Protocals and Standards. Hoboken, USA: John Wiley & Sons

Gubbi J., Buyya S., Marusic S. & Palaniswami M. (2013). Future Generation Computer Systems - Internet of Things (IoT): A vision, architectural elements, and future directions. Haettu 19.2.2019 osoitteesta: http://www.buyya.com/papers/Internet-of-Things-Vision-Future2013.pdf

Heino P. (2010). *Pilvipalvelut.* Helsinki: Talentum Media Oy.

IoT Analytics GmbH (2015). IOT PLATFORMS - The central backbone for the Internet of Things. Haettu 17.4.2019 osoitteesta http://iot-analytics.com/wp/wp-content/uploads/2016/01/White-paper-IoT-platforms-The-central-backbone-for-the-Internet-of-Things-Nov-2015-vfi5.pdf

JCutrer (2015). Python Tutorial: Get Temperature and other environmental data from a Dell PowerEdge Server using pysnmp. Haettu 13.3.2019 osoitteesta https://jcutrer.com/python/python-tutorial-query-dell-poweredge-temperature-snmp-data

Kamal R. (2017). Internet of Things – Architecture and Design Principles. Chennai, Intia: McGraw Hill Education

Kunbus (2019). Kunbus RevPi Core 3. Haettu: 26.2.2019 osoitteesta https://revolution.kunbus.com/revpi-core/

Microsoft (2018a). Azure IoT Edge Runtime. Haettu: 16.1.2019 osoitteesta https://docs.microsoft.com/en-us/azure/iot-edge/iot-edge-runtime

Microsoft (2019a). Azure IoT Subsystems. Haettu 29.5.2019 osoitteesta https://azure.microsoft.com/en-us/solutions/architecture/azure-iot-subsystems/

Microsoft (2019b). Install Azure IoT Edge runtime on Linux (ARM32v7/armhf). Haettu 14.5.2019 osoitteesta https://docs.microsoft.com/en-us/azure/iot-edge/how-to-install-iot-edge-linux-arm

Microsoft (2018b). IoT Edge. Haettu: 15.1.2019 osoitteesta https://docs.microsoft.com/en-us/azure/iot-edge/about-iot-edge

Microsoft (2018c). IoT Hub. Haettu: 4.2.2019 osoitteesta https://docs.microsoft.com/en-us/azure/iot-hub/about-iot-hub

Microsoft (2018d). IoT Hub REST. Haettu: 6.3.2019 osoitteesta https://docs.microsoft.com/en-us/rest/api/iothub/

Microsoft (2019c). Introduction to Azure Blob storage. Haettu: 6.3.2019 osoitteesta https://docs.microsoft.com/fi-fi/azure/storage/blobs/storage-blobs-introduction.

Microsoft (2018e). Overview of device management with IoT Hub Haettu: 4.2.2019 osoitteesta: https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-management-overview

Microsoft (2018f). Understand and use device twins in IoT Hub. Haettu 4.2.2019 osoitteesta https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-device-twins

Microsoft (2018g). Understand and use module twins in IoT Hub. Haettu: 4.2.2019 osoitteesta https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-module-twins

Microsoft (2018h). Understand the identity registry in your IoT hub. Haettu: 4.2.2019 osoitteesta https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-identity-registry

Microsoft (2018i). Welcome to Azure Cosmos DB Haettu: 5.2.2019 osoitteesta https://docs.microsoft.com/en-us/azure/cosmos-db/introduction

Microsoft (2019d). What is Azure Data Lake Storage Gen1? Haettu 23.5.2019 osoitteesta https://docs.microsoft.com/en-us/azure/data-lake-store/data-lake-store-overview

Microsoft (2019e) What is Azure Databricks? Haettu 29.5.2019 osoitteesta https://docs.microsoft.com/en-us/azure/azure-databricks/what-is-azure-databricks

Microsoft (2019f). What is Azure HDInsight and the Apache Hadoop technology stack. Haettu 5.3.2019 osoitteesta https://docs.microsoft.com/en-us/azure/hdinsight/hadoop/apache-hadoop-introduction

Microsoft (2018j). What is Azure SignalR Service. Haettu: 6.3.2019 osoitteesta https://docs.microsoft.com/en-us/azure/azure-signalr/signalr-overview

Microsoft (2018k). What is Azure Stream Analytics? Haettu: 4.2.2019 osoitteesta https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction

Murugesan S. & Bojanova I. (2016). Encyclopedia of Cloud Computing. Chichester, Iso-Britannia: John Wiley & Sons.

Phoenix Contact (2019). Security Appliance - TC MGUARD RS4000 4G VPN – 2903586. Haettu 23.5.2019 osoitteesta https://www.phoenixcontact.com/online/portal/fi?uri=pxc-oc-itemdetail:pid=2903586&library=fifi&tab=1

Salo I. (2014). *Big Data & Pilvipalvelut.* Jyväskylä: Docendo Oy

SNMP Labs (2018). SNMP library for Python. Haettu 13.3.2019 osoitteesta http://snmplabs.com/pysnmp/docs/snmp-history.html

Liite 1 LÄHDEKOODI

```
# Copyright (c) Microsoft. All rights reserved.
# Licensed under the MIT license. See LICENSE file in the project root for
# full license information.
#main.py

import random
import time
import sys
import iothub_client

from iothub_client import IoTHubModuleClient, IoTHubClientError, IoTHubTransportProvider
from iothub_client import IoTHubMessage, IoTHubMessageDispositionResult, IoTHubError

# messageTimeout - the maximum time in milliseconds until a message times out.
# The timeout period starts at IoTHubModuleClient.send_event_async.
# By default, messages do not expire.
MESSAGE_TIMEOUT = 10000

# global counters
RECEIVE_CALLBACKS = 0
SEND_CALLBACKS = 0
SNMP = snmp.useSNMP()


# Choose HTTP, AMQP or MQTT as transport protocol.  Currently only MQTT is supported.
PROTOCOL = IoTHubTransportProvider.MQTT

# Callback received when the message that we're forwarding is processed.
def send_confirmation_callback(message, result, user_context):
    global SEND_CALLBACKS
    print ( "Confirmation[%d] received for message with result = %s" % (user_context, result) )
    map_properties = message.properties()
    key_value_pair = map_properties.get_internals()
    print ( "    Properties: %s" % key_value_pair )
    SEND_CALLBACKS += 1
    print ( "    Total calls confirmed: %d" % SEND_CALLBACKS )


# receive_message_callback is invoked when an incoming message arrives on the specified
```

```
# input queue (in the case of this sample, "input1").  Because this is a filter module,
# we will forward this message onto the "output1" queue.
def receive_message_callback(message, hubManager):
    global RECEIVE_CALLBACKS
    message_buffer = message.get_bytearray()
    size = len(message_buffer)
    print ( "      Data: <<<%s>>> & Size=%d" % (message_buffer[:size].decode('utf-8'), size) )
    map_properties = message.properties()
    key_value_pair = map_properties.get_internals()
    print ( "    Properties: %s" % key_value_pair )
    RECEIVE_CALLBACKS += 1
    print ( "    Total calls received: %d" % RECEIVE_CALLBACKS )
    hubManager.forward_event_to_output("output1", message, 0)
    return IoTHubMessageDispositionResult.ACCEPTED

def snmp_request():
    location = None

    while True:
        if SNMP is not None:
            current_location = SNMP.get_coordinates()
            if current_location is not location:
                try:
                    message = IoTHubMessage(current_location)

                    # Send the message.
                    print( "Sending message: %s" % message.get_string() )
                    self.client.send_event_async(message,      self.send_confirmation_callback, None)
                    location = current_location

                except IoTHubError as iothub_error:
                    print ( "Unexpected error %s from IoTHub" % iothub_error )
                    return

class HubManager(object):

    def __init__(
            self,
            protocol=IoTHubTransportProvider.MQTT):
        self.client_protocol = protocol
        self.client = IoTHubModuleClient()
        self.client.create_from_environment(protocol)

        # set the time until a message times out
        self.client.set_option("messageTimeout", MESSAGE_TIMEOUT)
```

```
        # sets the callback when a message arrives on "input1" queue.  Messages sent to
        # other inputs or to the default will be silently discarded.
        self.client.set_message_callback("input1", receive_message_callback, self)

    # Forwards the message received onto the next stage in the process.
    def forward_event_to_output(self, outputQueueName, event, send_context):
        self.client.send_event_async(
            outputQueueName, event, send_confirmation_callback, send_context)

def main(protocol):
    try:
        print ( "\nPython %s\n" % sys.version )
        print ( "IoT Hub Client for Python" )

        hub_manager = HubManager(protocol)

        print ( "Starting the IoT Hub Python sample using protocol %s..." % hub_manager.client_protocol )
        print ( "The sample is now waiting for messages and will indefinitely. Press Ctrl-C to exit. ")

        while True:
            snmp_request()
            time.sleep(5)

    except IoTHubError as iothub_error:
        print ( "Unexpected error %s from IoTHub" % iothub_error )
        return
    except KeyboardInterrupt:
        print ( "IoTHubModuleClient sample stopped" )

if __name__ == '__main__':
    main(PROTOCOL)


#snmp.py

from pysnmp.hlapi import *
from pysnmp import hlapi

GEO_JSON = "{\"type\":\"Feature\",\"geometry\":{\"type\":\"Point\",\"coordinates\":{\"longitude\":%.5f,\"latitude\":%.5f}},\"properties\":{\"name\":\"Device_X\"}}"
```

```
SNMP_HOST = '192.168.1.1'
SNMP_PORT ='161'
SNMP_COMMUNITY = 'public'
LATITUDE = '.1.3.6.1.4.1.15450.2.4.12.5.0'
LONGITUDE = '.1.3.6.1.4.1.15450.2.4.12.4.0'
CONNECTION_STRING = "<Add connection string here> "
PROTOCOL = IoTHubTransportProvider.MQTT

class useSNMP:
   client = None

   def __init__(self):
      # Create an IoT Hub client
      self.client = IoTHubClient(CONNECTION_STRING, PROTOCOL)

   def snmpget(self, oid):
         cmdGen = cmdgen.CommandGenerator()
         errorIndication, errorStatus, errorIndex, varBinds =
cmdGen.getCmd(
            cmdgen.CommunityData(SNMP_COMMUNITY),
            cmdgen.UdpTransportTarget((SNMP_HOST, SNMP_PORT)),
            oid
         )

         # Check for errors and print out results
         if errorIndication:
            print(errorIndication)
         else:
            if errorStatus:
               print('%s at %s' % (
                  errorStatus.prettyPrint(),
                  errorIndex and varBinds[int(errorIndex)-1] or '?'
                  )
                  )
            else:
               for name, val in varBinds:
                  #print('3: ','%s = %s' % (name.prettyPrint(), val.prettyPrint()))
                  return val

   def get_coordinates(self):
      latitude = self.snmpget(LATITUDE)
      longitude = self.snmpget(LONGITUDE)

      geo_format = GEO_JSON % (longitude, latitude)

      return geo_format
```