jamk.fi

# Liikennedatan keruu- ja analysointi-järjestelmä Microsoft Azuressa

Valtteri Ahonen

Opinnäytetyö
Maaliskuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), ohjelmistotekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

**Kuvailulehti**

| Tekijä(t) | Julkaisun laji | Päivämäärä |
|---|---|---|
| Ahonen, Valtteri | Opinnäytetyö, AMK | Maaliskuu 2017 |
| | Sivumäärä<br>30 | Julkaisun kieli<br>Suomi |
| | | Verkkojulkaisulupa myönnetty: x |

Työn nimi
**Liikennedatan keruu- ja analysointijärjestelmä Microsoft Azuressa**

Tutkinto-ohjelma
Ohjelmistotekniikka

Työn ohjaaja(t)
Esa Salmikangas

Toimeksiantaja(t)
Nodeon Finland Oy

Tiivistelmä

Pilvipalveluita pidetään kiinnostavana, sillä ne ovat itse ylläpidettäviin konesaleihin ja palvelimiin verrattuna joustavia ja kustannustehokkaita. Tätä silmällä pitäen toteutettiin älyliikenteen ratkaisuihin keskittyneelle Nodeonille testiohjelma, joka keräsi liikennedataa eri lähteistä Microsoft Azure -pilvipalveluun ja myös prosessoi dataa pilvessä. Tavoitteena oli selvittää pilvipalvelun mahdollisuuksia älyliikennekontekstissa tiedon keräämisen apuvälineenä.

Ratkaisu toteutettiin Microsoftin tekniikoita ja rajapintoja hyödyntäen C#-kielellä, sillä se sopii erityisen hyvin käytettäväksi Azure-pilvipalvelun kanssa. Dataa kerättiin Liikenneviraston Digitraffic-palvelun avoimesta rajapinnasta. JSON-muotoinen data ladattiin pilvipalvelussa itsenäisesti toimivalla ohjelmalla Azuren pilvitietokantaan, josta sitä voitiin hakea tarkasteltavaksi ASP.NET-pohjaiseen web-käyttöliittymään, joka isännöitiin niin ikään Azuren pilvessä.

Tuloksena syntyi yritykselle hyvä tietopohja siitä, mitä Azure tällä hetkellä tarjoaa, sekä demo-ohjelma, joka kerää oikeaa liikennedataa arkistoon tietokantaan. Demoa tehdessä selvitettiin, kuinka sovellus saadaan Azureen, mitä vaihtoehtoja sen toteuttamiseen on olemassa sekä kuinka paljon toteutus maksaa erilaisilla skaaloilla.

Avainsanat (asiasanat)
Liikenne, pilvipalvelut, Microsoft Azure

Muut tiedot

jamk.fi

**Description**

| Author(s) | Type of publication | Date |
|---|---|---|
| Ahonen, Valtteri | Bachelor's thesis | March 2017 |
| | | Language of publication: Finnish |
| | Number of pages: 30 | Permission for web publication: x |

Title of publication

**Traffic data collection and analyzing system in Microsoft Azure**

Degree programme

Software engineering

Supervisor(s)

Salmikangas Esa

Assigned by

Nodeon Finland Oy

Abstract

Cloud services are considered interesting because they are flexible and cost effective compared to traditional datacenters, which is why Nodeon, a company focused on smart traffic solutions, decided to construct a demo program that runs in the cloud collecting data from various sources and saving it into a cloud database. The data could then be browsed and analyzed with web user interface.

The solution was crafted using Microsoft's techniques and frameworks because they work well with Microsoft Azure cloud. The data was collected from Digitraffic service. Digitraffic is a service operated by the Finnish Transport Agency offering real time traffic information in JSON format for free. The data was downloaded to Azure Table storage with a piece of software running on Azure Cloud Service. The user interface was built with ASP.NET framework. The user interface was also hosted in the cloud.

As a result, the company acquired a great amount of valuable information about what the cloud can nowadays offer and also a demo program that could be easily refined to be actually used in the field to gather data. Interesting knowledge was also gained about the costs of cloud computing on different scales.

Keywords/tags (subjects)

Traffic, cloud, cloud services, Microsoft Azure

Miscellaneous

# Sisältö

**Kuviot**



**Taulukot**

# 1 Lähtökohdat

## 1.1 Pilvipalvelut ja niiden hyödyntäminen

Esineiden internet, teollinen internet ja pilvipalvelut ovat moderneja muotisanoja, jotka enteilevät suuria muutoksia IT-alalle ja sen myötä myös monelle muulle toimialueelle. Tällä hetkellä pilvipalvelut kehittyvät erittäin nopeasti, ja siksi on tärkeää kerätä tuoretta tietoa ja kokemusta yrityksen tekniikkavalintojen tueksi.

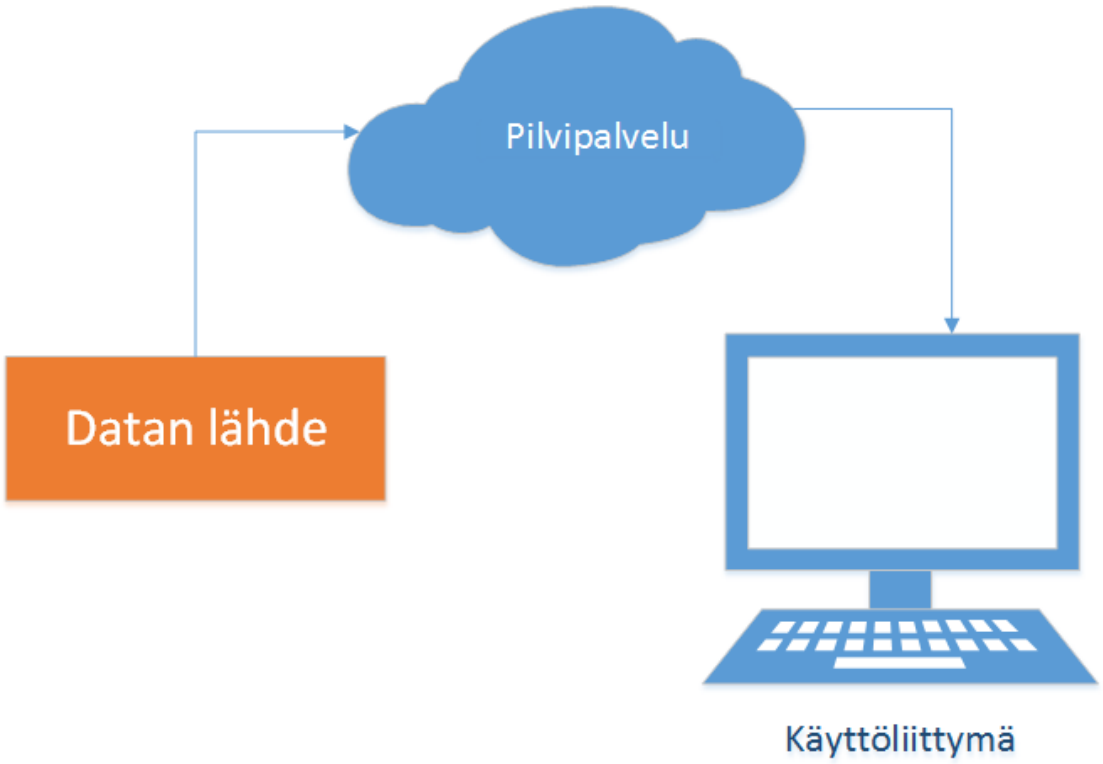
Tietokantojen ja laskennan siirtyminen pilvipalveluihin houkuttelee monia yrityksiä, sillä pilviympäristössä toimiminen on kustannustehokasta verrattuna omaan palvelimeen tai konesaliin. Pilvestä ostettu kapasiteetti on joustavaa, ja resurssit (ja sen myötä kustannukset) voidaan säätää nopeastikin tarpeiden muuttuessa. Pilvipalvelut eroavat myös web-hotelleista ja virtuaalipalvelimista juuri muun muassa käytöstä maksamisen ja skaalautuvuuden osalta. (Hafren 2014)

## 1.2 Toimeksiantaja

Opinnäytetyön toimeksiantajana on vuonna 2005 perustettu, älyliikenteen ja teollisen internetin liiketoiminta-alueella toimiva Nodeon Finland Oy. Tällä hetkellä (vuonna 2016) noin 30 ihmistä työllistävä yritys on ollut mukana esimerkiksi suurissa tunneliprojekteissa, muun muassa Tampereen Rantatunnelissa. Maantieteellisesti yrityksen toiminta sijoittuu pääasiassa Suomeen ja Ruotsiin. (Nodeon lyhyesti n.d.)

## 1.3 Tavoite ja vaatimukset

Opinnäyteyön tarkoituksena oli selvittää Microsoft Azure -pilvipalveluiden tällä hetkellä tarjoamia mahdollisuuksia älyliikenteen toimintakentällä. Erityisesti aiheena oli yksinkertainen datan keruu erilaisista rajapinnoista verkon yli sekä datan prosessoiminen ja tallentaminen pilvessä. Havainnekuva ideasta on esitetty kuviossa 1.




Kuvio 1. Havainnekuva projektin aiheesta

Tarkoituksena oli kirjoittaa pieni demo-ohjelma, joka hoitaa keruun ja tallennuksen. Opinnäytetyön puitteissa oli myös tarkoitus rakentaa niin ikään Azuressa toimiva, perustoiminnot kattava verkkokäyttöliittymä, jonka avulla tallennettua dataa voidaan selailla. Tämän tuotoksen sekä kehityksen aikana tehtyjen havaintojen pohjalta voitiin arvioida Azuren sopivuutta yrityksen tuleviin ratkaisuihin.

## 1.4 Valitut tekniikat ja valintaperusteet

### 1.4.1 Azure

Pilvialustaksi valikoitui toimeksiantajan pyynnöstä Microsoft Azure. Yrityksellä oli muutoinkin käytössään Microsoftin ohjelmistoja, joten saman valmistajan ratkaisu oli luonteva valinta.

Azure on myös yksi tämän hetken monipuolisimmista pilvialustoista ja tarjoaa kaiken opinnäytetyön vaatiman toiminnallisuuden (Azure Solutions n.d.). Microsoftin pilvialusta on myös ensimmäinen palvelu, joka on läpäissyt EU:n tietoturvatarkastelun ja

saanut siitä tunnustuksen. Tämä antaa yritykselle uskallusta luottaa alustan tulevaisuuteen. (Privacy authorities across Europe approve Microsoft's cloud commitments 2014.)

### 1.4.2 Visual Studio

Valitun pilvipalvelun myötä kehitysympäristöksi järkevä valinta oli Visual Studio: kehitysympäristöstä voi suoraan julkaista koodia Azureen. Visual Studiossa oli myös valmiita koodipohjia Azure-kehitystä varten. Koska Visual Studio -ohjelmisto on Azuren tapaan Microsoftin kehittämä, voidaan luottaa sen olevan yksi parhaista työkaluista Azure-kehitykseen myös tulevaisuudessa. Näiden lisäksi niin toimeksiantajalla kuin tekijälläkin oli Visual Studiosta aiempaa kokemusta.

### 1.4.3 ASP.NET

Lopputuotteelle haluttiin toteuttaa yksinkertainen käyttöliittymä. Nykytrendien mukaan kallistuttiin web-käyttöliittymän suuntaan, ja edellä mainittujen valintojen pohjalta järkevä toteutustekniikka on Microsoftin ASP.NET.

ASP.NET on Microsoftin avoimena lähdekoodina kehittämä ohjelmistokomponenttikirjasto web-sovellusten kehittämiseen. Kirjaston avulla voi hyödyntää C#-ohjelmointia sekä esimerkiksi MVC-mallia helposti HTML5-, JavaScript- ja CSS-tekniikoiden kanssa. (ASP.NET, n.d.)

### 1.4.4 Datarajapinnat

Liikennevirasto tarjoaa avointa liikennetietodataa Digitraffic-palvelun rajapinnan kautta. Tämä koettiin toimeksiantajan puolesta mielekkääksi ottaa mukaan. Lisäksi tarkasteltiin mahdollisuutta kerätä dataa suoraan toimeksiantajan tietyiltä mittalaitteilta, jotta työhön saataisiin laajempi tietopohja ja erilaisia näkökulmia.

# 2 Suunnittelu

## 2.1 Suunnittelun lähtökohdat

Teknisesti lavea toimeksianto antoi mahdollisuuden monenlaisiin toteutuksiin. Tämän vuoksi oli myös tärkeää aluksi tehdä teknologiavalinnat ja täsmentää niiden pohjalta vaatimusmäärittelyä, jottei työstä tulisi liian laaja, ja jotta se toisaalta palvelisi tarkoitustaan mahdollisimman hyvin. Lisäksi suunnitteluvaiheessa oli tarpeen tutustua Azuren tarjontaan ja valita käyttöön sopivat palvelut.

## 2.2 Vaatimusmäärittelyn tekninen täsmennys

Sovellukseen tuli toteuttaa kolme osaa: dataa keräävä osa Azuren valittuun pilvipalveluun, pilvitietokanta sekä käyttöliittymänä toimiva nettisivu, josta dataa voi yksinkertaisessa muodossa katsella. Työn lopussa esitellään myös lyhyesti, miten Azuren koneoppimispalvelua voidaan alkaa hyödyntää datan analysointiin.

### 2.2.1 Keräin

Dataa keräävän osan on toimittava käynnistämisen jälkeen pitkiä aikoja itsenäisesti ja virheet on käsiteltävä asianmukaisesti. Olisi hyvä, jos kerääjää voisi konfiguroida jollakin joustavammalla tavalla kuin kääntämällä ohjelma uudestaan.

Keräimen on pystyttävä käsittelemään ainakin yksi pyyntö per sekunti. Tästä voidaan joustaa, mikäli yhdellä pyynnöllä tarkoitetaan useamman datalähteen (vaikkapa mittalaitteen) datan tallentamista kerralla. Lisäksi palvelun tulisi olla saatavilla yli 99 % ajasta.

### 2.2.2 Tietokanta

Tietokannan tulee sijaita Azuressa ja se on valittava tarpeisiin sopivaksi. Tietokannan tulee vastata nopeasti, jotta käyttöliittymästä saadaan sulava. Tietokantaa valittaessa on tarkasteltava myös kustannuskysymyksiä, ja vaihtoehdoista on hyvä kirjata mainta raporttiin.

### 2.2.3 Käyttöliittymä

Käyttöliittymän tulee olla selainpohjainen ja sen on toimittava Azuressa. Käyttöliittymään tulee tehdä yksinkertainen autentikointi, ja siitä tulee pystyä tarkastelemaan ladattua dataa järkevästi jaoteltuna ja selkokielisenä. Dataa ei tarvitse pystyä muokkaamaan käyttöliittymästä. Käyttöliittymän muotoiluun voi tarvittaessa käyttää esimerkiksi Bootstrap-kirjastoa.

## 2.3 Datan muodon tarkastelu ja vaikutusten arviointi

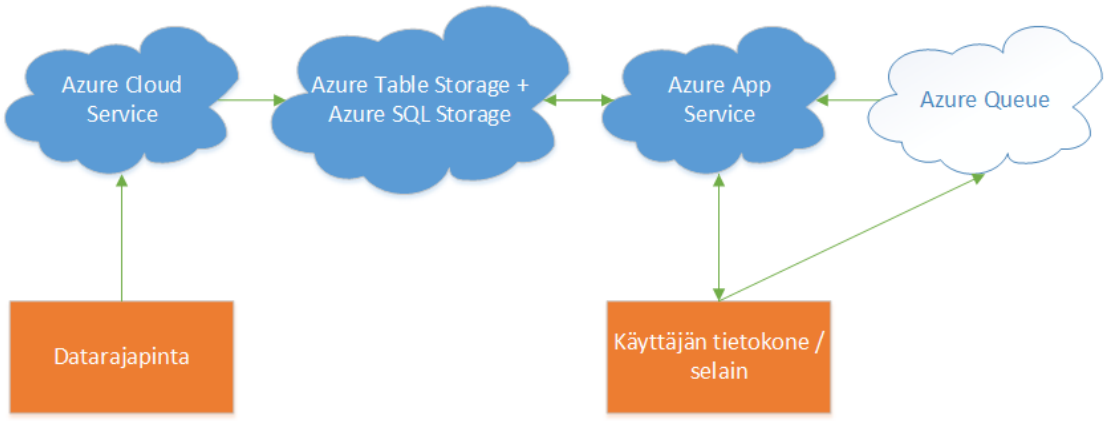
Datalähteeksi valittiin Liikenneviraston Digitraffic-palvelu, joka tarjoaa tie- ja rautatieliikenteen tietoja sekä säätietoja. Tulevaisuudessa Liikennevirasto aikoo avata palveluun myös meriliikenteen dataa. Digitraffic sopii tarkoitukseen hyvin, sillä kaiken palvelusta saatavan tiedon käyttölupa on Creative Commons 4.0 Nimeä. (Digitrafficin palvelukuvaus, N.d.)

Juuri työn alettua Digitraffic ilmoitti uusista rajapinnoista, jotka tulevaisuudessa korvaavat täysin vanhat SOAP-rajapinnat. Päätettiin tietysti käyttää uusia REST-rajapintoja, sillä vanhojen ilmoitettiin poistuvan pian kokonaan. (Mt.)

## 2.4 Azuren palveluiden valinta

### 2.4.1 Yleistä

Microsoft Azuresta löytyy nykyään todella paljon palveluita ja ratkaisuja erilaisiin tarkoituksiin aina tietokannoista IoT-Hubiin ja App Servicestä koneoppimiseen. Oikeiden palveluiden valinta on yhtä lailla tärkeää kuin vaikeaa. Nyt tilannetta helpotti projektin demoluontoisuus, joten vaatimuksena olivat vain mahdollisimman matalat kustannukset sekä toimivuus. Kuviossa 2 näkyy, minkälaiseen ratkaisuun opinnäytetyössä päädyttiin.




Kuvio 2. Korkean tason arkkitehtuurikuva

### 2.4.2 Tietokanta

Azure tarjoaa useita vaihtoehtoja tietokannan pystyttämiseen. Tarjolla on palveluna esimerkiksi perinteinen SQL-kanta, useita NoSQL-vaihtoehtoja sekä käytännössä mikä tahansa tietokanta itse hallittavalla virtuaalikoneella. SQL-tietokantojen etu on tiedon eheys, sillä tietokanta suunnitellaan tarkasti ja tiedon muoto määrätään ennalta tiukoin rajoin. NoSQL-kannat puolestaan tallentavat tietoa joustavammin, mutta tämä aiheuttaa koodauspuolella uusia huomioon otettavia asioita datan käsittelyssä. (Azure Services, n.d.)

SQL-kanta oli toimeksiantajalle tuttu, joten päätettiin lähteä hakemaan lisäarvoa valitsemalla sen sijasta avain–arvo-pareja käyttävä Azure Table Storage. Se vaikutti edulliselta ratkaisulta sekä joustavuutensa puolesta erittäin hyvin tarkoitukseen soveltuvalta. Sovelluksen jatkokehityksessä voisi valitun tallennusmuodon myötä helposti lisätä uusia, erimuotoisia datan lähteitä kajoamatta aiempiin tallennuksiin.

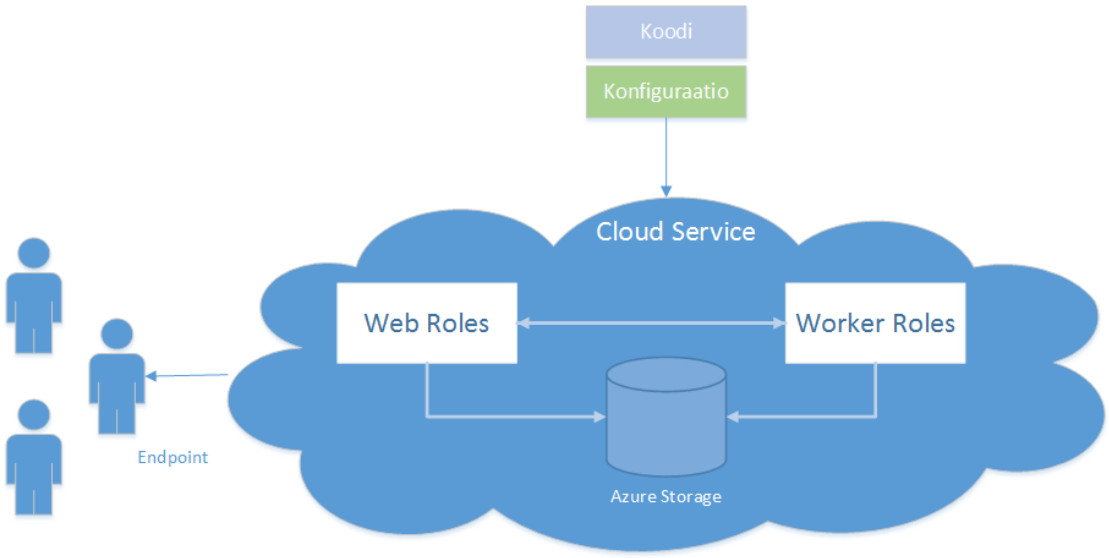
Table Storage maksaa ensimmäisen teratavun osalta 0,059 € / GB ja 0,003 € / 100 000 transaktiota, eli myös datan lukeminen ja kirjoittaminen maksavat. (Azure Storage Pricing n.d.)

Koska Azure Table Storage soveltuu heikosti esimerkiksi käyttäjähallinnan toteuttamiseen (Meredith n.d.), päätettiin käyttäjähallinta toteuttaa erikseen omaan SQL-tietokantaan. Käytännössä kahden eri tietokannan käyttö saattaisi olla kustannustehokasta jopa tuotantoympäristössä, sillä datamäärän kasvaessa SQL-tietokanta tulee

huomattavasti Table Storagea kalliimmaksi. Käyttäjämäärät pysyvät usein kuitenkin hallittavan kokoisina ja mahtuvat siis pienempäänkin SQL-tietokantaan. Esimerkiksi 500 gigatavun tallentaminen SQL-tietokantaan tulee Azuressa jo 10 kertaa kalliimmaksi Table Storageen verrattuna (Azure SQL-tietokantojen hinnat, N.d.).

### 2.4.3 Azure Cloud Service

Itse keräin päätettiin toteuttaa Worker Role -moduulina Azure Cloud Service -alustalla. Worker Role soveltuu projektin tarpeisiin hyvin, sillä se pystyy toimimaan itsenäisesti ja jatkuvasti pitkiä aikoja ns. taustalla. Myös keräimen yksinkertaisuus rohkaisi käyttämään ensisijaisesti pieniin taustaprosesseihin tarkoitettua Worker Role -moduulia. Kuviossa 3 on esitelty Azure Cloud Servicen rakenne.




Kuvio 3. Azure Cloud Service

### 2.4.4 Azure App Service

Web-käyttöliittymää varten otettiin käyttöön Azure App Service. Se tukee verkkosivujen ylläpitoa ja käyttöliittymän voi julkaista kätevästi suoraan Visual Studio -kehitysympäristöstä. App Service on tuoreehko palvelu, joka yhdistää aiemmin erillään olleet palvelut verkkosivuille, rajapinnoille, mobiilisovelluksille ja logiikkasovelluksille. (App Service n.d.)

### 2.4.5 Azure Queue

Vaatimusmäärittelystä löytyvä konfiguroitavuustoive päätettiin toteuttaa Azure Queue -jonopalvelulla. Käytännössä jono mahdollistaa vapaamuotoisten viestien välittämisen kahden sovelluksen välillä. Tällä kertaa päätettiin rakentaa hyvin pieni konsoliohjelma, jolla ohjausviestejä voi lähettää. Azure Cloud Service -alustalla toimivaan sovellukseen implementoitiin kuunteluominaisuus sekä valmius reagoida komentoihin esimerkiksi datankeruun pysäyttämisestä ja käynnistämisestä.

## 2.5 Tietokantasuunnittelu

### 2.5.1 Käyttäjähallinta

Koska käyttäjähallinnalle ei ole asetettu suurempia vaatimuksia, arvioitiin kustannustehokkaimmaksi vaihtoehdoksi pitäytyä Identity-kirjaston valmiissa perustauluissa, jotka mahdollistavat uusien käyttäjien luomisen, sisään- ja uloskirjautumisen sekä käyttöliittymän osien näkyvyyden ja käytön rajaamisen esimerkiksi roolien avulla. Taulujen nimet voidaan siisteyssyistä muuttaa, sillä oletuksena ne ovat muotoa ”AspNetUsers”, eikä esimerkiksi ”User”, kuten olisi tyypillisempää.

### 2.5.2 Kerätty data

Kerätty data deserialisoidaan JSON-muodosta suunnitellun mallin mukaisiksi olioiksi (ks. Datamalli). Olioihin on lisättävä/valittava Azure Table Storagen vaatimat PartitionKey- ja RowKey–kentät. Tämän jälkeen kerätty data voidaan lähettää kantaan sellaisenaan Azuren rajapinnan tarjoamilla metodeilla.

**Datamalli**

Digitraffic-palvelun data on JSON-muodossa. Rakenne on kuvattu lyhyesti taulukossa 1.

Taulukko 1. Digitraffic-palvelun datan rakenteen havainnekuva

| Kenttä | Esimerkkidata |
|---|---|
| dataUpdatedTime | 2016-12-19T15:58:12+02:00 |
| tmsStations | |
| id | 23001 |
| tmsNumber | 1 |
| measuredTime | 2016-12-19T15:58:12+02:00 |
| sensorValues | |
| id | 5054 |
| roadStationId | 23001 |
| name | OHITUKSET_60MIN_KIINTEA_SUUNTA1 |
| oldName | ohitukset_60min_kiintea_suunta1 |
| shortName | kpl/h1 |
| sensorValue | 470.0 |
| sensorUnit | kpl/h |

SensorValues-taulukossa on 18 arvoa, muun muassa autojen määrä tunnissa ja autojen määrä viidessä minuutissa kaistoittain. Nämä kaikki arvot eriteltiin Table Storageen omiksi sarakkeikseen.

## 2.6 Arkkitehtuurin suunnittelu

### 2.6.1 Käyttöliittymän arkkitehtuurisuunnittelu

Käyttöliittymän arkkitehtuuri ei vaatinut paljoa suunnittelua, sillä siinä on erittäin vähän toiminnallisuutta ja sivuja. Lisäksi, koska päätettiin ottaa käyttöön Microsoftin ASP.NET MVC –arkkitehtuuri, oli suurin osa suunnittelusta jo valmiiksi tehty. MVC-arkkitehtuuri tarkoittaa käytännössä ohjelman jakamista kolmeen itsenäiseen osaan (näkymään, kontrolleriin ja malliin), joita on myöhemmin mahdollisimman helppo vaihtaa tai muokata muiden osien rikkoutumatta.

### 2.6.2 Keräimen arkkitehtuurisuunnittelu

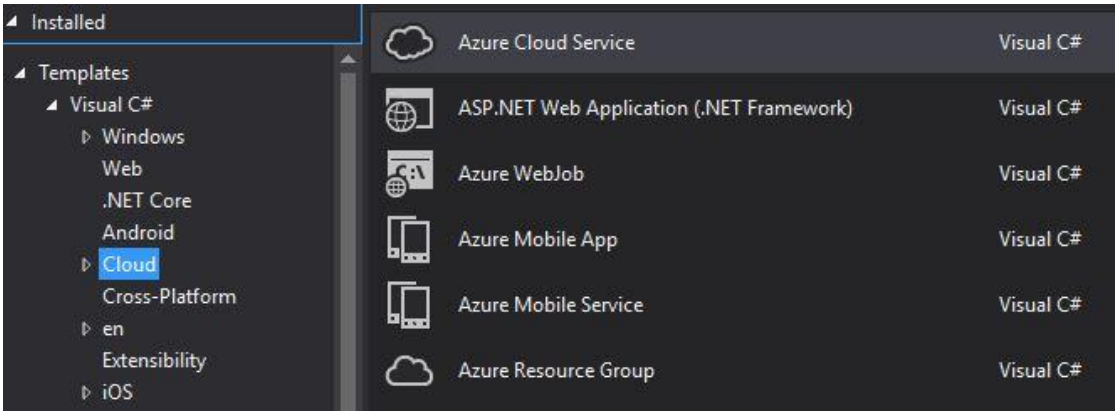
Keräimen arkkitehtuuri pidettiin käyttöliittymän tapaan yksinkertaisena. Laajennettavuuden helpottamiseksi toiminnallisuus jaettiin kuitenkin model, service ja mapper –luokkiin itse Worker Role –ohjelmaluokan lisäksi. Worker Role sisältää Azuren tarvitseman rakenteen, jonka saa Visual Studioon valmiina pohjana. Käytännössä Worker Rolessa on Run-metodi, josta ohjelman suoritus alkaa, sekä RunAsync-metodi, jota Run kutsuu. RunAsync sisältää while-silmukan, joka voidaan pysäyttää antamalla sille CancellationToken.

# 3 Toteutus

## 3.1 Azure-palvelun toteutus

### 3.1.1 Yleistä

Azureen päätettiin toteuttaa Cloud Service ja sinne niin sanottu Worker Role. Worker Role on tarkoitettu pieniin taustaprosesseihin, jotka pyörivät jatkuvasti, joten ne sopivat tarkoitukseen mainiosti. Worker Rolen pohja löytyy suoraan Visual Studion projektipohjista Cloud Servicen alta (ks. kuvio 4). Pohjan nähdäkseen on asennettava Azure SDK.




Kuvio 4. Worker Role -pohja löytyy Visual Studiosta Cloud Servicen alta.

Käytännössä Azuren Worker Role -mallin rakenne on varsin yksinkertainen. Ohjelman suoritus alkaa Run-metodista, jossa ei tarvita muuta kuin yksi lyhyt try-rakenne. Run-metodista kutsutaan RunAsync-metodia, jolle annetaan parametrina CancellationToken. RunAsync-metodi on nimensä mukaisesti asynkronisesti suoritettava koodi, joka tarkistaa CancellationTokenin avulla, onko lopetuskäsky annettu, ja jos ei ole, suorittaa halutun ohjelmakoodin. Metodin lopussa voidaan antaa ”await Task.Delay()”-käsky, jolla saadaan suoristuskertojen väliin jäämään haluttu määrä millisekunteja. Käytännössä metodilla voidaan synkronoida datan hakeminen datan päivitysväliin, joka tässä projektissa oli 5 minuuttia.

Digitrafficin datan mallintamista ja käsittelyä varten toteutettiin datan sisäkkäisen rakenteen takia kolme luokkaa: LAMModel, TmsStation ja SensorValueModel. Näiden luokkien ominaisuudet kopioitiin suoraan Digitrafficin datan muodosta ja ne olivat lähinnä deserialisointia varten. Jotta data saataisiin siistiin muotoon Table Storageen, päätettiin käyttää myös ylimääräistä DTO-luokkaa[1], joka purkaa kerrosrakenteen yhteen tasoon (yhden mittauspisteen yhden mittauskerran kaikkien sensorien data yhdelle riville tietokantaan).

Koska käytössä oli Table Storage, piti DTO-luokkaan periyttää Azure-kirjastoista TableEntity. Sen myötä luokkaan on lisättävä itse datan ominaisuuksien lisäksi PartitionKey ja RowKey -ominaisuudet. PartitionKeyn avulla dataa voi jaotella nopeammin haettaviin osioihin, ja yhdessä RowKeyn kanssa ne muodostavat tietokantataulun pääavaimen (primary key), jonka on oltava uniikki.

PartitionKey aiheutti pohtimista, sillä jotta Table Storagen kustannuksia vähentävää datan paketoimista[2] (batch) voitaisiin käyttää, tulisi jokaisen batchiin laitettavan datarivin PartitionKey olla sama. Tästä syystä esimerkiksi tienvarsiaseman id:n valitseminen PartitionKeyksi olisi huono vaihtoehto, mutta taas dataa luettaessa id:n haluttaisiin olevan nimenomaan PartitionKey, jotta yhden aseman historiadata saataisiin luettua mahdollisimman nopeasti. Avaimet ovat string-muotoisia, ja niissä on paljon

[1] DTO, eli Data Transfer Object, on luokka, jota käytetään usein, kun on tarve erotella datan todellinen muoto sekä muoto, jossa se kannattaa siirtää tai käsitellä. Joskus tämä tehdään tietoturvasyistä, kun halutaan vaikkapa antaa käyttäjän muokata jotain osaa datasta, mutta piilottaa tietyt ominaisuudet. Tässä tapauksessa luokalla haluttiin yksinkertaistaa tiedonsiirtoa.

[2] Table Storagessa datan voi lähettää pienissä paketeissa sen sijaan, että se lähetettäisiin rivi kerrallaan. Yhteen pakettiin voi liittää korkeintaan sata objektia. Jokainen paketti maksaa yhden transaktion verran, eli parhaimmillaan paketoimatta jättäminen voi satakertaistaa siirtokulut tietokantaan.

rajoituksia (Understanding the Table Service Data Model 2016). Koska Table Storagen käyttö maksaa vain 0,003 € / 100 000 transaktiota (Azure Storage Pricing n.d.), koettiin datan nopea tieasemakohtainen hakeminen kustannuksien minimointia järkevämmäksi painotukseksi.

### 3.1.2 Azure Queuen implementointi

Keräimen etäohjaus mahdollistettiin hyödyntämällä Azure Queue -palvelua. Jonopalvelu kuuluu osaksi Storage-palvelua, joten rajapinta sen käyttöön on hyvin samanlainen kuin Table Storagen kohdalla. Ensin ohjelmallisesti jäsennetään connection string CloudStorageAccount-olioksi, jonka avulla luodaan CloudQueueClient. Tämä osaa ottaa yhteyden palveluun ja esimerkiksi luoda tietyn nimisen jonon, jos sitä ei ole olemassa.

Tässä tapauksessa Worker Role pantiin kuuntelemaan tiettyä jonoa jokaisella ajokerralla ennen datan latausta. Jos jonossa oli viestejä, ne haettiin ja käsiteltiin (ks. kuvio 5). Yksinkertaiseen switch-case-rakenteen avulla tallennettiin mahdollisten viestien sisällöt. Jos esimerkiksi jonoon lähetti viestin ”digitraffic road”, käänsi keräin kyseistä datanlähdettä koskevan lipun. Jos lippu oli tosi, ladattiin dataa, ja jos eipätosi, niin lataus jätettiin tekemättä. Näin keräimeen saatiin mukaan hieman älyä, eikä sitä tarvinnut erikseen käydä sammuttamassa Azuren portaalista.

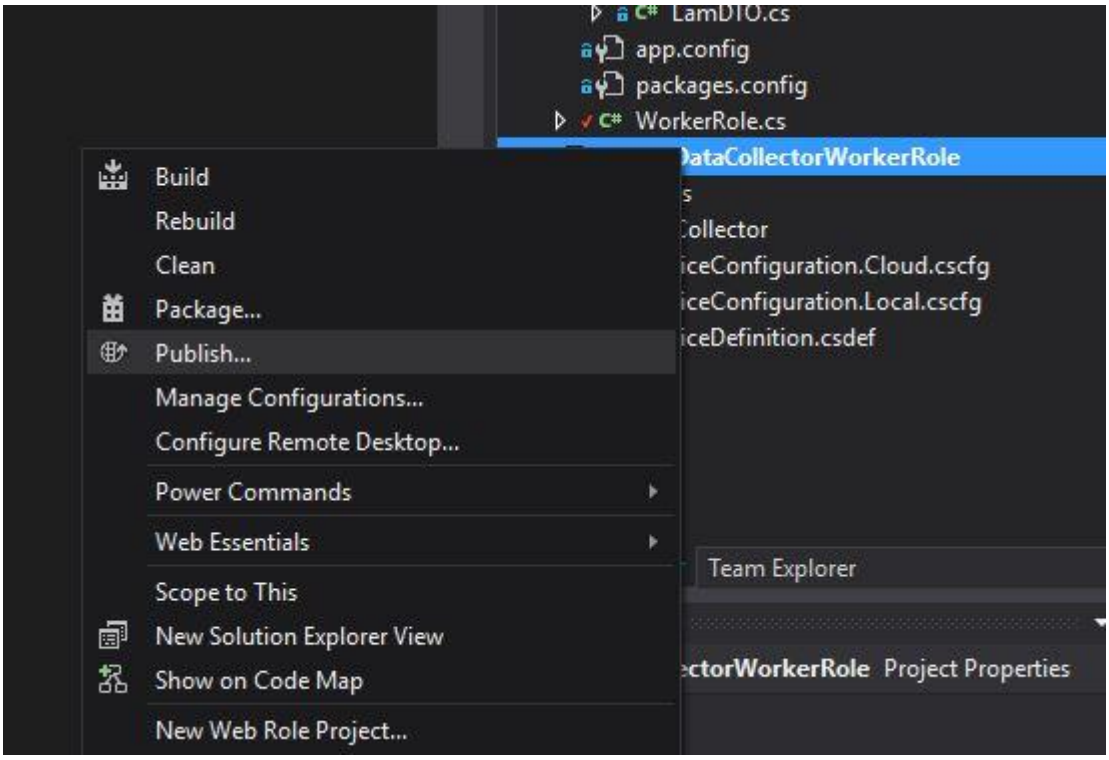
```
212     public void CheckQueue()
213     {
214         try
215         {
216             CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
217                 CloudConfigurationManager.GetSetting("StorageConnectionString"));
218             CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
219             CloudQueue queue = queueClient.GetQueueReference("                    ");
220 
221             queue.FetchAttributes();
222 
223             int? cachedMessageCount = queue.ApproximateMessageCount;
224 
225             // Jos viestien määrä on yli 0 niin loopataan
226             // Käsitellään viesti ja poistetaan se jonosta (pelkkä getMessage ei poista jonosta, vaan piilottaa)
227             // Lopuksi päivitetään viestilukumäärä
228             while (cachedMessageCount != null && cachedMessageCount > 0)
229             {
230                 Trace.TraceInformation("Jonossa tulkittiin olevan viestejä.");
231 
232                 var message = queue.GetMessage();
233 
234                 // Tähän täytyy lisätä kaikki tulevat rajapinnat, kun niille on lisätty lippu
235                 switch (message.AsString)
236                 {
237                     case "digitraffic road":
238                         digiTrafficLamData = !digiTrafficLamData;
239                         Trace.TraceInformation("DigitrafficLamData-lippu käännetty.");
240                         break;
```

Kuvio 5. Azure Queue -koodia

Viestien lähettämistä varten tehtiin pieni konsoliohjelma, joka otti vastaan käyttäjän kirjoitusta ja lähetti sen jonoon. Mikäli viestin sisältö ei vastannut mitään komentoa, se vain luettiin ja poistettiin ilman muita toimenpiteitä, joten järjestelmä ei ollut kovin altis virheille, paitsi siinä mielessä, että käyttäjä saattoi tahattomasti kirjoittaa väärän viestin.

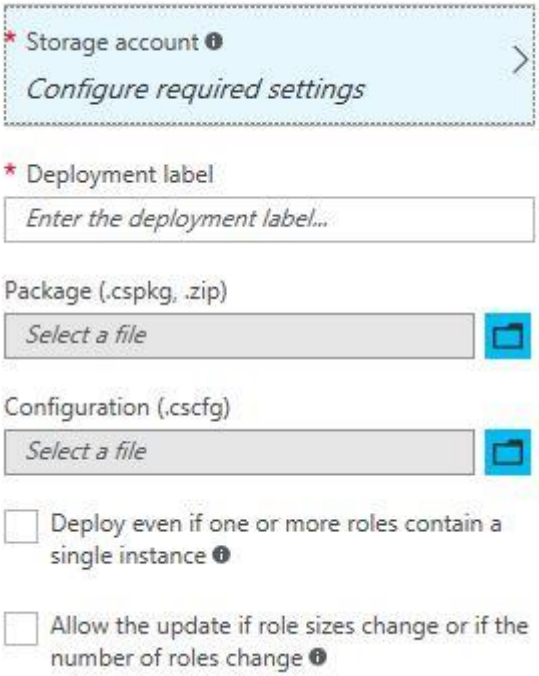
### 3.1.3 Worker Role -moduulin asennus Azureen

Worker Role toimii Azure Cloud Service -palvelussa. Microsoft on tehnyt asentamisesta varsin yksinkertaista Visual Studion avulla: kehitysympäristöstä löytyy valmiiksi kuviossa 6 esitetty julkaisupainike, jonka kautta roolin saa pilveen. Publish-vaihtoehto löytyy Visual Studiosta esimerkiksi klikkaamalla hiiren kakkospainikkeella Worker Role -objektin päältä Solution Explorer -näkymässä.




Kuvio 6. Publish-vaihtoehto Visual Studiossa

Tarvittavat tiedostot voidaan ladata Azureen myös käsin Azure portaalin kautta. Tämä keino voi toimia paremmin, jos lataamisessa suoraan kehitysympäristöstä on ongelmia. Tällöin on Visual Studiosta ensin valittava kuviossa 6 näkyvä Package. Tämä toiminto luo kaksi tarvittavaa tiedostoa, jotka ovat päätteiltään cspkg ja cscfg. Tämän jälkeen Azureen luodun Cloud Servicen kohdalta valitaan Update ja annetaan kaavakkeeseen tiedostot. Malli kaavakkeesta löytyy kuviosta 7.




Kuvio 7. Azure Cloud Service -tiedostonlatauskaavake

## 3.2 Käyttöliittymän toteutus

### 3.2.1 Yleistä

Käyttöliittymä toteutettiin web-sivuna käyttäen MVC-mallia. Microsoftin MVC tarjoaa tehokkaat työkalut web-käyttöliittymien tekemiseen.

Sivustosta tehtiin mahdollisimman yksinkertainen: siinä on vain kaksi näkymää. Toiselta sivulta pääsee katselemaan viimeisintä tuhatta riviä kannasta ja toiselta yhden mittauspisteen viimeisintä tuhatta riviä. Käytännössä tämä toteutettiin LamController-luokkaan, jossa otettiin yhteys Table Storageen ja ladattiin tallennettu data. Tämä

data siirrettiin IEnumerable-joukkona View-tyyppiseen luokkaan, jossa data purettiin HTML-koodin sekaan siistiksi taulukoksi.

### 3.2.2 Data transfer -luokka

DTO (data transfer object) kopioitiin suoraan keräimestä. Käytössä on siis sama LamDTO-luokka, joka sisältää Digitrafficilta tulevan datan rakenteen C#-luokaksi muutettuna. Käytännössä datan sisäkkäinen hierarkia on purettu yhteen luokkaan, sillä Table Storage ei tue mutkikkaita ja sisäkkäisiä rakenteita.

### 3.2.3 Controller-luokat

LamController-luokka on vastuussa Digitrafficin dataa esittävistä näkymistä. Details-näkymää varten otetaan vastaan kokonaisluku id, jonka perusteella Table Storagesta haetaan kaikki id:tä vastaavat rivit (ks. kuvio 8). Id on tässä tapauksessa tienvarsiaseman numero.

```
26          // GET: Lam/Details/{id}
27          public ActionResult Details(int id)
28          {
29              // Retrieve the storage account from the connection string.
30              CloudStorageAccount storageAccount = CloudStorageAccount.Parse(storageConnectionString);
31
32              // Create the table client.
33              CloudTableClient tableClient = storageAccount.CreateCloudTableClient();
34
35              // Retrieve a reference to the table.
36              CloudTable table = tableClient.GetTableReference("paattotyo");
37
38              var name = table.Name;
39
40              // Construct the query operation for all customer entities where PartitionKey="Smith".
41              TableQuery<LamDTO> query = new TableQuery<LamDTO>()
42                      .Where(TableQuery.GenerateFilterCondition("PartitionKey",
43                                                                QueryComparisons.Equal,
44                                                                id.ToString()));
45
46              var result = table.ExecuteQuery(query);
47
48              var list = result.ToList();
49
50              return View(list);
51          }
```

Kuvio 8. LamController-luokan Details-toiminto käyttää Azuren Table Storagen rajapintaa.

Datan hakeminen Table Storagesta on hyvin vaivatonta ja onnistuu pienellä määrällä koodia, mikä oli ilahduttavaa huomata. Käyttöohjeita selatessani huomasin, että rajapintaa oli uusittu suhteellisen vähän aikaa sitten, ja vanhemmissa verkkokeskusteluissa nousivat usein esiin hankalat ja vanhat tavat tehdä asioita. Microsoft on siis

selvästi panostanut myös Azuren helppokäyttöisyyteen, eikä vain toimintojen lisäämiseen.

### 3.2.4 Näkymät

Näkymät pidettiin erittäin yksinkertaisina taulukkonäkyminä, ja esimerkiksi kirjautumisruutua ei lähdetty lainkaan muokkaamaan Microsoftin tarjoamasta valmiista pohjasta. Esimerkiksi Lam/Details/{id}-näkymä on pelkkä taulukko, johon voi DTO:n avulla helposti koodissa valita haluamansa kentät näkyviin. Aivan paljaan HTML-taulukon sijasta käytettiin kuitenkin DataTables-JavaScript-lisäosaa. Sen avulla taulukosta saa erittäin helposti hieman siistimmän. Kaikki DataTablesin käyttöön tarvittava koodi näkyy kuviossa 9. Lisäosan asennus onnistui Visual Studion Package Managerilla. Lisäosassa on paljon optioita, joilla generoitavan taulukon ulkonäköä voi muokata, mutta niitä ei tässä projektissa käytetty.

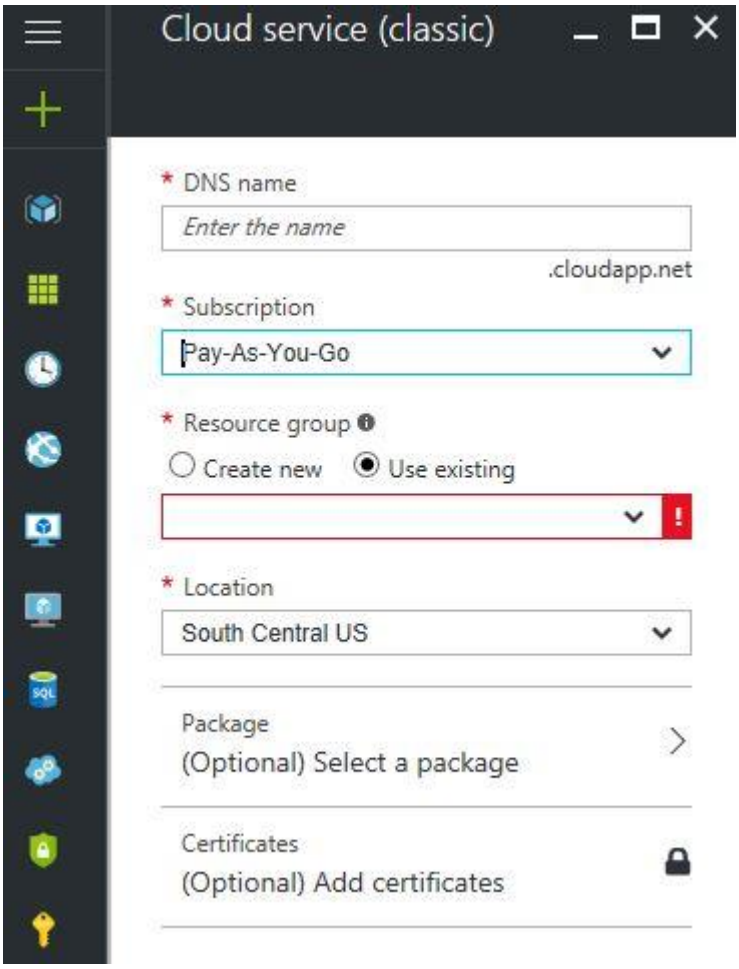
```
30          <script>
31              $(document).ready(function () {
32                  $("#lamdata").DataTable();
33              });
34          </script>
```

Kuvio 9. DataTables-lisäosan käyttö

## 3.3 Tietokannan toteutus

### 3.3.1 Yleistä

Tietokanta, johon liikennedataa tallennetaan, ei vaatinut varsinaista toteutusta, mutta toki Table Storagen valinta tallennusalustaksi vaikutti muuhun toteutukseen. Itse palvelu aktivoitiin Azuren portaalin kautta (ks. kuvio 10), mutta taulu luotiin dynaamisesti (ks. kuvio 8, rivi 36).




Kuvio 10. Azure Cloud Service -luomiskaavake

Käyttäjätietokanta päätettiin eriyttää SQL-kannaksi, joten se piti luoda erikseen. Sekään ei kuitenkaan vaatinut erityistä toteutusta, sillä käytössä oli Microsoft Identity -kirjasto, joka luo tietokantataulut käytännössä automaattisesti. Tietokantaan tehdyt muokkaukset määriteltiin ohjelmakoodissa (ks. kuvio 11).

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<User>().ToTable("User").Property(p => p.Id)
        .HasColumnName("UserId");
    modelBuilder.Entity<IdentityUserRole>().ToTable("UserRole");
    modelBuilder.Entity<IdentityUserLogin>().ToTable("UserLogin");
    modelBuilder.Entity<IdentityUserClaim>().ToTable("UserClaim")
        .Property(p => p.Id).HasColumnName("UserClaimId");
    modelBuilder.Entity<IdentityRole>().ToTable("Roles")
        .Property(p => p.Id).HasColumnName("RoleId");
```

Kuvio 11. IdentityModel-luokkaan tehdyt muutokset

### 3.3.2 Azure Table Storage

Koska päätettiin käyttää itse datan tallentamiseen Azuren Table Storagea, ei kantaa tarvinnut varsinaisesti tehdä etukäteen. Datan muoto pitää vain suunnitella järkeväksi. Kun Azureen on luotu tili ja avattu tallennuspalvelu (storage account), voi kantaan yksinkertaisesti ottaa ohjelmallisesti yhteyden ja alkaa siirtää dataa. Jos kohdetaulua ei ole olemassa, Azure luo sen lennosta ja sen sarakkeet muotoutuvat siirrettävän datan mukaan.

### 3.3.3 Entity Framework koodi edellä

Nykyään Entity Framework (EF) osaa luoda SQL-komennot suoraan C#-koodista (ks. kuvio 12), ja tätä tapaa päätettiin myös käyttää. Toinen vaihtoehto olisi ollut Database First –malli, jossa ensin suunniteltaisiin tietokanta, josta tehtäisiin EF:llä malli, joka liitettäisiin projektiin ja johon voisi siitä viitata. Code First –mallissa tietokantataulut luodaan koodin perusteella. Code First on siitä parempi tapa, että muutokset näkyvät kannassa heti ja kehittäminen on ohjelmoijalle helpompaa ja nopeampaa. Haittapuolena tietokannan syvempi hienosäätö vaatii uusien asioiden opettelua, kun tietokantaan ei suoraan ole tarkoitus koskea. (Entity Framework Code First Conventions. n.d.)

```
CREATE TABLE [dbo].[User] (
    [UserId] [nvarchar](128) NOT NULL,
    [Email] [nvarchar](256),
    [EmailConfirmed] [bit] NOT NULL,
    [PasswordHash] [nvarchar](max),
    [SecurityStamp] [nvarchar](max),
    [PhoneNumber] [nvarchar](max),
    [PhoneNumberConfirmed] [bit] NOT NULL,
    [TwoFactorEnabled] [bit] NOT NULL,
    [LockoutEndDateUtc] [datetime],
    [LockoutEnabled] [bit] NOT NULL,
    [AccessFailedCount] [int] NOT NULL,
    [UserName] [nvarchar](256) NOT NULL,
    CONSTRAINT [PK_dbo.User] PRIMARY KEY ([UserId])
)
```

Kuvio 12. Esimerkki Entity Frameworkin automaattisesti luomasta SQL-koodista

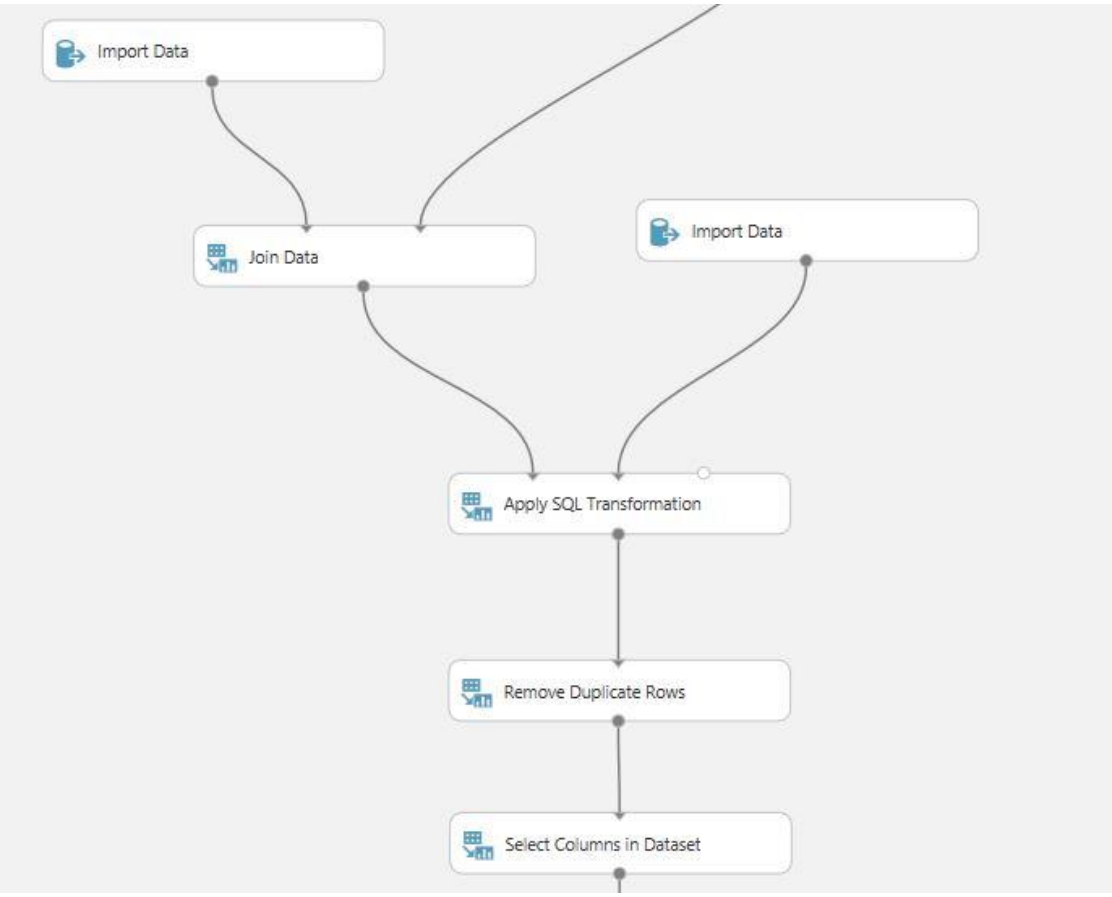
Code First -migraatiot otetaan Visual Studiossa käyttöön Package Manager -konsolin avulla komennolla Enable-Migrations. Tämän jälkeen uuden migraation voi lisätä komennolla Add-Migration <nimi>. Tämä komento ei vielä vie muutoksia tietokantaan, vaan ainoastaan ryhmittelee havaitut muutokset samaan tapaan kuin commit-komento git-ympäristössä. Update-Database-komennolla muutokset tallentuvat tietokantaan. Migraatioilla voi luoda ja muokata tauluja ja niiden rivejä sekä muuttaa käytännössä mitä tahansa tietokannan asetuksia. (Entity Framework Code First Migrations. 2016)

# 4 Koneoppimisen hyödyntäminen Azuressa

Azuresta löytyy varsin vaikuttava kokoelma erilaisia koneoppimisen työkaluja, joita nykyään hyödynnetään data-analyysissä paljon (Tanskanen 2015). Niihin pääsee käsiksi Machine Learning Workspaces -kohdan kautta. Käytännössä koneoppimista Azuressa käytetään Machine Learning Studion kautta.

Studio näyttää ohjelmoijan näkökulmasta hyvin yksinkertaiselta, mutta on todellisuudessa voimakas työkalu datan analysointiin, minkä lisäksi sillä voi toteuttaa jopa Web Servicen, joka hyödyntää analyysin tuloksia.

Studion keskellä olevaan kenttään vedetään vasemmalta erilaisia moduuleja, jotka näkyvät laatikoina ja joita voi yhdistellä viivoilla loogiseksi kokonaisuudeksi, kuten kuviosta 13 voi nähdä. Näitä moduuleita ovat esimerkiksi datan haku erilaisista lähteistä, kuten Azuren omista tallennusratkaisuista, tiedostosta tai jostakin ulkopuolisesta rajapinnasta. Sen jälkeen toisilla moduuleilla dataa voi yhdistellä ja siivota ennen analyysin aloittamista. Itse analyysikin toteutetaan samoilla laatikoilla. Tarjolla on kymmeniä erilaisia algoritmeja, joiden tarkoitusta matemaattisesti harjaantumattoman voi olla vaikea ymmärtää. Erilaiset algoritmit soveltuvat erilaisiin aineistoihin.




Kuvio 13. Azure Machine Learning Studion käyttöliittymä

Algoritmin suorittamisen jälkeen tuloksia vielä arvioidaan ja niistä rakennetaan malli. Tämä tapahtuu edelleen laatikkomoduuleilla. Lopuksi tulosta voi joko hyödyntää itse tai sen voi julkaista verkkopalveluksi. Julkaistuun palveluun voi syöttää uutta dataa, ja palvelu osaa siitä luodun mallin perusteella kertoa jotakin, yleensä ennustaa tuloksia.

Tämän projektin aikana koneoppimista ei hyödynnetty, sillä se vaatisi datan keräämistä pitkältä aikaväliltä. Tulevaisuudessa se voi kuitenkin tarjota toimeksiantajalle mielenkiintoisia uusia näkökulmia.

# 5 Lopputuote ja sen arviointi

## 5.1 Testaamisesta

Keräimen yksikkötestaaminen oli sen pilviluonteen vuoksi hankalaa. Käytännössä testaamisen onnistuminen olisi vaatinut koodin uudelleenjärjestelyä. Niinpä projektin testaus suoritettiin integraatio- ja järjestelmätasolla manuaalisesti. Demoluontoisuutensa vuoksi testaaminen uskallettiin jättää suhteellisen vähälle, mutta muutamat suuret oksakohdat saatiin kuitenkin karsittua pois. Esimerkkejä näistä ovat esimerkiksi pilvikomponenttien olemassaolon tarkistus, yhteyksien luonti datan lähteeseen ja tallennuskohteeseen sekä erilaisten palautusten arvot.

## 5.2 Keräin

Lopputuotteen pääosa on ohjelma, joka hakee rajapinnasta dataa, jäsentää sen sopivaan muotoon ja tallentaa pilvitietokantaan. Oleellista on, että myös tämä ohjelma ajetaan pilvessä, eikä sen toimintaan käynnistämisen jälkeen tarvitse juuri puuttua.

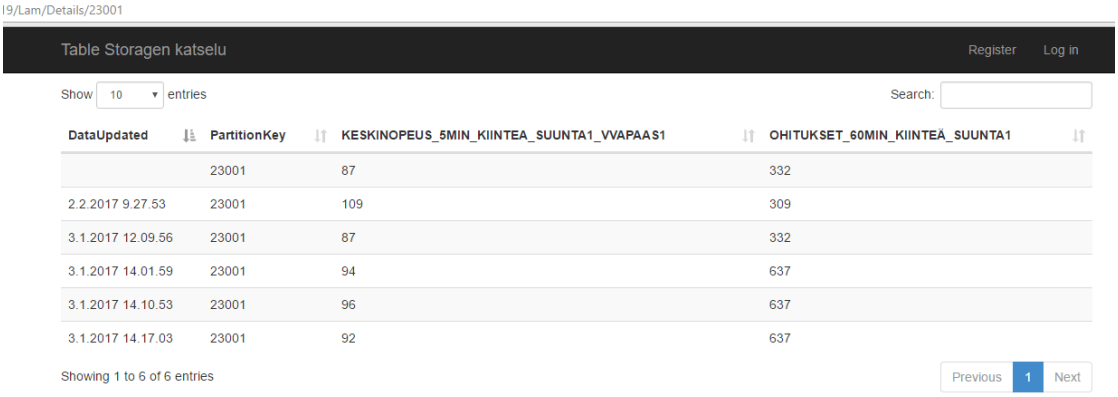
Keräimelle asetetut vaatimukset täyttyivät täysin niin toiminnallisten kuin ei-toiminnallisten vaatimusten osalta. Uusien datalähteiden antaminen keräimelle tulevaisuudessa vaatii jonkin verran koodaamista, mutta suurin pohjatyö on kuitenkin tehty. Käytännössä datan hankkiminen ja muokkaaminen Table Storagelle sopivaan muotoon jäävät tehtäviksi.

## 5.3 Käyttöliittymä

Osana työtä toteutettiin web-käyttöliittymä. Käyttöliittymän vaatimuksena oli, että pilveen tallennettua tietoa pitää pystyä katselemaan selaimella. Lisäksi piti toteuttaa jonkinlainen autentikointi, jottei kuka tahansa pääse tietoihin käsiksi. Nämä vaatimukset täyttyivät. Kerättyä dataa pystyy katselemaan siistissä taulukossa. Dataa ei

pysty käyttöliittymän kautta muokkaamaan, kuten ei ollut tarvettakaan. Azure tarjoaa oman, helppokäyttöisen ja kevyen Azure Storage Explorer -ohjelman, jolla dataa pääsee halutessaan myös tarkastelemaan sekä muokkaamaan (Microsoft Azure Storage Explorer. N.d.).

Esimerkkinä käyttöliittymästä kuvio 14, jossa LamControllerin Details-metodilta pyydetään tienvarsiaseman 23001 tallennetut tapahtumat. Taulukkoon on valittu muutama satunnainen sarake näytettäväksi, mutta niitä voi helposti vaihtaa näkymän koodista.




Kuvio 14. Web-käyttöliittymä

## 5.4 Tietokannat

### 5.4.1 Table Storage

Pienen työstön jälkeen tietokannasta saatiin suhteellisen järkevän muotoinen. Kuviossa 15 näkyy, minkälaisia sarakkeita päädyttiin käyttämään. Loput sarakkeet jatkuvat samaan tyyliin ja ovat suoraan Digitraffic-palvelun datassa olevien kenttien nimiä, jotka on leikattu kuviosta tilankäyttösyistä. PartitionKey, RowKey ja Timestamp ovat Table Storagessa pakollisia sarakkeita, ja viimeksi mainittu luodaan aina tietokantamoottorin toimesta automaattisesti.


Kuvio 15. Tietokannan sarakkeita

Datan pakkaaminen paketteihin ei onnistunut, mikä jäi hieman harmittamaan. Siihen olisi varmasti löytynyt ajan kanssa jonkinlainen ratkaisu.

### 5.4.2 Käyttäjähallinta (Azure SQL)

Käyttäjähallinta toteutettiin Entity Frameworkia käyttäen Azure SQL -tietokantaan, sillä se vaati vähiten konfigurointia ja on varmatoiminen valinta. Toisena vaihtoehtona harkittiin Table Storagea, mutta tuki sen osalta nähtiin huonoksi.

Käyttäjähallinta on käytännössä automaattisesti mukana Microsoft Identityn muodossa, mutta muutamia muokkauksia tehtiin esimerkiksi taulujen nimiin periaatteen ja esimerkin vuoksi. Käyttäjähallinta toimii hyvin ja on luotettava autentikaattori. Itse käyttäjien muokkaaminen on kuitenkin työlästä ja vaatii kolmannen osapuolen ratkaisuja, itse koodattua hallintatyökalua tai tietojen muuttamista suoraan kannasta. Azure SQL -tietokantaan voi ottaa yhteyden Microsoftin SQL Server Management Toolilla, joten sen hallinta onnistuu samalla tavalla kuin paikallisenkin kannan.

## 5.5 Etäohjaus

Hieman ylimääräisenä osana, mutta kuitenkin vaatimuksissa toiveena mainittuna, toteutettiin mahdollisuus ohjata pilvessä toimivaa keräintä jonoviestein. Azure

Queue -palvelua hyödyntämällä ohjelmalle voitiin lähettää kirjoitettuja viestejä. Pilvikeräin kuunteli tiettyä jonoa ja reagoi ennalta määrätyn muotoisten viestien saapuessa asianmukaisesti esimerkiksi lopettamalla keräämisen.

Etäohjaus toimi testeissä hyvin ja teki mitä sen kuuluikin. Etäohjauksen ansiosta ihan jokaista muutosta varten ei tarvinnut kääntää keräimestä uutta versiota ja ladata sitä pilveen. Digitraffic-palveun kohdalla dataa ei välttämättä olisi mennyt hukkaan vaikka olisikin tarvinnut, mutta jos data tulevaisuudessa päivittyy tiuhemmin, olisi etäohjauksella epäilemättä arvoa.

# 6 Pohdinta

Opinnäytetyön aihe oli mielenkiintoinen eikä vähiten ajankohtaisuutensa vuoksi. Pilvipalveluja rummutetaan lujaa nyt vähän joka suunnasta, joten varmasti niiden merkitys tulee lähiaikoina kasvamaan. Yrityksillä ei kuitenkaan ole useinkaan valmiiksi töissä pilviasiantuntijoita, joten selvitystyötä ja opiskelua tarvitaan.

Opinnäytetyönä toteutettu kokonaisuus oli hyvä esimerkki pilvipalvelusta: se toimii itsenäisesti paikallisesta toteutuksesta riippumattomana ja mahdollistaa helpon skaalauksen.

Olennaisimmat kysymykset, joihin opinnäytetyöllä haluttiin vastata, olivat:

1. Soveltuuko pilvipalvelu liikennedatan keräämiseen ja analysointiin hyvin?
2. Onko pilvipalvelun käyttäminen kustannustehokasta annetussa kontekstissa?

Ensimmäiseen vastaus on selvästi kyllä. Liikennedatan muoto on moninaista, mutta toisaalta pilvessä toimivat palvelut ovat monipuolisia ja joustavia. Azure-pilvipalvelu tarjoaa nyt käytettyjen ratkaisujen lisäksi paljon muitakin palveluita, joita voidaan tulevaisuudessa hyödyntää IoT-kehityksessä (Internet of Things). Tällaisia ovat esimerkiksi IoT Hub sekä Stream Analytics.

Toiseen kysymykseen vastaus lienee monitahoisempi, mutta yhtä kaikki edelleen kyllä. Azuren avulla on suhteellisen yksinkertaista toteuttaa pieniä demoja ja näitä demoja puolestaan on helppo skaalata myöhemmin suuremmiksi yksinkertaisesti kasvattamalla valittuja resursseja lennosta (esimerkiksi lisäämällä SQL-kannan kokoa ja nopeutta). Muutamalla eurolla kuukaudessa tai jopa ilmaiseksi on mahdollista

tehdä kokeiluja, joista voi myöhemmin syntyä suuria tulonlähteitä tai hyviä ajatuksia ja kokemusta.

Palveluiden valinta onnistui jälkeenpäin tarkastellen hyvin, ja oli järkevää lähteä tutkimaan esimerkiksi Table Storagea. Myös jonopalveluun tutustuminen tuotti lisäarvoa. Näiden palveluiden toteutukset ovat demotasolla toimivia ja uskon, että molempia voidaan jatkossa hyödyntää tavalla tai toisella myös oikeassa tuotantokäytössä IoT-ratkaisujen osana.

Nyt tehty toteutus opetti tekijälleen paljon niin pilvipalveluista yleisesti kuin eritoten Azuresta, sen tarjoamista moninaisista palveluista sekä itse pilvitoteutuksista. Työ onnistui hyvin ja tarjosi haluttuja vastauksia.

# Lähteet

App Service. N.d. Microsoft Azuren App Servicestä kertova verkkosivu. Viitattu 23.1.2017. https://azure.microsoft.com/en-us/services/app-service/

ASP.NET. N.d. Microsoft ASP.NET -kirjaston verkkosivu. Viitattu 12.12.2016. https://www.asp.net/

Azure Services. N.d. Lista Microsoft Azuren palveluista. Viitattu 28.11.2016. https://azure.microsoft.com/fi-fi/services/

Azure Solutions. N.d. Microsoftin Azuresta kertova verkkosivu. Viitattu 22.11.2016. https://azure.microsoft.com/fi-fi/solutions/

Azure SQL -tietokantojen hinnat. N.d. Microsoft Azure SQL -tietokantaratkaisujen hintatiedot. Viitattu 12.12.2016. https://azure.microsoft.com/en-us/pricing/details/sql-database/

Azure Storage Pricing. N.d. Microsoft Azure Table Storage -palvelun hintatiedot. Viitattu 28.11.2016. https://azure.microsoft.com/fi-fi/pricing/details/storage/tables/

Digitraffic-palvelun palvelukuvaus. N.d. Liikenneviraston Digitraffic-palvelun Github-sivun wiki-osio. Viitattu 22.11.2016. https://github.com/finnishtransportagency/digitraffic/wiki

Entity Framework Code First Conventions. N.d. Microsoftin Entity Frameworkin Code First -metodeja selventävä opas. Viitattu 25.1.2017. https://msdn.microsoft.com/en-us/library/jj679962(v=vs.113).aspx

Entity Framework Code First Migrations. 2016. Microsoftin Entity Frameworkin Code First -migraatioita selventävä opas. Viitattu 1.2.2017. https://msdn.microsoft.com/en-us/library/jj591621(v=vs.113).aspx

Hafren, M. 2014. Webhotellin, virtuaalipalvelimen ja pilven ero. Primeweb-blogi. Viitattu 22.11.2016. http://www.primeweb.fi/fi/blog/webhotellin-virtuaalipalvelimen-ja-pilven-ero

Meredith, N. 2015. Why you shouldn't use Azure Table Storage as your website back end. Viitattu 12.12.2016. https://www.neilmeredith.com/why-you-shouldnt-use-azure-table-storage-as-your-website-backend/

Microsoft Azure Storage Explorer. N.d. Azure Storage Explorer -ohjelman kotisivu. Viitattu 30.1.2017. http://storageexplorer.com/

Nodeon lyhyesti. N.d. Artikkeli Nodeon Oy:n sivuilta. Viitattu 22.11.2016. http://www.nodeon.com/fi/yritys/lyhyesti/

Privacy authorities across Europe approve Microsoft’s cloud commitments. 2014. Tiedote koskien Euroopan unionin Azurelle antamaa hyväksyntää Microsoftin blogissa. Julkaistu 10.4.2014. Viitattu 16.1.2017. https://blogs.microsoft.com/blog/2014/04/10/privacy-authorities-across-europe-approve-microsofts-cloud-commitments/

Tanskanen, M. 2015. Koneoppiminen tuo konkretian IoT -hankkeisiin. Artikkeli. Viitattu 25.1.2017. http://www.tekniikkatalous.fi/kumppaniblogit/sas/koneoppiminen-tuo-konkretian-iot-hankkeisiin-6061179

Understanding the Table Service Data Model. 2016. Artikkeli Azure Table Storagen datamallista ja sen rajoitteista. Julkaistu 30.11.2016. Viitattu 20.12.2016. https://docs.microsoft.com/en-us/rest/api/storageservices/fileservices/understanding-the-table-service-data-model