Data-analyse en visualisatie van verkeersdata

Ruben Vervust

Promotoren: prof. dr. Jan Cnops, dhr. Hans Fraiponts (Digipolis)

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Bart Dhoedt
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2017-2018

UNIVERSITEIT
GENT

Gent, juni 2018

De promotor | De auteur

Prof. dr. Jan Cnops | Ruben Vervust

Data-analyse en visualisatie van verkeersdata

Ruben Vervust

Promotoren: prof. dr. Jan Cnops, dhr. Hans Fraiponts (Digipolis)

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Bart Dhoedt
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2017-2018

UNIVERSITEIT
GENT

Woord vooraf

Dit eindwerk betekent het einde van mijn vier jaar als student industrieel ingenieur. Vier jaar vol met behulpzame docenten en enthousiaste medestudenten. Stuk voor stuk personen zonder wie dit eindwerk niet zou gerealiseerd kunnen zijn. Uit deze grote groep mensen zijn er nog een paar personen die ik persoonlijk zou willen bedanken.

Ten eerste wil ik mijn promotor, prof. Jan Cnops bedanken voor de tweewekelijkse consultaties en de goede raad tijdens het uitvoeren van het project.

Als tweede wil ik Hans Fraiponts bedanken voor het mogelijk maken van dit eindwerk en het zorgen voor de nodige resources, inclusief het nodige duwtje in de rug hier en daar. Ook Frederik Waeyaert wil ik bedanken voor de expertise over de Azure Cloud.
Daarnaast ook veel dank gericht aan Pieter Morlion, voor de steun, ideetjes en databronnen vanuit het mobiliteitsbedrijf Gent.

Tenslotte wil ik mijn ouders bedanken voor de vierjarige financiële en morele steun. Zowel in leuke en lastige tijden. Ik wil hen oprecht bedanken voor de kansen die ik heb gekregen.

Abstract

Deze masterproef heeft als doel het verwerken van verkeersdata in en rond Gent. Initieel worden de data gedumpt in bestandsformaten die moeilijk bruikbaar zijn. Een belangrijk aspect is het omzetten van de data naar een uniforme dataset. Ze moeten zowel uniform toegankelijk zijn, als een gelijkaarde structuur hebben. Daartoe worden verschillende ETL-processen vergeleken en verschillende databaseopties aangekaart. De masterproef bestaat uit een drietal afzonderlijke onderdelen. Ten eerste het centraal verzamelen van de data en deze converteren naar een uniforme structuur. Daarna moet dit volledige proces, in de mate van het mogelijke, geautomatiseerd worden. Tenslotte moeten deze data verwerkt worden en eventuele anomaliën worden gedetecteerd. Indien mogelijk kunnen aan deze resultaten ook bepaalde visualisaties gekoppeld worden. Het uiteindelijke resultaat wordt dan een platform, waarin de data centraal toegankelijk zijn, en makkelijk verwerkbaar zijn voor verdere analyses.

Traffic data analysis and visualization

Ruben Vervust

Supervisor(s): Jan Cnops, Hans Fraiponts

Abstract—**In this article the process of creating a coherent data set from different input sources will be described. A number of sources provide information concerning the traffic state in and around Ghent. The data will be processed and transformed to provide meaningful information about travel times, road alerts and average speeds in Gent.**

Keywords—**Traffic, Ghent, Cosmos Database, Azure Cloud**

I. Introduction

IN Ghent a lot of traffic control is done digitally. A lot of sources provide digital (sensor-) data to officials who monitor the state of traffic. Currently this is done mostly in real-time. As seen at the website *verkeer.gent* [1]. A digital, cloud based platform handles as much as possible traffic issues automatically. When needed, the system notifies an official who takes over. This happens in unexpected cases where severe accidents occur.

All this data is currently just processed real-time for the above use case. They noticed that this is a huge waste of information and started saving said data to a cold storage. However, the data is not processed any further yet.

This project will mainly use cloud instances from the Microsoft Azure cloud. Multiple *Software as a Service* (SaaS) stacks will be considered in this paper.

In this abstract, we will first picture the usable data and process it for further use. Used technologies and software stacks will also be explained.

II. Dumping real-time data

A. Azure Blob Storage

The *Microsoft Azure Blob Storage Account* is a handy storage platform with an extensive API. It's low cost and easy to use. In this project it serves as a cold, not frequently accessed, storage for the data.

B. Azure Functions

Azure Functions are executable programs or scripts that can be uploaded into the cloud. They can be triggered by events, manual or by a timer.

C. Using the Blob Storage as a datadump

How can real-time data be used to create a historical data set? The combination of the Azure Functions and Blob Storage can help in this matter. For each different data provider a new script is uploaded to the Azure Functions cloud. Each script performs a *http-request* agains the real-time API and dumps the result on the storage account. There the data can be accessed at a later data for further processing

III. Azure Cosmos Database

The database itself is at the core of this project. The Azure Cosmos database is used for this use case, instead of using a monolithic data warehouse approach. The Cosmos database is fully scalable, distributed and responsive [2]. The possibility to elastically scale this database makes it very useful. The database can handle the processing of data and accessing of data at the same time when enough resources are available. This means that the database should never be offline.

IV. Processing data

A. Creating an uniform dataset

The datasets from different providers are all processed in similar ways. First the raw data is imported in the database using an *Azure Data Factory* (ADF). To use the ADF one has to configure a source, a sink and configure the dataset. The input can be in a JSON-format and the output can be a XML-format. It doesn't matter, as long as the ADF can recognise the data type. A preliminary transformation is done using the ADF while the data is copied to the Cosmos database.

Once the data is copied, it is transformed to a generic dataset using custom scripts and programs. Once this step is completed the data from all different sources should be indistinguishable from each other.

B. Predicting missing data

Not all data sets are perfect. Sometimes data is missing. To handle this issue a prediction is made using the *Apache Spark Machine Learning module*. Each available record is giving a label according to speed: *below average*, *average* and *above average*. Using this label and the day of the week and hour of the day as features, a *decision tree* model is created. A simple example is shown in figure 1. This model can be used to classify a messing speed record when date and time are known. Using only these two features a model with an error rate of 30-40% was trained. This is still a very big rate of error. More features, such as weather and nearby events, can be used to minimize the error.

C. Filtering data

Not all data should be blindly considered as without error. A sensor can malfunction, someone can enter a wrong value or a network error could cause corruption. In al these cases a wrong value could be inserted in the database. To detect these errors, a records value is compared to a specific calculated average value. If the value differs too much from that average additional actions should be taken. It is completely possible it's a faulty value, but the anomaly can also have another source. For example an accident could also cause strange values in the data set.

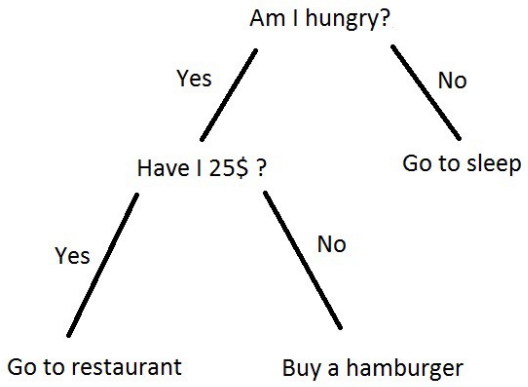


Fig. 1. Simple example of a decision tree model

V. Publishing the resulting data

The data will be published in three different ways. The first option is to interface with the database directly and access the data. A second option will allow users to get the data they need by using a *REST API*. These first two options tend to appeal to the more advances users with programming backgrounds. The third options allow users to interface with the database using Microsofts PowerBI desktop tools. It is a graphical interface allow users to create visual diagrams by simple dragging and dropping records [3].

VI. Conclusion

The NoSQL approach is also a lot cheaper than its SQL counterparts. The resulting dataset allows users to easily access and process the traffic data. It bundles a lot of data sources that do not appear to coincide at first glance.

References

[1] *Dashboard Verkeerscentrum Gent*, https://verkeerscentrumgent.waylay.io/dashboard
[2] Microsoft, *Welcome to Azure Cosmos DB*, https://docs.microsoft.com/en-us/azure/cosmos-db/introduction
[3] Microsoft, *What is Power BI Desktop?*, https://docs.microsoft.com/en-us/power-bi/desktop-what-is-desktop

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van codefragmenten

Lijst van afkortingen

ETL	Extract, Transform, Load
SDK	Software Development Kit
SaaS	Software as a Service
PaaS	Platform as a Service
IaaS	Infrastructure as a Service
SSD	Solid State Drive
RU	Request Unit
POCO	Plain Old CLR Object
SQL	Structured Query Language
API	Application Programming Interface
AWS	Amazon Web Services
JAR	Java Archive

Hoofdstuk 1

Inleiding

1.1 Probleemstelling

Stad Gent beschikt over een aantal digitale datasets omtrend de verkeersdata. Tellusen op de stadsring, parkeerbezetting op straat en in de ondergrondse parkings, fietsentellingen,... Ook worden een aantal datasets aangekocht van externe bedrijven. Een eerste is Coyote, dit bedrijf levert realtime reistijden aan op de stadsring. Daarnaast maakt men ook gebruik van de data geleverd door Be-Mobile, een organisatie die zich specialiseert in het verzamelen van allerhande verkeersdata. Met behulp van deze datasets zou men in staat moeten zijn om het verkeer in en rond Gent te visualiseren. Voor realtime toepassingen is dit reeds het geval. Onlangs is immers de site *https://verkeer.gent/* in het leven geroepen. Op deze site kunnen inwoners en bezoekers van Gent in één oogopslag de huidige verkeersituatie bekijken.
Voor historische verkeersanalyses voldoet de huidige dataset echter nog niet. Er zijn immers enkel data over de huidige toestand en niet elke dataset is makkelijk te koppelen aan elkaar. Momenteel worden een aantal rapporten aangemaakt in programma's als Microsoft Excel. Dit gebeurt manueel en is zeer arbeidsintensief.

1.2 Doelstelling

Onderzoeksvraag

De bedoeling van deze masterproef is om eens af te tasten wat de mogelijkheden zijn van bepaalde *big data* oplossingen toegepast op de verschillende datasets van het verkeer in Gent. Er moet als het ware een antwoord gezocht worden op de vraag: **Hoe kan men alle verschillende datasets op een efficiente manier bundelen en verwerken tot een eenduidige en complete database van verkeersdata waarop zinvolle analyses kunnen worden uitgevoerd?**

Opbouw van het project

In grote lijnen zijn er drie zaken die aan bod zullen komen. Ten eerste moeten de data vanuit verschillende bronnen worden verzameld zodat deze dan kunnen worden verwerkt. Ook moet een oplossing worden gezocht voor het verkrijgen van historische data. Ten tweede zullen de data getransformeerd worden en zullen eventuele inconsistenties moeten worden aangepakt. Tenslotte zullen de data worden verwerkt tot een bruikbare dataset voor de eindgebruiker.

1.2.1 Verzamelen van de data

Centrale toegang

Een eerste probleem is het verzamelen van de verschillende data. Deze data zijn afkomsten van verschillende bronnen: externe online API's, open data van Gent, handmatig ingevoerde tellingen, aangekochte data die maandelijks worden aangeleverd... Het is dan ook een hele opgave om die datasets te verzamelen en centraal aanspreekbaar te maken.

Historische data

De aangeleverde data is vooral realtime, met uitzondering van Be-Mobile. Dit heeft als gevolg dat er geen historische data worden opgeslagen. Dit is natuurlijk een probleem, om analyses te kunnen uitvoeren is het nodig om historische data aant te spreken. Er moet dus een oplossing voor dit probleem worden gezocht.

1.2.2 Transformeren en filteren van de data

Eens de data allemaal beschikbaar zijn op een centraal toegankelijke opslagplaats moeten deze nog verwerkt worden. Door het feit dat deze data aangeleverd worden door veel verschillende instanties, zien deze er dan ook compleet anders uit. De ene dataset is in JSON-formaat, de andere in CSV- of XML-formaat... In een volgende stap moeten deze data dan getransformeerd worden naar een bruikbaar, uniform standaardformaat.
De indeling van de data is niet het enige probleem. Ook is niet elke dataset even betrouwbaar. Ze bevatten soms een aantal fouten: verkeerd geijkte sensoren, sensoren die niet werken, onnauwkeurigheden... Het detecteren en elimineren van dergelijke fouten vormt een extra opgave voor het bruikbaar maken van de verschillende datasets.

1.2.3 Verwerken van de data

Tenslotte moeten de data verwerkt worden. Het resultaat wordt een bruikbare dataset die een totaalbeeld geeft van de verkeersituatie. De database moet verwerkte data bevatten die eenvoudig kan begrepen worden zonder al te veel documentatie te moeten doornemen. Deze dataset moet zowel bruikbaar zijn voor gebruikers met programmeerervaring, analisten of

andere eindgebruikers zonder informaticakennis. De voornaamste eindgebruiker is dan ook Stad Gent zelf, die de data kunnen gebruiken om het wegennet nog efficienter in te delen.

Hoofdstuk 2

Eindresultaat

2.1 Doelgroepen

Er zijn een aantal verschillende soorten eindgebruikers waar rekening mee moet gehouden worden. Elk van deze eindgebruikers heeft een aantal verschillende use cases. Niet iedereen is immers geïnteresseerd in dezelfde data en niet iedereen heeft dezelfde programmeerervaring.
Een eerste doelgroep zijn de inwoners van Gent zelf. Voor hen zijn gemiddelde reistijden naar het werk of de drukte van de wegen in hun buurt interessante gegevens.
Vervolgens kan het ook nuttig zijn om bepaalde datasets als open API beschikbaar te maken voor andere onderzoekers of programmeurs. Zij kunnen dan de bekomen data gebruiken in hun onderzoeken of applicaties zonder de datasets zelf nog te moeten ophalen en te bewerken. Hiervoor kan het platform van open data Gent gebruikt worden.
Tenslotte is dit gegeven ook zeer interessant voor het mobiliteitsbedrijf van Gent. Dit is de belangrijkste doelgroep. Ze leveren de ruwe data aan en bepalen ook wat wel en niet publiek gezet kan worden. In dit geval is een *data warehouse* een interessant gegeven. Hiermee kunnen er grondige analyses worden uitgevoerd op de grote hoeveelheid beschikbare data.
Om elke eindgebruiker te kunnen voorzien van een gepast eindproduct zijn dus verschillende technologieën nodig die elk zullen focussen op een speciek aanleverplatform voor de verschillende data.

2.2 Eindproducten

2.2.1 Data Warehouse

Het uiteindelijke product is geen echte data warehouse geworden. Dit komt eerst en vooral door de hoge prijs om dergelijke platformen te onderhouden. Ook lenen de data zich eerder naar een goedkopere NoSQL variant. Er wordt gebruikgemaakt van de Microsoft Azure Cosmos DB (Microsoft, 2017a). De Cosmos DB werd gekozen omdat dit een volledige oplossing is in de Microsoft Azure cloud. Digipolis maakt immers grotendeels gebruik van deze

cloudtechnologie. Ook omdat de SQL-syntax grotendeels ondersteund wordt, hierdoor kan men toch nog steeds klassieke queries gebruiken om data te zoeken in de documentdatabase. Het resultaat is dus geen volwaardige *data warehouse*, maar wel een database die meer dan voldoende is om de doeleinden van dit project te verwezenlijken.

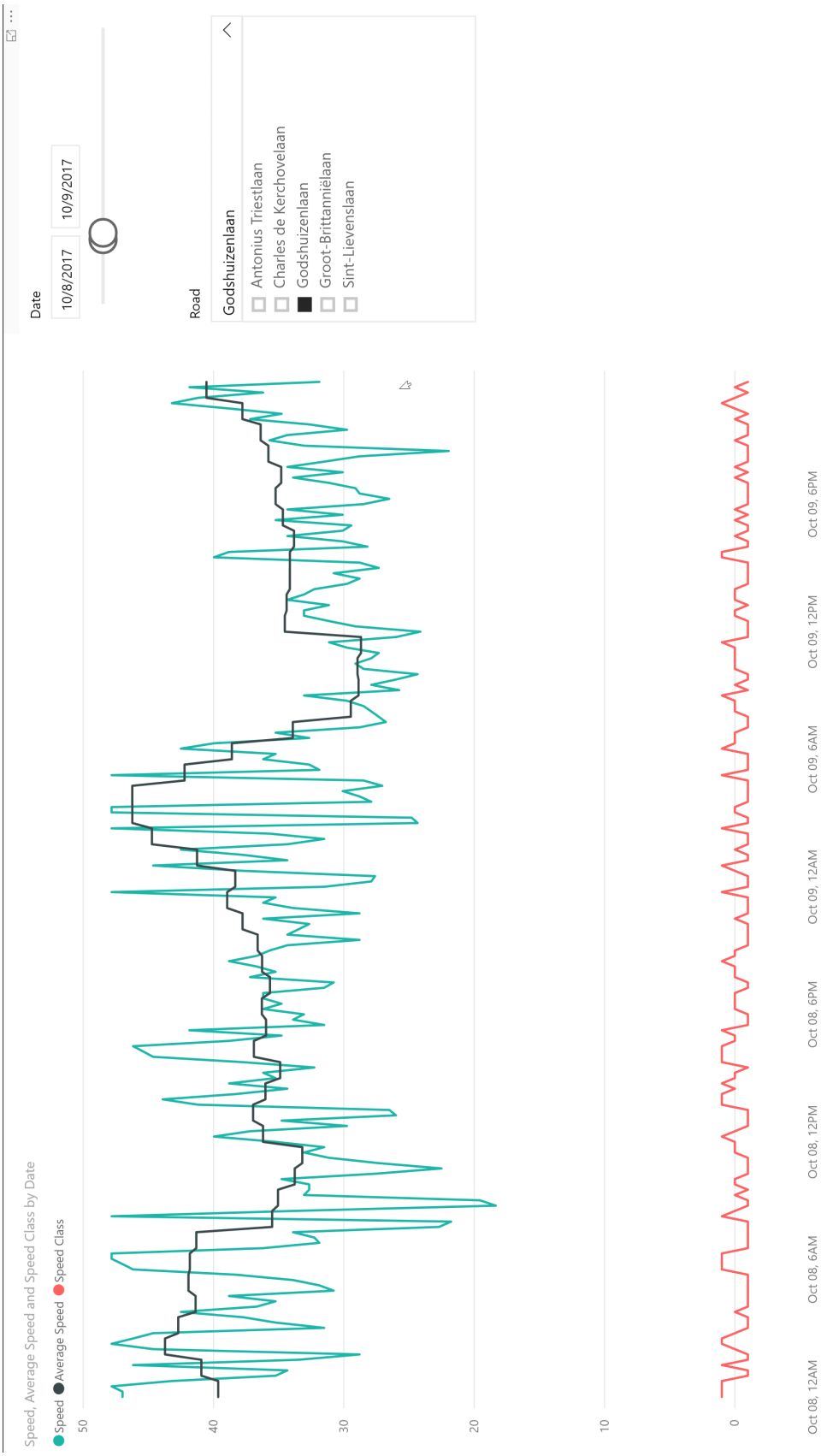
2.2.2 Open Data / Rest API

Stad Gent stelt een aantal datasets ter beschikking die raadpleegbaar zijn voor iedereen. Deze zijn te vinden op de site van stad Gent. Het is interessant om een aantal datasets uit dit project beschikbaar te maken op dit portaal. Met een GET-request naar een vaste url in dit dataportaal wordt achter de schermen een query uitgevoerd op de hierbovenvermelde Cosmos DB. Het resultaat is een JSON-document dat de opgevraagde informatie bevat.

Figuur 2.1: Open data portaal Gent

2.2.3 Microsoft PowerBI

PowerBI is een tool van Microsoft die gebruikers in staat stellen om grote datasets op een eenvoudige manier te verwerken tot prachtige visualisaties zoals grafieken, taardiagrammen, histogrammen... PowerBI is zo aantrekkelijk omdat het visualiseren van *business intelligence* zo eenvoudig maakt. Het zal ook mogelijk zijn om via een connector direct verbinding te maken met de verkeersdata in de Cosmos database. Op die manier kunnen organisators interfacen met de database en zelf handige visualisaties, zoals in figuur 2.2, maken.
Zo zal het mobiliteitsbedrijf ook hoogstwaarschijnlijk met PowerBI werken om een handig dashboard op te zetten met historische verkeersdata.



Figuur 2.2: PowerBI visualisatie voor verkeersdata in oktober 2017

Hoofdstuk 3

Mobiliteitsdata

3.1 Opslag in de cloud

Er zijn een groot aantal verschillende datasets, afkomstig uit verschillende bronnen. Het Mobiliteitsbedrijf Gent heeft zelf een groot aandeel aan datasets ter beschikking. Sommige komen van eigen sensordata, anderen worden aangekocht van externe partners. In het kader van deze masterproef stellen zij dan ook deze data beschikbaar.
De meeste bronnen zijn echter real-time API's, hiermee kunnen natuurlijk geen historische analyses gedaan worden. Om dit probleem te overbruggen wordt gebruikgemaakt van een datadump in de Azure cloud. Digipolis is reeds een tijdje bezig met het opslaan van de realtime datasets. Er wordt automatisch een call gedaan naar de betreffende API's, het resultaat wordt vervolgens opgeslagen in de cloud. Deze calls gebeuren met een gemiddelde frequentie van 1 call elke 5 minuten, dit voor elke API. Deze datadump kan dus gebruikt worden om historische data op te vragen.

3.2 Databronnen

In dit onderdeel worden de verschillende datasets kort aangekaart. Dit zijn de datasets zoals ze worden aangeboden. Vooraleer de data bruikbaar zijn voor dit project zullen nog een aantal aanpassingen moeten gebeuren.
Enkel van de meest relevante datasets worden een voorbeeld getoont in dit hoofdstuk. In appendix A zullen alle datasets ook nog eens worden opgelijst.

3.2.1 Be-Mobile

Informatie Be-Mobile is een extern bedrijf die zelf ook data verzamelt over het verkeer in steden. Stad Gent koopt deze data aan. Het is de meest complete dataset.

Formaat CSV

Voorbeeld

```
Traject;timestamp;TravelTimeMs
B_OW;01/06/2017 00:00;85779
B_OW;01/06/2017 00:15;77056
B_OW;01/06/2017 00:30;73282
B_OW;01/06/2017 00:45;63459
```

3.2.2 Reistijden Coyote

Informatie Deze dataset bevat informatie over de reistijden op de stadsring.

Formaat JSON

Voorbeeld

```
{"Gand":{
   "Rooigemlaan (R40) Northbound - Drongensesteenweg -
      Palinghuizen":{
      "normal_time":40.2,
      "real_time":40.2,
      "diff_time":0,
      "length":1340,
      "geometries":[
         [
            {
               "lng":3.69508,
               "lat":51.05617
            },
            {
               "lng":3.69487,
               "lat":51.05631
            },
            ...
         ]
      ],
      "sections":[

```

```
22        ],
23        "alerts":[
24
25        ]
26    }}}
```

3.2.3 Tellussen Agentschap Wegen en Verkeer

Informatie Deze dataset bevat het aantal wagens die over bepaalde meetpunten rijden in Gent. Deze worden onderhouden door het Agentschap Wegen en Verkeer, verder AWV genoemd en worden opgengesteld voor gebruik(AWV, 2017). De tellussen geven informatie weer over het aantal wagens en hun gemiddelde snelheid.

Formaat JSON

Voorbeeld

```
1  {
2     "type":"FeatureCollection",
3     "features":[
4        {
5           "type":"Feature",
6           "geometry":{
7              "type":"Point",
8              "coordinates":[
9                 3.7202457700359748,
10                51.041201364632059
11             ]
12          },
13          "properties":{
14             "attributes":[
15                {
16                   "attributeName":"speed",
17                   "value":35
18                },
19                {
20                   "attributeName":"OCC",
21                   "value":1
22                },
```

```
                {
                    "attributeName":"Count",
                    "value":60
                },
                {
                    "attributeName":"Timestamp",
                    "value":1491004635
                }
            ],
            "contextEntity":"M004"
        }
    },
        ...
  ],
  "EventProcessedUtcTime":"2017-04-01T00:00:00.7027260Z",
  "PartitionId":0,
  "EventEnqueuedUtcTime":"2017-04-01T00:00:02.4120000Z"
}
```


3.2.4 Bluebike

Informatie Bluebike is een initiatief van de NMBS, De Lijn, B-Parking, FIETSenWERK en Tec. Een aantal fietsen worden op gemakkelijk te bereiken locaties geplaats. Via het systeem van Bluebike kunnen gebruikers deze fietsen dan gemakkelijk lenen voor een bepaalde periode(Bluebike, 2018). Bluebike stelt een API ter beschikking die per fietslocatie het aantal fietsen weergeeft. In Gent zijn dit twee locaties: Sint-Pietersstation en Gent Dampoort.

Formaat JSON

3.2.5 iRail

Informatie IRail stelt een groot aantal datasets open voor gebruik(iRail npo, 2011). Voor dit project wordt gefocust op de vertrekuren van de treinen in het Sint-Pietersstation. Elke vertrekkende trein wordt opgeslagen met eventuele vertragingen.

Formaat JSON

3.2.6 Parkeergarages Stad Gent

Informatie Via het open data portaal van stad Gent kan men de status van de parkeergarages raadplegen. De datasets bevatten onder meer informatie over het aantal vrije plaatsen, totaal aantal plaatsen en openingsuren van de parkeergarages.

Formaat JSON

3.2.7 Waze

Informatie Waze is een mobiele applicatie waarop gebruikers zelf kunnen aangeven waar er zich files bevinden op de weg(Waze Mobile, 2018). Deze op de community gebaseerde app stelt zijn eigen data open naar ondermeer nieuwskanalen en steden. De dataset bevat vooral data over files en opstoppingen verdeeld in verschillende categorieën.

Formaat JSON

Voorbeeld

```
{
   "usersOnJams":[]
      {
         "wazersCount":0,
         "jamLevel":0
      },
      {
         "wazersCount":0,
         "jamLevel":1
      },
      {
         "wazersCount":0,
         "jamLevel":2
      },
      {
         "wazersCount":0,
         "jamLevel":3
      },
      {
         "wazersCount":0,
         "jamLevel":4
```

```
        }
    ],
    "routes":[
        {
            "historicTime":430,
            "line":[
                {
                    "x":3.672124,
                    "y":51.08643799999998
                },
                                ...
            ],
            "bbox":{
                "minY":51.013156941216529,
                "minX":3.6558570000000006,
                "maxY":51.08643799999998,
                "maxX":3.728930471796887
            },
            "length":11408,
            "type":"STATIC",
            "jams":[

            ],
            "alerts":[

            ],
            "toName":"Sluisweg",
            "name":"Buitenring-Drongen (R4) southbound",
            "fromName":"Industrieweg",
            "jamLevel":0,
            "id":4096,
            "time":433
        },
    ],
    "irregularities":[]

    ],
    "broadcasterId":147,
    "areaName":"Belguim, Gent",
```

```
    "bbox":{
        "minY":51.007618616290713,
        "minX":3.6749267578125,
        "maxY":51.096966571798838,
        "maxX":3.7937164306640625
    },
    "name":"Verkeerscentrum Gent",
    "isMetric":1,
    "restrictions":{}

    },
    "lengthOfJams":[]
        {
            "jamLevel":1,
            "jamLength":0
        },
        {
            "jamLevel":2,
            "jamLength":0
        },
        {
            "jamLevel":3,
            "jamLength":0
        },
        {
            "jamLevel":4,
            "jamLength":0
        },
        {
            "jamLevel":5,
            "jamLength":3702
        }
    ],
    "updateTime":1496277282266,
    "EventProcessedUtcTime":"2017-06-01T00:35:10.1622563Z",
    "PartitionId":0,
    "EventEnqueuedUtcTime":"2017-06-01T00:35:09.5450000Z"
}
```

Hoofdstuk 4

Cloudtechnologieën

4.1 Inleiding

Cloudapplicaties zijn nog steeds aan een hevige opmars bezig. Bij Digipolis wordt alvast meer en meer gebruikgemaakt van cloud-gebaseerde systemen. Deze trend heeft niet alleen de keuze voor het gebruik van de cloud beïnvloed, maar ook het cloudplatform bepaald. Doordat er in het bedrijf reeds een contract aanwezig was met Microsoft Azure leek dit de meest vanzelfsprekend keuze. Om de historische datasets te creëren wordt gebruikgemaakt van een aantal *Azure Functions*, vervolgens worden de bekomen data opgeslagen op een *Azure Blob storage*. In deze storage worden ongewijzige JSON -of XML-bestanden op datum gerangschikt. Als databasesysteem werd gekozen voor de Azure Cosmos database. Dit is een NoSQL clouddatabase oplossing in de Azure Cloud. De brug tussen de data in de blob-storage en de Cosmos database wordt gelegd met behulp van een *Data Factory*. De backend bestaat dus grotendeels uit deze onderdelen. De verschillende bouwstenen worden verder in dit hoofdstuk besproken.

4.2 De Azure Cloud

Microsoft Azure, voorheen Windows Azure, werd in 2008 aangekondigd door Microsoft en was in 2010 wereldwijd beschikbaar(Sanders, 2017). Het is Microsofts antwoord op cloudplatformen als die van Google of Amazon.

Figuur 4.1: Microsoft Azure

Vooral de laatste jaren is Microsoft Azure aan een stevige opmars bezig. Meer en meer bedrijven maken de overstap naar de cloud. Microsofts cloudplatform is op dit moment de leider in het aanbieden van zowel *Software as a Service* (SaaS), *Platform as a Service* (PaaS) en *Infrastructure as a Service*(IaaS)(Team Trefis, 2016). In tabel 4.1 wordt de omzet per cloudprovider eens opgesomd zoals ondervraagd door de nieuwssite ZDNet (Dignan, 2018). De tabel maakt meteen duidelijk dat Microsoft Azure en Amazon Web Services (AWS) aan de absolute top staan wat betreft cloudgebaseerde applicaties.

4.3 Microsoft Azure Cosmos Database

De clouddatabase van Microsoft onderscheidt zich van andere databases door de grote verscheidenheid aan modellen die worden ondersteund. Zowel key-value stores, document database, tabelgebaseerde databases en graafgebaseerde modellen worden ondersteund. Het is volgens Microsoft het antwoord en een beter alternatief op veel andere NoSQL-softwarepakketten. De documentatie van Microsoft laat uitschijnen dat het goed is in alles. Zowel consistentie, beschikbaarheid en partitionering zijn volgens Microsoft troeven van dit databasemodel(Asay, 2017).

4.3.1 Kenmerken

De Azure Cosmos database bevat een hele reeks aan kenmerken en functionaliteiten. Verder in dit onderdeel worden de belangrijkste kenmerken eens op een rijtje gezet (Microsoft, 2017b).

Globale Distributie Via het administratiepaneel van de Azure Cosmos database heeft men een groot aantal opties om de data te repliceren naar verschillende datacenter wereldwijd. Met een paar muisklikken is dit ingesteld en wordt de replicatie volledig automatisch georganiseerd achter de schermen. Dankzij de *multi-homing API's* wordt altijd het dichtsbijzijnde datacenter geselecteerd en wordt op die manier de opvraagtijd van data tot een mimimum behouden.

Cloud Provider	Jaarlijkse omzet
Microsoft Azure	$21,2 miljard
Amazon Web Services	$20,4 miljard
IBM	$10,3 miljard
Oracle	$6,08 miljard
Google Cloud Platform	$4 miljard
Alibaba	$2.2 miljard

Tabel 4.1: Jaaromzet cloudproviders 2017

Verschillende databasemodellen en API's Zoals reeds vermeld ondersteunt de Cosmos database een breed gamma aan database modellen. Is het nu key-value, document, table of graph, het maakt voor het systeem niet veel uit. Ook de database API is zelf te kiezen. Dit zorgt voor een hoge compatibiliteit met bestaande applicaties. Als men bijvoorbeeld een MongoDB applicatie heeft, is het mogelijk om zonder enige aanpassing om te schakelen naar Cosmos. De MongoDB API wordt volledig ondersteund. Een aantal andere mogelijkheden zijn: SQL, Cassandra en Table.

Elasticiteit en schaling Zoals bij de meeste cloudtoepassingen is schaling van groot belang. Een beginnende database heeft misschien geen nood aan terabytes opslag en zware rekenkracht, maar na verloop van tijd kan dit wel het geval worden. De Cosmos database laat de gebruiker volledig vrij in het kiezen van opslag en rekenkracht. Men betaalt enkel wat men effectief gebruikt.

4.3.2 Kritiek

Azure Cosmos belooft veel, volgens sommige developers te veel. Er kunnen een aantal bedenkingen worden gemaakt. Ten eerste, het systeem ondersteunt een groot aantal databasemodellen. Hoe efficient is de implementatie van deze modellen dan (Asay, 2017)? Tenslotte kan het CAP-theorema (Brewer, 2000) in dit geval ook worden toegepast. Het CAP acroniem staat voor:

Consistentie (Consistency) In welke mate het volledige databasesysteem dezelfde data bevat na een update.

Beschikbaarheid (Availability) In welke mate het systeem toegankelijk blijft tijdens operaties of na het crashen van bepaalde nodes

Partitionering (Partitioning) De verdeling van de data over verschillende nodes

Brewers theorema stelt dat slechts aan hoogstens twee van bovenstaande eigenschappen kan worden voldaan. Bij Amazons Dynamo database wordt geopteerd voor een hoge beschikbaarheid en hoge partitioneringsgraad. Het geeft echter geen zekerheden over de consistentie van de database. Als tweede voorbeeld is er Googles Bigtable database. Hier is de consistentie en beschikbaarheid groot, maar werkt het systeem niet zo goed in een gepartitioneerde netwerkomgeving (De Tré & Bronselaer, 2017). Toch geeft Microsoft aan dat Cosmos elk van de drie bovenstaande kenmerken tegelijk kan implementeren.

4.4 Azure Blob Storage

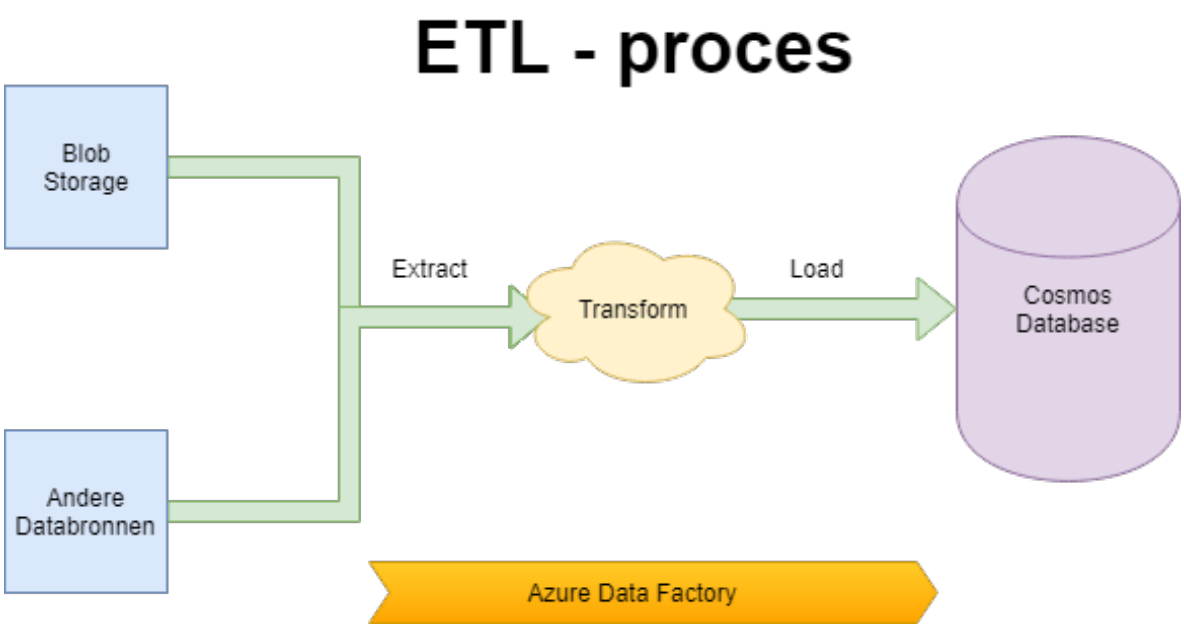
Om een historische dataset te creeëren wordt al langer gebruikgemaakt van een permanente opslagplaats in de cloud. Elke 5 minuten wordt een request gestuurd naar de verschillende

databronnen. Het resultaat wordt zonder enige aanpassing gedumpt op de blob storage. Via een REST API of via verscheidene Software Development Kits (SDK's) zijn de blobs dan zonder al te veel moeite toegankelijk voor developers(Bisson, 2017).

4.5 Azure Data Factory

Er is nood aan een mechanisme om de data vanuit de blob storage te exporteren naar de database. Ook mogen de data niet zomaar gekopieerd worden. De data in hun ruwe vorm zijn immers niet bruikbaar om queries op uit te voeren. Gelukige bevat de Azure Cloud hier ook een oplossing voor. Azure Data Factory stelt gebruikers in staat om pipelines aan te maken die handmatig, of door middel van bepaalde triggers bepaalde data van de ene bron over te zetten naar een andere locatie. Het is een vorm van een Extract, Transform, Load (ETL) proces. In figuur 4.2 wordt het ETL-proces weergegeven zoals hier toegepast.

Figuur 4.2: ETL-proces



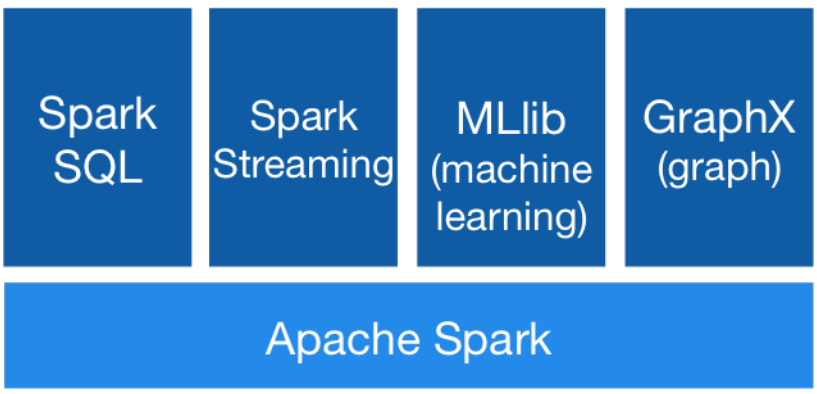
Hoofdstuk 5

Apache Spark

5.1 Inleiding

Het verwerken van grote hoeveelheden data is niet vanzelfsprekend. Naast het kopiëren en normalizeren van de data is het ook nodig om bepaalde aggregaties uit te voeren. Hier komt Apache Spark, verder Spark genoemd, van pas. Spark maakt het mogelijk om de werklast van dergelijke geheugen -en processorintensieve taken te verdelen over verschillende nodes. Spark is een gegevensverwerkingssysteem die gebruik maakt van clusters, het is snel en betrouwbaar. Het framework biedt interfaces voor zowel Java, Python en Scala. De ontwikkeling van Apache Spark had het versnellen van big data processing als doel. In de meeste gevallen is men daar ook in geslaagd. Voor veel use cases is Spark sneller dan zijn grote broer, Apache Hadoop. Een van de grote oorzaken van deze snelheidswinst is het ondersteunen van *in-memory* gegevensverwerking, in tegenstelling tot *disk-based* oplossing als Hadoop (Shoro & Soomro, 2015).
Ook implementeert Spark een aantal *high-level tools*, bijvoorbeeld *MLlib* voor machine learning, *Spark SQL*, *GraphX* voor graafbewerkingen en *Spark Streaming*. Een overzicht is te zien in figuur 5.1.

Figuur 5.1: De Spark stack



5.2 Apache Spark MLlib

Een van de grootste voordelen van Apache Spark is de brede waaier aan reeds geïmplementeerde tools. Machine learning is daar één van en is ook de reden waarom voor Spark werd gekozen in dit project. Daar wordt later verder op ingegaan.

5.2.1 Kernfuncties

In dit onderdeel worden een aantal functies en mogelijkheden van deze module opgelijst. Extra informatie is te vinden in de MLlib documententatie (MLlib, 2018).

Gegevensstructuren en algoritmen
: De library voorziet een reeks snelle en efficiente implementaties van veelgebruikte algoritmes. Een aantal van deze algoritmes zijn: naive Bayes, verschillende implementaties van beslissingsbomen voor classificatie, k-means clustering en verscheidene modellen voor dimensiereductie. Data kunnen aangevoerd worden via verscheidene kanalen: Spark SQL, Spark datastreaming en een aantal externe API's

Slimme optimalisaties
: Spark staat niet meer in zijn kinderschoenen en vele algoritmes hebben reeds een aantal herzieningen gekend. Telkens met een hogere efficiëntie tot gevolg. Spark maakt ten volle gebruik van de Java en Scala functies om maximale performantie te garanderen (Meng *et al.*, 2016).

Pipeline API
: Zelden bestaat het aanpassen en prepareren van een dataset voor verdere verwerking uit één enkel stap. Meestal is het nodig om een aantal transformaties toe te passen om de dataset te kunnen gebruiken voor het trainen van bijvoorbeeld een classificatiemodel. Juist om deze dataflow makkelijker te kunnen implementeren maakt Sparks implementatie gebruik van pipelines. Op die manier kunnen opeenvolgende stappen gedefinieerd worden en op een efficiente manier worden uitgevoerd. Een voorbeeld van dit proces wordt voorgesteld in codefragment 5.1. Eerst worden twee pipelines gedefinieerd (regels 1-5) gevolgd door de methode voor classificatie (regel 10). Eens het model voor elke stap in orde is, worden de drie stappen meegegeven aan de pipeline (regel 13). Om het model vervolgens te trainen wordt de pipeline uitgevoerd (regel 16), eerst worden de labels en features geïndexeerd en vervolgens het model zelf opgebouwd aan de hand van de indexen.

Spark integraties

Sparks overvloed aan modules, integraties en externe bibliotheken vormen een enorm voordeel voor elke module. De meeste onderdelen zijn op een zodanige manier geïmplementeerd dat ze zonder al te veel extra configuratie met elkaar kunnen interfacen. In dit project wordt bijvoorbeeld gebruikgemaakt van een extra module om te kunnen communiceren met de Azure Cosmos database in de cloud.

codefragment 5.1: Definitie van een pipeline in Apache Spark MLlib

```
# Define indexes used to create the classificationmodel
labelIndexer = StringIndexer(inputCol="speedClass",outputCol="indexedLabel
    ").fit(data2)

featureIndexer =\
     VectorIndexer(inputCol="features", outputCol="indexedFeatures",
         maxCategories=4).fit(data2)
# Split data in trainingdata and testdata
(trainingData, testData) = data2.randomSplit([0.7, 0.3])

# Define classifier, note that the used columns are the indexed ones.
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="
    indexedFeatures")

# Chain indexers and tree in a Pipeline
pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dt])

# Train model. This also runs the indexers.
model = pipeline.fit(trainingData)
```

Hoofdstuk 6

Configuratie en gebruiksaanwijzing

6.1 Inleiding

In dit project wordt gebruikgemaakt van een aantal verschillende softwarepakketten. In dit hoofdstuk wordt stap voor stap beschreven hoe de verschillende componenten moeten worden geconfigureerd en afgesteld. Eerst zullen alle componenten en hun vereisten worden opgesomd, gevolgd door een beschrijving per component hoe deze moet worden ingesteld.

6.2 Componenten

6.2.1 Azure Cloud

In dit project werd gekozen voor de cloudoplossingen van Microsoft. Vanzelfsprekend kan een soortgelijke softwarestack opgebouwd worden bij een andere cloudprovider.

Data Opslag — Een platform voor dataopslag, voor dit project werd gebruikgemaakt van de Azure Blob Storage.

Data Factory v2 — De datafactory zorgt voor het kopiëren van de data van de dataopslag naar de Cosmos Database.

Cosmos Database — NoSQL database oplossing in de cloud.

App Services — Web host in de cloud

6.2.2 Hulpprogramma's

Niet alle software draait op de Azure cloud. Een aantal programma's dienen op een server-instantie te draaien. Dit is eventueel ook mogelijk in de Azure cloud, maar helemaal geen vereiste. Een aantal programma's zijn geschreven in C# en hebben het .Net Core framework van Microsoft nodig. Daarnaast wordt naar Spark geïnterfaced via Python en is dus ook Python 3 vereist alsook Apache Spark.

.Net Core 2.0 host — Een .Net Core 2.0 compatibele Windows, Linux of Mac OS X instantie met werkende netwerkverbinding.

Apache Spark en PySpark host — Linux instantie met Java 8 en Python 3 en een direct adresseerbaar IP-adres. 8gb RAM-geheugen is aan te raden, maar echter niet vereist. In deze masterproef werd gebruikgemaakt van Spark versie 2.2

6.3 Configuratie van de Azure Cloud

Het afstellen van resources in de cloud is vaak een ingewikkeld proces. In dit hoofdstuk worden de instellingen van elke afzonderlijke instantie aangehaald en verklaard. Om verwarring te vermijden zal naar elk configuratieveld verwezen worden met de Engelstalige naam, zoals die wordt weergegeven op het cloudportaal.

6.3.1 Belangrijke termen

In dit onderdeel worden nog enkele belangrijke termen verklaard die een aantal keer zullen terugkomen in het configureren van de verschillende intanties. Deze zullen verder niet meer verklaard worden, tenzij er specifieke opmerkingen zijn bij een bepaalde cloudinstantie.

Resource Group — Een resource group is een verzameling van instanties van cloudtoepassingen. Typisch worden alle resources voor een bepaald project toegevoegd aan één resource group. Het is dan ook mogelijk om specifieke gebruikersrechten aan een hele resource group te te kennen, in plaats van aan elke resource afzonderlijk (Taylor, 2017).

Resource Replication — Bij elke soort van storage in de cloud is het mogelijk om een graad van replicatie toe te kennen. Hoe meer replicatie, hoe lager de kans dat de data verloren gaan. Toch heeft een hogere replicatiegraad meestal een negatieve invloed op de performantie en vanzelfsprekend de prijs van de resource.

Location — Het primaire datacenter waar de resource zal gehost worden. Hier kiest men dus best voor een datacenter dichtbij de meest waarschijnlijke eindgebruiker om onnodige vertragingen over het netwerk te vermijden. Als er een hoge replicatiegraad is ingesteld, kan deze instelling meerdere datacenters omvatten. Op die manier zijn de data niet verloren als een datacenter faalt.

Subscription — Hiermee wordt het model van facturatie ingesteld. Dit kan zowel als particulier, via kredietkaart of via een enterprise subscription als er in

het bedrijf contracten zijn afgesloten met Microsoft.

Resource Endpoint — Unieke identificatie van de instantie. Bestaat meestal uit een unieke naam gevolgde door een welbepaald Azure domein. Applicaties kunnen deze naam gebruiken om de resource te identificeren en als resource-URL.

6.3.2 Configuratie van het Storage Account

Deze instantie houdt alle ruwe data bij. De returnwaarden van de periodieke calls naar de verschillende API's worden hierop gedumpt zonder ze te verwerken. Een screenshot van de interface is te zien in figuur 6.1.

Tabel 6.1: Verschillende replicatiemodellen

Scenario	**LRS**	**ZRS**	**GRS**	**RA-GRS**
Node binnen één datacenter faalt	Ja	Ja	Ja	Ja
Volledig datacenter is onbeschikbaar	Nee	Ja	Ja	Ja
Volledige regio is onbeschikbaar	Nee	Nee	Ja	Ja
Lees operaties mogelijk in geval van grootschalige black-outs	Nee	Nee	Nee	Ja

Instellingen

Name — Naam van de resource, moet uniek zijn binnen het domein. De naam, gevolgd door het domein van de instantie wordt gebruikt als resource endpoint.

Deployment Model — Azure heeft reeds een aantal verschillende manieren van deployment gekend. De huidige manier, Resource Manager genaamd, ondersteunt bijvoorbeeld het gebruik van resource groups om de resources te groeperen. De klassieke (en verouderde) manier van deployen ondersteunt dit niet. Hoewel deze instelling niet meteen een impact heeft op dit project, wordt ten sterkste aangeraden om gebruik te maken van de Resource Manager (FitzMacken *et al.*, 2017).

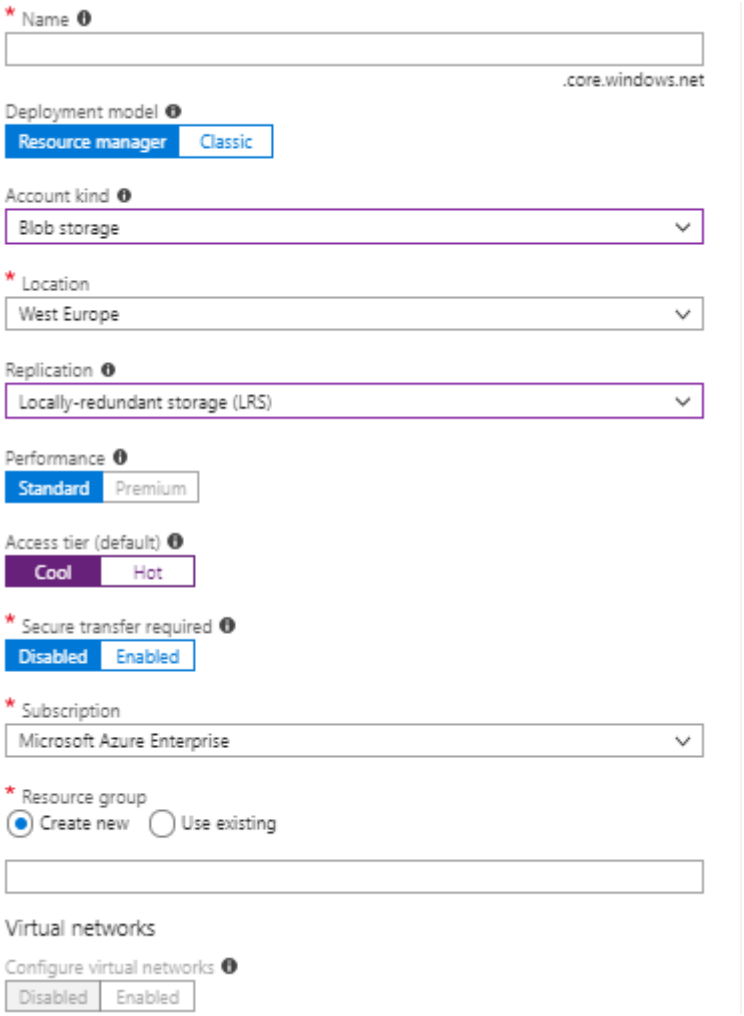
Account kind — Deze instelling bepaalt het type storage. Men kan kiezen tussen general purpose of blob. Elk type heeft zijn voordelen, maar hier wordt gekozen voor de blob storage variant.

Location

Replication — Data in een Azure storage account worden altijd gerepliceerd. Toch kan men aanduiden hoe deze data moeten worden gerepliceerd. Zie

	tabel 6.1 voor de verschillen tussen *locally-redundant storage* (LRS), *zone-redundant storage* (ZRS), *geo-redundant storage* (GRS) en *read-access geo-redundant storage* (RA-GRS) (Myers, 2018).Voor dit project werd gekozen voor LRS.
Performance	Afhankelijk van het gekozen datacenter kan men ofwel kiezen voor standard of premium performance. Premium performance biedt snellere hardware, bijvoorbeeld solid state drives (SSD's).
Access Tier	Deze instelling is afhankelijk van het gebruik van de data. De optie cold is typisch voor data die niet vaak zullen worden geraadpleegd. Hot is dan weer aangeraden voor data die vaak zullen opgevraagd of aangepast worden.
Secure transfer required	Met deze instelling wordt bepaald of alle toegang vanaf niet-beveiligde bronnen moet geblokkeerd worden.
Subscription	
Resource group	

Figuur 6.1: Storage Account Configuratie



6.3.3 Configratie van de Cosmos database

De algemene instellingen van de Cosmos Database komen grotendeels overeen met de configuratie van het Storage Account, beschreven in het vorige onderdeel. Toch zijn er een aantal verschillen. Eerst en vooral moet er een API gekozen worden, deze API bepaald hoe programma's zullen interfacen naar de database. Verder zullen de verschillende opties besproken worden. Ook maakt de database gebruik van collecties in plaats van tabellen. Elke collectie heeft dan ook een aantal eigen instellingen.

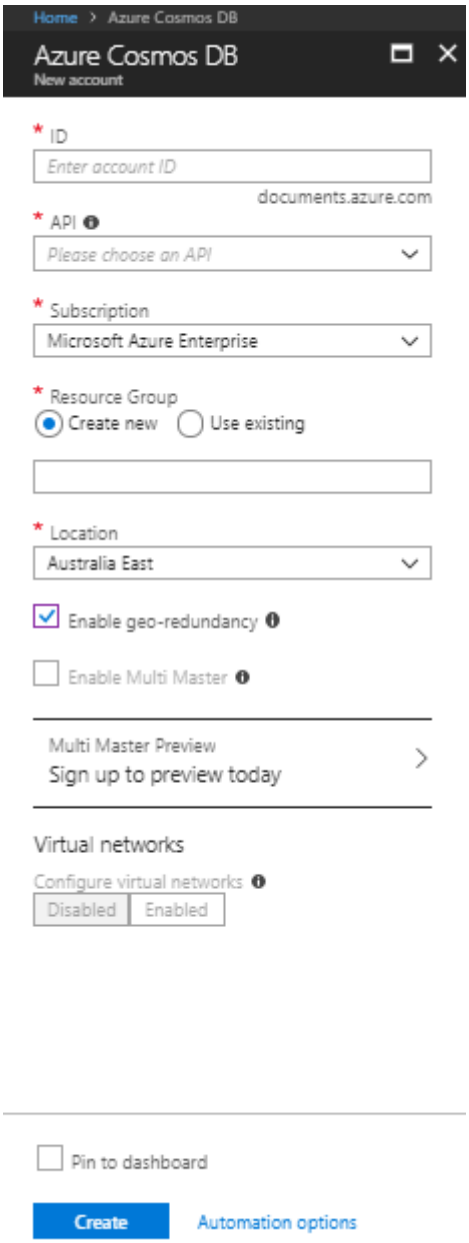
Algemene Instellingen

De interface is te zien in figuur 6.2.

ID
: Unieke identificatie van de database, wordt net zoals de *name* property gebruikt als resource endpoint.

API	Deze instelling bepaald op welke manier andere programma's zullen interfacen met de database. De verschillende opties worden verder in dit hoofdstuk aangehaald. Voor dit project werd gebruikgemaakt van de SQL API, omdat de query-mogelijkheden zeer handig zijn om data te filteren.
Enable geo-redundancy	Het aanvinken van deze optie zorgt ervoor dat de data automatisch worden gerepliceerd naar een secundair datacenter.
Subscription	
Resource group	
Location	

Figuur 6.2: Cosmos Databases Configuratie



Beschikbare data modellen en API's

Cosmos maakt gebruik van een *atomic-record-sequence* (ARS). Elk stukje data wordt omgezet in ofwel een *atom*, *record* of een *sequence*. Door dit systeem toe te passen kunnen op een hoger level een brede waaier aan verschillende databasemodellen worden geïmplementeerd (Microsoft, 2017b).

SQL API Schemaloze JSON database met query-mogelijkheden via SQL. Welliswaar

worden minder functies ondersteund dan bij een klassieke relationele database.

MongoDB API MongoDB-as-a-Service, volledige compatibel met reeds bestaande MongoDB-applicaties.

Cassandra API Cassandra-as-a-Service, compatibel met reeds bestaande Cassandra drivers.

Gremlin API Horizontaal schaalbare graafdatabase, compatibel met Open Graph API's.

Table API Een key-value database.

Configureren van een collectie

Zoals reeds werd aangehaald bestaat een Cosmos Database uit een aantal collecties. Elke collectie kan afzonderlijk geconfigureerd worden met volgende instellingen:

Database id Naam van de database waartoe de collectie zal behoren.

Collection id Unieke naam van de collectie.

Storage capacity Hier kan men kiezen tussen *Fixed (10 GB)* of *Unlimited*. Indien men kiest voor unlimited zal de data over verschillende nodes worden verspreid.

Partition key Deze instelling is enkel van toepassing bij een *unlimited* storage capacity en bepaald op basis van welke variabelen de data zullen worden gesplitst.

Throughput Het aantal *Request Units* (RU) die moet worden toegewezen aan de collectie. RU is een maat voor de processorkracht die een collectie gereserveerd krijgt.

Aangezien veel opvragingen gebeuren op basis van een datum of tijdstip, kan het voordelig zijn om bepaalde collecties extra indexeringsregels mee te geven. Cosmos voert zelf al een aantal indexeringen uit om de data vlotter toegankelijk te maken, maar die automatisch gegenereerde regels kan men zelf aanpassen. Indexering wordt bepaald door de *Indexing Policy*, die is te vinden onder de *Scale & Settings* tab bij elke collectie. Door het JSON-bestand, zie codefragment 6.1, aan te passen kan je indexeringsregels toevoegen of verwijderen. De meeste indexeringsregels zijn vanzelfsprekend. Een uitvoerige beschrijving van alle opties is te vinden in de Microsoft Docs (Sarosh, 2018).

codefragment 6.1: Cosmos Indexing Configuratie

```
{
    "indexingMode": "consistent",
    "automatic": true,
    "includedPaths": [
        {
            "path": "/*",
            "indexes": [
                {
                    "kind": "Range",
                    "dataType": "Number",
                    "precision": -1
                },
                {
                    "kind": "Range",
                    "dataType": "String",
                    "precision": -1
                },
                {
                    "kind": "Spatial",
                    "dataType": "Point"
                }
            ]
        },
         {
            "path": "/timestamp/*",
            "indexes": [
                {
                    "kind": "Range",
                    "dataType": "String",
                    "precision": -1
                }
            ]
        }
    ],
    "excludedPaths": []
}
```

6.3.4 Creëren van een datapipeline

Nu zowel de *blob storage* en de Cosmos database geconfigureerd zijn kan de link tussen beide systemen gelegd worden. Door middel van de *Azure Data Factory* kan men een input, transformatie en output specifiëren. Er wordt als het ware een ETL-proces gecreëerd, die men in deze context een *pipeline* noemt. Dit proces kan eenmalig, door bepaalde triggers of periodiek uitgevoerd worden. Deze *pipelines* kunnen via de commandline worden ingesteld of via een grafische interface. Hier wordt gebruikgemaakt van de grafische interface.

Data kopiëren

De grafische interface bevat een handige *wizard* die stap voor stap helpt bij het maken van een pipeline. Men start de *wizard* via de *Copy Data* knop. Vervolgens kan in zes verschillende stappen een nieuwe kopieeropdracht aangemaakt worden.

Ten eerste moeten een aantal algemene properties zoals *Task Name* en *Description* worden meegegeven. Vervolgens selecteert men de *Source*, die de te kopiëren data bevat. Afhankelijk van het type bron dat werd gekozen krijg men een aantal aanvullende opties te zien, die ervoor zorgen dat de *Data Factory* de records correct kan parsen en eventueel transformeren naar een ander formaat. Ten derde wordt de *Sink* waarnaar de data zullen worden gekopieerd geconfigureerd aangeduid. Gevolgd door, indien van toepassing, de doeltabel of doelcollectie. Een vierde stap bevat enkele aanvullende opties, bijvoorbeeld over fouttolerantie en timeouts. Tenslotte worden alle instellingen nog eens opgesomd en kan in een laatste stap de *pipeline* gedeployd worden.

Bij de het maken van de *pipeline* komt het voordeel van *resource groups* nog maar eens naar boven. De *wizard* herkent immers alle bronnen binnen eenzelfde groep zonder extra configuratie. Dat maakt het instellen van een kopieeropdracht een stuk eenvoudiger.

6.4 Configuratie van Apache Spark

De configuratie van Spark is sterk afhankelijk van besturingssysteem. Voor deze configuratie wordt verwezen naar de documentatie van Apache Spark (Apache, 2017). Let op dat de Cosmos connector de laatste versie op het moment van schrijven nog niet ondersteund. Er werd gebruikgemaakt van **Spark 2.2.0** en **Java 8**.

Hoofdstuk 7

Verwerking van de data

7.1 Inleiding

In vorige hoofdstukken werden de inputdata reeds beschreven. In de hoofdstuk wordt er verder gebouwd op deze datasets. Hoe worden de data verwerkt en wat is het resultaat? Elke stap van het verwerkingsproces komt aan bod en de input en output zal telkens worden aangekaart.
Ter vereenvoudiging zal de dataset van *Be-Mobile* als referentie worden gebruikt. Deze dataset vergt een aantal extra stappen die niet nodig zijn bij andere datasets als Waze en Coyote. Met uitzondering van die extra stappen en een aantal transformatieregels, de datasets zien er immers compleet anders uit, is de verwerking volledig analoog.
In dit hoofdstuk komen de kopieeropdrachten en *blob storage* niet meer aan bod. Dit werd uitvoerig besproken in het vorige hoofstuk. Elk transformatieproces verzamelt data uit één of meerdere collecties van de Cosmos database, verwerkt deze en dumpt het resultaat in een bestaande of nieuwe Cosmos database collectie. Sommige stappen kunnen als overbodig of makkelijk samen te voegen met een andere stap beschouwd worden. Verder in dit hoofdstuk worden een aantal redenen voor het opsplitsen van de transformaties in meerdere stappen aangekaard. Daarna volgen de verschillende stappen.

7.2 Splitsing in verschillende stappen

Er is weldegelijk een reden voor het opdelen van elke transformatie in een aantal kleinere stappen. Eerst en vooral worden opgehaalde datasets relatief klein gehouden tegenover de volledige database. Dit wordt bijvoorbeeld door filters of *where*-clausules in de queries bekomen. Het klein houden van de datasets is van belang om de impact van fouten zo minimaal mogelijk te houden. Een netwerkfout heeft dan enkel een invloed op de kleinere dataset die wordt opgehaald en niet op de volledige database. Ook is het voordeliger voor de Cosmos database, door de datasets zodanig te filteren op geïndexeerde velden kan men het aantal

request units die een transactie vergt een stuk verlagen. Op die manier wordt de kans op databasefouten ook een stuk kleiner. Indien een transactie meer dan de toegewezen *request units* van de collectie nodig heeft om te voltooien, zal de database een *request rate is too large*-error teruggeven in plaats van de gevraagde data. Dit moet zo veel mogelijk vermeden worden.

7.3 Verwerken van Be-Mobile data

codefragment 7.1: Be-Mobile dynamische data

```
{
    "id": "5e813766-cc29-eb5f-2be7-b5ad6e3241a0",
    "_rid": "ctZsALQ52gABAAAAAAAAAA==",
    "_self": "dbs/ctZsAA==/colls/ctZsALQ52gA=/docs/ctZsALQ52
      gABAAAAAAAAAA==/",
    "_etag": "\"00000b01-0000-0000-0000-5aea4fc60000\"",
    "TrajectID": "B_OW",
    "timestamp": "02/04/2017 00:45",
    "travel time (ms)": 71427,
    "_attachments": "attachments/",
    "_ts": 1525305286
}
```

codefragment 7.2: Be-Mobile statische data

```
{
    "TrajectID": "B_OW",
    "SegmentID": 1200035,
    "lengthmm": 7340,
    "optimalspeedkph": 60,
    "beginnodelatitude": 51.038433,
    "beginnodelongitude": 3.730925,
    "endnodelatitude": 51.038431,
    "endnodelongitude": 3.73082,
    "id": "91c5e633-eff5-49ec-b731-2d55a98fc64f",
    "_rid": "ctZsAIcMNQABAAAAAAAAAA==",
    "_self": "dbs/ctZsAA==/colls/ctZsAIcMNQA=/docs/
      ctZsAIcMNQABAAAAAAAAAA==/",
    "_etag": "\"02005b5d-0000-0000-0000-5acc8a380000\"",
    "_attachments": "attachments/",
```

```
15     "_ts": 1523354168
16 }
```

7.3.1 Aggregatie van de statische data

In codefragment 7.2 is een fragment uit de statische dataset van *Be-Mobile* weergegeven. Het gaat om één Traject, wat meestal te vergelijken is met een (deel van) een straat. Merk het veld *SegmentID* op. Elk traject is nog eens opgedeeld in een groot aantal segmenten. Voor verdere verwerking is dit niet zo interessant, de totale lengte van de volledige route zou veel makkelijker zijn om mee te werken. De eerste stap zorgt er dan voor dat elk segment aan een ander segment wordt geplakt, tot de route enkel nog bestaat uit één groot segment.

codefragment 7.3: Samenvoegen van segmenten

```
1 {
2
3             while (toProcess.Count > 0)
4             {
5                 if (processedAfterAdd > toProcess.Count)
6                 {
7                     tolerance *= 10; //increase tolerance if no matches
                          found
8                 }
9
10                TraveltimeSegmentStatic current = toProcess.Dequeue();
11                if (Math.Abs(biggest.endnodelatitude - current.
                      beginnodelatitude) < tolerance && Math.Abs(biggest.
                      endnodelatitude - current.beginnodelatitude) <
                      tolerance)
12                {
13                    //new segment comes behind current one
14                    biggest.lengthmm += current.lengthmm;
15                    biggest.endnodelatitude = current.endnodelatitude;
16                    biggest.endnodelongitude = current.endnodelongitude;
17                    biggest.SegmentID++;
18                    processedAfterAdd = 0;
19                //    Console.WriteLine($"Matching segment found, new
                      length: {biggest.lengthmm}");
20                }
21                else if (Math.Abs(current.endnodelatitude - biggest.
                      beginnodelatitude) < tolerance &&
22                         Math.Abs(current.endnodelatitude - biggest.
                               beginnodelatitude) < tolerance)
23                {
24                    //new segment comes before current one
25                    ...
26                }
```

```
27                  else
28                  {
29                      //new segment not adjacent to segment, putting in back
                            of the queue
30                  }
31
32  }
```

Samenvoegen van de segmenten

Codefragment 7.3 wordt het *merge*-proces weergegeven. Een eerste segment wordt ad random gekozen en als huidige grootste segment gezien. Alle andere segmenten worden op een queue geplaatst, de operatie gaat verder zolang de queue niet leeg is. Begin -en eindcoördinaten worden met elkaar vergeleken (regel 11), opeenvolgende segmenten hebben immer ook opeenvolgende coördinaten. Is het verschil tussen de coördinaten kleiner dan de tollerantie dan worden de segmenten aan elkaar geplakt (regels 12-20 en 23-26), afhankelijk van welke coördinaten elkaar opvolgen wordt het segment vooraan of achteraan toegevoegd. Het resultaat vormt het nieuwe grootste segment.
Als er geen match gevonden wordt, wordt het segment opnieuw achteraan in de queue geplaatst (regels 28-30) en wordt met het volgende segment opnieuw geprobeerd. Als er na de volledige queue te overlopen nog steeds geen match gevonden wordt, dan wordt de tollerantie verhoogt (reguls 6-8) en wordt opnieuw geprobeerd. Dit proces herhaalt zich tot de queue leeg is. Uit tests en waarnemingen blijkt dat de meeste routes kunnen worden geconstrueerd met een maximale tollerantie van *0.00000001* graden verschil in coördinaten.

Ophalen van locatie

Een tweede probleem is het ontbreken van locatie-informatie. De statische informatie bevat geen enkele verwijzing naar een straatnaam of andere locatie. In datasets van andere bronnen, zoals Waze en Coyote, ontbreekt deze ook of is de locatie niet eenduidig genoeg. Om dit op te lossen werd een module geschreven die via coördinaten generieke informatie over de locatie ophaalt. De coördinaten van een locatie zijn een goed startpunt, ze worden met elke databron weergegeven en zijn ook absoluut. Er werd gekozen voor de *Google Locations API*, omdat deze het meeste informatie bevat en ook goedkoper is dan tegenhangers zoals de *Here Geocoding API*. Een voorbeeld van het resultaat is te zien in codefragment 7.4. Door deze manier van werken kan het programma automatisch omgaan met locatieverwijzingen vanuit verschillende bronnen.

codefragment 7.4: Opvraging van een locatie

```
"googleLocationInfo": [
            {
                "html_attributions": [],
                "result": {
                    "address_components": [
                        {
                            "long_name": "179-121",
                            "short_name": "179-121",
                            "types": [
                                "street_number"
                            ]
                        },
                        {
                            "long_name": "Sint-Lievenslaan",
                            "short_name": "Sint-Lievenslaan",
                            "types": [
                                "route"
                            ]
                        },
                        ..
                        ],
                    "formatted_address": "Sint-Lievenslaan 179-1
                       21, 9000 Gent, Belgium",
            }
        }
    ]
```

7.3.2 Mergen van statische en dynamische data tot een genormaliseerd document

De volgende stap is het samenvoegen van de verwerkte statische data en de dynamische data zoals in fragment 7.1. In *big data NoSQL* omgevingen is het een stuk voordelig om data dubbel op te slaan en op die manier de nodige processorkracht te verminderen, dan met queries *joins* uit te voeren. Het joinen van data wordt zelfs maar in beperkte mate ondersteund. De output van deze stap is een genormaliseerd document dat hetzelfde is voor elke databron. In fragment 7.5 is in *C#* te zien hoe zo'n *Plain old CLR object* (POCO) eruit ziet. Na deze stap zal geen onderscheid meer moeten gemaakt worden tussen de verschillende databronnen.

codefragment 7.5: Uniforme POCO voor verkeersdata

```
1 public class GenericTraveltimeSegment : IDocumentPOCO
2     {
3         [JsonProperty("id")]
4         public string Id { get; set; } //id = Source + segment + timestamp
5         public string Source { get; set; }
6         public DocumentDate Timestamp { get; set; }
7         public string SegmentName { get; set; }
8         public string FromPoint { get; set; }
9         public string ToPoint { get; set; }
10
11        public Coordinate[] Coordinates;
12        public double FromLatitude { get; set; }
13        public double ToLatitude { get; set; }
14        public double FromLongitude { get; set; }
15        public double ToLongitude { get; set; }
16
17        public double Length { get; set; } //in meters
18        public double Speed { get; set; }//in km/h
19        public double OptimalSpeed { get; set; }//in km/h
20        public int Duration { get; set; }//in seconds
21  }
```

7.3.3 Toevoegen van compleet nieuwe datasets

In codefragment 7.5 is te zien hoe uiteindelijk een universeel, genormaliseerd document eruit ziet. Na deze stap wordt er immers geen onderscheidt meer gemaakt tussen verschillende bronnen van data. Dit heeft als gevolg dat nieuwe bronnen relatief makkelijk kunnen worden toegevoegd. Dit kan deels via de *Data Factory* door de pipeline zodanig te configureren dat er al een groot deel van het werk gedaan wordt. Via deze tool kan men probleemloos XML of CSV omzetten naar een JSON-document in de Cosmos Database. Er moet immers geen verband zijn tussen input en output wat betref dataformaat. Zolang beide maar worden herkend en ondersteund door de *Data Factory*. Verdere transformaties kunnen dan gebeuren via scripts

in Python of programma's in C#. Er werd zoveel mogelijk gewerkt met *Class Libraries* en abstractie om het implementeren van een nieuwe use case zo gemakkelijk mogelijk te maken.

7.4 Verdere verwerking van de data

In de vorige stappen werden de data samengevoegd en genormaliseerd tot een uniforme dataset. Met behulp van *Apache Spark* kunnen er nu verdere bewerkingen op worden uitgevoerd. Via *Python* wordt geïnterfaced naar de verschillende worker nodes van Spark. Spark maakt op zijn beurt verbinding met de database door middel van *Azure Cosmos DB Connector for Apache Spark*. Deze connector vergt wat extra configuratie en veel documentatie is er niet over beschikbaar. In het volgende onderdeel zal daarom wat extra aandacht aan deze connector worden besteed.

7.4.1 Azure Cosmos DB Connector voor Apache Spark

De connector is beschikbaar voor zowel Python, Scala en Java. In dit project wordt gebruikgemaakt van Python om te interfacen met de Spart worker nodes. Er werd reeds vermeld dat een extra *JAR* moet meegegeven worden om van de functionaliteiten van de connector gebruik te kunnen maken.

Ophalen van informatie uit de database

Men configureert een verbinding door een object aan te maken zoals te zien in codefragment 7.6. Via deze configuratie kan een *Spark dataframe* aangemaakt worden met de records uit de database zonder al te veel extra overhead.

Endpoint	Url naar de database, dit is meestal de *ID* van de database gevolgd door *documents.azure.com*.
Masterkey	De secret key van de database. Voor leesopdrachten mag dit een *read-only* key zijn, voor schrijfopdrachten moet dit uiteraard een *read-write* key zijn. Opgelet: deze sleutels moeten altijd geheim blijven, ze worden gebruikt voor authenticatie.
Database	Naam van de database.
preferredRegions	Een lijst van datacenterlocaties die de voorkeur krijgen. Soms wordt een database gerepliceerd naar enorm veel datacenters. Het kan dan voordelig zijn om het datacenter met de laagste netwerkvertraging te kiezen.
Collection	De naam van de collectie die moet worden geraadpleegd.
SamplingRatio	De gebruikte bemonstering voor het afleiding van databaserecords.

codefragment 7.6: Cosmos leesconfiguratie in Python

```
1 dbReadConfig = {
2         "Endpoint" : "EndpointUrl",
3         "Masterkey" : "12345678910111213141516171819202122232425...===",
4         "Database" : "mbdvoiot-cdb-1",
5         "preferredRegions" : "West Europe",
6         "Collection" : "mbdvoiot-cdb-partial",
7         "SamplingRatio" : "1.0",
8         "schema_samplesize" : "1000",
9         "query_pagesize" : "2147483647",
10        "query_custom" : "SELECT * FROM c"
11        }
12
13 result = spark.read.format("com.microsoft.azure.cosmosdb.spark").options(**
      dbReadConfig).load()
```

schema_samplesize Aantal documenten die initiëeel zullen worden doorzocht in de database

query_pagesize De groote van de resultaatpagina van een query. Hoe groter deze pagina hoe minder overhead voor het ophalen van queryresultaten.

query_custom De query die zal worden uitgevoerd op de database. Deze variabele kan worden gebruikt om de data te filteren door bijvoorbeeld een filter toe te voegen.

Schrijven van informatie naar de database

In codefragment 7.7 is de configuratie om terug naar de database te schrijven weergegeven. Merk op dat de variabele *upsert* is toegevoegd. *Upsert* heeft het voordeel dat eerst en vooral wordt gecontroleerd of het document nog niet bestaat. Is dat niet het geval, dan zal een nieuw document aangemaakt worden. In het geval dat er reeds een document bestaat met dezelfde identificatie dan zal een andere actie ondernomen worden. Die actie wordt meegegeven met de methode die het schrijven effectief zal uitvoeren (regel 13), via de *mode* optie. Indien de optie *upsert* niet wordt meegegeven, of als deze de waarde *False* heeft, dan zal een fout worden teruggegeven bij het schrijven van een reeds bestaand document.

7.4.2 Genereren van statistische data

Het is interessant om een aantal statistieken te berekenen over de volledige dataset. Het is bijvoorbeeld zeer handig om te weten wat de gemiddelde snelheid in de Sint-Lievenslaan is om 8 uur 's morgens op een maandag. Of hoeveel wagens er via elke invalsweg op de ring rijden per uur per dag. Hoewel de Cosmos database het gebruik van functies als *AVG()* en

codefragment 7.7: Cosmos schrijfconfiguratie in Python

```
dbWriteConfig = {
        "Endpoint" : "EndpointUrl",
        "Masterkey" : "12345678910111213141516171819202122232425...===",
        "Database" : "mbdvoiot-cdb-1",
        "preferredRegions" : "West Europe",
        "Collection" : "mbdvoiot-cdb-partial",
        "SamplingRatio" : "1.0",
        "schema_samplesize" : "1000",
        "query_pagesize" : "2147483647",
        "upsert": "True"
        }

        dfJoined.write.format("com.microsoft.azure.cosmosdb.spark").mode("
            overwrite").options(**dbWriteConfig).save()
```

SUM() ondersteund, wordt de *group by*-clausule nog niet aangeboden. De oplossing voor dit probleem is het gebruik van *Spark SQL*, zo kan men op een dataframe die de velden *FromPoint*, *DayOfWeek*, *Hour* en *Speed* bevat de query uitvoeren die wordt weergegeven in codefragment 7.8.

Het resultaat is een dataset die voor elke straat op elke dag van de week en voor elk uur van die dag een gemiddelde snelheid bevat. Nu werd deze query initieel uitgevoerd over de volledige dataset. Achteraf bleek deze data echter niet zo bruikbaar. Een gemiddelde varieert immers doorheen het jaar door een aantal factoren. De Gentse Feesten en schoolvakanties hebben een grote invloed op deze data, daarom is het beter om de gemiddeldes te berekenen over kleinere periodes.

Eens de statische data zijn gegenereerd is het voordelig om deze meteen te mergen met de dynamische data. Voorlopig gebeurt dit via een join operatie met behulp van Spark, maar dit lijkt niet efficient te zijn. Een betere optie zou zijn om dit te doen bij het invullen van de snelheidsklassen, wat verder in dit hoofdstuk wordt aangehaald, aangezien elk document daar toch opnieuw moet worden ingeladen en verwerkt.

codefragment 7.8: Spark query met group by clausule

```
SELECT c.FromPoint, c.DayOfWeek, c.Hour, AVG(c.Speed)
FROM c
GROUP BY c.FromPoint, c.DayOfWeek, c.Hour
```

7.4.3 Predictieve modellen opbouwen

De verkeersdata is niet perfect, af en toe zijn er inconsistenties of zelfs ontbrekende data. Om dit probleem toch in beperkte mate op te vangen, werd een voorspelling gedaan over het verkeer op de ontbrekende momenten. Spark bevat een module die een aantal *machine learning* algoritmes ondersteund. Op die manier kan een bepaalde periode ingedeeld worden in een snelheidsklasse die meest waarschijnlijk is op dat moment.

Definiëren van snelheidsklassen

Een getal plakken op de te voorspellen snelheidswaarde is niet mogelijk, of toch niet met voldoende zekerheid. Om dit probleem op te lossen worden drie verschillende snelheidsklassen gedefinieerd, relatief tegenover de gemiddelde snelheid op dat moment. De gemiddelde snelheid werd reeds berekend in de vorige stap, nu proberen we de werkelijke snelheid te voorspellen relatief tegenover dat gemiddelde. Er zijn drie mogelijkheden: **sneller dan gemiddeld**, **gemiddeld** en **trager dan gemiddeld**. Via classificatie zal er gepoogd worden om de meest waarschijnlijke klasse toe te wijzen aan een bepaalde periode.

Classificatie in snelheidsklassen

Eerst en vooral moet een dataset worden opgebouwd waarop men het model kan trainen en testen. Aan elk document wordt een extra veld toegevoegd die de juiste snelheidsklasse bevat voor dat record. De snelheidsklasse is dan het label voor de te identificeren klasse.
Vervolgens worden de *features* van de records gedefinieerd. Deze features zijn de velden die bepalen tot welke klasse een record behoort. Op dit moment zijn er slechts twee velden die dienst doen als features: de dag van de week en het uur op de dag. Toch zijn er een aantal uitbreidingen die een positief effect kunnen hebben op de predictie. Zo zou het toevoegen van een extra veld over het weer, over ongevallen in de buurt of over stakingen een meer exacte predictie kunnen opleveren. Die data zijn echter nog in te beperkte mate beschikbaar om toe te voegen aan de lijst met features.

Voorspelling door middel van een beslissingsboom

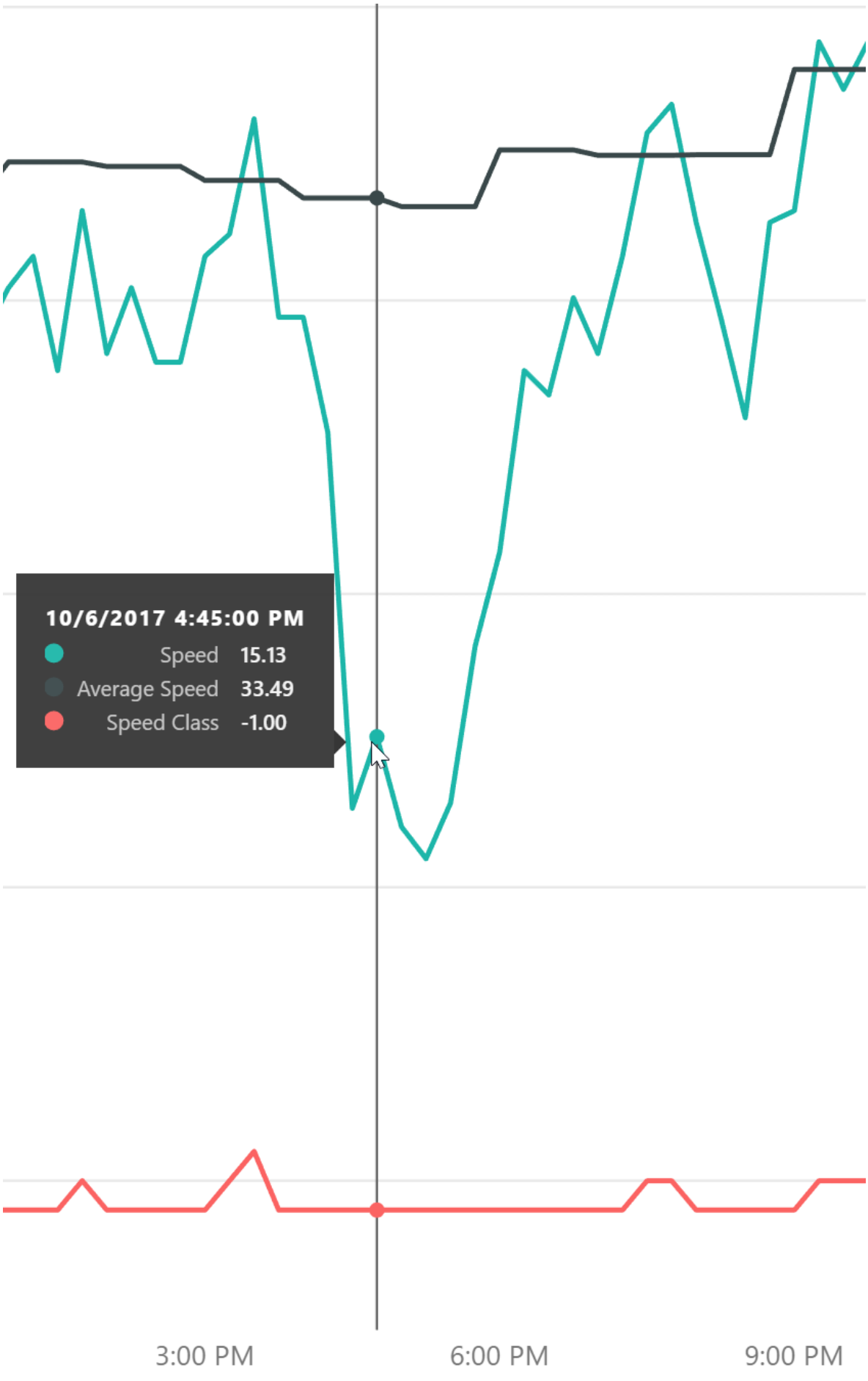
Nadat de *features* en het *label* bepaald zijn, kan een predictie uitgevoerd worden. Er wordt gebruikgemaakt van Sparks ingebouwde beslissingsboomalgoritme. De beschikbare dataset wordt at random in twee gesplitst. Een grootste deel trainingsdata en een kleiner deel testdata. De beslissingsboom wordt vervolgens geconstrueerd met behulp van de trainingsdata en toegepast op de testdata. Het resultaat is een beslissingsboom met een bepaald foutenpercentage. De beslissingsboom had een gemiddelde diepte van 5 met 33 nodes.
Jammergenoeg werd hier telkens nog een foutenpercentage van 20-30% bereikt. Dit komt

waarschijnlijk door het ontbreken van een aantal belangrijke features zoals het weer. Er moet dus nog wat aan het model worden gesleuteld.

7.5 Filteren van foutieve data

Na het visualiseren van de bekomen datasets, zijn hier en daar een aantal pieken te zien in de bekomen grafieken, een voorbeeld is te zien op figuur 7.1. Het is voor de analyse uiterst interessant om deze pieken te onderscheiden van de normale dataset. Ten eerste kunnen deze maxima of minima wijzen op een ongeval of andere abnormale toestand op de weg. Een tweede mogelijkheid is een foutieve meetwaarde van de databron. Beide gevallen kunnen zonder bijkomende data moeilijk van elkaar onderscheiden worden. Gelukkig is er een bijkomende dataset beschikbaar die hier hulp kan bieden. Waze biedt naast snelheden en reistijden ook zogenaamde *alerts*. Een *alert* treed op als een aantal gebruikers ofwel stilstaan of sterk vertragen op hetzelfde moment, of als iemand manueel een melding geeft via de applicatie.
Wanneer er dus een correlatie bestaat tussen het tijdstip van een dergelijke vertraging in de grafiek en een melding van de Waze-applicatie kan men dus met een tamelijke zekerheid zeggen dat de vertragin te wijden is aan een abnormale omstandigheid op de weg. Is dat niet het geval, dan is er ofwel geen melding via Waze doorgekomen ofwel zijn de data foutief.

Figuur 7.1: Abnormale waarden in dataset



7.5.1 Huidige implementatie van de filter

Deze implementatie is niet de meest betrouwbare, maar wel het meest haalbaar binnen de korte tijdspanne van het project. Eerst en vooral worden zoals hiervoor vermeld de gemiddelde waarden per uur per weekdag berekend. Bijkomende wordt de variantie, de maat waarin de data gemiddeld afwijken van het gemiddelde, mee berekend. Afhankelijk van de gemiddelde waarde, de variantie en de gemeten waarde op een specifiek moment wordt een waarde gemarkeerd als *mogelijk foutief*.
Via een query wordt dan nagegaan of er een melding van vertraging bestaat in de database. De meldingen van Waze werden ook ingeladen in de database, samen met de reistijden. De bewerking is volledig analoog en wordt daarom niet verder besproken.

Hoofdstuk 8

Conclusies en perspectieven

8.1 Inleiding

In dit onderdeel wordt eerst en vooral het eindproduct bekeken en vergeleken met het initiële voorstel. De vergelijking wordt opgevolgd door opmerkingen en aanpassingen die nodig waren bij het originele idee voor het project

8.2 Evaluatie van het eindproduct

8.2.1 Beschrijving van het resultaat

Het eindproduct bestaat uit 4 componenten, de blob opslag, de Cosmos database, PowerBI visualisaties en een simpele *REST API* voor het opvragen van bepaalde datasets.
De blob opslag doet dienst als een permanente opslag en backup voor onverwerke datasets. Vervolgens doet de Cosmos database dienst als database voor de verwerkte datasets, niet als permanente opslag voor data die reeds op de blob opslag te vinden zijn. Het heeft geen nut om data dubbel te stockeren en de blob opslag was de goedkoopste en efficiëntste oplossing om data in een soort van koude opslag te dumpen.
Via PowerBI is het inderdaad mogelijk om handige visualisaties te maken en weer te geven online. Na het overleggen met de klant, was dat voor hen ook de makkelijkste en handigste optie.
De *REST API* is toegankelijk en stelt de gebruiker in staat om simpele queries uit te voeren op de dataset. De grootte van het resultaat en query opties zijn echter wel een pak gelimiteerder dan een directe interface met de database.

8.2.2 Afwijkingen van het oorspronkelijke werkplan

Er zijn een aantal afwijkingen met het originele werkplan. Ze worden in dit onderdeel aangekaart en verklaard.

Automatische imports Er zijn nog steeds een aantal scripts en imports die niet automatisch kunnen gebeuren. Dit komt doordat een aantal datasets handmatig worden aangeleverd en omdat er geen garantie kan worden gegeven van de aanleverfrequentie. Daardoor leek het beter en makkelijker om de scripts niet periodiek te triggeren, maar eerder handmatig als er een nieuwe dataset beschikbaar is. Het tijdstip en frequentie van het uitvoeren van de scripts en programma's heeft weinig impact op de database zelf, die blijft altijd online.

Filteren van de data Het filteren van de data was ook een hekelpunt. Hoe kan men efficient fouten detecteren in een dataset van deze grootorde, waarbij de metingen zo kunnen variëren. De uiteindelijke implementatie werd verklaard in het hoofdstuk over de dataverwerking.

Datawarehouse Initiëel was het plan om een datawarehouse op te zetten met de verschillende verkeersdata. Door de hoge kosten en lagere schaalbaarheid werd afgestapt van dit idee. Na wat onderzoek werd geconstateerd dat een NoSQL database, de Cosmos database, meer dan voldoende functies bevat voor deze use case. Bovendien is deze implementatie niet alleen goedkoper, ze is ook robuuster en kan dynamische geschaald worden in de cloud.

Webdashboard Het is nog niet zeker hoe en op welke manier de data naar buiten zullen worden gebracht. Er werd beslist om geen volledig dashboard ter beschikking te stellen. Er zijn echter wel een aantal visualisaties gemaakt die een beeld geven over bepaalde aspecten van het verkeer via PowerBI zoals weergegeven in figuur 2.2, die kunnen dan weergegeven worden op bestaande website en dashboards indien gewenst. Een volledig nieuw platform maken was overbodig.

8.3 Onderzoeksvraag

In deze masterproef werd een manier ontwikkeld om verkeersdata op een consistente manier te verzamelen, te verwerken en te gebruiken. De onderzoeksvraag luidde: **Hoe kan men alle verschillende datasets op een efficiente manier bundelen en verwerken tot een eenduidige en complete database van verkeersdata waarop zinvolle analyses kunnen worden uitgevoerd?**
De initiële voorwaarden zijn voldaan. Verschillende databronnen worden aangesproken via periodieke calls en gedumpt in een filestorage. Vervolgens worden ze geïmporteerd in een database in verder verwerkt tot een samenhangende verkeersdataset. Die dataset omvat

voorlopig reistijden en verkeersmeldingen. Dit is voldoende voor de meeste analyses. Toch zijn uitbreidingen mogelijk, die worden besproken in het volgende onderdeel.

8.4 Uitbreidingen en aanpassingen

Er zijn nog een aantal zaken die de resultaten van de verkeersanalyses positief kunnen beïnvloeden, maar waarvoor geen tijd meer is in dit project. Dit kunnen ofwel uitbreidingen zijn die extra informatie kunnen geven of aanpassingen voor reeds geïmplementeerde functies.

8.4.1 Extra features gebruiken voor predicties

Momenteel wordt enkel het tijdstip gebruikt om de snelheid op een route te voorspellen. Factoren zoals wegenwerken, weer, evenementen... hebben ook een invloed op dit resultaat. Het toevoegen van extra datasets kan het foutpercentage enorm naar beneden halen.

8.4.2 Geavanceerde foutdetectie

De manier waarop de data worden gefilterd is momenteel voldoende, maar kan een stuk beter. Er werd onderzoek gedaan naar het gebruik van een *Kalmanfilter* om eventuele inconsistenties aan te duiden. Een kalmanfilter voorspelt een volgende waarde aan de hand van de vorige meetresultaten. Wijkt het effectieve meetresultaat te veel af van het berekende resultaat van de kalmanfilter, dan wordt dit gemarkeerd. Op die manier kunnen fouten dynamisch worden gedetecteerd. Deze filter werkt met een datastream, dit is meteen de moeilijkheid van deze implementatie. Een oplossing zou zijn om de *Apache Spark streaming API* te gebruiken om met de dataset te interfacen als een datastream. Deze implementatie zou de foutenmarge van detecties sterk kunnen verbeteren.

8.4.3 Meer datasets over reistijden

Zoals reeds aangekaart is het vrij eenvoudig om nieuwe datasets toe te voegen aan de database. Indien meer datasets beschikbaar zouden zijn in de toekomst, kan het de analyses alleen maar positief beïnvloeden.

8.4.4 Extensiever gebruik van Apache Spark

Sommige transformaties worden uitgevoerd met Spark, anderen niet. Achteraf gezien kon Spark voor meer zaken gebruikt worden. Het is sneller en efficienter. Transformaties die gebeuren door middel van een aantal C# programma's konden bijvoorbeeld ook door Spark gebeuren.

Bibliografie

Apache (2017). Apache spark docs. https://spark.apache.org/docs/2.2.0/. Geraadpleegd op 1 maart 2018.

M. Asay (2017). Does microsoft's cosmos db promise too much? https://www.infoworld.com/article/3197627/database/does-microsofts-cosmos-db-promise-too-much.html. Geraadpleegd op 19 maart 2018.

AWV (2017). Vlaamse overheid stelt actuele verkeersdata open. https://wegenenverkeer.be/persberichten/vlaamse-overheid-stelt-actuele-verkeersdata-open. Geraadpleegd op 10 maart 2018.

S. Bisson (2017). Data storage in azure: Everything you need to know. https://www.infoworld.com/article/3228151/cloud-storage/data-storage-in-azure-everything-you-need-to-know.html. Geraadpleegd op 12 maart 2018.

Bluebike (2018). Ontdek bluebike. https://www.blue-bike.be/nl/ontdek. Geraadpleegd op 10 maart 2018.

E. A. Brewer (2000). Towards robust distributed systems. Keynote op de PODC symposium in het jaar 2000 [Geraadpleegd op 11 maart 2018] via http://awoc.wolski.fi/dlib/big-data/Brewer_podc_keynote_2000.pdf.

G. De Tré & A. Bronselaer (2017). Information managment syllabus. Universiteit Gent. Geraadpleegd via Minerva.

L. Dignan (2018). Top cloud providers 2018: How aws, microsoft, google cloud platform, ibm cloud, oracle, alibaba stack up. http://www.zdnet.com/article/cloud-providers-ranking-2018-how-aws-microsoft-google-cloud-platform-ibm-cloud-oracle-a Geraadpleegd op 21 maart 2018.

T. FitzMacken, J. Dial, R. Squillace & K. Crider (2017). Azure resource manager vs. classic deployment: Understand deployment models and the state of your resources. https://docs.microsoft.com/en-us/azure/azure-resource-manager/resource-manager-deployment-model. Geraadpleegd op 21 mei 2018.

iRail npo (2011). Nmbs. `https://data.irail.be/`. Geraadpleegd op 10 maart 2018.

X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia & A. Talwalkar (2016). Mllib: Machine learning in apache spark. *Journal of Machine Learning Research*, 17(1):1–7.

Microsoft (2017a). Azure cosmos db documentation. `https://docs.microsoft.com/en-us/azure/cosmos-db/`. Geraadpleegd op 3 maart 2018.

Microsoft (2017b). Welcome to azure cosmos db. `https://docs.microsoft.com/en-us/azure/cosmos-db/introduction`. Geraadpleegd op 3 maart 2018.

MLlib (2018). Mllib: Rdd-based api. `https://spark.apache.org/docs/latest/mllib-guide.html`. Geraadpleegd op 11 mei 2018.

T. Myers (2018). Azure storage replication. `https://docs.microsoft.com/en-us/azure/storage/common/storage-redundancy`. Geraadpleegd op 21 mei 2018.

J. Sanders (2017). Microsoft azure: The smart person's guide. `https://www.techrepublic.com/article/microsoft-azure-the-smart-persons-guide/`. Geraadpleegd op 20 maart 2018.

R. Sarosh (2018). How does azure cosmos db index data? `https://docs.microsoft.com/en-us/azure/cosmos-db/indexing-policies`. Geraadpleegd op 23 mei 2018.

A. G. Shoro & T. R. Soomro (2015). Big data analysis: Ap spark perspective. *Global Journal of Computer Science and Technology*, 15(1):7–14.

P. Taylor (2017). What is an azure resource group? `https://docs.microsoft.com/en-us/azure/architecture/cloud-adoption-guide/adoption-intro/resource-group-explainer`. Geraadpleegd op 21 mei 2018.

Team Trefis (2016). A close look at microsoft's azure cloud. `https://www.forbes.com/sites/greatspeculations/2016/09/08/a-close-look-at-microsofts-azure-cloud-part-1-cloud-computing/#6954cb7128ec`. Geraadpleegd op 20 maart 2018.

Waze Mobile (2018). Waze. `https://waze.com/`. Geraadpleegd op 12 maart 2018.

Appendices

Bijlage A

Aanvullende mobiliteitsdata

subsectionBe-Mobile

Formaat CSV

Voorbeeld

```
Traject;timestamp;TravelTimeMs
B_OW;01/06/2017 00:00;85779
B_OW;01/06/2017 00:15;77056
B_OW;01/06/2017 00:30;73282
B_OW;01/06/2017 00:45;63459
```

A.0.1 Reistijden Coyote

Formaat JSON

Voorbeeld

```
{"Gand":{
   "Rooigemlaan (R40) Northbound - Drongensesteenweg -
      Palinghuizen":{
      "normal_time":40.2,
      "real_time":40.2,
      "diff_time":0,
      "length":1340,
      "geometries":[
         [
```

```
            {
                "lng":3.69508,
                "lat":51.05617
            },
            {
                "lng":3.69487,
                "lat":51.05631
            },
            ...
        ]
    ],
    "sections":[

    ],
    "alerts":[

    ]
}}}
```

A.0.2 Tellussen Agentschap Wegen en Verkeer

Formaat JSON

Voorbeeld

```
{
   "type":"FeatureCollection",
   "features":[
      {
         "type":"Feature",
         "geometry":{
            "type":"Point",
            "coordinates":[
               3.7202457700359748,
               51.041201364632059
            ]
         },
         "properties":{
            "attributes":[
```

```
                {
                    "attributeName":"speed",
                    "value":35
                },
                {
                    "attributeName":"OCC",
                    "value":1
                },
                {
                    "attributeName":"Count",
                    "value":60
                },
                {
                    "attributeName":"Timestamp",
                    "value":1491004635
                }
            ],
            "contextEntity":"M004"
        }
    },
        ...
  ],
  "EventProcessedUtcTime":"2017-04-01T00:00:00.7027260Z",
  "PartitionId":0,
  "EventEnqueuedUtcTime":"2017-04-01T00:00:02.4120000Z"
}
```


A.0.3 Bluebike

Formaat JSON

Voorbeeld


```
{
  "type":"Feature",
  "geometry":{
     "type":"Point",
     "coordinates":[
        3.70964,
```

```
            51.1688
         ]
      },
      "properties":{
         "contextEntity":"Blue-bikes_Gent-Sint-Pieters",
         "description":"Parking Blue-bike fietsen Gent-Sint-Pieters
            ",
         "attributes":[
            {
               "attributeName":"CapacityTotal",
               "value":85
            },
            {
               "attributeName":"CapacityInUse",
               "value":24
            },
            {
               "attributeName":"CapacityAvailable",
               "value":59
            },
            {
               "attributeName":"CapacityInMaintenance",
               "value":2
            },
            {
               "attributeName":"PriceEuro",
               "value":3
            },
            {
               "attributeName":"RouteNl",
               "value":"Neem de uitgang aan de achterkant van het
                  station. Daar aan de rechterkant zie je de
                  sleutelautomaat en de Blue-bikes. Het fietspunt
                  ligt nog iets verder achter de fietsenparking."
            },...
         ]
      },
      "EventProcessedUtcTime":"2017-04-04T00:00:01.5000827Z",
      "PartitionId":1,
```

```
42     "EventEnqueuedUtcTime":"2017-04-04T00:00:00.9210000Z"
43 }
```

A.0.4 iRail

Formaat JSON

Voorbeeld

```
1  {
2     "@context":{
3        "delay":"http:\/\/semweb.mmlab.be\/ns\/rplod\/delay",
4        "platform":"http:\/\/semweb.mmlab.be\/ns\/rplod\/platform"
            ,
5        "scheduledDepartureTime":"http:\/\/semweb.mmlab.be\/ns\/
            rplod\/scheduledDepartureTime",
6        "headsign":"http:\/\/vocab.org\/transit\/terms\/headsign",
7        "routeLabel":"http:\/\/semweb.mmlab.be\/ns\/rplod\/
            routeLabel",
8        "stop":{
9           "@id":"http:\/\/semweb.mmlab.be\/ns\/rplod\/stop",
10          "@type":"@id"
11       }
12    },
13    "@graph":[
14       {
15          "@id":"https:\/\/irail.be\/stations\/nmbs\/008892007\/
               departures\/20180101011105d3f5bd993fa9c6d48b0f53b9ffa
               493a",
16          "delay":"0",
17          "platform":"12",
18          "canceled":"0",
19          "scheduledDepartureTime":"2018-01-01T01:10:00+01:00",
20          "stop":"https:\/\/irail.be\/stations\/NMBS\/008892007",
21          "headsign":"Oostende",
22          "routeLabel":"IC 545"
23       },
24       ...
25    ]
```

```
}
```

A.0.5 Parkeergarages Stad Gent

Formaat JSON

Voorbeeld


```
[
    {
        "id":18408,
        "lastModifiedDate":"2016-04-12T12:58:15.673Z",
        "name":"P07 Sint-Michiels",
        "description":"Sint-Michiels",
        "latitude":51.05367,
        "longitude":3.7186,
        "address":"Sint-Michielsplein 8\n9000 Gent",
        "contactInfo":"Tel.: 09 266 29 20",
        "city":{
            "id":1004,
            "name":"Gent"
        },

        "parkingServer":{
            "id":1005,
            "name":"ITG Gent"
        },
        "suggestedFreeThreshold":4,
        "suggestedFullThreshold":4,
        "capacityRounding":1,
        "totalCapacity":450,
        "openingTimes":[
            {
                "days":[
                    "MONDAY",
                    "TUESDAY",
                    "WEDNESDAY",
                    "THURSDAY",
                    "FRIDAY",
```

```
                "SATURDAY",
                "SUNDAY"
            ],
            "from":"00:00",
            "to":"23:59"
        }
    ],
    "parkingStatus":{
        "availableCapacity":369,
        "totalCapacity":450,
        "open":true,
        "suggestedCapacity":"ACTUAL",
        "activeRoute":"",
        "lastModifiedDate":"02/01/2018 00:57:00"
    }
},
...
]
```

A.0.6 Waze

Formaat JSON

Voorbeeld

```
{
   "usersOnJams":[]
      {
         "wazersCount":0,
         "jamLevel":0
      },
      {
         "wazersCount":0,
         "jamLevel":1
      },
      {
         "wazersCount":0,
         "jamLevel":2
      },
```

```
        {
            "wazersCount":0,
            "jamLevel":3
        },
        {
            "wazersCount":0,
            "jamLevel":4
        }
    ],
    "routes":[
        {
            "historicTime":430,
            "line":[
                {
                    "x":3.672124,
                    "y":51.08643799999998
                },
                                ...
            ],
            "bbox":{
                "minY":51.013156941216529,
                "minX":3.6558570000000006,
                "maxY":51.08643799999998,
                "maxX":3.728930471796887
            },
            "length":11408,
            "type":"STATIC",
            "jams":[

            ],
            "alerts":[

            ],
            "toName":"Sluisweg",
            "name":"Buitenring-Drongen (R4) southbound",
            "fromName":"Industrieweg",
            "jamLevel":0,
            "id":4096,
            "time":433
```

```
        },
    ],
    "irregularities":[]

    ],
    "broadcasterId":147,
    "areaName":"Belguim, Gent",
    "bbox":{
        "minY":51.007618616290713,
        "minX":3.6749267578125,
        "maxY":51.096966571798838,
        "maxX":3.7937164306640625
    },
    "name":"Verkeerscentrum Gent",
    "isMetric":1,
    "restrictions":{}

    },
    "lengthOfJams":[]
        {
            "jamLevel":1,
            "jamLength":0
        },
        {
            "jamLevel":2,
            "jamLength":0
        },
        {
            "jamLevel":3,
            "jamLength":0
        },
        {
            "jamLevel":4,
            "jamLength":0
        },
        {
            "jamLevel":5,
            "jamLength":3702
        }
```

```
    ],
    "updateTime":1496277282266,
    "EventProcessedUtcTime":"2017-06-01T00:35:10.1622563Z",
    "PartitionId":0,
    "EventEnqueuedUtcTime":"2017-06-01T00:35:09.5450000Z"
}
```

Data-analyse en visualisatie van verkeersdata

Ruben Vervust

Promotoren: prof. dr. Jan Cnops, dhr. Hans Fraiponts (Digipolis)

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Bart Dhoedt
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2017-2018

UNIVERSITEIT
GENT