TABLE OF CONTENTS

Literature is included as end notes to this paper, referring to the respective paragraphs.

1. Summary

Project Goal and Problem Statement

This project's goal is to develop an integrated framework that analyzes data from an IOT smartwatch. It will provide a platform to gain insights into raw data as well as aggregated metrics.

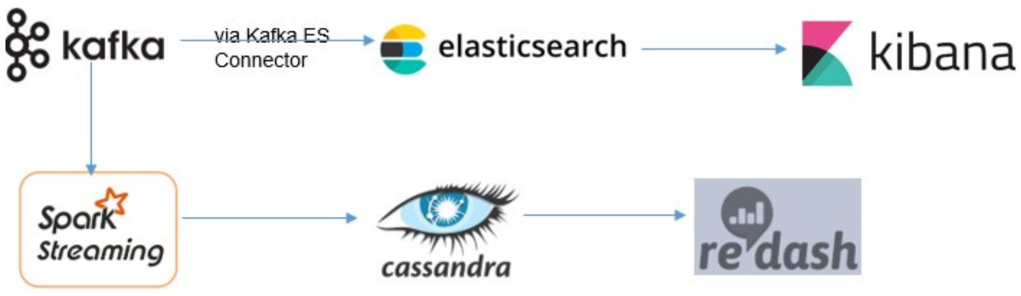
Big Data Source

Kafka will be used as data generator to produce a constant message stream simulating incoming data from an IOT smartwatch/fitness tracker.

Results

As a result of my employed architecture, I enabled the user to show and analyze (a) raw data features (heart rate, stress level) as a function of time and (b) aggregated metrics (e.g. total number of users aggregated by different sports in total and within the last 10 minutes).

Processing Pipeline



Pipeline Overview and Technologies Used

Collection Tier: Apache Kafka Producer will be used to simulate a stream of incoming messages by multiple fitness tracker devices used in Frankfurt.

Processing Tier: Spark Streaming will be used to calculate total number of users and the total number of users aggregated by different sports within the last hour.

Storage Tier: Apache Cassandra will be used to store the calculated metrics as tables in a database.

Indexing Tier: Kafka ES Connector will be used to push data from the initial Kafka simulator to Elasticsearch.

Search Tier/Storage Tier: Elasticsearch will be used to act as a storage for the raw data. An event will be the signal send by a smartwatch, where each event will receive a unique index for logging purposes.

Visualization Tier: Kibana will be used to produce time-series graphs showing the data for heart rate and stress level over time for different users

2. Implementation Details

a. Setting Up the Environment

I employed the architecture on Windows 10, but it should – with minor adjustments – work on other operating systems as well. This architecture requires the following software components and versions:

Software/Packages	Version
Java Development Kit	1.8
Maven	3.5.2
Eclipse	Java Oxygen
Zookeeper	3.4.9
Kafka	2.10 // 0.10.0.0
Cassandra	3.11.1
Spark	1.6.2
Elasticsearch	5.0.0
Kibana	5.0.0
Kafka ES Connector	4.0.0

I tried to previously implement the solution in Python but ended up abandoning this approach and switched to Java because the **Python support** for an integration of Spark Streaming and Kafka was **discontinued.** Please refer to https://spark.apache.org/docs/2.2.0/streaming-kafka-integration.html for more information.

Furthermore, I ran into **issues** with integrating the **Kafka ES Connector** with **Elasticsearch 6.0.0** which is why I switched to Elasticsearch 5.0.0. The reason for this is that Elasticsearch 6.0.0 deprecated the use of multiple mapping types which are needed to use the connector. However, they are still supported in version 5.0.0, so I just used that instead of reprogramming the connector. Please refer to https://www.elastic.co/guide/en/elasticsearch/reference/6.x/removal-of-types.html for more information.

With regards to the other software components, please follow the installation guidelines given below to install them on your respective operating system:

JDK:
https://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html

Maven:
https://maven.apache.org/install.html

Eclipse:
http://www.eclipse.org/downloads/eclipse-packages/?show_instructions=TRUE

Zookeeper:

https://zookeeper.apache.org/doc/r3.1.2/zookeeperStarted.html

Kafka:
https://kafka.apache.org/quickstart

Cassandra:
http://cassandra.apache.org/doc/latest/getting_started/installing.html

Spark:
https://spark.apache.org/docs/1.6.2

Elasticsearch:
https://www.elastic.co/guide/en/elasticsearch/reference/5.0/install
- elasticsearch.html

Kibana:
https://www.elastic.co/guide/en/kibana/5.0/install.html

Kafka ES Connector:
https://github.com/confluentinc/kafka-connect-elasticsearch

In the following, I describe the process to get Zookeeper, Apache Kafka, Apache Cassandra, Elasticsearch, Kibana and the Kafka ES Connector running. I will not go into further details with regards to JDK, Maven, Eclipse and Spark. Please refer to the guidelines for help on installation issues.

Zookeeper

After downloading and unzipping the Zookeeper package, the configuration file "zoo_sample.cfg" needs to be renamed to "zoo.cfg" and

```
dataDir=/tmp/zookeeper
```

needs to be changed to

```
dataDir=\Zookeeper\data
```

Furthermore, the following entries should be added to the system environment variables:

```
ZOOKEEPER_HOME = C:\Zookeeper
```

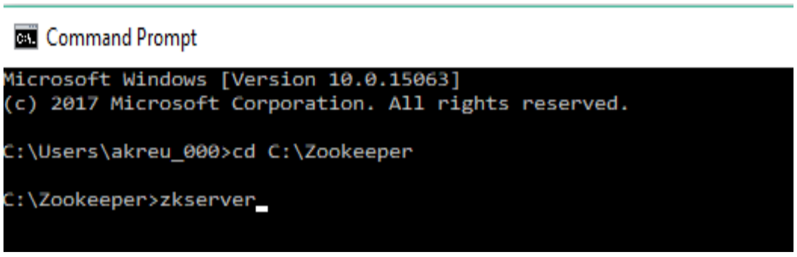
and

```
PATH = %ZOOKEEPER_HOME%\bin
```

Eventually, the Zookeeper server can be started by typing

```
> zkserver
```

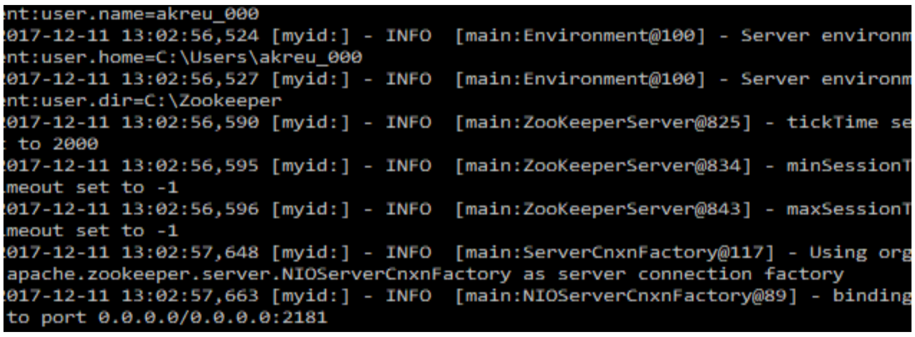
in the Windows command prompt:



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\akreu_000>cd C:\Zookeeper

C:\Zookeeper>zkserver_
```



```
nt:user.name=akreu_000
017-12-11 13:02:56,524 [myid:] - INFO  [main:Environment@100] - Server environm
nt:user.home=C:\Users\akreu_000
017-12-11 13:02:56,527 [myid:] - INFO  [main:Environment@100] - Server environm
nt:user.dir=C:\Zookeeper
017-12-11 13:02:56,590 [myid:] - INFO  [main:ZooKeeperServer@825] - tickTime se
 to 2000
017-12-11 13:02:56,595 [myid:] - INFO  [main:ZooKeeperServer@834] - minSessionT
meout set to -1
017-12-11 13:02:56,596 [myid:] - INFO  [main:ZooKeeperServer@843] - maxSessionT
meout set to -1
017-12-11 13:02:57,648 [myid:] - INFO  [main:ServerCnxnFactory@117] - Using org
apache.zookeeper.server.NIOServerCnxnFactory as server connection factory
017-12-11 13:02:57,663 [myid:] - INFO  [main:NIOServerCnxnFactory@89] - binding
to port 0.0.0.0/0.0.0.0:2181
```

Kafka

After downloading and unzipping the Kafka package, the ‘server.properties’ file in the config folder. Again, change

```
> log.dirs=/tmp/kafka-logs
```

to

```
> log.dir=C:\Kafka\kafka-logs
```

Kafka will run on the default port 9092 and Zookeeper will run on 2181. Afterwards, the Kafka server can be started by invoking

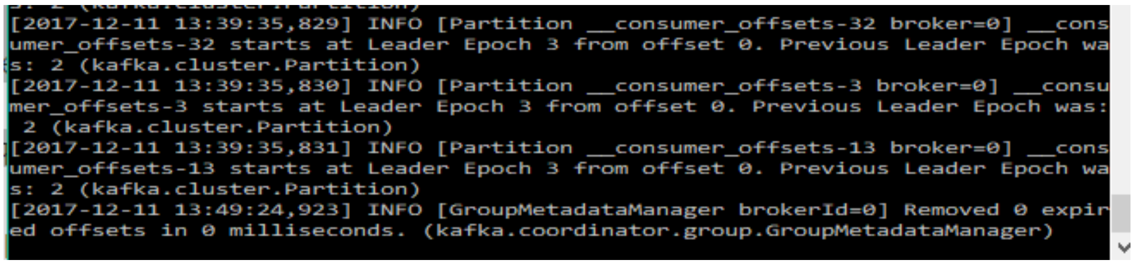
```
> cd C:\Kafka
> .\bin\windows\kafka-server-start.bat .\config\server.properties
```

in the Windows command line or

```
> .\bin\windows\kafka-server-start.sh .\config\server.properties
```

in the Cygwin tool.

The console output look like the following to indicate a successful start of the Kafka server:



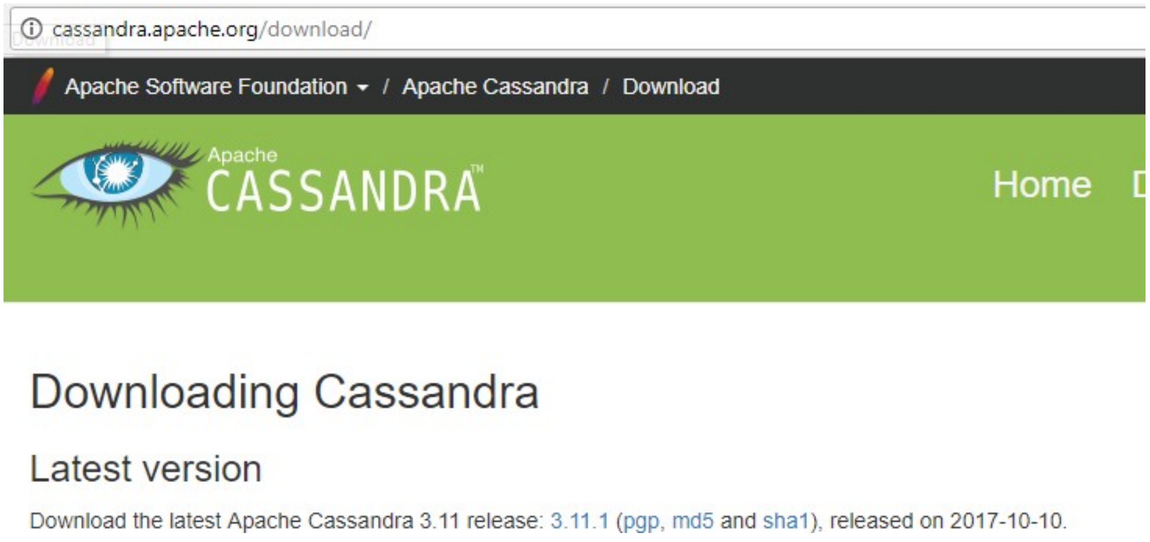
```
[2017-12-11 13:39:35,829] INFO [Partition __consumer_offsets-32 broker=0] __cons
umer_offsets-32 starts at Leader Epoch 3 from offset 0. Previous Leader Epoch wa
s: 2 (kafka.cluster.Partition)
[2017-12-11 13:39:35,830] INFO [Partition __consumer_offsets-3 broker=0] __consu
mer_offsets-3 starts at Leader Epoch 3 from offset 0. Previous Leader Epoch was:
 2 (kafka.cluster.Partition)
[2017-12-11 13:39:35,831] INFO [Partition __consumer_offsets-13 broker=0] __cons
umer_offsets-13 starts at Leader Epoch 3 from offset 0. Previous Leader Epoch wa
s: 2 (kafka.cluster.Partition)
[2017-12-11 13:49:24,923] INFO [GroupMetadataManager brokerId=0] Removed 0 expir
ed offsets in 0 milliseconds. (kafka.coordinator.group.GroupMetadataManager)
```

In a second command prompt, create a topic ‘iotdata’ by invoking the following command:

```
> .\bin\windows\kafka-topics.bat --create --zookeeper
localhost:2181 --replication-factor 1 --partitions 1 --topic
iotdata
```

Cassandra

After downloading the most recent Cassandra version from Cassandra.apache.org/download, unzip the package and move it to your desired Cassandra directory.


cassandra.apache.org/download/
Apache Software Foundation / Apache Cassandra / Download
Apache CASSANDRA
Home

Downloading Cassandra

Latest version

Download the latest Apache Cassandra 3.11 release: 3.11.1 (pgp, md5 and sha1), released on 2017-10-10.

To start Cassandra, invoke the following commands in the command prompt:

```
> cd C:\Cassandra\bin
> .\cassandra -f
```

Navigate to a new terminal and execute

```
> cqlsh
```

to start the command line tool to insert CQL statements and interact with your Cassandra database.

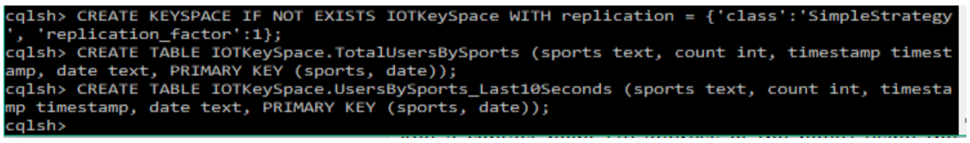
We need to create a key space and two different[i] tables[ii] before starting to execute our smartwatch simulation. We do this by invoking the following commands in the CQLSH prompt[iii]:

```
> CREATE KEYSPACE IF NOT EXISTS IOTKeySpace WITH
replication = {'class':'SimpleStrategy',
'replication_factor':1};

> CREATE TABLE IOTKeySpace.TotalUsersBySports (sports
text, count int, timestamp
timestamp, date text, PRIMARY KEY
(sports, date));

> CREATE TABLE IOTKeySpace.UsersBySports_Last10Seconds
(sports
text, count int, timestamp
timestamp, date text, PRIMARY KEY
(sports, date));
```

The primary keys were chosen to support the inherent structure of the queries and optimize queuing time.



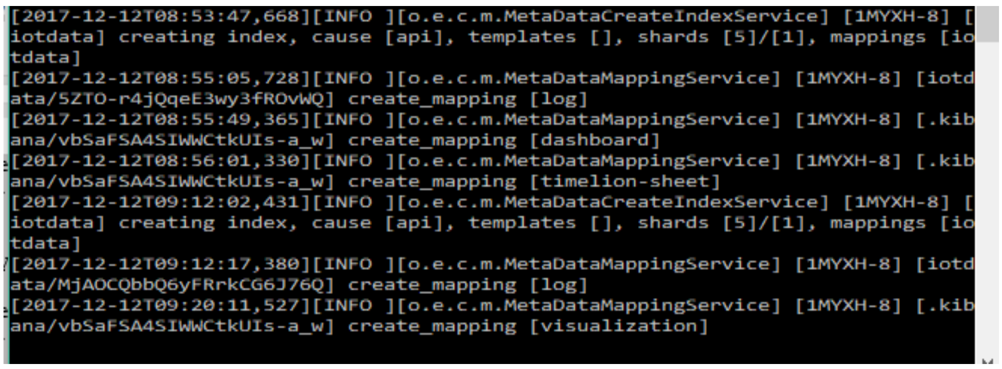
```
cqlsh> CREATE KEYSPACE IF NOT EXISTS IOTKeySpace WITH replication = {'class':'SimpleStrategy
', 'replication_factor':1};
cqlsh> CREATE TABLE IOTKeySpace.TotalUsersBySports (sports text, count int, timestamp timest
amp, date text, PRIMARY KEY (sports, date));
cqlsh> CREATE TABLE IOTKeySpace.UsersBySports_Last10Seconds (sports text, count int, timesta
mp timestamp, date text, PRIMARY KEY (sports, date));
cqlsh>
```

Elasticsearch

The setup of Elasticsearch is straight-forward. After downloading and unzipping the respective package for Elasticsearch 5.0.0 (IMPORTANT! Don't use 6.0.0, see above for the reasoning behind this), the following commands can be used to start Elasticsearch:

```
> cd C:\Elasticsearch
> .\bin\elasticsearch
```

The console output should look like this:



```
[2017-12-12T08:53:47,668][INFO ][o.e.c.m.MetaDataCreateIndexService] [1MYXH-8] [
iotdata] creating index, cause [api], templates [], shards [5]/[1], mappings [io
tdata]
[2017-12-12T08:55:05,728][INFO ][o.e.c.m.MetaDataMappingService] [1MYXH-8] [iotd
ata/5ZTO-r4jQqeE3wy3fROvWQ] create_mapping [log]
[2017-12-12T08:55:49,365][INFO ][o.e.c.m.MetaDataMappingService] [1MYXH-8] [.kib
ana/vbSaFSA4SIWWCtkUIs-a_w] create_mapping [dashboard]
[2017-12-12T08:56:01,330][INFO ][o.e.c.m.MetaDataMappingService] [1MYXH-8] [.kib
ana/vbSaFSA4SIWWCtkUIs-a_w] create_mapping [timelion-sheet]
[2017-12-12T09:12:02,431][INFO ][o.e.c.m.MetaDataCreateIndexService] [1MYXH-8] [
iotdata] creating index, cause [api], templates [], shards [5]/[1], mappings [io
tdata]
[2017-12-12T09:12:17,380][INFO ][o.e.c.m.MetaDataMappingService] [1MYXH-8] [iotd
ata/MjAOCQbbQ6yFRrkCG6J76Q] create_mapping [log]
[2017-12-12T09:20:11,527][INFO ][o.e.c.m.MetaDataMappingService] [1MYXH-8] [.kib
ana/vbSaFSA4SIWWCtkUIs-a_w] create_mapping [visualization]
```

Use a simple PUT request to ensure that Elasticsearch is up and running:

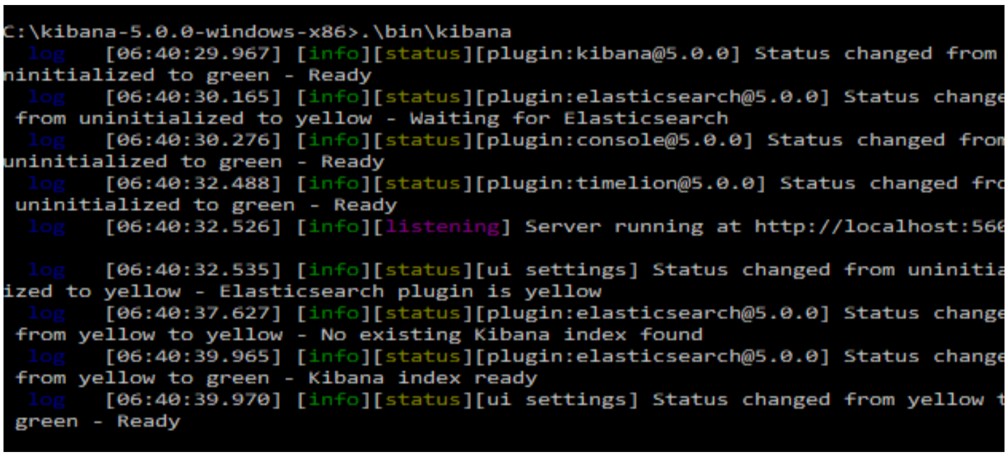
```
curl -XPUT 'http://localhost:9200/_search
```

Kibana

As simple to setup as Elasticsarch. However, it is important to start the Elasticsearch server before starting Kibana. Also, we need to use Kibana 5.0.0 instead of 6.0.0 to ensure compatibility within the architecture. After downloading and unzipping the respective package for Kibana 5.0.0, the following commands can be used to start Kibana:

```
> cd C:\Elasticsearch\Kiabana
> .\bin\kibana
```

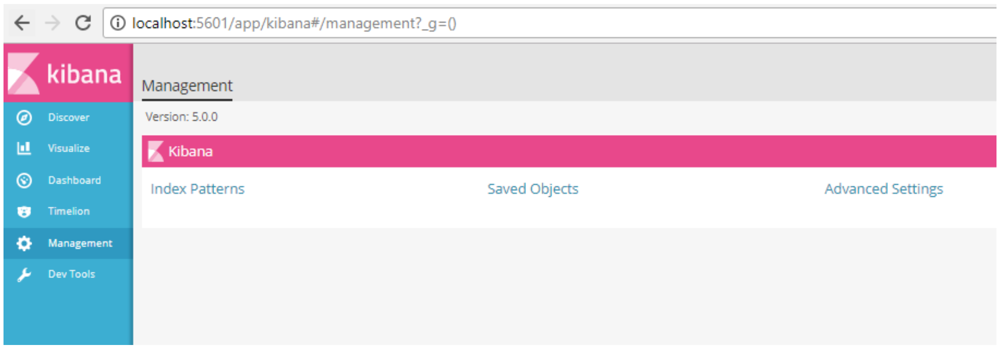
The console output should look like this:



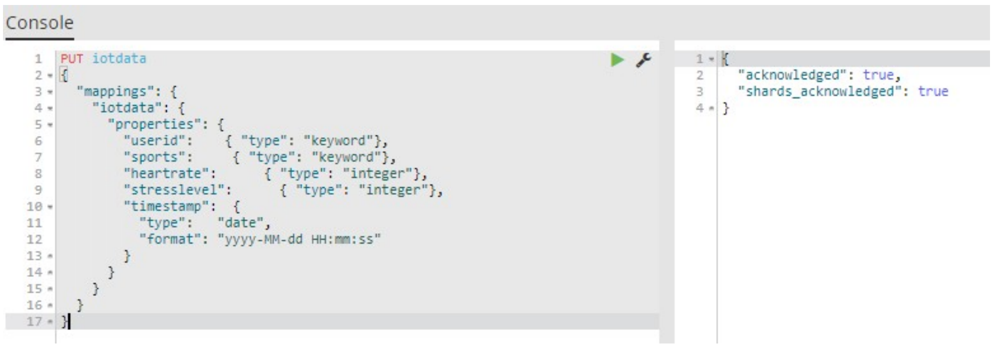
```
C:\kibana-5.0.0-windows-x86>.\bin\kibana
  log   [06:40:29.967] [info][status][plugin:kibana@5.0.0] Status changed from
ninitialized to green - Ready
  log   [06:40:30.165] [info][status][plugin:elasticsearch@5.0.0] Status change
 from uninitialized to yellow - Waiting for Elasticsearch
  log   [06:40:30.276] [info][status][plugin:console@5.0.0] Status changed from
uninitialized to green - Ready
  log   [06:40:32.488] [info][status][plugin:timelion@5.0.0] Status changed fro
 uninitialized to green - Ready
  log   [06:40:32.526] [info][listening] Server running at http://localhost:560

  log   [06:40:32.535] [info][status][ui settings] Status changed from uninitia
ized to yellow - Elasticsearch plugin is yellow
  log   [06:40:37.627] [info][status][plugin:elasticsearch@5.0.0] Status change
 from yellow to yellow - No existing Kibana index found
  log   [06:40:39.965] [info][status][plugin:elasticsearch@5.0.0] Status change
 from yellow to green - Kibana index ready
  log   [06:40:39.970] [info][status][ui settings] Status changed from yellow t
 green - Ready
```

If everything worked according to plan, Kibana can be reached under http://localhost:5601 from a browser of your choice. You should see a working Kibana environment:



Once Kibana is active, we use the included Dev Tools (left menu, last entry) to create an index iotdata_index and assign it a mapping, so that Elasticsearch knows right away which data types are being sent via the stream:



```
PUT iotdata
{
  "mappings": {
    "iotdata": {
      "properties": {
        "userid":     { "type": "keyword"},
        "sports":     { "type": "keyword"},
        "heartrate":      { "type": "integer"},
        "stresslevel":      { "type": "integer"},
        "timestamp":  {
          "type":   "date",
          "format": "yyyy-MM-dd HH:mm:ss"
        }
      }
    }
  }
}
```

```
{
  "acknowledged": true,
  "shards_acknowledged": true
}
```

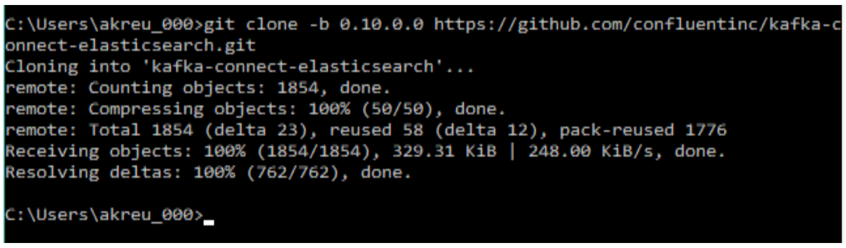
The mapping is:

```
PUT iotdata
{
  "mappings": {
    "iotdata": {
      "properties": {
        "userid":    { "type": "keyword"},
        "sports":     { "type": "keyword"},
        "heartrate":      { "type": "integer"},
        "stresslevel":      { "type": "integer"},
        "timestamp":{
          "type":   "date",
          "format": "yyyy-MM-dd HH:mm:ss"
        }
      }
    }
  }
}
```

Kafka Elasticsearch Connector

Starting the Kafka Elasticsearch Connector is a bit more involved process[iv]. First, the version compatible with Kafka version 0.10.0.0 needs to be cloned from github using

```
> git clone -b 0.10.0.0
https://github.com/confluentinc/kafka-connect-
elasticsearch.git
```



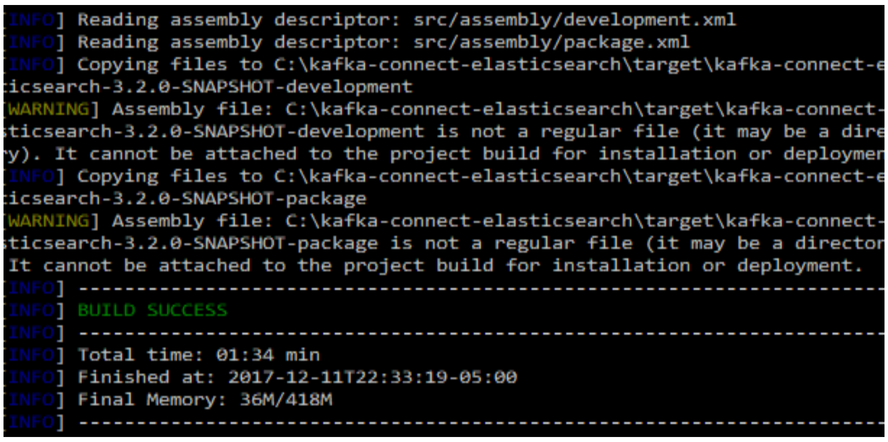
```
C:\Users\akreu_000>git clone -b 0.10.0.0 https://github.com/confluentinc/kafka-c
onnect-elasticsearch.git
Cloning into 'kafka-connect-elasticsearch'...
remote: Counting objects: 1854, done.
remote: Compressing objects: 100% (50/50), done.
remote: Total 1854 (delta 23), reused 58 (delta 12), pack-reused 1776
Receiving objects: 100% (1854/1854), 329.31 KiB | 248.00 KiB/s, done.
Resolving deltas: 100% (762/762), done.

C:\Users\akreu_000>
```

Afterwards, we build the package by invoking the following commands[v] in the command line:

```
> cd kafka-connect-elasticsearch
> mvn clean package
```

This should lead to the following output:



```
INFO] Reading assembly descriptor: src/assembly/development.xml
INFO] Reading assembly descriptor: src/assembly/package.xml
INFO] Copying files to C:\kafka-connect-elasticsearch\target\kafka-connect-e
icsearch-3.2.0-SNAPSHOT-development
WARNING] Assembly file: C:\kafka-connect-elasticsearch\target\kafka-connect-
ticsearch-3.2.0-SNAPSHOT-development is not a regular file (it may be a dire
y). It cannot be attached to the project build for installation or deploymen
INFO] Copying files to C:\kafka-connect-elasticsearch\target\kafka-connect-e
icsearch-3.2.0-SNAPSHOT-package
WARNING] Assembly file: C:\kafka-connect-elasticsearch\target\kafka-connect-
ticsearch-3.2.0-SNAPSHOT-package is not a regular file (it may be a director
It cannot be attached to the project build for installation or deployment.
INFO] ------------------------------------------------------------------------
INFO] BUILD SUCCESS
INFO] ------------------------------------------------------------------------
INFO] Total time: 01:34 min
INFO] Finished at: 2017-12-11T22:33:19-05:00
INFO] Final Memory: 36M/418M
INFO] ------------------------------------------------------------------------
```

After the build is finished, go to the *target/kafka-connect-elasticsearch- 3.2.0-SNAPSHOT-package/share/java/kafka-connect-elasticsearch/* directory and copy all the files in there into the Kafka *libs* directory.

We will run the ES connector in standalone mode. To be able to do that, we need to create a new configuration file, elasticsearch-connect.properties in the conf-folder of the Kafka directory. In that file, insert

```
name=elasticsearch-sink
connector.class=io.confluent.connect.elasticsearch.Ela
sticsearchSinkConnector
tasks.max=1
topics=iotdata
topic.index.map=iotdata:iotdata_index
connection.url=http://localhost:9200
type.name=log
key.ignore=true
schema.ignore=true
```

key.ignore and schema.ignore are set to true because we already assigned a mapping in elasticsearch to that index.

Furthermore, we need to alter the connect-standalone.properties file in the Kafka conf directory. It should include the following content:

```
bootstrap.servers=localhost:9092
key.converter=org.apache.kafka.connect.json.JsonConver
ter
value.converter=org.apache.kafka.connect.json.JsonConv
erter
key.converter.schemas.enable=false
value.converter.schemas.enable=false
internal.key.converter=org.apache.kafka.connect.json.J
sonConverter
internal.value.converter=org.apache.kafka.connect.json
.JsonConverter
internal.key.converter.schemas.enable=false
internal.value.converter.schemas.enable=false
offset.storage.file.filename=/tmp/connect.offsets
offset.flush.interval.ms=10000
```

An important thing to notice here is that the way we set up the connector, we can only parse JSON streams with it, no other format.

We start the connector with the following command:

```
> bin\connect-standalone.sh config\connect-
standalone.properties config\elasticsearch-
connect.properties
```

Elasticsearch will now “subscribe” to topic “iotdata” and every message that gets send to this topic will be directed to and parsed by Elasticsearch.

b. Code

In the following, the JAVA classes[vi] necessary to simulate the smartwatch data[vii] and to send it to the 'iotdata' topic are explained.

Package: watchdata.producer

This package is responsible for creating Smartwatch data and sending it via the Kafka Producer.

Smartwatch class

The Smartwatch class is the cornerstone of the data producer. It is responsible for connecting to Kafka and Zookeeper and it creates events in the required JSON format before sending them off via the Kafka Producer to the 'iotdata' topic.

Data Encoder

The Data Encoder Class encodes the events.

Package: watchdata.processor

The processor package is used to parse and process the data from the Kafka Producer. It is responsible for calculating aggregated metrics and sending them to the Cassandra database.

TotalUserData

In this class, the object structure for the first of the two aggregated measures is set: The count of total users per sports.

WindowUserData

In this class, the object structure for the second of the two aggregated measures is set: The count of total users per sports within the last 30 seconds.

DataProcessor

The data processor is the core of the processor package. Here, we will parse the incoming JSON objects and calculate our metrics before writing the results to Cassandra. For the first metric, the total user count by sports parsed by date, we need to process both current streaming data and historical data of that day. For the second measure, which counts the users per sports within the last 30 seconds, we need to employ a windowed datastream.

DataDecoder

The Data Decoder class corresponds to the Data Encoder Class within the producer package and is responsible for decoding the incoming objects.

Data

This class is a copy of the Data Class within the producer package. Again, it gives structure to the data that the smartwatches are sending.

Aggregate Key

This class is used to construct aggregate for mapping the data streams in Cassandra.

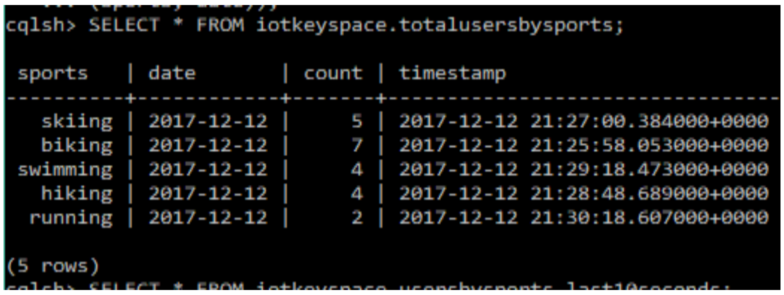
Data Streamer

This class sets the Spark-Context and parses data from the incoming JSON stream of events sent by the smartwatches. It furthermore ensures that the incoming stream batches are being reduced by key (here: userId) to allow for counting unique users in the aggregated metrics we'll later store in Cassandra.

3. Results

The code works as expected. In Cassandra, we start the CQLSH console by executing the following command in the Windows CLI:

```
> cd \path\to\Cassandra\directory\bin
> cqlsh
> SELECT * FROM iotkeyspace.totalusersbysports;
```

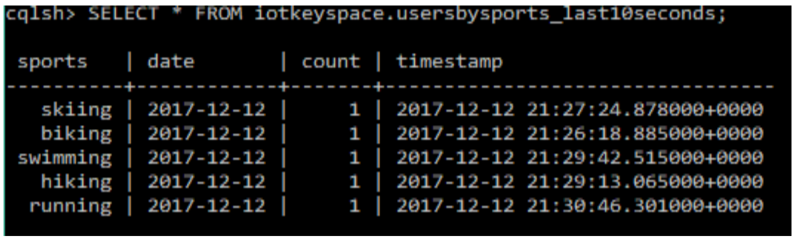


```
cqlsh> SELECT * FROM iotkeyspace.totalusersbysports;

 sports   | date       | count | timestamp
----------+------------+-------+---------------------------------
   skiing | 2017-12-12 |     5 | 2017-12-12 21:27:00.384000+0000
   biking | 2017-12-12 |     7 | 2017-12-12 21:25:58.053000+0000
 swimming | 2017-12-12 |     4 | 2017-12-12 21:29:18.473000+0000
   hiking | 2017-12-12 |     4 | 2017-12-12 21:28:48.689000+0000
  running | 2017-12-12 |     2 | 2017-12-12 21:30:18.607000+0000

(5 rows)
```

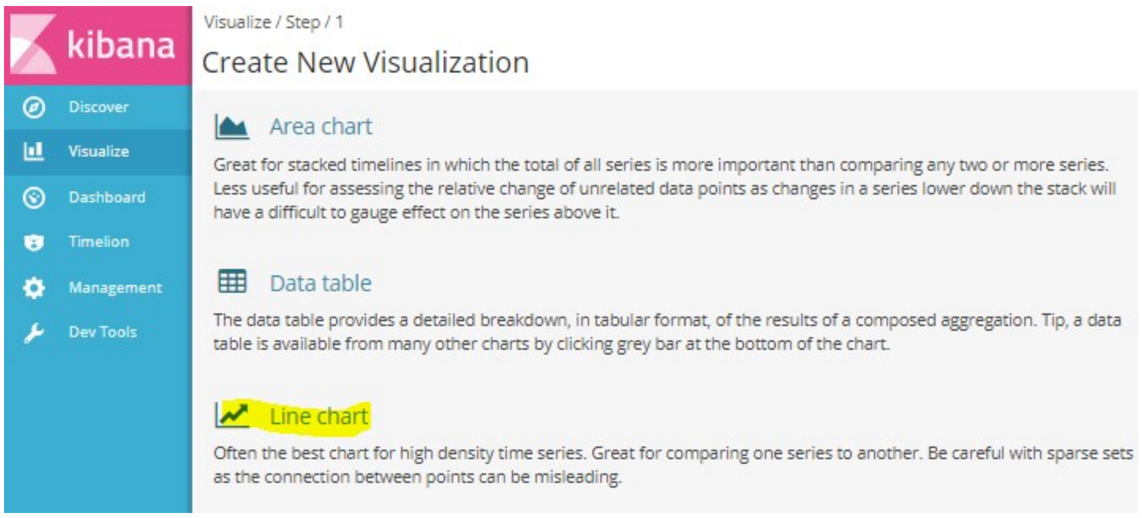
```
> SELECT * FROM iotkeyspace.usersbysports_last10seconds;
```



```
cqlsh> SELECT * FROM iotkeyspace.usersbysports_last10seconds;

 sports   | date       | count | timestamp
----------+------------+-------+---------------------------------
   skiing | 2017-12-12 |     1 | 2017-12-12 21:27:24.878000+0000
   biking | 2017-12-12 |     1 | 2017-12-12 21:26:18.885000+0000
 swimming | 2017-12-12 |     1 | 2017-12-12 21:29:42.515000+0000
   hiking | 2017-12-12 |     1 | 2017-12-12 21:29:13.065000+0000
  running | 2017-12-12 |     1 | 2017-12-12 21:30:46.301000+0000
```

In Kibana, we navigate to "Visualize" on the left-hand side menu of the screen. We are prompted to create a new visualization. To obtain the time-series graphs we are interested in, we click on "Line Chart" and select our "iotdata" index.



Visualize / Step / 1

Create New Visualization

Area chart

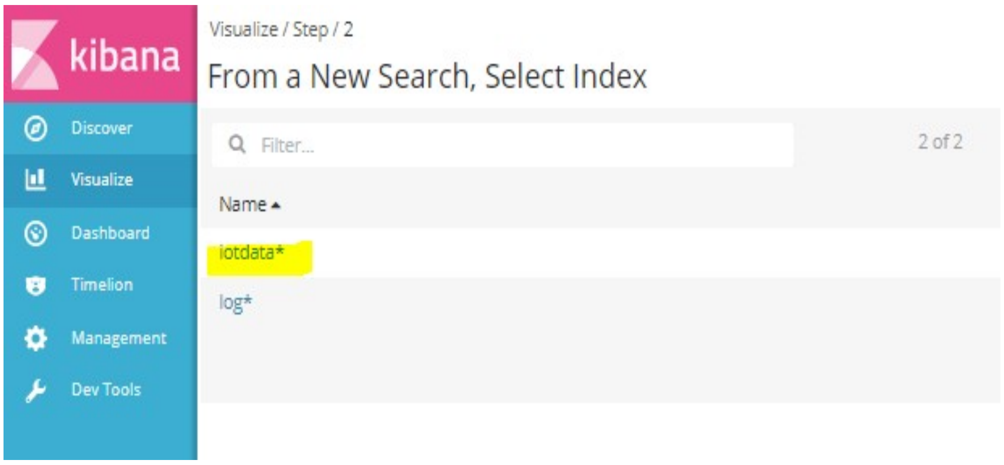
Great for stacked timelines in which the total of all series is more important than comparing any two or more series. Less useful for assessing the relative change of unrelated data points as changes in a series lower down the stack will have a difficult to gauge effect on the series above it.

Data table

The data table provides a detailed breakdown, in tabular format, of the results of a composed aggregation. Tip, a data table is available from many other charts by clicking grey bar at the bottom of the chart.

Line chart

Often the best chart for high density time series. Great for comparing one series to another. Be careful with sparse sets as the connection between points can be misleading.



Visualize / Step / 2

From a New Search, Select Index

Filter...

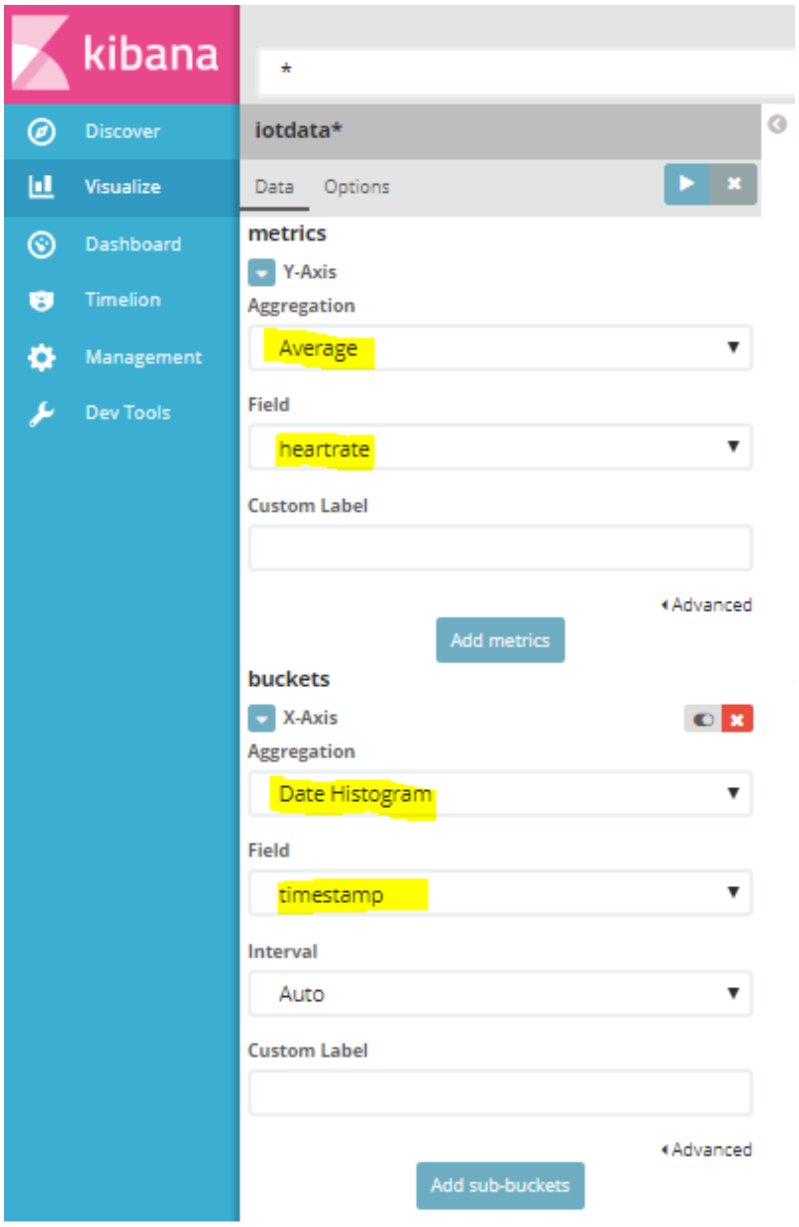
2 of 2

Name ▲

iotdata*

log*

Afterwards, we select “avg” and “heartrate” (or “stresslevel”, respectively) for the y-axis and “Date Histogram” and “timestamp” for the x-axis.



*

iotdata*

metrics

Y-Axis

Aggregation: Average

Field: heartrate

Custom Label

buckets

X-Axis

Aggregation: Date Histogram

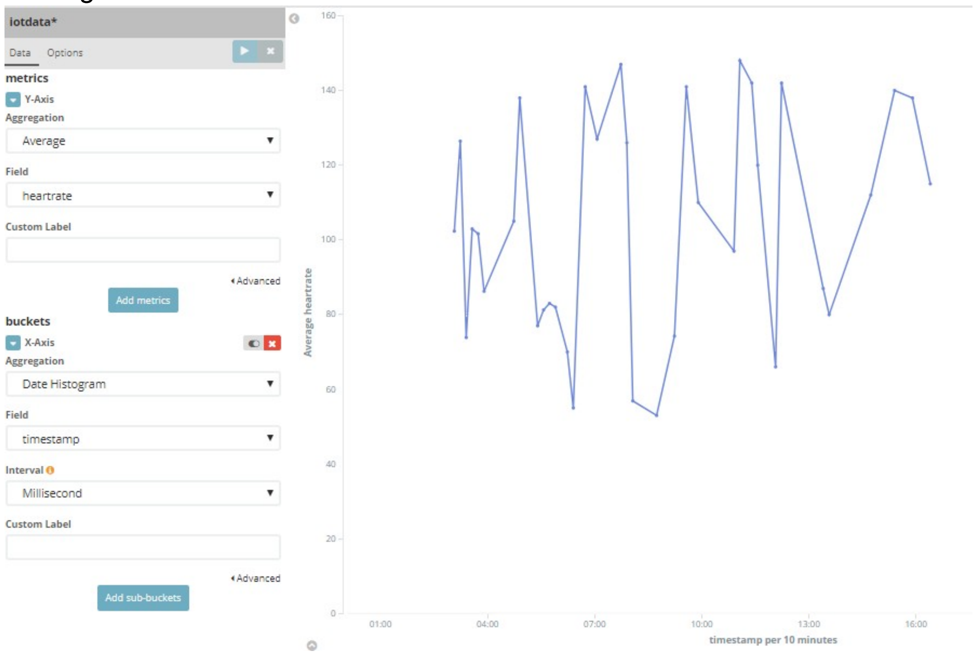
Field: timestamp

Interval: Auto

Custom Label

This yields to the following graphs:

Average of variable “heartrate” over time:


iotdata*

Data Options

metrics

Y-Axis

Aggregation: Average

Field: heartrate

Custom Label

buckets

X-Axis

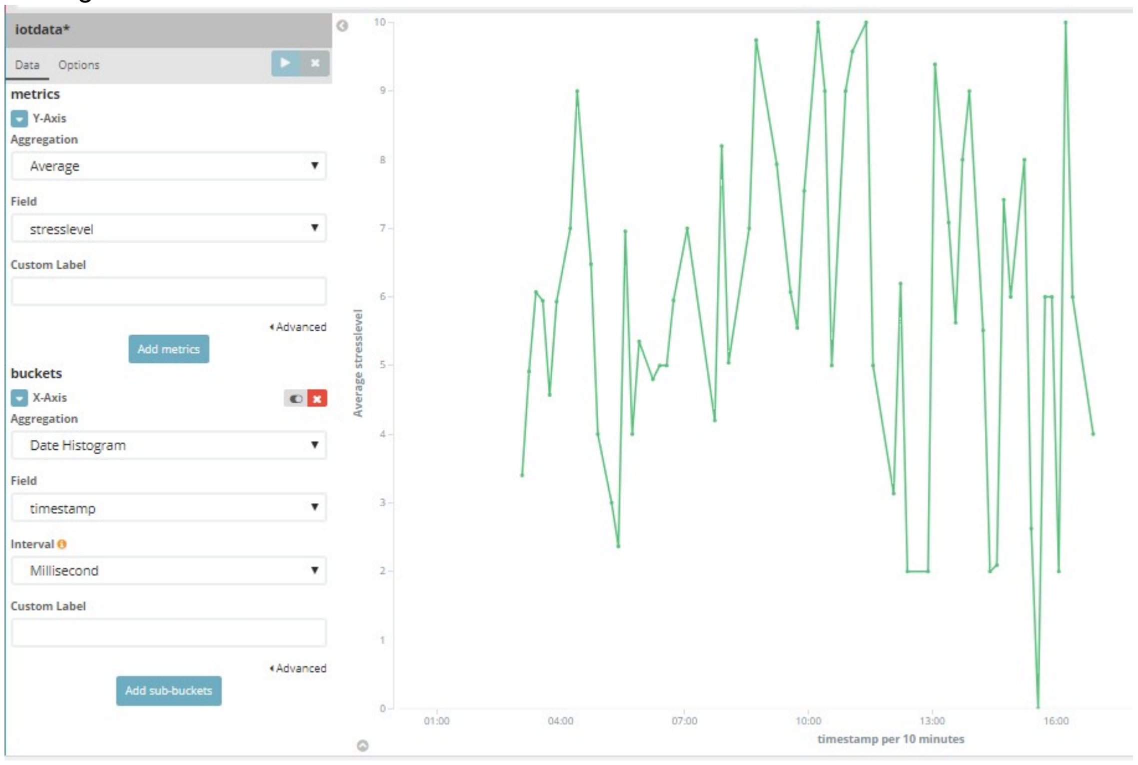
Aggregation: Date Histogram

Field: timestamp

Interval: Millisecond

Custom Label

Average of variable “stresslevel” over time:


iotdata*

Data Options

metrics

Y-Axis

Aggregation: Average

Field: stresslevel

Custom Label

buckets

X-Axis

Aggregation: Date Histogram

Field: timestamp

Interval: Millisecond

Custom Label

We successfully developed an architecture to visualize and analyte data from multiple smartwatches.

4. Conclusions and Lessons Learned

Originally, I wanted to include Re:dash as a visualization tool for Apache Cassandra. However, I missed that Re:dash doesn't run on my local machine but on remote servers. This means that I would've had to allow access from the internet to my database and to configure my data source accordingly. While this would have been possible, especially with a tool like Ngrok (https://ngrok.com/) to facilitate this, it would have meant an increased security risk. As I am no professional in security concerns, I did not feel comfortable to implement this solution and share it with the community, in case someone wanted to use it for professional purposes.

Another thing that I learned is that when planning an architecture, one not only has to pay attention to the high-level structure, i.e. how the individual components could fit together, but also to the version compatibility between the components. This cost me a lot of time and, as mentioned above, I started out with writing the code in Python until I realized that the Spark Streaming and Kafka integration support was discontinued, which let me start my project again in Java. So next time, I'll check version compatibility first before head starting into programming my code.

i https://databricks.com/blog/2017/02/23/working-complex-data-formats-structured-streaming-apache-spark-2-1.html
ii http://abiasforaction.net/a-practical-introduction-to-cassandra-query-language/
iii https://www.slideshare.net/mattdennis/cassandra-nyc-2011-data-modeling
iv https://sematext.com/blog/kafka-connect-elasticsearch-how-to/
v https://docs.confluent.io/current/connect/userguide.html
vi https://acadgild.com/blog/streaming-twitter-data-using-kafka/
vii https://www.infoq.com/articles/traffic-data-monitoring-iot-kafka-and-spark-streaming