Homework 9: Kafka

E-63 Big Data Analytics
Harvard University, Autumn 2017

Tim Hagmann

November 04, 2017

Problem 1 (25%)

Question: Install Kafka on you Linux VM.

Install Kafka

I installed the kafka server on my EC2 Ubuntu 16.04 instance. On the instance are R-Studio Server 1.1, Spark 2.2 and python 2.7 as well as python 3 installed (see previous homeworks for the installation instructions). The following commands install kafka on an ubuntu machine:

```
wget http://mirror.switch.ch/mirror/apache/dist/kafka/0.11.0.1/kafka_2.11-0.11.0.1.tgz
sudo mkdir /opt/kafka
sudo tar -xvf kafka_2.11-0.11.0.1.tgz
sudo mv kafka_2.11-0.11.0.1/* /opt/kafka
sudo rm -r kafka_2.11-0.11.0.1
```

Add path

Question: Create an environmental variable KAFKA_HOME pointing to that directory. Place the directory KAFKA_HOME/bin in the PATH variable in the .bash_profile file in your home directory. Source .bash_profile.

The following lines add kafka to the path:

```
# Add Kafka to the home directory
echo "export KAFKA_HOME='/opt/kafka'" >> /home/tim/.bashrc
echo "export PATH=$PATH:$KAFKA_HOME/bin" >> /home/tim/.bashrc

# Source file
source /home/tim/.bashrc
```

Question: Make sure that Zookeeper server is started. Kafka configuration files reside in the directories: $KAFKA_HOME/config. Create a topic. Demonstrate that provided scripting client kafka-console-producer.sh receives and displays messages produced by kafka-console-consumer.sh client.

Run zookeper

After installing Kafka, we use the following to launch Zookeeper:

```
sudo /opt/kafka/bin/zookeeper-server-start.sh /opt/kafka/config/zookeeper.properties
```

Test zookeper.

The zookeper installation kan be tested with the following command:

```
netstat -ant | grep :2181
```

```
tim@ip-172-31-24-35:~/e63$ netstat -ant | grep :2181
tcp6       0      0 :::2181                 :::*                    LISTEN
tcp6       0      0 127.0.0.1:2181          127.0.0.1:55044         ESTABLISHED
tcp6       0      0 127.0.0.1:55044         127.0.0.1:2181          ESTABLISHED
```

As can be seen above, zookeper is listening on port 2181.

Start spark

Next we have to start spark on the machine. The option *localhost[2]* indicates that 2 threads are beeing used.

```
sudo /home/tim/spark/sbin/start-all.sh --host localhost[2]
```

Start kafka

With the following command is kafka started:

```
sudo /opt/kafka/bin/kafka-server-start.sh /opt/kafka/config/server.properties
```

Create partition

I'm creating the topic *tim* with *1 partition*. We could also speed up data processing by adding more partitions. Kafka would check, that the order accross the partictions is kept. However, 1 partition is easier to visualize with he output. That is why only 1 partition is chosen.

```
sudo /opt/kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 \
--replication-factor 1  --partitions 1 --topic tim
```

List topics

We can list the topic with the following command:

```
sudo /opt/kafka/bin/kafka-topics.sh --describe --zookeeper localhost:2181
```

```
tim@ip-172-31-24-35:~/e63$ sudo /opt/kafka/bin/kafka-topics.sh --describe \
--zookeeper localhost:2181
Topic:tim       PartitionCount:1        ReplicationFactor:1     Configs:
        Topic: tim      Partition: 0    Leader: 0       Replicas: 0     Isr: 0
```

Producer Terminal

Now we start our producer with:

```
sudo /opt/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic tim
```

```
tim@ip-172-31-24-35:~/e63$ sudo /opt/kafka/bin/kafka-console-producer.sh \
--broker-list localhost:9092 --topic tim
>test
```

```
>Hello World!
>
```

Consumer Terminal

And then our consumer with

```
sudo /opt/kafka/bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic tim
```

```
tim@ip-172-31-24-35:~/e63$ sudo /opt/kafka/bin/kafka-console-consumer.sh \
--zookeeper localhost:2181 --topic tim
Using the ConsoleConsumer with old consumer is deprecated and will be removed
in a future major release. Consider using the new consumer by passing
[bootstrap-server] instead of [zookeeper].
test
Hello World!
```

Problem 2 (25%)

Question: Make supplied python script kafka_consumer.py receive messages produced by supplied python script kafka_producer.py. Modify kafka_producer.py so that you can pass server name and the port of the Kafka broker and the name of Kafka topic on the command line.

Get data

Before we can start sending data to the producer we have to get the data onto the image. We're doing that trough git/github.

```
git clone https://github.com/greenore/e63.git
cd e63

# Untar
tar xvzf orders.tar-1.gz
```

Install python kafka library

Next we have to install the kafka library to python

```
pip3 install kafka
```

i) Make Script work

I had to rewirte the *kafka_producer.py* file, so that it works with python 3.

```
from kafka import KafkaProducer
import time
import sys

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: kafka_producer.py <broker_list> <topic>", file=sys.stderr)
```

```
        exit(-1)

    broker_list, topic = sys.argv[1:]

    producer = KafkaProducer(bootstrap_servers=broker_list)

    for batch in range(3):
           print('Starting batch #' + str(batch))
           for i in range(4):
                   print('sending message #' + str(i))
                   message = 'test message #' + str(i)
                   producer.send(topic, value=message.encode())
           print('Finished batch #' + str(batch))
           print('Sleeping for 5 seconds ...')
           time.sleep(5)

    print('Done sending messages')
```

Next we're starting the producer file on *port 9092*.

```
python3 kafka_producer.py localhost:9092 tim
```

```
tim@ip-172-31-24-35:~/e63$ python3 kafka_producer.py localhost:9092 tim
Starting batch #0
sending message #0
sending message #1
sending message #2
sending message #3
Finished batch #0
Sleeping for 5 seconds ...
Starting batch #1
sending message #0
sending message #1
sending message #2
sending message #3
Finished batch #1
Sleeping for 5 seconds ...
Starting batch #2
sending message #0
sending message #1
sending message #2
sending message #3
Finished batch #2
Sleeping for 5 seconds ...
Done sending messages
```

With that we can start the kafka_consumer file.

```
python3 kafka_consumer.py localhost:9092 tim
```

```
tim@ip-172-31-24-35:~/e63$ python3 kafka_consumer.py localhost:9092 tim
Topic is:  tim
Group is:  my-group1

...
```

```
got msg:  ConsumerRecord(topic='tim', partition=0, offset=200967,
timestamp=1509816176691, timestamp_type=0, key=None, value=b'test message #0',
checksum=-501606557, serialized_key_size=-1, serialized_value_size=15)
partition:  0 message offset:  200967
got msg:  ConsumerRecord(topic='tim', partition=0, offset=200968,
timestamp=1509816176692, timestamp_type=0, key=None, value=b'test message #1',
checksum=-1543171966, serialized_key_size=-1, serialized_value_size=15)
partition:  0 message offset:  200968
got msg:  ConsumerRecord(topic='tim', partition=0, offset=200969,
timestamp=1509816176693, timestamp_type=0, key=None, value=b'test message #2',
checksum=-1138154465, serialized_key_size=-1, serialized_value_size=15)
partition:  0 message offset:  200969
got msg:  ConsumerRecord(topic='tim', partition=0, offset=200970,
timestamp=1509816176706, timestamp_type=0, key=None, value=b'test message #3',
checksum=1566005260, serialized_key_size=-1, serialized_value_size=15)
partition:  0 message offset:  200970
got msg:  ConsumerRecord(topic='tim', partition=0, offset=200971,
timestamp=1509816181710, timestamp_type=0, key=None, value=b'test message #0',
checksum=-281509412, serialized_key_size=-1, serialized_value_size=15)
partition:  0 message offset:  200971
got msg:  ConsumerRecord(topic='tim', partition=0, offset=200972,
timestamp=1509816181711, timestamp_type=0, key=None, value=b'test message #1',
checksum=421188717, serialized_key_size=-1, serialized_value_size=15)
partition:  0 message offset:  200972
got msg:  ConsumerRecord(topic='tim', partition=0, offset=200973,
timestamp=1509816181711, timestamp_type=0, key=None, value=b'test message #2',
checksum=-2146204201, serialized_key_size=-1, serialized_value_size=15)
partition:  0 message offset:  200973
got msg:  ConsumerRecord(topic='tim', partition=0, offset=200974,
timestamp=1509816181711, timestamp_type=0, key=None, value=b'test message #3',
checksum=-149637823, serialized_key_size=-1, serialized_value_size=15)
partition:  0 message offset:  200974
got msg:  ConsumerRecord(topic='tim', partition=0, offset=200975,
timestamp=1509816186716, timestamp_type=0, key=None, value=b'test message #0',
checksum=-1485067759, serialized_key_size=-1, serialized_value_size=15)
partition:  0 message offset:  200975
got msg:  ConsumerRecord(topic='tim', partition=0, offset=200976,
timestamp=1509816186717, timestamp_type=0, key=None, value=b'test message #1',
checksum=1364852640, serialized_key_size=-1, serialized_value_size=15)
partition:  0 message offset:  200976
got msg:  ConsumerRecord(topic='tim', partition=0, offset=200977,
timestamp=1509816186717, timestamp_type=0, key=None, value=b'test message #2',
checksum=-934236646, serialized_key_size=-1, serialized_value_size=15)
partition:  0 message offset:  200977
got msg:  ConsumerRecord(topic='tim', partition=0, offset=200978,
timestamp=1509816186717, timestamp_type=0, key=None, value=b'test message #3',
checksum=-1084776820, serialized_key_size=-1, serialized_value_size=15)
partition:  0 message offset:  200978
```

ii) Read only input

Question: Also, modify that script so that it continuously reads your terminal inputs and sends every line to Kafka consumer. Demonstrate that kafka_consumer.py can read and display messages of modified

kafka_producer.py. Provide working code of modified kafka_producer.py. Describe to us the process of installing Python packages, if any, you needed for this problem.

The following producer is able to read continuously the terminal messages.

```
from kafka import KafkaProducer
import time
import sys

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: kafka_producer2.py <broker_list> <topic>", file=sys.stderr)
        exit(-1)

    broker_list, topic = sys.argv[1:]

    producer = KafkaProducer(bootstrap_servers=broker_list)

    for i in sys.stdin:
        message = 'typed message #' + str(i)
        producer.send(topic, value=message.encode())
```

When starting the *kafka_producer2.py* script we can enter different text.

```
python3 kafka_producer2.py localhost:9092 tim
```

```
tim@ip-172-31-24-35:~/e63$ python3 kafka_producer2.py localhost:9092 tim
Hello World!
Test
```

The *kafka_consumer.py* skript is able to read the upper intput.

```
python3 kafka_consumer.py localhost:9092 tim
```

```
tim@ip-172-31-24-35:~/e63$ python3 kafka_consumer.py localhost:9092 tim
Topic is:  tim
Group is:  my-group1

...

got msg:  ConsumerRecord(topic='tim', partition=0, offset=25,
timestamp=1509730033483, timestamp_type=0, key=None,
value=b'typed message #Hello World!\n', checksum=-2029186398,
serialized_key_size=-1, serialized_value_size=28)
partition:  0 message offset:  25
got msg:  ConsumerRecord(topic='tim', partition=3, offset=19,
timestamp=1509730034996, timestamp_type=0, key=None,
value=b'typed message #Test\n', checksum=-1409034517,
serialized_key_size=-1, serialized_value_size=20)
partition:  3 message offset:  19
```

Problem 3 (25%)

Question: Rather than using splitAndSend.sh bash script to generate traffic towards Spark Streaming engine, write a Kafka Producer which will read orders.txt file and send 1,000 orders to a Kafka topic every second. Write a Kafka consumer that will deliver those batches of orders to Spark Streaming engine. Base your Kafka

consumer on provided *direct_word_count.py* script. Let Spark streaming engine count the number of orders different stocks where bought in each batch. Display for us a section of results in your solution. Describe to us the process of installing and invoking Python packages, if any, you needed for this problem.

Install dependencies

First we're installing the necessary dependencies

```
sudo pip3 install kafka-python
sudo pip3 install future
```

Create producer

Then we're adapting the producer script that reads the *orders.txt*, splits them into batches of the size 1'000 and transfers them to kafka.

```
# Libraries
from __future__ import print_function
import sys
import time
import itertools

import findspark
findspark.init("/home/tim/spark")

from kafka import KafkaProducer

# __name__
if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: direct_kafka_wordcount.py <broker_list> <topic>", file=sys.stderr)
        exit(-1)

    broker_list, topic = sys.argv[1:]
    producer = KafkaProducer(bootstrap_servers=broker_list)

    # Open file
    file = open('/home/tim/e63/orders.txt')
    rdd_file = file.read()
    rdd_split = rdd_file.split('\n')

    # Loop trough file (the orders file has 500'000 rows)
    for batch in range(500):
        print('Start batch #' + str(batch))
        for i in range(1000):
            place = i + (batch * 1000)
            print(rdd_split[place])
            producer.send(topic, rdd_split[place].encode())
        print('Finish batch #' + str(batch))
        print('Sleep 1 second')
        time.sleep(1)
```

```
    # End
    print('Finished!')
```

Running the scripts leads to the following output.

```
python /home/tim/e63/kafka_producer3.py localhost:9092 tim
```

```
tim@ip-172-31-24-35:~/e63$ python3 kafka_producer3.py localhost:9092 tim
Start batch #0
Finish batch #0
Sleep 1 second
Start batch #1
Finish batch #1

...

Sleep 1 second
Start batch #498
Finish batch #498
Sleep 1 second
Start batch #499
Finish batch #499
Sleep 1 second
Finished!
```

Create consumer

Next we're adapting the *direct_word_count.py* script into a new *kafka_consumer3.py* script. We're transforming the *buy* variable into a binary *true/false* variable in the *parse_data* function. The goal is count the number of different stocks that where bought in each batch. We're doing that by aggregating the stock *symbol* variable by it's *buy* variable.

```
# Libraries
from __future__ import print_function
from operator import add, sub
import sys

import findspark
findspark.init("/home/tim/spark")

from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *
from datetime import datetime

# Parse function
def parse_data(line):
    s = line.rstrip().split(",")
    try:
        if s[6] != "B" and s[6] != "S":
```

```
            raise Exception('Wrong format')
        return [
            {"time": datetime.strptime(s[0], "%Y-%m-%d %H:%M:%S"),
             "orderId": int(s[1]), "clientId": int(s[2]),
             "symbol": s[3], "amount": int(s[4]), "price": float(s[5]),
             "buy": s[6] == "B"}]
    except Exception as err:
        print("Wrong line format (%s): " % line)
        return []

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: kafka_consumer3.py <broker_list> <topic>", file=sys.stderr)
        exit(-1)

    # Get brokers and topic
    broker_list, topic = sys.argv[1:]

    # Open spark context
    conf = SparkConf().setAppName("PythonStreamingDirectKafkaWordCount")
    conf = conf.setMaster("local[2]")
    sc = SparkContext(appName="PythonStreamingDirectKafkaWordCount")
    ssc = StreamingContext(sc, 2)

    # Open kafka stream
    kvs = KafkaUtils.createDirectStream(ssc, [topic],
                                        {"metadata.broker.list": broker_list})
    filestream = kvs.transform(lambda rdd: rdd.values())

    # Parse file
    rdd_orders = filestream.flatMap(parse_data)

    # Sum buys (1) and sells (0)
    result = rdd_orders.map(lambda x: (x['symbol'], x['buy'])).reduceByKey(add)

    # Print result
    result.pprint()

    ssc.start()

ssc.awaitTermination()
```

We can run the script with the following command

```
python /home/tim/e63/kafka_consumer3.py localhost:9092 tim
```

```
tim@ip-172-31-24-35:~/e63$ python3 kafka_consumer3.py localhost:9092 tim
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
For SparkR, use setLogLevel(newLevel).
17/11/04 19:29:09 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
-------------------------------------------
Time: 2017-11-04 19:29:14
```

```
-------------------------------------------
('LEU', 13)
('C', 14)
('AMD', 11)
('PHG', 14)
('FCEL', 16)
('PYPL', 10)
('GE', 13)
('CHU', 8)
('WLL', 15)
('TOT', 9)
...

-------------------------------------------
Time: 2017-11-04 19:29:16
-------------------------------------------
('LEU', 10)
('C', 14)
('AMD', 20)
('TSU', 16)
('FCEL', 7)
('PYPL', 13)
('GE', 11)
('CHU', 7)
('WLL', 17)
('TOT', 6)
...
```

As can be seen above, the script is counting the number of stocks that were bought for each batch.

Problem 4 (25%)

Question: Install Cassandra server on your VM.

Install cassandra

We're installing cassandra on the ubuntu instance. That means we have to add the debian source to the *sources.list.d*.

```
echo "deb http://www.apache.org/dist/cassandra/debian 311x main" | sudo tee -a \
/etc/apt/sources.list.d/cassandra.sources.list
curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -
sudo apt-key adv --keyserver pool.sks-keyservers.net --recv-key A278B781FE4B2BDA
sudo apt-get update
sudo apt-get install cassandra
```

Install cqlsh and cassandra driver

We also need to install *cqlsh* and the *python driver*.

```
pip install cqlsh
pip install cassandra-driver
```

```
pip3 install cassandra-driver

conda install -c conda-forge cassandra-driver
```

Start cassandra

Next we can start cassandra with the following commands

```
systemctl enable cassandra
systemctl start cassandra
systemctl -l status cassandra
```

Start shell

The cqlsh version is not compatible with all cassandra versions. However, we can add the cqlversion number when starting cqlsh.

```
cqlsh --cqlversion=3.4.4
```

Question: Use Cassandra SQL Client, *cqlsh*, to create and populate table *person*. Let every *person* be described by his or her *first* and *last name*, and *city* where he or she lives. Let every person possess up to *three cell phones*. Populate your table with three individuals using *cqlsh* client. Demonstrate that you can select the content of your table *person* including individuals' cell phones.

Create Keyspace "hw9"

We're first creating the keyspace hw9.

```
CREATE KEYSPACE IF NOT EXISTS hw9
WITH replication = {'class':'SimpleStrategy','replication_factor':1};
```

Use keyspace

Next we're using the keyspace

```
USE hw9;
```

Create table person

Creating a table can be done with the following sql statement.

```
CREATE TABLE person (id int,
                     first_name text,
                     last_name text,
                     city text,
                     cell1 text,
                     cell2 text,
                     cell3 text,
                     PRIMARY KEY (id));
```

Verify

Next we're verifing if the table was created as programmed above.

```
DESCRIBE person;
```

```
cqlsh:hw9> DESCRIBE person;

CREATE TABLE hw9.person (
    id int PRIMARY KEY,
    cell1 text,
    cell2 text,
    cell3 text,
    city text,
    first_name text,
    last_name text
) WITH bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class':
    'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy',
    'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64',
    'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';
```

Insert data

Next we can insert data into our table.

```
INSERT INTO person (id, first_name, last_name, city, cell1, cell2, cell3)
VALUES (1, 'Tim','Hagmann','Basel','111-111-0000', '111-222-0000', '111-333-0000');

INSERT INTO person (id, first_name, last_name, city, cell2, cell3)
VALUES (2, 'Michael','Müller','Zürich','222-222-0000', '222-333-0000');

INSERT INTO person (id, first_name, last_name, city, cell1, cell3)
VALUES (3, 'Linda','Meier','Bern','333-111-0000', '333-333-0000');
```

Select data

Select *all data* from the table person:

```
SELECT * FROM person;
```

```
cqlsh:hw9> SELECT * FROM person;
```

```
 id | cell1        | cell2        | cell3        | city   | first_name | last_name
----+--------------+--------------+--------------+--------+------------+-----------
  1 | 111-111-0000 | 111-222-0000 | 111-333-0000 |  Basel |        Tim |   Hagmann
  2 |         null | 222-222-0000 | 222-333-0000 | Zürich |    Michael |    Müller
  3 | 333-111-0000 |         null | 333-333-0000 |   Bern |      Linda |     Meier
```

With the following commands can we select individual elements

```
SELECT first_name, last_name, city, cell1, cell2 FROM person;
```

```
cqlsh:hw9> SELECT first_name, last_name, city, cell1, cell2 FROM person;

 first_name | last_name | city   | cell1        | cell2
------------+-----------+--------+--------------+--------------
        Tim |   Hagmann |  Basel | 111-111-0000 | 111-222-0000
    Michael |    Müller | Zürich |         null | 222-222-0000
      Linda |     Meier |   Bern | 333-111-0000 |         null
```

Question: Write a simple client in a language of your choice that will populate *3 rows* in Casandra's table *person*, subsequently update one of those rows, for example change the city where a person lives, and finally retrieve that modify row from Cassandra and write its content to the console.

I wrote a client in the *cassandra_client.py* script. However, I splited the script up into the following code chunks so that it is easier to see what I did.

Setup

```
# Load libraries
from cassandra.cluster import Cluster
from cassandra.query import SimpleStatement

# Setup connection
cluster = Cluster()
session = cluster.connect('hw9')
```

Functions

```
# Create a person record
def insert_person(id, first_name, last_name, city, cell1, cell2, cell3):
  session.execute("""INSERT INTO person (id, first_name, last_name, city,
                                         cell1, cell2, cell3)
                     VALUES (%s,%s,%s,%s,%s,%s,%s)""",
                     (id, first_name, last_name, city, cell1, cell2, cell3))

# All
def select_all():
  statement = SimpleStatement("SELECT * FROM person")
  result = session.execute(statement)
  return result

# By id
def select_one(id):
  result = session.execute("""SELECT id, first_name, city FROM person
```

```
                                WHERE id = %s""", [id])[0]
  return result

# Update city
def update_city(id, city):
  session.execute("UPDATE person SET city = %s where id = %s", (city,id))
```

Add three rows

```
# Add three new records
insert_person(4, 'Hans-Peter', 'Hugentobler', 'Grenchen', '444-111-0000',
              '444-222-0000', '444-333-0000')
insert_person(5, 'Christof', 'Hausmann', 'Bottmingen', '555-111-0000',
              '555-222-0000', '555-333-0000')
insert_person(6, 'Gabriela', 'Tschaggelar', 'Bottmingen',
              '666-111-0000','666-222-0000','666-222-0000')

# Select persons
df_persons = select_all()

# Print output
for i in df_persons:
  print(i)
```

```
Row(id=5, cell1='555-111-0000', cell2='555-222-0000', cell3='555-333-0000',
city='Bottmingen', first_name='Christof', last_name='Hausmann')
Row(id=1, cell1='111-111-0000', cell2='111-222-0000', cell3='111-333-0000',
city='Bettlach', first_name='Tim', last_name='Hagmann')
Row(id=2, cell1=None, cell2='222-222-0000', cell3='222-333-0000', city='Zürich',
first_name='Michael', last_name='Müller')
Row(id=4, cell1='444-111-0000', cell2='444-222-0000', cell3='444-333-0000',
city='Grenchen', first_name='Hans-Peter', last_name='Hugentobler')
Row(id=6, cell1='666-111-0000', cell2='666-222-0000', cell3='666-222-0000',
city='Bottmingen', first_name='Gabriela', last_name='Tschaggelar')
Row(id=3, cell1='333-111-0000', cell2=None, cell3='333-333-0000', city='Bern',
first_name='Linda', last_name='Meier')
```

Update row

```
# Updade id 1
update_city(1, 'Bettlach')

# Select and print
df_person = select_one(1)
print(df_person.first_name + " has a new city with the name: " + df_person.city)
```

```
Tim has a new city with the name: Bettlach
```

As mentioned before, all of the above code can also be found in the *cassandra_client.py* script. It can be executed with the following command:

```
python3 cassandra_client.py
```