Exercise HandsOn – Stream Processing

Advanced DBMS

MIT 2019 - Data Science
June 2020

Dennis Timis
Bojan Makivic
Johannes Wieser

1. Team

1.1. Members

Name	Student #
Bojan Makivic	01002146
Dennis Timis	51818530
Johannes Wieser	00608831

1.2. Work distribution

Task	Team Member
Task 1	Dennis Timis
Task 2	Johannes Wieser
Task 3	Bojan Makivic

2. Description of Use Case

Our Idea to fulfil the requirement of using several data sources to combine and stream them with a stream processor was to use Tweets and financial Stock data and combine them to find out if there are some correlations. So the specific example looks like this:

- Continuously get stock data on Microsoft Stock
- Continuously get stock data on Tesla Stock
- Continuously get all tweets containing the word “Gates”
- Continuously get all tweets containing the word “Musk”

As we wanted to find out a relation between twitter and stock market price we took the following metrics:

- Stock market price in USD
- Amount of tweets per time
- Calculate average sentiment of the tweets per time (from -1 to 1)

Later (see visualization) we combined and compared the output while it’s dynamically updating.

3. Web-APIs for data retrieval

3.1. Twitter

In order to use the Twitter API, first a developer account needs to set up. This procedure will be described in chapter 5.1. However, once everything is properly set up, you can easily engage with the Twitter API and search tweets, post content and so on.

Twitter offers three different API modalities:

- Standard Search API
- Premium Search API
- Enterprise Search API

For this project the Standard Search API was taken because it is useable for free (check the restrictions, though!) and sufficient for the uses cases.

For a more precise documentation refer to the documentation:
https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets

In order to avoid doing to the HTTP Post and Receive action by ourselves, we used an existing twitter library for Python called twitter. See chapter 5.3.2. for more information.

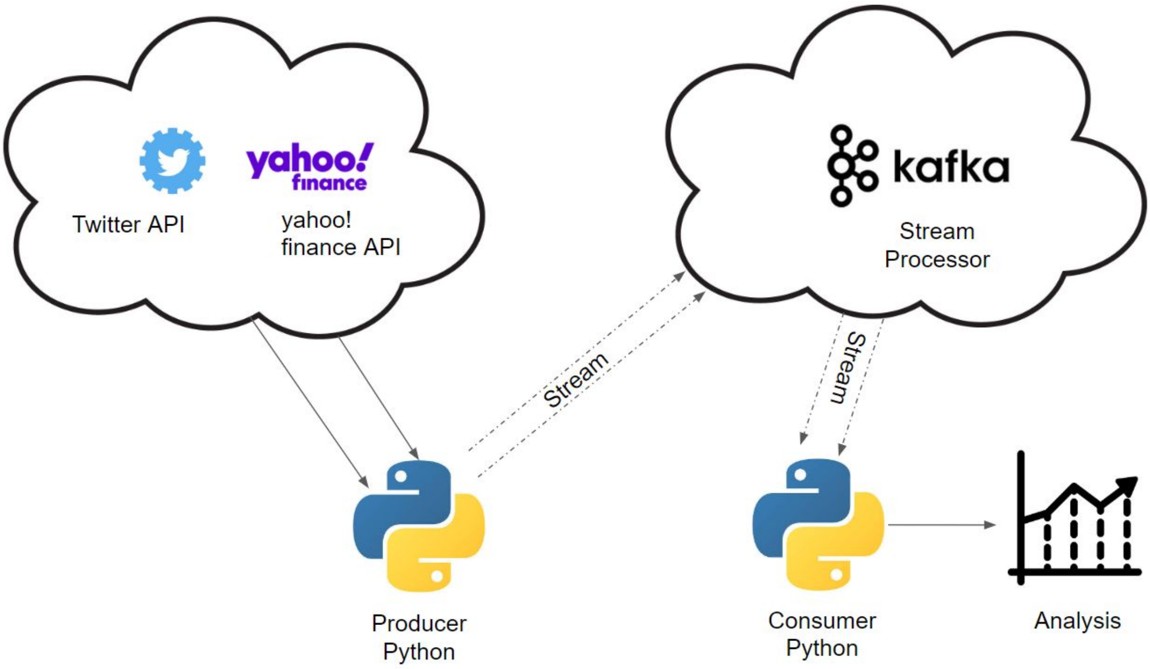
3.2. Yahoo! Finance

For the retrieval of the stock prices Yahoo! Finance was used because it is not only simple to use, but it also does not need any user account, authentication or whatsoever. Basically all you need to do is perform your request and you will get your result.

For this the library yfinance was used. See chapter 5.3.3. for more information.

4. Overview of the solution

4.1. High Level Architecture Overview



The overall approach high level overview consists of all systems involved in the project. The Data is collected from the twitter and yahoo finance APIs and distributed via python in the stream processing system, which is hosted in the cloud. The data is then streamed continuously and later collected and polled with python and analyzed.

4.2. Used Technologies

4.2.1. Python

Python was used to get the data from the Twitter and Yahoo finance APIs and produce Messages to the kafka stream processor. It was also used to poll the data out of the kafka in the form of a stream and create upon this analysis, in kafka this is called the “consumer”.

4.2.2. Kafka

Apache Kafka is an open-source stream-processing software platform developed by the Apache Software Foundation, written in Scala and Java. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. Its storage layer is essentially a “massively scalable pub/sub message queue architected as a distributed transaction log,” making it highly valuable for enterprise infrastructures to process streaming data. Additionally, Kafka connects to external systems (for data import/export) via Kafka Connect and provides Kafka Streams, a Java stream processing library. (Wikipedia)

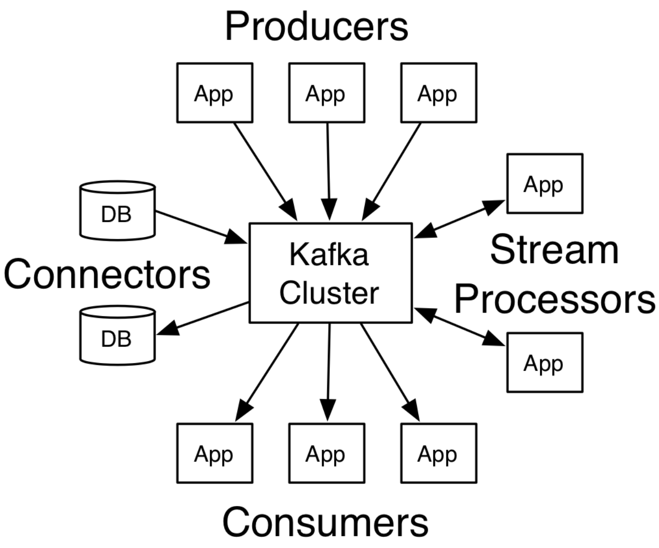


Image Source: Kafka

Topics

Every message that is fed into the system must be part of some topic. The topic is nothing but a stream of records. The messages are stored in key-value format. Each message is assigned a sequence, called Offset. The output of one message could be an input of the other for further processing.

Producers

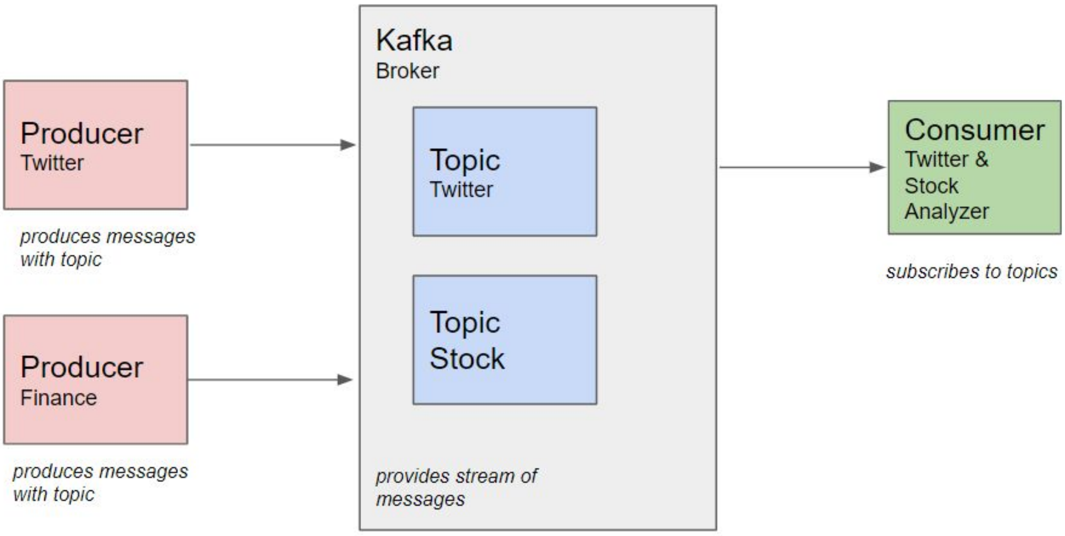
Producers are the apps responsible to publish data into the Kafka system. They publish data on the topic of their choice.

Consumers

The messages published into topics are then utilized by Consumers apps. A consumer gets subscribed to the topic of its choice and consumes data.

Broker

Every instance of Kafka that is responsible for message exchange is called a Broker. Kafka can be used as a stand-alone machine or a part of a cluster.



In our case, the Producers create Messages for Stock Data, as well as Tweets from Twitter andf eed them into Kafka which is hosted in the cloud using Confluent Kafka. The Consumer can choose which Topics to subscribe to, in our case both Topics “Twitter” and “Stock” are used in the consumer application. In the Consumer the data is analyzed and visualized dynamically as the data changes continuously.

5. Installation & Setup

5.1. Twitter Developer Accounts

The procedure how to set up a developer account is pretty straight forward and explained here: https://developer.twitter.com/en/docs/basics/developer-portal/overview

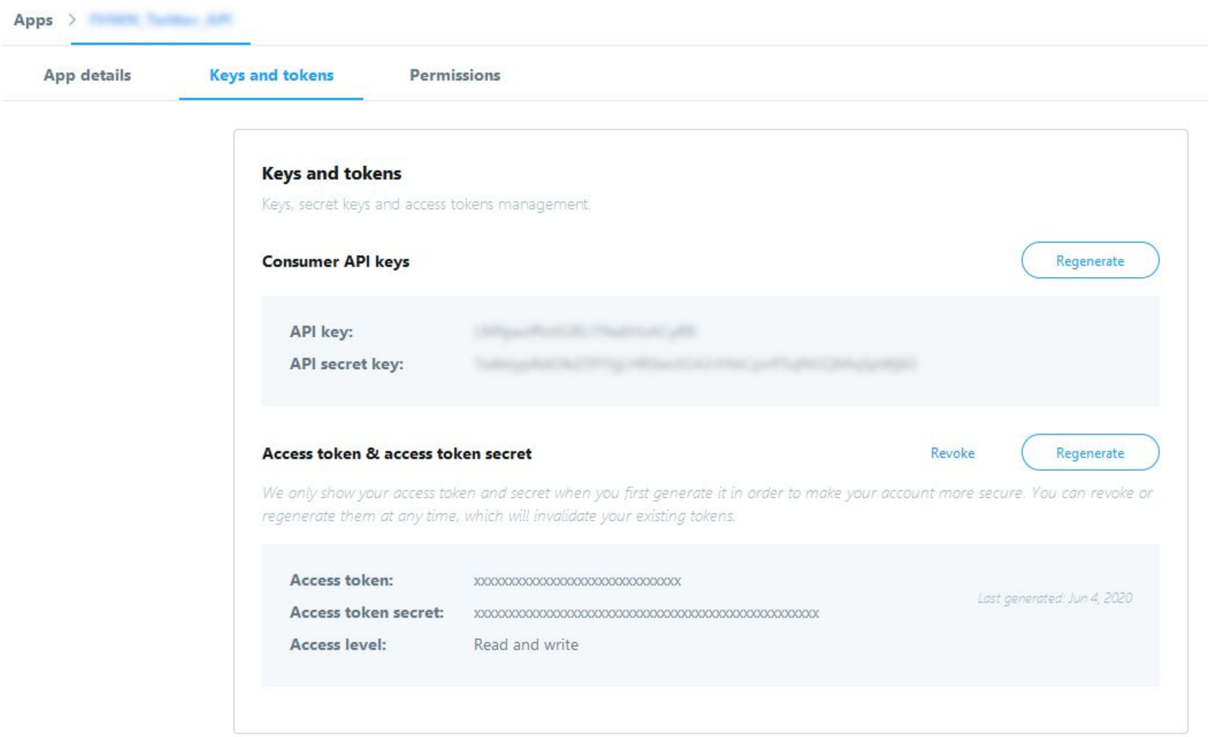
Once you have the developer account, you need to know your login credentials in order to interact with the API. For this follow these steps:

1) Login and access https://developer.twitter.com/en/apps .
2) Choose your app and click on "Details"..
3) Navigate to the "Keys and token" tab.

Here we are presented with the "Consumer API keys" which are already existing. What needs to be done is the generation of the "Access token & access token secret".

4) Click on the "Generate" button in the section "Access token & access token secret".

Once you have generated them, store them somewhere secretly (if needed) because they will be hidden afterwards! With these 4 tokens you can now use the twitter API.



Keys and tokens

Keys, secret keys and access tokens management.

Consumer API keys

API key: [redacted]

API secret key: [redacted]

Access token & access token secret

We only show your access token and secret when you first generate it in order to make your account more secure. You can revoke or regenerate them at any time, which will invalidate your existing tokens.

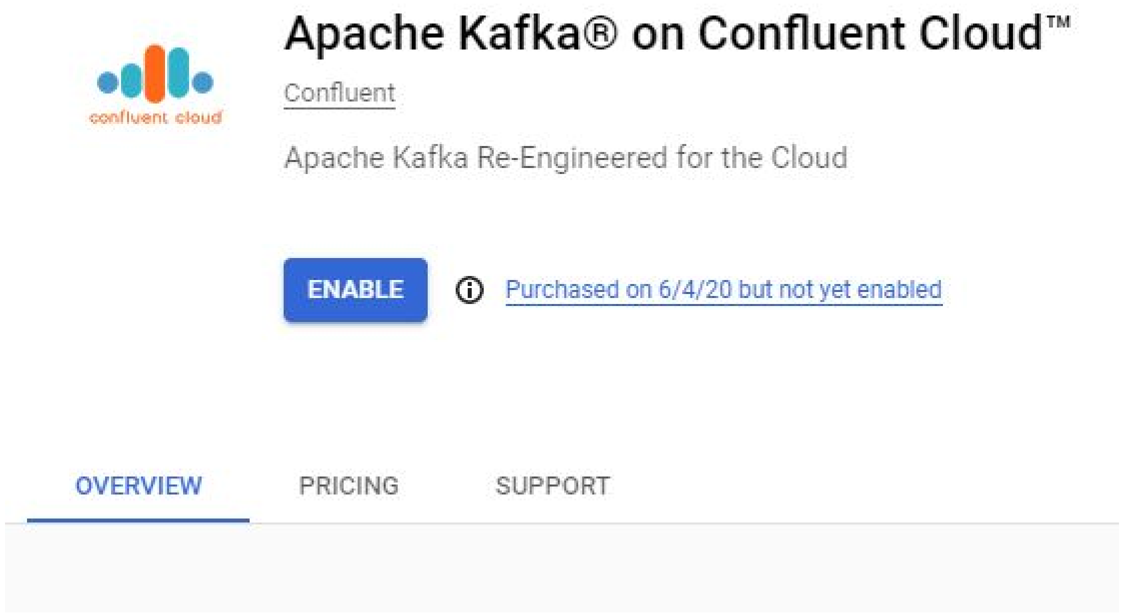
Access token: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Access token secret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Access level: Read and write

Last generated: Jun 4, 2020

5.2. Kafka Setup


Apache Kafka® on Confluent Cloud™

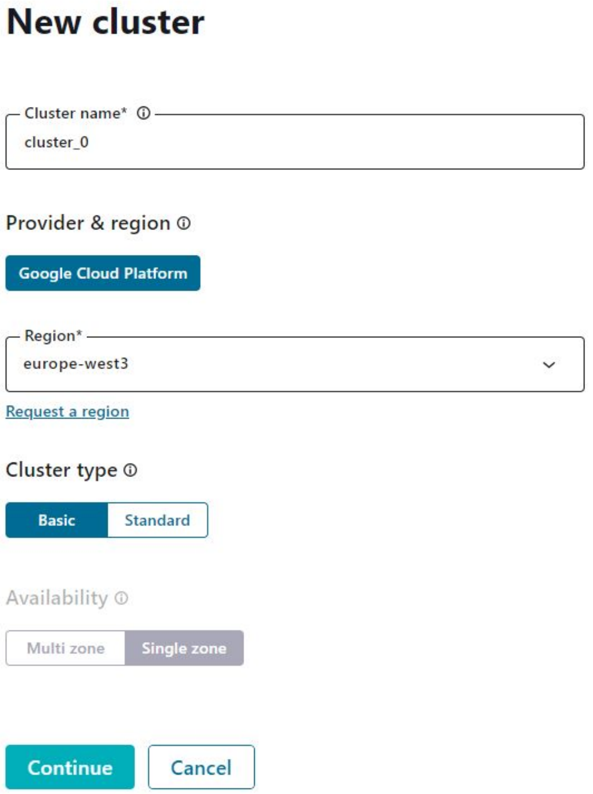
Confluent

Apache Kafka Re-Engineered for the Cloud

Purchased on 6/4/20 but not yet enabled

On the Google Cloud Platform (GCP), Apache Kafka was enabled and can be billed with your Google Cloud account. We registered a collectively used gmail account to use GCP services.

After Registration a new Kafka Cluster is created:


New cluster

Cluster name*: cluster_0

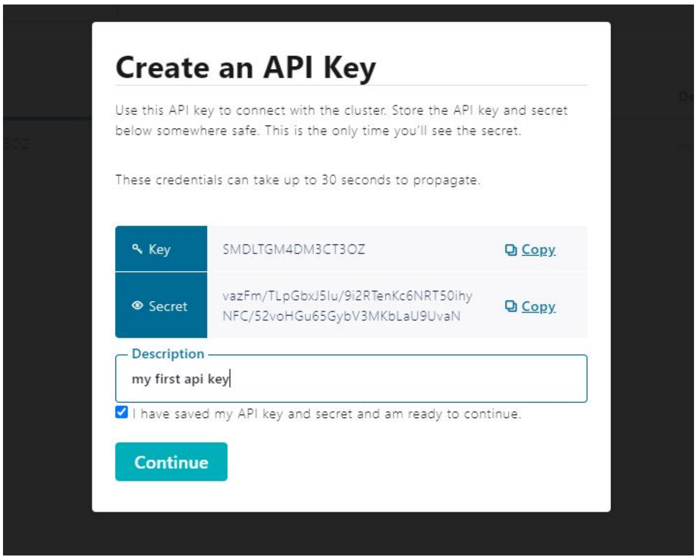
Provider & region: Google Cloud Platform

Region*: europe-west3

Cluster type: Basic

Availability: Single zone

And the cluster is later launched. After this the API Key and secret were generated. These are required for external access to the Kafa Cloud Cluster.



Create an API Key

Use this API key to connect with the cluster. Store the API key and secret below somewhere safe. This is the only time you'll see the secret.

These credentials can take up to 30 seconds to propagate.

Key	SMDLTGM4DM3CT3OZ	Copy
Secret	vazFm/TLpGbxJ5Iu/9i2RTenKc6NRT50ihy NFC/52voHGu65GybV3MKbLaU9UvaN	Copy

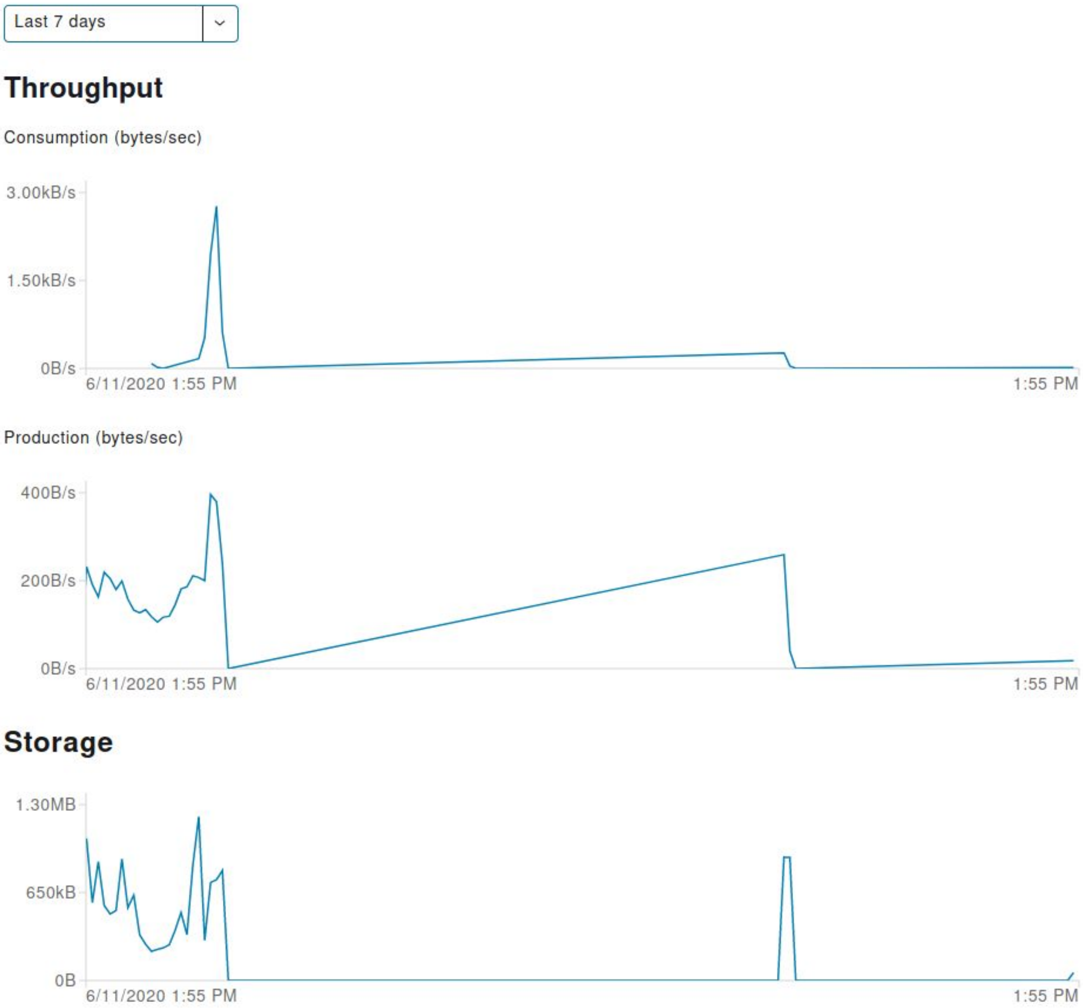
Description: my first api key

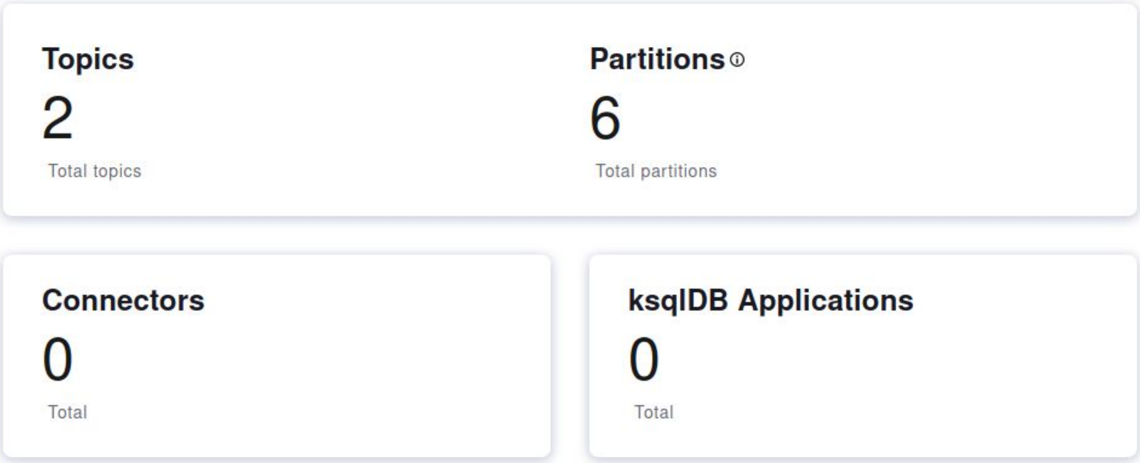
☑ I have saved my API key and secret and am ready to continue.

You can log in and monitor the system on https://confluent.cloud/login

Kafka Dashboard

The overall “check” for your Kafka cluster can be done on the “Overview” after the login. It gives instant information if data is streaming, which topics are in use, and basically lets you see if everything is working or if there are some Problems.





Topics
2
Total topics

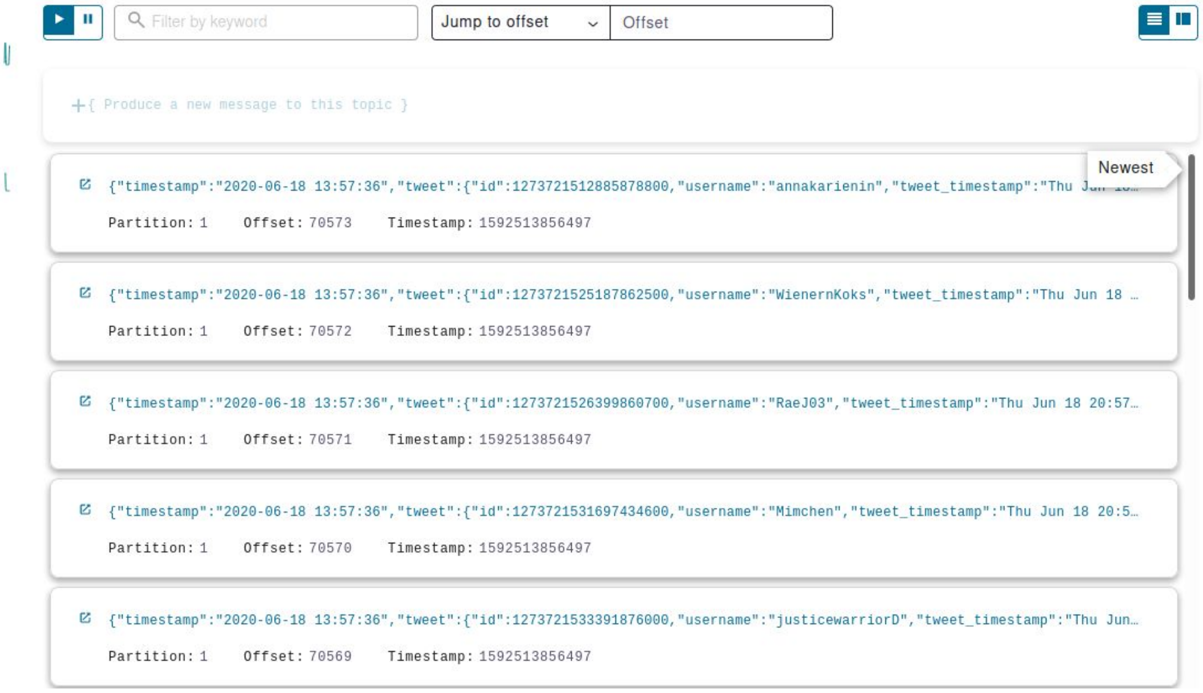
Partitions
6
Total partitions

Connectors
0
Total

ksqlDB Applications
0
Total

Message View per Topic

In real time the messages per topic can be viewed, and also messages can be produced for debugging etc. It’s accessed via Topics > “topic” > Message

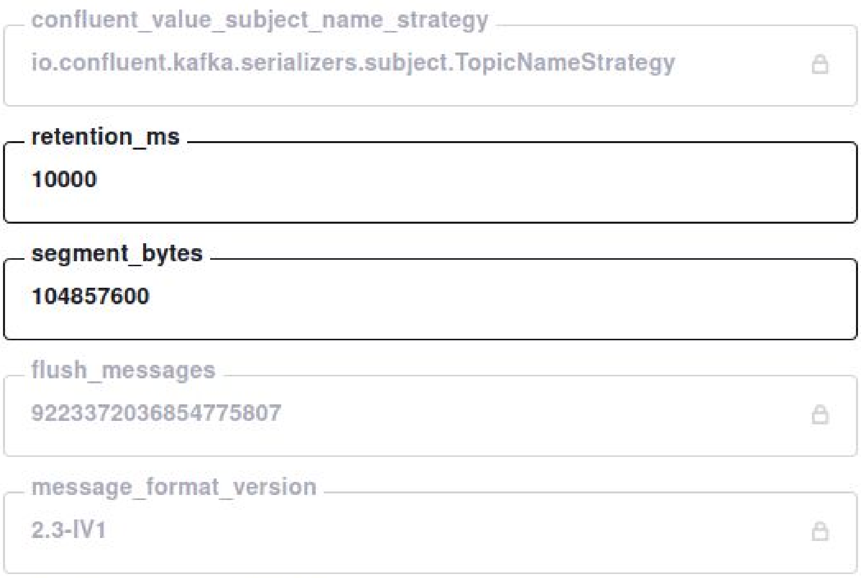


```
{"timestamp":"2020-06-18 13:57:36","tweet":{"id":1273721512885878800,"username":"annakarienin","tweet_timestamp":"Thu Jun 18…
Partition: 1     Offset: 70573     Timestamp: 1592513856497

{"timestamp":"2020-06-18 13:57:36","tweet":{"id":1273721525187862500,"username":"WienernKoks","tweet_timestamp":"Thu Jun 18 …
Partition: 1     Offset: 70572     Timestamp: 1592513856497

{"timestamp":"2020-06-18 13:57:36","tweet":{"id":1273721526399860700,"username":"RaeJ03","tweet_timestamp":"Thu Jun 18 20:57…
Partition: 1     Offset: 70571     Timestamp: 1592513856497

{"timestamp":"2020-06-18 13:57:36","tweet":{"id":1273721531697434600,"username":"Mimchen","tweet_timestamp":"Thu Jun 18 20:5…
Partition: 1     Offset: 70570     Timestamp: 1592513856497

{"timestamp":"2020-06-18 13:57:36","tweet":{"id":1273721533391876000,"username":"justicewarriorD","tweet_timestamp":"Thu Jun…
Partition: 1     Offset: 70569     Timestamp: 1592513856497
```

A very useful feature is also that a message can be produced here.

Streaming Window

Per Topic the streaming window size can be configured. Either in maximum amount of bytes or in time. We choose to leave all messages for 10 seconds in the window. The configuration can be found in Confluent Kafka > Topics > Configuration > Switch to expert mode > retention_ms was set to 10000. For a retention maximum byte size, “retention_bytes” can be used. We set it to “-1” which is “maximum”



confluent_value_subject_name_strategy: io.confluent.kafka.serializers.subject.TopicNameStrategy

retention_ms: 10000

segment_bytes: 104857600

flush_messages: 9223372036854775807

message_format_version: 2.3-IV1

5.3. Messages

5.3.1. Overall

The messages are in json format and handled via Kafka. We have agreed on, but not enforced a schema on each topic's messages.

5.3.2. Stock Messages

The stock messages contain these values:

- timestamp: the timestamp created by the producer when the data is received
- value: stock price in USD
- stock_timestamp: timestamp from the yahoo! finance api

Example Message

```
{
  "timestamp": "2020-06-19 07:49:31",
  "value": 198.12,
  "stock_timestamp": "2020-06-19 10:49:20-04:00"
}
```

5.3.3. Twitter Messages

The twitter messages contain these values:

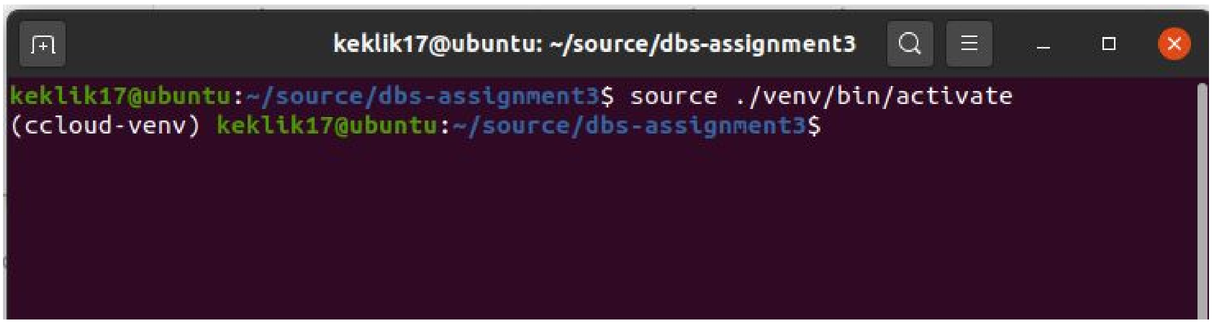
- timestamp: the timestamp created by the producer when the data is received
- tweet
    - id: tweet unique id
    - username: twitter username from the tweet author
    - tweet_timestamp: twitters timestamp
    - text: the content of the tweet

Example Message

```
{
  "timestamp": "2020-06-19 07:51:00",
  "tweet": {
    "id": 1273991666022731800,
    "username": "BornintheUSA100",
    "tweet_timestamp": "Fri Jun 19 14:50:57 +0000 2020",
    "text": "RT @MzMugzzi: This is the real Gates! https://t.co/gmsZVgsp36"
  }
}
```

5.4. Python Environment

A virtual Environment was used to install all a version of python and all required libraries to run this project on your machine. It is not included in the project because it’s to big for edunet upload, but can be downloaded here. It can be activated with the activation script. Required for the use is only “python”, “pip” and “virtualenv”. All other packages as well as the specific are included in the venv folder, and used from the terminal in which it’s activated. See the name of the environment in brackets before the command line:



```
keklik17@ubuntu:~/source/dbs-assignment3$ source ./venv/bin/activate
(ccloud-venv) keklik17@ubuntu:~/source/dbs-assignment3$
```

5.5. How to run the project

Configure the in the main.py which producers you want to choose.

```
factory.register_twitter('Musk')
factory.register_twitter('Gates')
factory.register_yahoo('TSLA')
factory.register_yahoo('MSFT')
```

- Install python, pip and virtualenv
- Download the virtual environment with all pre-installed packages here
- Open a terminal
- Activate the virtual environment by using “source path/venv/bin/activate”
- run “python path/main.py”
- you see the producers starting to fire messages to kafka
- Open a new terminal
- Activate the virtual environment by using “source path/venv/bin/activate”
- run “python path/twitter-analysis-consumer.py”

5.6. Python libraries

5.6.1. confluent_kafka

This python library is a client which abstracts the communication with the Kafka server. The documentation on how to work with the client is found here: https://docs.confluent.io/current/clients/confluent-kafka-python/

In order to install this library on your machine open the command-line and run:

```
pip install confluent-kafka
```

5.6.2. twitter

This python library is a wrapper around the Twitter API. The documentation on how to work with this wrapper is found here: https://python-twitter.readthedocs.io/en/latest/

In order to install this library on your machine open the command-line and run:

```
pip install python-twitter
```

5.6.3. yfinance

This python library is a wrapper around Yahoo's stock market API. The documentation on how to work with this wrapper is found here: https://pypi.org/project/yfinance/

In order to install this library on your machine open the command-line and run:

```
pip install yfinance
```

5.6.4. json

This python library is responsible for the conversion of Python data to a json string (that is the format the Kafka server reliably works with) and vice versa. The documentation on how to work with the client is found here: https://docs.python.org/3/library/json.html
This library does not need to be separately installed because it is part of the standard library.

5.6.5. matplotlib

This python library is a comprehensive library for creating static, animated, and interactive visualizations. The documentation on how to work with this library is found here: https://matplotlib.org/contents.html

Regarding the installation it depends on your exact environment. Take look at this documentation: https://matplotlib.org/users/installing.html#building-on-windows

5.6.6. nltk (Natural Language Toolkit)

This python library is a leading platform for building Python programs to work with human language data. One of its relevant functionalities for this project is the ability to perform sentiment analysis. The documentation on how to work with this library is found here:
https://www.nltk.org/

In order to install this library on your machine open the command-line and run:

```
pip install nltk
```

6. Task 1 - Producer

This task represents the creation of a producer, using the `confluent_kafka` client-library, to periodically send content to the Kafka server. The producer is split into multiple projects files which will be not shortly explained.

6.1. twitter_api.py

This module holds the functionality to interact with the twitter API.

```
class TwitterAPI(Requestor):

  # constructor
  # @ authentication: Represents the Twitter credentials.
  # @ search_word: Represents the name after which to be searched.
  def __init__(authentication: TwitterAuthentication,
               search_word: str) -> TwitterAPI

  # Implementation of the abstract method to retrieve new data.
  # - returns: Returns a list of dictionaries containing the search result.
  def request_new() -> [{}]
```

```
class TwitterAuthentication():

  # constructor
  # @ consumer_key: Represents Twitter’s “Consumer Key”.
  # @ consumer_secret: Represents Twitter’s “Consumer Secret”.
  # @ access_token_key: Represents Twitter’s “Access Token Key”.
  # @ access_token_secret: Represents Twitter’s “Access Token Secret”.
  def __init__(consumer_key: str, consumer_secret: str,  access_token_key: str,
               access_token_secret: str) -> TwitterAuthentication
```

6.2. yahoo_api.py

This module holds the functionality to interact with the yahoo API.

```
class YahooFinanceAPI(Requestor):

  # constructor
  # @ stock: Represents the name of the stock to be analyzed.
  def __init__(stock: str) -> YahooFinanceAPI

  # Implementation of the abstract method to retrieve new data.
  # - returns: Returns a list of dictionaries containing the search result.
  def request_new() -> [{}]
```

6.3. requestor.py

This module holds the abstract class which needs to be implemented by the twitter API and the yahoo API.

```
class Requestor():

  # Represents the abstract method which needs to be implemented by all subclasses.
  # - returns: Returns a list of dictionaries containing the search result.
  @abstractmethod
  def request_new() -> [{}]
```

6.4. producer.py

This module wraps a requestor and periodically calls its interface method to retrieve new data. The class is run in a separate thread, so that the main thread is not blocked.

```
class Producer():

  # constructor
  # @ host: Represents the host name of the Kafka server.
  # @ port: Represents the port where the Kafka server is run.
  # @ username: Represents the username for authentication.
  # @ password: Represents the password for authentication.
  def __init__(host: str, port: int, username: str, password: str) -> Producer

  # Starts the producer to periodically retrieve new data.
  def start(requestor: Requestor,
            topic_name: str,
            topic_key: str,
            timeout: int) -> [{}]

  # Stops the producer.
  def stop() -> None
```

6.5. producer_factory.py

This module represents a factory, holding and enabling the creation of multiple producers. First of all the producers need to be created and afterwards the factory can start the registered producers. Once the factory has initiated the start, no further producers can be registered to be run. Stopping the factory will reset it and remove all registered producers.

```
class ProducerFactory():

  # constructor
  # @ host: Represents the host name of the Kafka server.
  # @ port: Represents the port where the Kafka server is run.
  # @ username: Represents the username for authentication.
  # @ password: Represents the password for authentication.
  def __init__(host: str, port: int, username: str, password: str)
    -> ProducerFactory

  # Starts all the registered producers.
  def start() -> None

  # Stops the the registered producers and empties the registered list.
  def stop() -> None

  # Registers a producer to retrieve Twitter content.
  # @ search_word: Represents the name after which to be searched.
  def register_twitter(search_word: str, ) -> None

  # Registers a producer to retrieve Twitter content.
  # @ stock: Represents the name of the stock to be analyzed.
  def register_yahoo(stock: str, ) -> None
```

6.6. main.py

This module represents the entry point of the producer’s application and contains the credentials for the authentication on the Kafka server.

```
import producer
import producer_factory
import requestor
import twitter_api
import yahoo_api

# Represents the credentials for the Kafka server.
HOST: str = ***.gcp.confluent.cloud'
PORT: int = 9092
USER_TOKEN: str = '***your password token here***'
PASSWORD_TOKEN: str = '***your password token here***'

# Created the factory.
factory = producer_factory.ProducerFactory(HOST, PORT, USER_TOKEN, PASSWORD_TOKEN)

# Registers producers for twitter and yahoo! finance.
factory.register_twitter('Tesla')
factory.register_twitter('Gates')
factory.register_yahoo('TSLA')
factory.register_yahoo('MSFT')

# Starts the factory.
factory.start()
```

7. Task 2 - Consumer

The purpose of the consumer is to get all messages which have the required topics - which in this case is “stock” and “twitter”. And further prepare and hold the data to later visualize it.

7.1. Connection

The connection is established using the confluent_kafka package and the key and secret as well as configuration data.

```
from confluent_kafka import Consumer

# create consumer and establish connection
c = Consumer({
    'bootstrap.servers': '***.confluent.cloud:9092',
    'sasl.mechanism': 'PLAIN',
    'security.protocol': 'SASL_SSL',
    'sasl.username': '***username***',
    'sasl.password': '***password***',
    'group.id': str(uuid.uuid1()),  # this will create a new consumer group on each
invocation.
    'auto.offset.reset': 'earliest'
})
```

7.2. Subscription to Topics & Polling

The package also makes it very simple to subscribe to one or several topics it’s.

```
# subscribe to both topics
c.subscribe(['twitter', 'stock'])
```

After the subscription it’s easily possible to poll messages. We do this in a while True loop to continuously poll for data.

```
try:
    while True:
        msg = c.poll(0.1)  # Wait for 0.1 secs for message
        if msg is None:
            # No message available within timeout.
            # Initial message consumption may take up to `session.timeout.ms` for
            #   the group to rebalance and start consuming.
            continue
        if msg.error():
            # Errors are typically temporary, print error and continue.
            print("Consumer error: {}".format(msg.error()))
            continue
```

It is also continuously checked for errors or null messages.

7.3. Message Selection

```
if msg.topic() == 'twitter':
    if str(msg.key(), 'utf-8').lower() == 'tesla':
        myjson = json.loads(str(msg.value(), 'utf-8'))
        list_tweets_tsla.append(myjson['tweet']['text'])
        count_tweets_tsla +=1

    if str(msg.key(), 'utf-8').lower() == 'gates':
        myjson = json.loads(str(msg.value(), 'utf-8'))
        list_tweets_ms.append(myjson['tweet']['text'])
        count_tweets_ms +=1
```

Basically the messages are selected by Key and Topic and separated and temporarily saved to lists, which are later used for the visualization.

7.4. Sentiment Analysis

On all tweets a sentiment analysis is performed using nltk. To get the pretrained model for that on the first time a lexicon is downloaded.

```
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

#sentiment analyzer
sid = SentimentIntensityAnalyzer()
```

After that the sentiment is easily calculated by using sid.polarity_socres(“some text”)[‘property’]. There are various properties. for our use the compound score gives us the most easily usable result which can vary from -1 (very bad sentiment) to 1 (very good sentiment). Every tweet text is analyzed, and the sentiment is averaged across all tweets.

```
if str(msg.key(), 'utf-8').lower() == stock_ms.lower():
    list_tweet_counts_ms.append(count_tweets_ms)
    count_tweets_ms = 0
    myjson = json.loads(str(msg.value(), 'utf-8'))
    list_stock_prices_ms.append(myjson['value'])
    if len(list_tweets_ms) > 0:
        temp = []
        for tweet in list_tweets_ms:
            temp.append(sid.polarity_scores(tweet)['compound'])
        list_sentiments_ms.append(sum(temp) / len(temp))
        list_tweets_ms = []
    else:
        list_sentiments_ms.append(0)
```

Also the sentiments are continuously hold in memory in lists to later be used in visualization to analyze if the tweets for the words are “good” or “bad”.

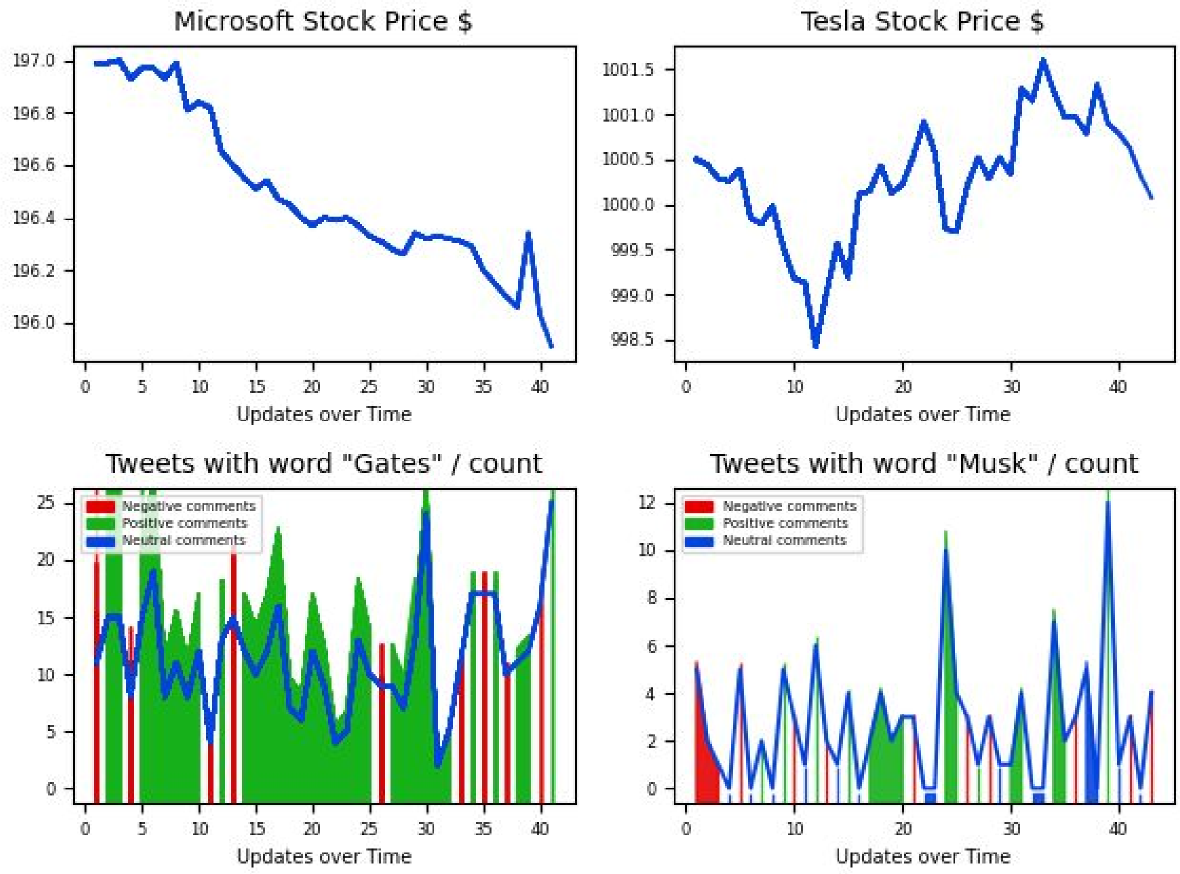
7.5. Tweet counting & Stock Price

Similarly but less complicated then the sentiment, the stock price and count of tweets are aggregated and saved in memory in lists. these lists are later used for visualization.

```
if msg.topic() == 'twitter':

if str(msg.key(), 'utf-8').lower() == twitter searchword_tsla.lower():
              myjson = json.loads(str(msg.value(), 'utf-8'))
              list_tweets_tsla.append(myjson['tweet']['text'])
              count_tweets_tsla +=1
```

8. Task 3 - Visualization

We have used the matplotlib package for further analysis and visualization of streaming data. The aim was to visualize four graphs; two graphs with different stock market prices for two different acteurs and two graphs considering the count of keywords on twitter that we have previously defined. In addition the sentiment values were integrated in the two lower graphs filled the three categories by three different colors (figure below).



The biggest challenge was the implementation of colored sentiment values in the curved area due to non-user friendly possibilities that matplotlib offers for such visualization. The one possibility was to scale the ‘y axis’ values from 0-1 and to add 0.03 more points in order to fill fully the area under the curve as in the code snippet below.

```
axs[1,0].fill_between(range(1,len(list_sentiments_ms)+1),[i/max(np_tweet_counts_ms)+0.03
for i in np_tweet_counts_ms],where=np_sent_ms > threshold, color='xkcd:green',
alpha=0.9, transform=axs[1, 0].get_xaxis_transform())

axs[1,0].fill_between(range(1,len(list_sentiments_ms)+1),
[i/max(np_tweet_counts_ms)+0.03 for i in np_tweet_counts_ms], where=np_sent_ms <
threshold, color='xkcd:red', alpha=0.9, transform=axs[1, 0].get_xaxis_transform())
```

```
axs[1,0].fill_between(range(1,len(list_sentiments_ms)+1),[i/max(np_tweet_counts_ms)+0.03
for i in np_tweet_counts_ms], where=np_sent_ms == threshold, color='xkcd:blue',
alpha=0.9, transform=axs[1, 0].get_xaxis_transform())
```