**One-way message subscription App using Apache Kafka, Python and Flask**

**Siddhartha Pachhai**

**The goal of the project**

The goal of this short project to create a simple single producer and consumer application that is able to visualize the data being streamed. The streamed data will be text-comments made by the producer which will be collected and converted into sentiment scores using Vader sentiment library in python. Also, the goal is to learn more about streaming.

**Libraries and software needed for this application**

*Python 3.6*

*Libraries {*

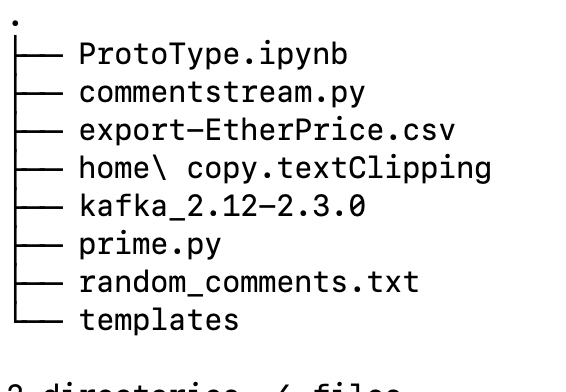
*Os, Flask, pandas, numpy, struct(optional), PyKafka, kafka-python, itertools, vaderSentiment*

*}*

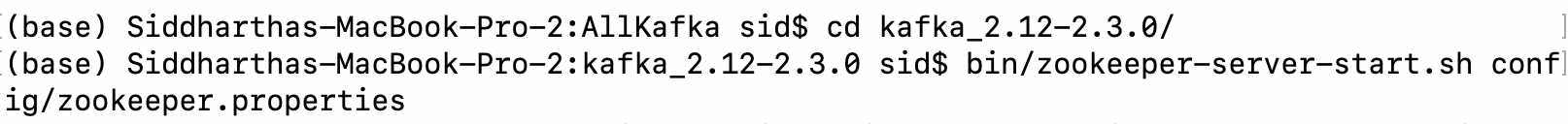
*Apache Kafka 2.3*

*D3.js*

**Starting the Kafka Server**



Within the folder I have the Kafka folder which contains all the Kafka files, the prime.py file, which is written in flask and some html templates, the other files aren’t as important and will be discussed later.



Within the folder I launch Zookeeper, using the following statement: ***bin/zookeeper-server-start.sh config/zookeeper.properties***

A close up of a mans face

Description automatically generated

After starting zookeeper, I initiate the kafka server: ***bin/kafka-server-start.sh config/server.properties***

**How Kafka works**

Kafka has two main users which are producers and consumers. The producers and consumers are able to subscribe to a particular topic, for a producer subscribing to a topic will allow them to send or post messages to the topic, and similarly consumers will be allowed to receive message from topics they are subscribed to. **In my example I have used a simple setup with only a single topic.** Topics allow records of messages to be broadcasted through the Kafka cluster, important information such as host, replication factor, partition and topics are needed for by the topics. **We can also configure how long records remain in the memory; this setting has not been changed here.**

**Writing the flask app**

Given that your zookeeper server is operating, we can now create a flask app which will allow a html based front end application to interact with our Kafka app. The first thing that is required are all the libraries described below.

from \_\_future\_\_ import division

import os

from flask import Flask

from flask import render\_template

from flask import request

from flask import redirect

from pykafka import KafkaClient

import pandas as pd

from pykafka import KafkaClient, SslConfig

from pykafka import topic

from pykafka.common import OffsetType

from kafka.admin import KafkaAdminClient, NewTopic

import struct

import time

import json

import numpy as np

import math

from itertools import islice

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

After loading the library, we will define our client. The docs for PyKafka calls this “A high-level pythonic client for Kafka”[[1]](#footnote-1). This basically allows the python library to run functions and communicate with the Kafka cluster.

client = KafkaClient(hosts="127.0.0.1:9092")

Our next step is to define the function that loads our homepage. **First, we will subscribe to the topic ‘sentiment’, creating the topic will be discussed later.** A simple consumer in PyKafka will collect data from the subscribed topic, we give it additional parameters, so the data being read is the latest. Next we limit the number of records that we want to read, the later parts of the code are used to extract pairs of values and record index and append them to “dat” which holds the values and offset which holds the index numbers. **The messages being consumed by the consumers are byte objects which need to be decoded accordingly**. In the end the flask app is returns the html template for the home page. From the function, 3 data structures are passed into the html, which are “sendata” the 15 latest sentiment scores, offset the index for these scores and hist which will be used to make a histogram and holds 25 of the latest sentiment values.

@app.route('/')

def home():

topic = client.topics['sentiment']

consumer = topic.get\_simple\_consumer(auto\_offset\_reset=OffsetType.LATEST,reset\_offset\_on\_start=True)

LAST\_N\_MESSAGES = 25

MAX\_PARTITION\_REWIND = int(math.ceil(LAST\_N\_MESSAGES / len(consumer.\_partitions)))

offsets = [(p, op.last\_offset\_consumed - MAX\_PARTITION\_REWIND)**for** p, op **in** consumer.\_partitions.items()]

offsets = [(p, (o **if** o > -1 **else** -2)) **for** p, o **in** offsets]

consumer.reset\_offsets(offsets)

dat = []

offset = []

**for** message **in** islice(consumer, LAST\_N\_MESSAGES):

incoming\_data = message.value.decode()

value = sentscore(incoming\_data)['compound']

dat.append(value)

offset.append(message.offset)

**return** render\_template("home.html", sentdata = dat[10:], ofs = offset[10:], hist = dat)

For the same function the html page will receive the values of being generated and **d3.js is used to convert everything into analytics.** Due to my limited experience with D3 and JS, I was unable to create a smooth tool that is able to show the data like medical devices do in real time, but to make it close enough I am using the **code below in the html page to reload the file every 5 seconds updating the data.**

<meta http-equiv="refresh" content="5;url=/" />

A close up of text on a white background

Description automatically generated

Figure Example of the Visualization created in the html file.

The text comments being published are converted into sentiment scores using the “sentscore” function.

analyser = SentimentIntensityAnalyzer()

def sentscore(sentence):

score = analyser.polarity\_scores(sentence)

return score

The sentiment analysis library vadarSentiment is an easy to use method which doesn’t require machine learning. The output is fair, for a simple app such as this one and works pretty well. Text is input and three scores are output. The compound score is used to determine the sentiment of the text. Below is the breakdown.[[2]](#footnote-2)

1. **positive sentiment**: compound score >= 0.05
2. **neutral sentiment**: (compound score > -0.05) and (compound score < 0.05)
3. **negative sentiment**: compound score <= -0.05

The output scores are then converted into bytes and published.

**The second function** is used to create a html form where users can input comments. The comment is converted into bytes and send through Kafka where consumers can get access to them.

@app.route('/admin', methods=["GET","POST"])

def admin():

**if** request.form:

incoming\_data = request.form.get("opinion")

print (request.form.get("opinion"))

topic = client.topics['sentiment']

with topic.get\_sync\_producer() as producer:

value = incoming\_data

producer.produce(value.encode())

**return** render\_template("admin.html")

Below is snippet of the form creation syntax used in the html page. When the user opens the URL “localXXX:/admin” they get access to a page where they can post comments.

<h1>Please tell me if you like this app</h1>

<form method="POST" action="/admin">

<input type="text" name="opinion">

<input type="submit" value="Add">

A screenshot of a cell phone

Description automatically generated

Figure Comment Upload Page

The very end of every flask app, is the “app.run” command, which is executed when the file is run through terminal, an additional create() command is executed which will prepare the topics, so that the app can run without failure.

if \_\_name\_\_ == "\_\_main\_\_":

create()

app.run(debug=True, threaded=True)

The create function will try and create a topic called ‘sentiment’, if it fails it means the topic likely exists, if not the error message will be printed in the terminal, but the app will still run. If the topic is being created for the first time, a temporary producer is created and 0 is published 30 times, so that the visuals can still be made, but the sentiment will not be affected. The only flaw is that for a certain period of time the sentiment histogram will be skewed towards neutral. (This will be dealt with in the future)

def create\_topic(name):

admin\_client = KafkaAdminClient(bootstrap\_servers="localhost:9092", client\_id='test')

topic\_list = []

topic\_list.append(NewTopic(name=str(name), num\_partitions=1, replication\_factor=1))

admin\_client.create\_topics(new\_topics=topic\_list, validate\_only=False)

def create():

try:

create\_topic("sentiment")

topic = client.topics['sentiment']

with topic.get\_sync\_producer() as producer:

**for** i **in** range(30):

value = 'SecRet5543Xf'

producer.produce(value.encode())

except Exception as e:

print (e)

**A video demonstration of the app can be found in the link below.**

https://www.youtube.com/watch?v=ThSVPFV\_Iv8

1. https://pykafka.readthedocs.io/en/latest/api/client.html [↑](#footnote-ref-1)
2. https://github.com/cjhutto/vaderSentiment#introduction [↑](#footnote-ref-2)