Big Data Pipeline to Capture and Visualize Twitter Data

Abstract

A pipeline that consumes twitter data to extract meaningful insights about a variety of topics can be constructed with the following technologies: twitter API, Kafka, MongoDB, and Tableau. These technologies were chosen for very specific reasons. Firstly, the twitter API is leveraged to obtain information to be processed. Secondly Kafka takes the data and connects the various other components of this pipeline. Thirdly, MongoDB stores the obtained tweets for later analysis. Fourthly and finally, Tableau creates meaningful visualizations.

Introduction

The COVID-19 pandemic, also known as the coronavirus pandemic, is an ongoing global pandemic of coronavirus disease 2019 (COVID-19) caused by severe acute respiratory syndrome coronavirus 2 (SARS-CoV-2). The World Health Organization declared a Public Health Emergency of International Concern regarding COVID-19 on 30 January 2020, and later declared a pandemic on 11 March 2020. As of 10 May 2021, more than 158 million cases have been confirmed, with more than 3.29 million deaths attributed to COVID-19, making it one of the deadliest pandemics in history. Researchers and scientists across the world are developing new strategies and plans to tackle this pandemic and especially create awareness amongst common people on how to adopt safety measures. As computer science students we as part of the course decided to create a data pipeline for tweets especially mentioning covid and showing interesting patterns (heat map) of the areas from where the tweets were generated for analysis.

We seek to analyze how the Covid-19 pandemic's effects can be felt through twitter. To this end we created a big data pipeline to consume, analyze, and display information gleaned from tweets that come from all over the world. Specifically, we intend to analyze how hashtags are affected by the pandemic with meaningful terms directly related to the pandemic. To accomplish this goal at scale we must take several big data technologies and combine them together in order to create an effective pipeline.

This pipeline will start with the twitter API accessed via a python module. Tweets will be fed into a cluster running Kafka which will next direct it to Spark Streaming for any calculations that need to be performed. Processed data will be fed back into the Kafka cluster to be directed to the next step. This step will require the path that the data flows to be split, firstly the data will be sent into a MongoDB database for long term storage, and secondly, the data will be fed into Tableau. The data sent to Tableau will then be transformed into meaningful data visualizations for the end user to interpret and draw conclusions from.

We anticipate that our solution will be horizontally scalable and highly maintainable. While we are limited to the amount of data that the twitter API allows us to collect, in theory our solution would allow for massively larger velocities.

Literature Survey

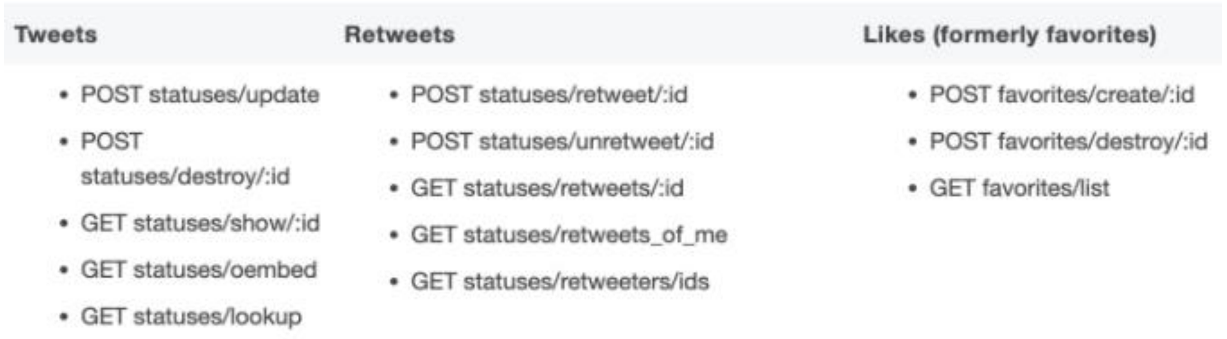
Twitter API Access

The first part of the project deals with accessing the data generated from twitter (Like live tweets or from a decided target).

The **Twitter API** lets you read and write **Twitter** data. Thus, you can use it to compose tweets, read profiles, and access your followers' data and a high volume of tweets on subjects in specific locations. **API** stands for Application Programming Interface. An API also provides a list of commands that can be executed and methods like GET, POST, PUT, DELETE.

Twitter API HTTP Methods and Endpoints

Following are the methods through which we can post, retrieve, and engage with tweets.



Tweets	Retweets	Likes (formerly favorites)
• POST statuses/update	• POST statuses/retweet/:id	• POST favorites/create/:id
• POST statuses/destroy/:id	• POST statuses/unretweet/:id	• POST favorites/destroy/:id
• GET statuses/show/:id	• GET statuses/retweets/:id	• GET favorites/list
• GET statuses/oembed	• GET statuses/retweets_of_me	
• GET statuses/lookup	• GET statuses/retweeters/ids	

After comparing pros and cons between python module and Twitter API we decided to implement python module in accessing twitter data as it is compatible with our further technologies, easy to debug and rectify and provides easy maintainability.

Accessing Twitter API and using tweets for creating big data pipeline

After carefully analyzing the ways in which the twitter data can be accessed, we finalized the following steps to get the tweets from twitter related to covid-19 -

1) The very first step is applying for a developer account to use the twitter API. This way we would be able to find tweets relating to the coronavirus in a real-time setting.
2) Then generate the access token, access token secret, consumer key and consumer secret for authentication.
3) After receiving the account and configuring it, we develop a python script to listen to tweets and use Kafka to transmit the data to an application.
4) The data visualization and representation part will be handled using MongoDB database and Tableau which is explained in further sections.

Kafka

We studied two platforms for our project to deal with huge data that we will receive from Twitter. They were Kafka and Kinesis.

Kafka, an open-source platform that makes it an incredibly easy way to move and ingest a large amount of data into multiple platforms or apps. Companies like **LinkedIn, Yelp, and Pinterest** all use Kafka in the platforms to handle the massive amount of data generated by users.

Amazon Kinesis is a product offered by AWS to collect and process streaming data. This product is highly scalable in terms of volume, whether that be the number of incoming streams or the velocity of those streams. ETL and real-time analytics are both supported by Amazon Kinesis (*Amazon Kinesis*, n.d.).

Amazon Kinesis has several subproducts such as Streams and Firehose for slightly different purposes. Amazon Kinesis Streams is more involved than Amazon Kinesis Firehose and will require more maintenance by the user which would be provided by AWS for Amazon Kinesis Firehose and data expires after a week unlike Firehose for which expires at a customizable speed. Regardless of the exact version that you use it tends to create a complex system that is difficult to manage in practice (Özal, 2021).
Amazon Kinesis makes it easy to collect, process, and analyze real-time, streaming data so you can get timely insights and react quickly to new information. Amazon Kinesis offers key capabilities to cost-effectively process streaming data at any scale, along with the flexibility to choose the tools that best suit the requirements of your application. With Amazon Kinesis, you can ingest real-time data such as video, audio, application logs, website clickstreams, and IoT telemetry data for machine learning, analytics, and other applications. Amazon Kinesis enables you to process and analyze data as it arrives and respond instantly instead of having to wait until all your data is collected before the processing can begin (Low, 2017).

Comparative Analysis between Kafka and Kinesis

	Kafka	**Amazon Kinesis**
Concepts	Kafka Streams	Kinesis Analytics
Stream of records container	Topic	Stream
Data Storage	Kafka Partition	Kinesis Shared
Unique ID of the record	Offset number	Sequence Number
Ordering under	Partition level	Shared level
SDK Support	Supports Java	Supports Android, Java, Go, .NET
Configuration and Features	More control and better performance	Number of days/shards can only be configured
Reliability	Replication factor can be configured	Kinesis writes synchronously to 3 different machines/data-centers
Performance	Kafka Wins	Kinesis is a bit slower than Kafka
Setup	Weeks	A couple of hours
Incident Risk/Maintenance	More In Kafka	Amazon takes care
Human Costs	Require human support for installing and managing their clusters, and also accounting for requirements such as high availability, durability, and recovery	Kinesis is just about paying and use

So based upon the overall analysis we decided to implement the project using **Kafka**.

MongoDB

MongoDB’s document data model naturally supports JSON and its expressive query language is simple for developers to learn and use. MongoDB is schema-less. It is a document database in which one collection holds different documents. But in RDBMS, you need to first design your tables, data structure, relations, and only then can you start coding. Functionality such as automatic failover, horizontal scaling, and the ability to assign data to a location are built in.

MongoDB is almost 100 times faster than traditional database systems like RDBMS, which is slower in comparison with the NoSQL databases.

The data that we fetch from twitter will be mostly in the form of tweets and hashtags. But taking social media into consideration, users share data in all formats, which could be textual, image, videos and audio. Hence, instead of going with traditional database systems, MongoDB is preferable to store all kinds of data irrespective of the format.

Tableau

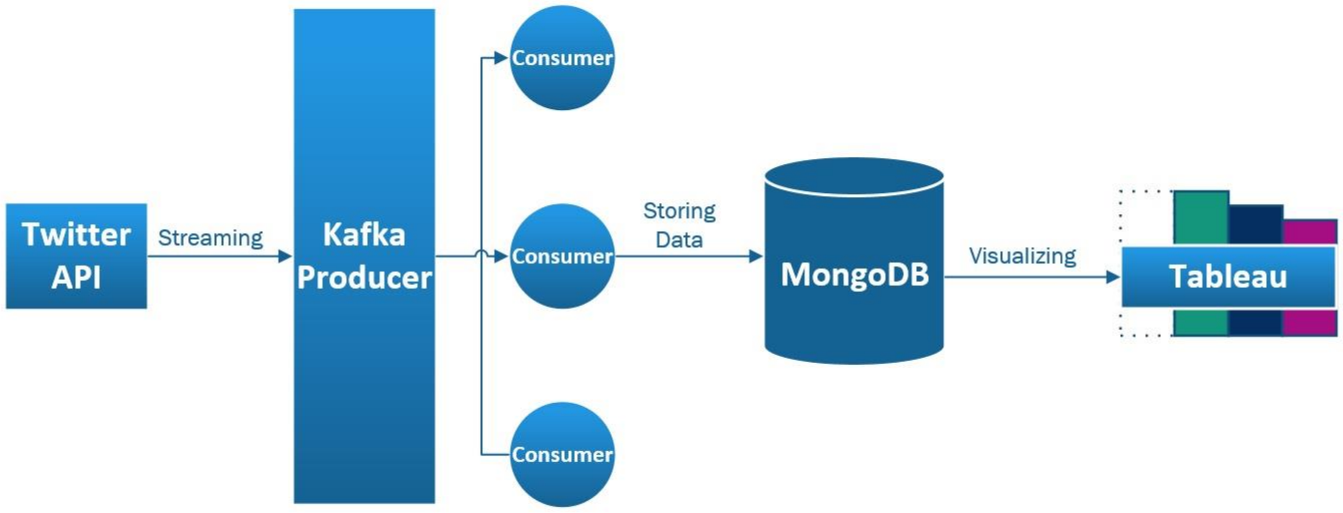
Tableau is a visual analytics platform which transforms the way people use data to solve problems. Many organizations of all sizes trust Tableau to help them be more data driven. State-of-the-art dashboards and plots are easy to create on the platform. We can deploy the results in the cloud, on-premises, or natively integrate with Salesforce CRM. It is easy to connect all of the data with fully integrated AI/ML capabilities, governance and data management, visual storytelling and collaboration.

Our project uses Kafka to live stream twitter data. Tableau can establish a live connection to the database for dynamic results shown on the dashboard which is helpful to monitor active changes in the data.

As it is an extensively used tool, incorporating it to analyze twitter data can be helpful for the team to understand what industry level plots look like and gain experience on trending tools that are readily used by various data-driven organizations.

Architecture of Data Processing Pipeline

The following is the architecture of the data processing pipeline for this project. We will be only creating one consumer node for this project, but multiple consumers can be connected to the same topic if and when needed.



Overview of the project

At a high level our project consists of four phases: gathering the data, the Kafka cluster, MongoDB, and Tableau visualization. Firstly, we gather data with tweepy, which is passed off to a Kafka cluster which then uploads the data to our MongoDB database. Finally, the data is downloaded in full for Tableau to visualize.

The following steps are required to execute the complete big data pipeline project:

Step 1 - Creation of the twitter developer account and generating the access keys.

Step 2 - Setting up the single node Kafka cluster with zookeeper.

Step 3 - Creation of the topic.

Step 4 - Setting up the producer and consumer.

Step 5 - Accessing twitter API using tweepy by python script - twitterproducer.py. Using the same script, tweets are received in the producer by connecting the twitter stream to the Kafka cluster.

Step 8 - Receiving tweets in the consumer.

Step 9 - Setting up the database and collection in the MongoDB compass.

Step 10 - Passing the tweets from the consumer to the MongoDB database through python script mongoDbconsumer.py.

Step 11 - Exporting the data to Tableau software through a .csv file named - Covid_filtered

Step 12 - Showing the visualization in the form of a heat map for depicting the density of tweets generated from different countries.

Technical Details

How does Kafka work in a nutshell?

Kafka is a distributed system consisting of servers and clients that communicate via a high-performance TCP network protocol. It can be deployed on bare-metal hardware, virtual machines, and containers in on-premises as well as cloud environments.

Servers: Kafka is run as a cluster of one or more servers that can span multiple data centers or cloud regions. Some of these servers from the storage layer, called the brokers. Other servers run Kafka Connect to continuously import and export data as event streams to integrate Kafka with your existing systems such as relational databases as well as other Kafka clusters. To let you implement mission-critical use cases, a Kafka cluster is highly scalable and fault-tolerant: if any of its servers fails, the other servers will take over their work to ensure continuous operations without any data loss.

System prerequisites

For setting up a Kafka Cluster we need the following installed:

1) **Installing Java 8 or higher -**

   Link - https://java.com/en/

2) **Downloading Kafka -**
   We downloaded the Apache Kafka Scala 2.13 version which is compatible with Java as well as Scala language for multipurpose use.

   Link - https://kafka.apache.org/downloads

Now along with the Kafka download we also need the zookeeper installed for the system. So we downloaded the Apache Zookeeper Release - 3.6.3 by going to the releases page.

3) **We downloaded the 3.7.0 version consisting of the bin.tar.gz file.**

Link - https://zookeeper.apache.org/

Setting up the Kafka Cluster is the most important part of the project and it involves the following steps. For cluster setup to function we need the zookeeper.

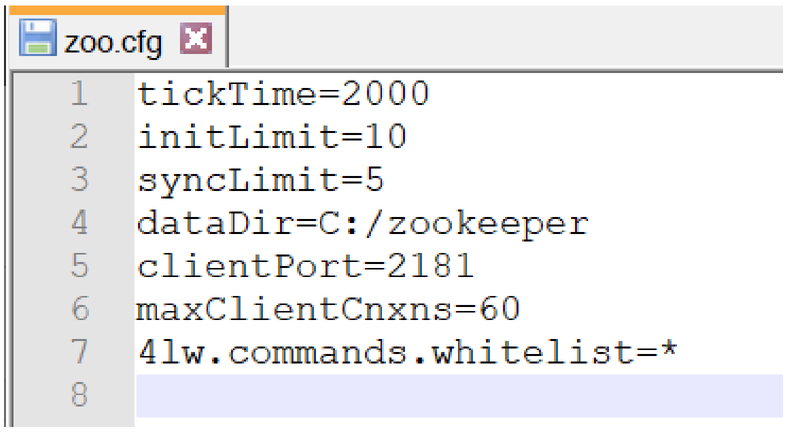
Setting up the zookeeper

1) Extract the apache zookeeper 3.6.3**.tar.gz** file.

2) We need to modify the properties config file for the zookeeper setup. So for that go to the **conf** directory and search for **zoo.cfg** file.

3) We noticed that the following file is not present, so we copied the zoo_sample.cfg file and renamed it to zoo.cfg file for the same.

4) Inside the zoo.cfg file we made some changes for our project according to our needs. We removed all the commented lines in the file and following was the final zoo.cfg file which we used:



```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=C:/zookeeper
clientPort=2181
maxClientCnxns=60
4lw.commands.whitelist=*

```

5) Now once all the settings are done, go to the bin directory amongst the extracted files. Then run the zkServer.cmd file for zookeeper setup so that we can run the zookeeper through the command prompt next.

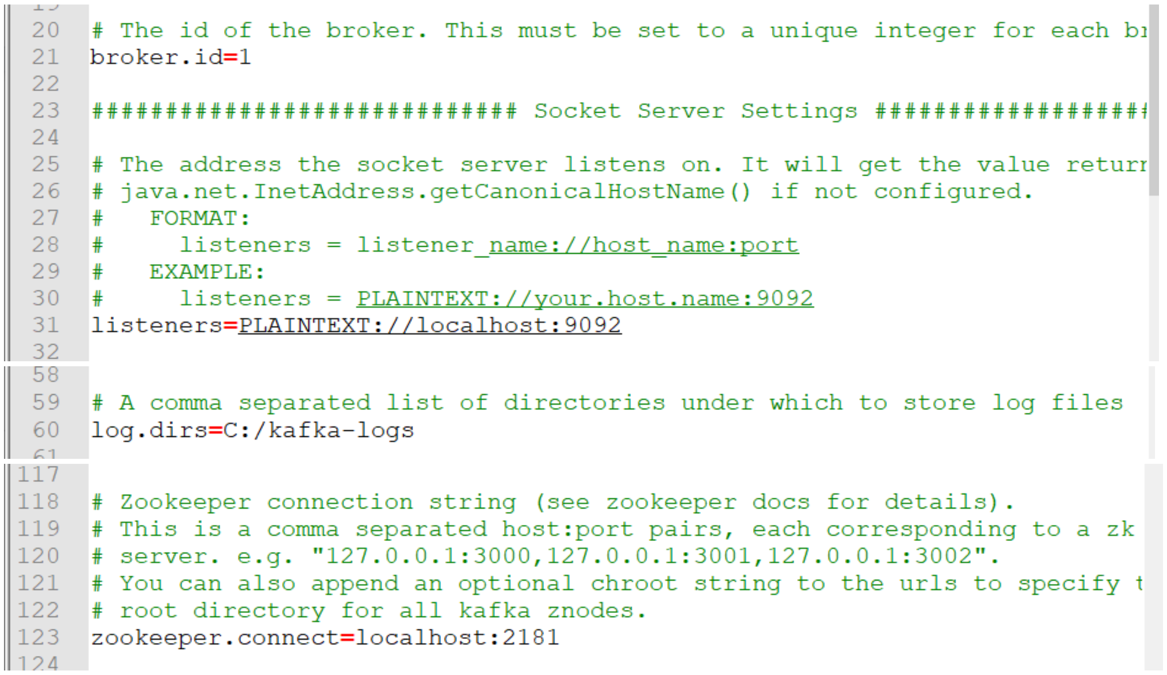
Starting the Kafka server

1) Extract the kafka2.13-2.8.0**.tar.gz** file.

2) Before we start the setup, we require the configuration file for setup which is present inside the config directory called as server.properties.

3) We edited the server.properties file and made some changes to keep the properties for our project. We learnt that following important properties need to be changed:

    1) Broker-id.
    2) Listeners.
    3) Log.dirs
    4) Zookeeper.connect

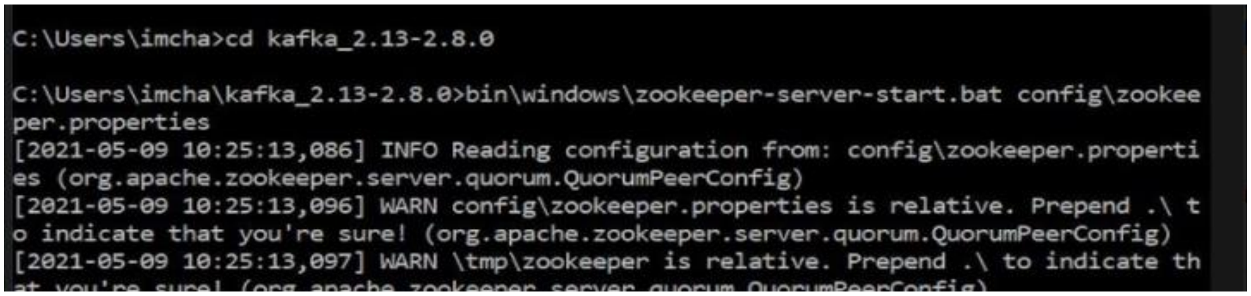


```
# The id of the broker. This must be set to a unique integer for each b
broker.id=1

############################# Socket Server Settings ##################

# The address the socket server listens on. It will get the value retur
# java.net.InetAddress.getCanonicalHostName() if not configured.
#   FORMAT:
#     listeners = listener_name://host_name:port
#   EXAMPLE:
#     listeners = PLAINTEXT://your.host.name:9092
listeners=PLAINTEXT://localhost:9092


# A comma separated list of directories under which to store log files
log.dirs=C:/kafka-logs


# Zookeeper connection string (see zookeeper docs for details).
# This is a comma separated host:port pairs, each corresponding to a zk
# server. e.g. "127.0.0.1:3000,127.0.0.1:3001,127.0.0.1:3002".
# You can also append an optional chroot string to the urls to specify t
# root directory for all kafka znodes.
zookeeper.connect=localhost:2181
```

4) In order to start the zookeeper server, we need the server file, search for zookeeper-server-start batch file inside bin\windows.

5) Open the command prompt and start the zookeeper server using the following command from the Kafka directory:

    bin\windows\zookeeper-server-start.bat config\zookeeper.properties

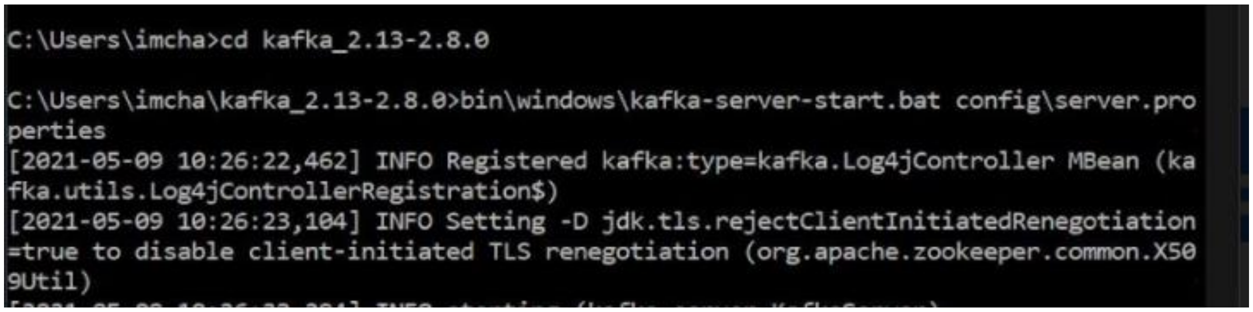


```
C:\Users\imcha>cd kafka_2.13-2.8.0

C:\Users\imcha\kafka_2.13-2.8.0>bin\windows\zookeeper-server-start.bat config\zookee
per.properties
[2021-05-09 10:25:13,086] INFO Reading configuration from: config\zookeeper.properti
es (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-05-09 10:25:13,096] WARN config\zookeeper.properties is relative. Prepend .\ t
o indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-05-09 10:25:13,097] WARN \tmp\zookeeper is relative. Prepend .\ to indicate th
at you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
```

6) To start the Kafka server we need the server file, search for kafka-server-start batch file inside bin\windows.

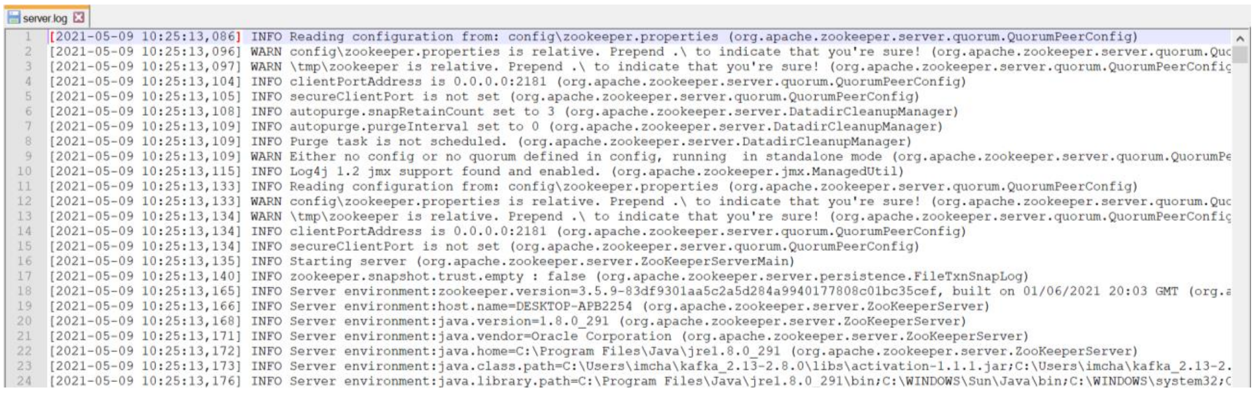
7) Open another command prompt and start the Kafka server using the following command from the Kafka directory:

   bin\windows\kafka-server-start.bat config\server.properties



```
C:\Users\imcha>cd kafka_2.13-2.8.0

C:\Users\imcha\kafka_2.13-2.8.0>bin\windows\kafka-server-start.bat config\server.pro
perties
[2021-05-09 10:26:22,462] INFO Registered kafka:type=kafka.Log4jController MBean (ka
fka.utils.Log4jControllerRegistration$)
[2021-05-09 10:26:23,104] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation
=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X50
9Util)
```

8) The logs for the Kafka server were generated inside the logs directory in the server.log file. We can check all of the logs generated for Kafka server.



```
[2021-05-09 10:25:13,086] INFO Reading configuration from: config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-05-09 10:25:13,096] WARN config\zookeeper.properties is relative. Prepend .\ to indicate that you're sure! (org.apache.zookeeper.server.quorum.Quc
[2021-05-09 10:25:13,097] WARN \tmp\zookeeper is relative. Prepend .\ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfic
[2021-05-09 10:25:13,104] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-05-09 10:25:13,105] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-05-09 10:25:13,108] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2021-05-09 10:25:13,109] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)
[2021-05-09 10:25:13,109] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2021-05-09 10:25:13,109] WARN Either no config or no quorum defined in config, running  in standalone mode (org.apache.zookeeper.server.quorum.QuorumPe
[2021-05-09 10:25:13,115] INFO Log4j 1.2 jmx support found and enabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2021-05-09 10:25:13,133] INFO Reading configuration from: config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-05-09 10:25:13,133] WARN config\zookeeper.properties is relative. Prepend .\ to indicate that you're sure! (org.apache.zookeeper.server.quorum.Quc
[2021-05-09 10:25:13,134] WARN \tmp\zookeeper is relative. Prepend .\ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfic
[2021-05-09 10:25:13,134] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-05-09 10:25:13,134] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-05-09 10:25:13,135] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2021-05-09 10:25:13,140] INFO zookeeper.snapshot.trust.empty : false (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2021-05-09 10:25:13,165] INFO Server environment:zookeeper.version=3.5.9-83df9301aa5c2a5d284a9940177808c01bc35cef, built on 01/06/2021 20:03 GMT (org.a
[2021-05-09 10:25:13,166] INFO Server environment:host.name=DESKTOP-APB2254 (org.apache.zookeeper.server.ZooKeeperServer)
[2021-05-09 10:25:13,168] INFO Server environment:java.version=1.8.0_291 (org.apache.zookeeper.server.ZooKeeperServer)
[2021-05-09 10:25:13,171] INFO Server environment:java.vendor=Oracle Corporation (org.apache.zookeeper.server.ZooKeeperServer)
[2021-05-09 10:25:13,172] INFO Server environment:java.home=C:\Program Files\Java\jre1.8.0_291 (org.apache.zookeeper.server.ZooKeeperServer)
[2021-05-09 10:25:13,173] INFO Server environment:java.class.path=C:\Users\imcha\kafka_2.13-2.8.0\libs\activation-1.1.1.jar;C:\Users\imcha\kafka_2.13-2.
[2021-05-09 10:25:13,176] INFO Server environment:java.library.path=C:\Program Files\Java\jre1.8.0_291\bin;C:\WINDOWS\Sun\Java\bin;C:\WINDOWS\system32;C
```

With this way we connected the Kafka with Zookeeper and a single node Kafka cluster was created.

Creating a Kafka Topic - covid-19.

Open a new terminal and create a topic using the following command:

```
bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic covid-19
```

After running this command, our topic is ready for use with name: **covid-19**

Setting up MongoDB

We decided to use MongoDB for the purposes of preserving our gathered data for later batch purposes. Our choice to MongoDB was a result of several considerations. Firstly, our data from Twitter is denormalized with complex columns, as such we needed a NoSQL solution. Secondly, MongoDB is very scalable and has a great deal of support available in both the community that uses it and tools already built. Our implementation of this database changed through our attempts at actualizing it. Initially we intended to use an AWS managed service called Amazon DocumentDB. This allowed for a cost-effective, scalable, and very easily manageable solution. Indeed, setting up the database could be done in minutes without much additional work. However, problems appeared when trying to connect to this database. When this did not fix the issue, we decided that we needed to pivot to another solution and thus changed which managed MongoDB service we employed. MongoDB Atlas is the service that we tried next to host and manage our MongoDB service. Code was originally written in python to pull data out of the database but was replaced with manually using MongoDB Compass.

Creating the Database

1) Navigate to the official MongoDB website and download the latest version of MongoDB Compass.

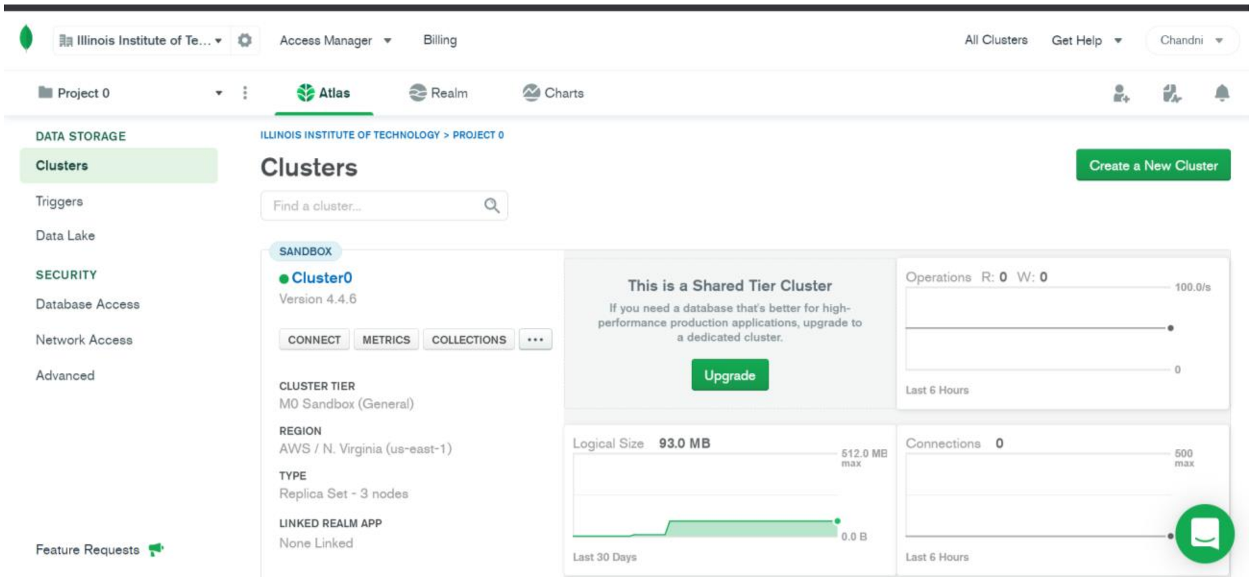
2) Navigate to the MongoDB Atlas section of the website and sign in or create an account.

3) Create a shared cluster (The free option).

4) Under the *SECURITY* section click on *Database Access*. Click *Add New Database User*. Fill in the username and password fields and finalize by clicking *Add User*.

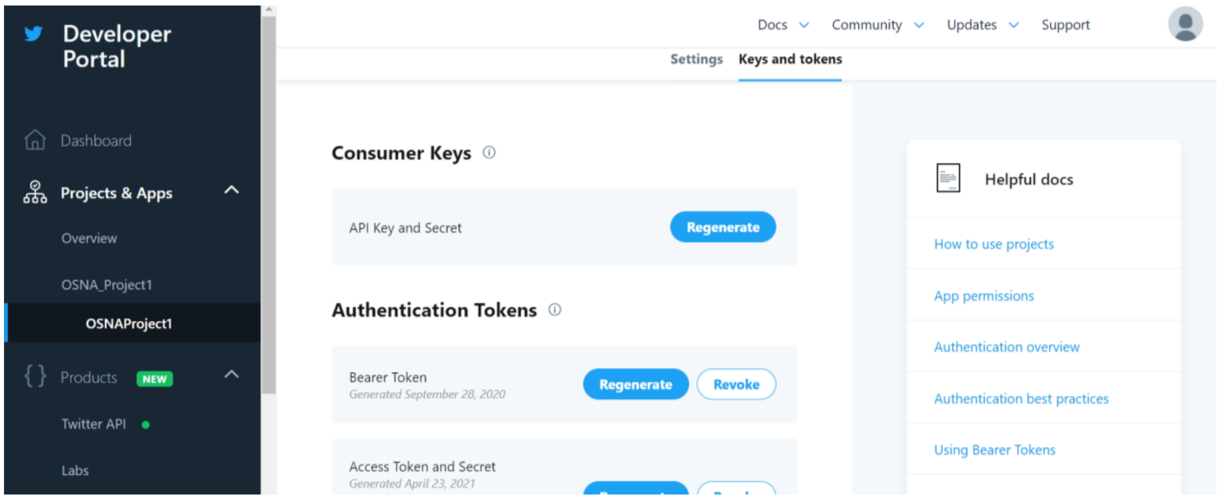
5) Under the *SECURITY* section click on *Network Access*. Click *Add IP Address* then *ALLOW ACCESS FROM ANYWHERE* and finalize by clicking *Confirm*.

6) Once the cluster is created (this process can take quite a while) click *COLLECTIONS* then *Add My Own Data*. Fill in *CSP554* in the *DATABASE NAME* field and *COVID* in the *COLLECTION NAME* field. Finalize by clicking *CREATE*.



Obtaining Twitter Data

Firstly, the initial part of the project was to fetch the tweets from twitter to use them for analysis while showcasing them in Tableau software.



Twitter provides a developer portal to is a set of self-serve tools that developers can use to manage their access as well as to create and manage their Projects and Apps. In the portal, you can: Create and manage your Twitter Projects and Apps.

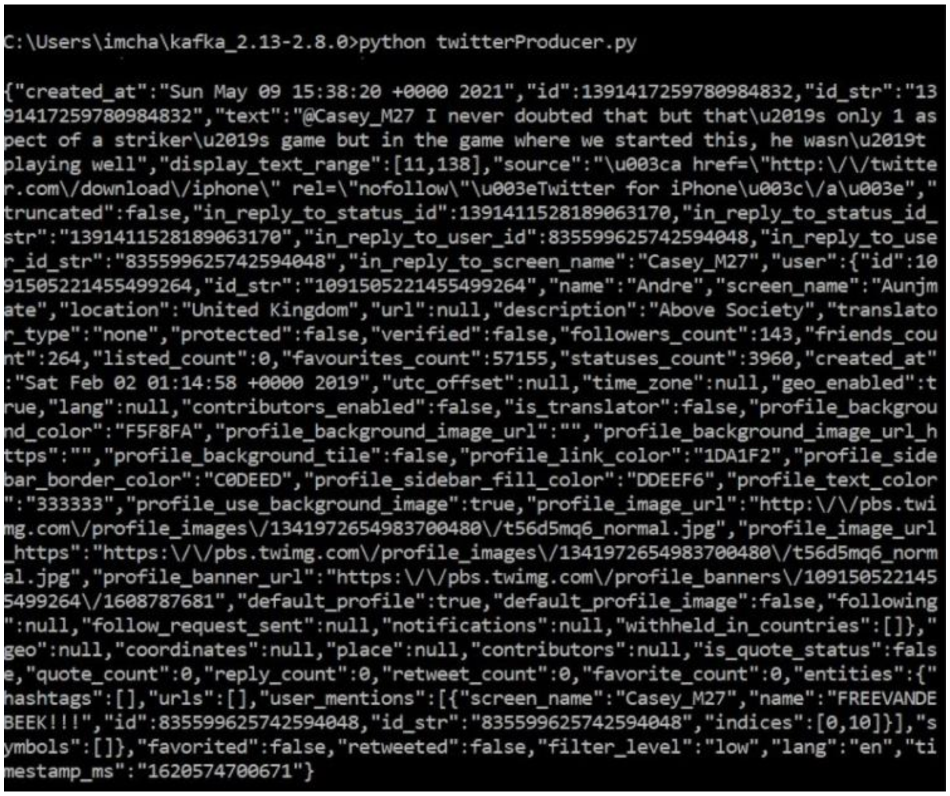
The purpose of creating or using an existing developer account is to access tweets through twitter API. In order to fetch the tweets, we used the python script. 4 keys for authentication required to connect to the twitter API and fetch tweets are:

1) Access Token
2) Access token secret
3) Consumer Key
4) Consumer Secret

Attached below is the python code file for fetching the tweets through the developer account. Interesting to note is that python provides an open-source package called Tweepy which needs to be downloaded prior to the development of the code as it is a very convenient way to access the Twitter API with Python. **Tweepy** includes a set of classes and methods that represent Twitter's models and API endpoints, and it transparently handles various implementation details, such as: Data encoding and decoding.

Producer

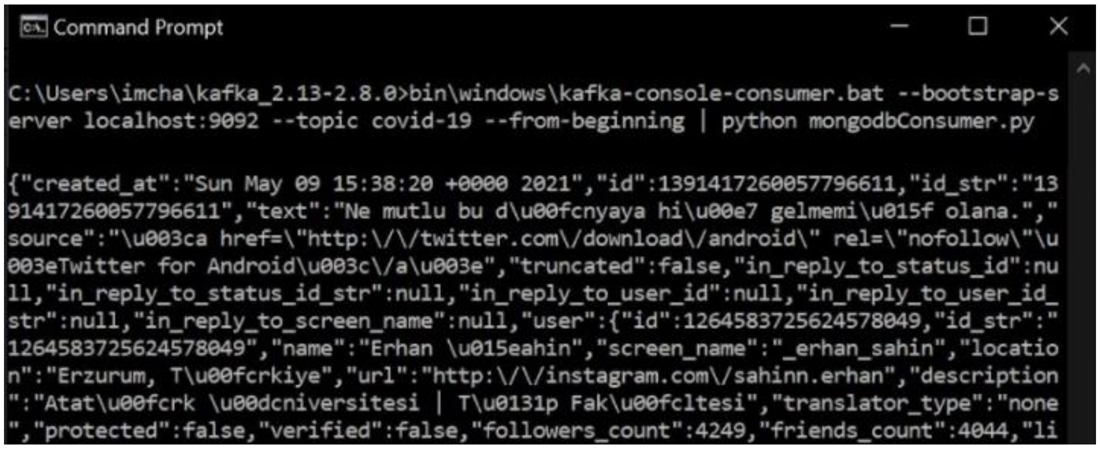
The twitterproducer.py script connects the twitter stream to the Kafka cluster as a producer and tweets are passed:



```
C:\Users\imcha\kafka_2.13-2.8.0>python twitterProducer.py

{"created_at":"Sun May 09 15:38:20 +0000 2021","id":1391417259780984832,"id_str":"13
91417259780984832","text":"@Casey_M27 I never doubted that but that\u2019s only 1 as
pect of a striker\u2019s game but in the game where we started this, he wasn\u2019t
playing well","display_text_range":[11,138],"source":"\u003ca href=\"http:\/\/twitte
r.com\/download\/iphone\" rel=\"nofollow\"\u003eTwitter for iPhone\u003c\/a\u003e","
truncated":false,"in_reply_to_status_id":1391411528189063170,"in_reply_to_status_id_
str":"1391411528189063170","in_reply_to_user_id":835599625742594048,"in_reply_to_use
r_id_str":"835599625742594048","in_reply_to_screen_name":"Casey_M27","user":{"id":10
91505221455499264,"id_str":"1091505221455499264","name":"Andre","screen_name":"Aunjm
ate","location":"United Kingdom","url":null,"description":"Above Society","translato
r_type":"none","protected":false,"verified":false,"followers_count":143,"friends_cou
nt":264,"listed_count":0,"favourites_count":57155,"statuses_count":3960,"created_at"
:"Sat Feb 02 01:14:58 +0000 2019","utc_offset":null,"time_zone":null,"geo_enabled":t
rue,"lang":null,"contributors_enabled":false,"is_translator":false,"profile_backgrou
nd_color":"F5F8FA","profile_background_image_url":"","profile_background_image_url_h
ttps":"","profile_background_tile":false,"profile_link_color":"1DA1F2","profile_side
bar_border_color":"C0DEED","profile_sidebar_fill_color":"DDEEF6","profile_text_color
":"333333","profile_use_background_image":true,"profile_image_url":"http:\/\/pbs.twi
mg.com\/profile_images\/1341972654983700480\/t56d5mq6_normal.jpg","profile_image_url
_https":"https:\/\/pbs.twimg.com\/profile_images\/1341972654983700480\/t56d5mq6_norm
al.jpg","profile_banner_url":"https:\/\/pbs.twimg.com\/profile_banners\/109150522145
5499264\/1608787681","default_profile":true,"default_profile_image":false,"following
":null,"follow_request_sent":null,"notifications":null,"withheld_in_countries":[]},"
geo":null,"coordinates":null,"place":null,"contributors":null,"is_quote_status":fals
e,"quote_count":0,"reply_count":0,"retweet_count":0,"favorite_count":0,"entities":{"
hashtags":[],"urls":[],"user_mentions":[{"screen_name":"Casey_M27","name":"FREEVANDE
BEEK!!!","id":835599625742594048,"id_str":"835599625742594048","indices":[0,10]}],"s
ymbols":[]},"favorited":false,"retweeted":false,"filter_level":"low","lang":"en","ti
mestamp_ms":"1620574700671"}
```

Consumer

Here, we used **mongodbConsumer.py** script to pass data to MongoDB from the consumer. On a new terminal run the following command to start the consumer and pass the data into MongoDB:

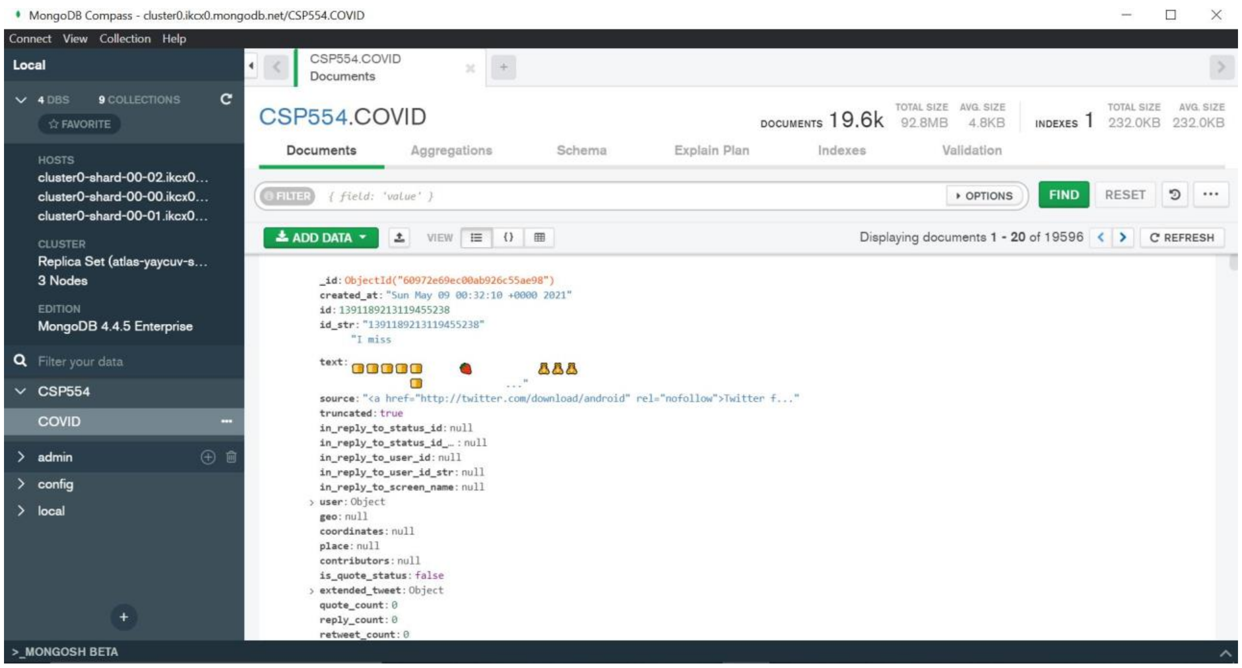


```
C:\Users\imcha\kafka_2.13-2.8.0>bin\windows\kafka-console-consumer.bat --bootstrap-s
erver localhost:9092 --topic covid-19 --from-beginning | python mongodbConsumer.py

{"created_at":"Sun May 09 15:38:20 +0000 2021","id":1391417260057796611,"id_str":"13
91417260057796611","text":"Ne mutlu bu d\u00fcnyaya hi\u00e7 gelmemi\u015f olana.","
source":"\u003ca href=\"http:\/\/twitter.com\/download\/android\" rel=\"nofollow\"\u
003eTwitter for Android\u003c\/a\u003e","truncated":false,"in_reply_to_status_id":nu
ll,"in_reply_to_status_id_str":null,"in_reply_to_user_id":null,"in_reply_to_user_id_
str":null,"in_reply_to_screen_name":null,"user":{"id":1264583725624578049,"id_str":"
1264583725624578049","name":"Erhan \u015eahin","screen_name":"_erhan_sahin","locatio
n":"Erzurum, T\u00fcrkiye","url":"http:\/\/instagram.com\/sahinn.erhan","description
":"Atat\u00fcrk \u00dcniversitesi | T\u0131p Fak\u00fcltesi","translator_type":"none
","protected":false,"verified":false,"followers_count":4249,"friends_count":4044,"li
```

In this way we connect the twitter stream to the Kafka cluster as a producer, and set up a producer that sends data to MongoDB.

Data Collected in MongoDB

The data was streamed into MongoDB in real-time and was collected in the following format:



```
_id: ObjectId("60972e69ec00ab926c55ae98")
created_at: "Sun May 09 00:32:10 +0000 2021"
id: 1391189213119455238
id_str: "1391189213119455238"
      "I miss
text: 🟨🟨🟨🟨🟨     🍓          ⚠⚠⚠
              🟨          ..."
source: "<a href="http://twitter.com/download/android" rel="nofollow">Twitter f..."
truncated: true
in_reply_to_status_id: null
in_reply_to_status_id_... : null
in_reply_to_user_id: null
in_reply_to_user_id_str: null
in_reply_to_screen_name: null
> user: Object
geo: null
coordinates: null
place: null
contributors: null
is_quote_status: false
> extended_tweet: Object
quote_count: 0
reply_count: 0
retweet_count: 0
```

MongoDB Compass - cluster0.ikcx0.mongodb.net/CSP554.COVID
Connect View Collection Help
Local
4 DBS 9 COLLECTIONS
FAVORITE
HOSTS
cluster0-shard-00-02.ikcx0...
cluster0-shard-00-00.ikcx0...
cluster0-shard-00-01.ikcx0...
CLUSTER
Replica Set (atlas-yaycuv-s...
3 Nodes
EDITION
MongoDB 4.4.5 Enterprise
Filter your data
CSP554
COVID
admin
config
local
CSP554.COVID
Documents
DOCUMENTS 19.6k TOTAL SIZE 92.8MB AVG. SIZE 4.8KB
INDEXES 1 TOTAL SIZE 232.0KB AVG. SIZE 232.0KB
Documents Aggregations Schema Explain Plan Indexes Validation
FILTER { field: 'value' }
OPTIONS FIND RESET
ADD DATA VIEW
Displaying documents 1 - 20 of 19596
REFRESH
>_MONGOSH BETA

Connecting to Tableau

Connecting Tableau to our data pipeline became a great source of trouble for our group. Firstly, it is important to note that we were utilizing the community edition of Tableau and were working on a tight budget as such could not afford tools that would require purchase. Secondly, during the stage on which we were attempting to set up our Kafka cluster we considered several tools to connect the cluster to our various other tools including Confluent and its suite of connectors. However, the set-up proved unacceptably difficult to set up despite many hours of effort. Ultimately, we abandoned our initial plans to connect Tableau directly to the Kafka cluster to do visualizations on stream data. While we still believe that this is a viable option, we also decided to switch to visualizing the batch data gathered by the pipeline for several reasons. Firstly, while this set up is intended to be highly scalable in each component our testing only goes as far as gathering a few thousand data points so the difference between visualizations of streaming data and batch data would be negligible in our testing. Secondly, visualizing streaming data does not assist in drawing conclusions about Covid-19 more than visualizing batch data from a specified time range. Thirdly, time-constraints forced us to shift towards connecting Tableau to MongoDB instead of the Kafka cluster. As this was decidedly the least important aspect of the project, we shifted our attention to solving the more important problems of the implementation.

Steps Taken to Obtain Data from MongoDB

1) Open MongoDB Compass.

2) Navigate to your cluster on MongoDB Atlas and click *CONNECT*. Click *Connect using MongoDB Compass* then *I have MongoDB Compass*, copy the string in step 2.

3) In MongoDB Compass paste the string you just copied and replace the username and password with the ones you created when setting up MongoDB. Finally hit *Connect*.

4) Open the collection inside the database and click on the *Export Collection* icon and then select required fields. Also select between JSON or CSV format (we used CSV format for this project) and click *EXPORT*.

Tableau Visualizations and Analysis

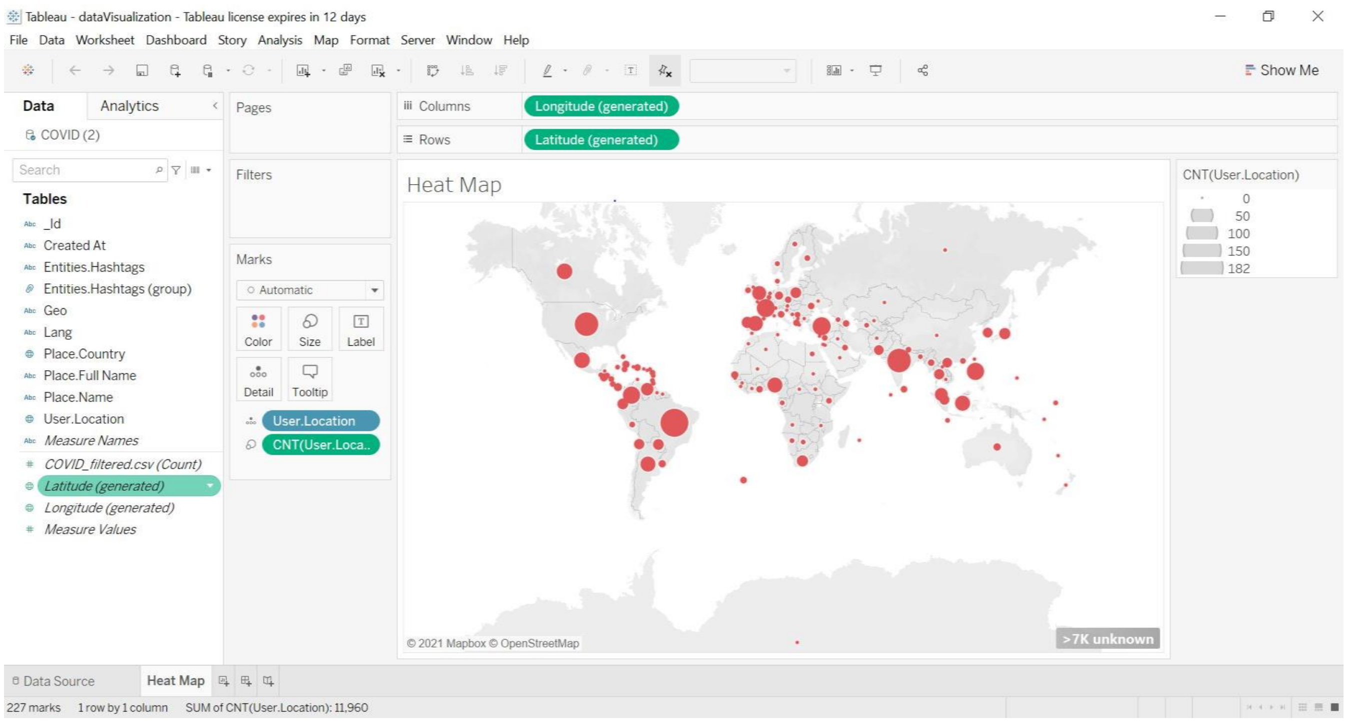
For data visualization in tableau, following are the steps:

1. We downloaded the installation link for tableau for desktop from the following source:

Link - https://www.tableau.com/trial/tableau-software

2. After the installation of tableau, we connected the COVID_filtered.csv file to import data into tableau.

3. Then the data was visualized using the User.Location and the density of the tweets from each location into a heat map.



Conclusion

This project resulted in several valuable insights both in the tools utilized to create the pipeline and the data analyzed. Firstly, upon examining the visualizations we see a relative concentration of tweets containing the *COVID* hashtag in the Americas, Europe, and Southern Asia, this seems to line up with expectations of areas that both have a high adoption of twitter and many Covid-19 cases. Further work needs to be done to validate this conclusion though. However, the focus of this project is on the methodologies of data collection and as such our insights are more focused on this area. So secondly, managed services are very valuable in data pipelines. We used a combination of managed services like MongoDB Atlas and self-run services like our Kafka cluster. The time spent to set up and manage these two were vastly different. The Kafka cluster took an entire day with three graduate students working together to get up and running, while the managed MongoDB database took one student a few minutes to set up. Thirdly, and finally, scalable solutions require independent components to be decoupled. Each individual component in our pipeline is decoupled from the other relying on Python scripts or like communicate with each other. This lack of coupling allows for individual tuning for the scale of deployment.

All implementations that combine multiple professional tools result in unexpected difficulties, this project was no exception. Initial plans to connect each component through Kafka was inspired by the Confluent which has a suite of tools that allow convenient connection between Kafka and other tools such as MongoDB. However, the tool proved to be too difficult to set up and was subsequently abandoned for python scripts that worked just as well but were far more manageable. Another instance of difficulties faced is our Kafka consumer which is a combination of a Windows Batch file and python script piped together. We had a working Windows Batch file and a python script that couldn't seem to connect to the Kafka cluster for reasons we were never able to discover. So instead of further debugging we simply piped the output of our working Batch file into the Python script which connected to the MongoDB cluster.

The data pipeline built here can be further built upon. Each component in this pipeline, with the sole exception of the Tableau visualization stage, is highly scalable. Horizontal scalability can be directly utilized to grow the cluster size in the Kafka cluster and MongoDB cluster. The connector scripts can also share the load of each component. Further work on this pipeline could include adapting the Python scripts to better handle load balancing and sharing for horizontal scalability.

Bibliography

Amazon Kinesis. (n.d.). AWS. https://aws.amazon.com/kinesis/

Apache Flink. (n.d.). AWS. https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-flink.html

Apache Flink vs Apache Spark – A comparison guide. (n.d.). Data Flair. https://data-flair.training/blogs/comparison-apache-flink-vs-apache-spark/#:~:text=Spark%20has%20already%20been%20deployed,as%204G%20of%20Big%20Data.

Low, C. (2017, May 26). AMAZON KINESIS VS. APACHE KAFKA FOR BIG DATA ANALYSIS. *Dataconomy*. https://dataconomy.com/2017/05/kinesis-kafka-big-data-analysis/

Özal, S. (2021, January). How and Why You Should Use Amazon Kinesis for Your Data Streams. *Thundra*. https://blog.thundra.io/how-and-why-you-should-use-amazon-kinesis-for-your-data-streams

Spark Streaming Programming Guide. (n.d.). Apache Spark. https://spark.apache.org/docs/latest/streaming-programming-guide.html

What is Apache Flink? —Architecture. (n.d.). Apache Flink. https://flink.apache.org/flink-architecture.html

MongoDB. https://www.mongodb.com/2

Kunal Sethi (July, 2018). MongoDB vs. RDBMS. https://dzone.com/articles/mongodb-vs-rdbms

Tableau. https://www.tableau.com/