Problem Decomposition:

The problem is divided into 2 parts
1.Getting all cleaned data into a database.
2.Firing efficient queries to revive the results, adding it to API services.

Part 1.
Selection of DataBase
Best options available for databases are Elasticsearch and mongoDB.
Looking at our requirements and availability we choose mongoDB.

Some points that support the use of mongoDB are.
1.Elasticsearch is slow in adding data as compared to mongoDB , and our data is going to be streaming tweets from some source.

2.In ElasticSearch Indexing semantics is defined on client side, so the actual indexing cannot be optimized as well as with real storages.

3.MongoDB doesn't create any issue when there is a need to handle a huge volume of data.A MapReduce system can simply be implemented and there are certain tasks it can perform manually.

4.Since twitter data is received in JSON format ,and both database options are compatible to JSON format, we could use any database.

5.MongoDB offers a high insert rate which is useful in a situation where the write load is high.In our problem this is relevant due to streaming tweets.

Other options available for DataBase:

1.OrientDB: OrientDB is another open source NoSQL multi-model database. Organizations can now unlock the true power of graph databases. This can be done without any need of deploying multiple systems to handle different data types. This helps optimize performance and security while supporting scalability.

2.CouchDB : CouchDB is also an open source NoSQL Database System. This tool is built to offer web accessibility which supports a variety of devices. Data here is stored in JSON format, and organized into key-value pairs which are similar to the MapReduce format.

3.PostgreSQL : PostgreSQL is a very popular and widely used open-source database management system. This alternative of MongoDB provides support for SQL for relational as

well as JSON for non-relational queries. Hence it performs efficiently with both, structured and unstructured data.

4.Apache Cassandra : The Apache Cassandra is an ideal choice for the user if scalability and high availability are required and at the same time not affecting its performance. This alternative of MongoDB offers support for replication of data across multiple data centers. Hence providing security and durability without compromising on the efficiency.

How to get data from twitter.
There are many API's available to collect data from twitter.
One such is twitter / hbc in java
While collecting the tweets,the Keywords are predefined related to coronavirus.

Understanding the tweets JSON Data
Twitter data is in json format containing multiple name/value pairs:

In order to solve given queries only mentioned fields are important
1.created_at -> time at which tweet is created.
2.source.screen_location-> to get the location of user
3.text -> to read the tweet text.

Some other useful info.
1.Geo location is not considered as mostly it is found to be null.
2.Null countries are not counted.

A Glimpse of twitter tweet Json format.


```
{
   "created_at": "Sun Apr 19 14:57:36 +0000 2020",
   "id": 1251887680134230016,
   "id_str": "1251887680134230016",
   "text": "RT @oflynnsocial: The word \"skipped\" is so loaded here. WHO did not
declare Covid a global pandemic till March 11 - Boris Johnson was alrea…",
   "source": "<a href=\"http://twitter.com/download/android\" rel=\"nofollow\">Twitter
for Android<\/a>",
   "truncated": false,
   "in_reply_to_status_id": null,
   "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": {},
   "geo": null,
   "coordinates": null,
   "place": null,
   "contributors": null,
```

```
   "retweeted_status": {⊞},
   "quoted_status_id": 1251563504118771712,
   "quoted_status_id_str": "1251563504118771712",
   "quoted_status": {⊞},
   "quoted_status_permalink": {⊞},
   "is_quote_status": true,
   "quote_count": 0,
   "reply_count": 0,
   "retweet_count": 0,
   "favorite_count": 0,
   "entities": {⊞},
   "favorited": false,
   "retweeted": false,
   "filter_level": "low",
   "lang": "en",
   "timestamp_ms": "1587308256549"
}
```


Information about user's


```
"user": {⊟
      "id": 61482552,
      "id_str": "61482552",
      "name": "TheNewMadmax",
      "screen_name": "rrajendram",
      "location": "Hemel Hempstead, UK",
      "url": null,
      "description": null,
      "translator_type": "none",
      "protected": false,
      "verified": false,
      "followers_count": 52,
      "friends_count": 98,
      "listed_count": 1,
      "favourites_count": 17741,
      "statuses_count": 401,
      "created_at": "Thu Jul 30 13:08:55 +0000 2009",
      "utc_offset": null,
      "time_zone": null,
      "geo_enabled": true,
      "lang": null,
      "contributors_enabled": false,
      "is_translator": false,
      "profile_background_color": "C0DEED",
      "profile_background_image_url":
"http://abs.twimg.com/images/themes/theme1/bg.png",
      "profile_background_image_url_https":
"https://abs.twimg.com/images/themes/theme1/bg.png",
      "profile_background_tile": false,
```

```
      "profile_link_color": "1DA1F2",
      "profile_sidebar_border_color": "C0DEED",
      "profile_sidebar_fill_color": "DDEEF6",
      "profile_text_color": "333333",
      "profile_use_background_image": true,
      "profile_image_url":
"http://pbs.twimg.com/profile_images/843988601113317376/k1bC9Obx_normal.jpg",
      "profile_image_url_https":
"https://pbs.twimg.com/profile_images/843988601113317376/k1bC9Obx_normal.jpg",
      "default_profile": true,
      "default_profile_image": false,
      "following": null,
      "follow_request_sent": null,
      "notifications": null
   }
```


Why Kafka to store the data.

We have chosen the kafka to store the data (Reasons , other options).

1.Since the given problem is streaming the data , kafka is a suitable technology.

2.We can store data into topics and proper log and offsets are maintained by default.

3.Kafka has a built-in framework called Kafka Connect for writing sources and sinks that either continuously ingest data in Kafka or into external systems (here we have used mongoDB as an external database to store our results).

4.It can be deployed at cluster level.

5.It can be connected easily to other technologies (which process streaming data)

6.KStream is available , if we want to process the streaming data.

Other options to store data

1.Data could be stored in simple file .json format too, but to process the file using spark might require loading a big file into ram which will require huge ram resources.It can not be said as a streaming environment or real time as data is first collected in files and then consumed.

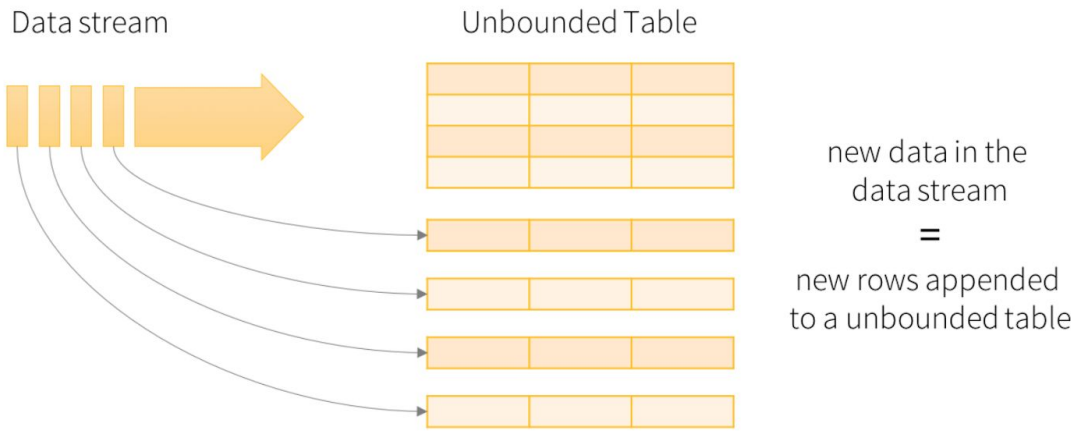
2.Flume: This can be used but Kafka has some advantages over Flume. One of the drawbacks of the Flume data streaming tool is that if it fails, data will be lost and hence there won't be any events replication.

Spark for Processing(Choosing Spark over KStreams)

1.KStreams can ingest data from Kafka topics and store the output in kafka topics only.While on the other hand Spark can ingest data from multiple technologies(Kafka topics also) and can output the result into the mongoDB.

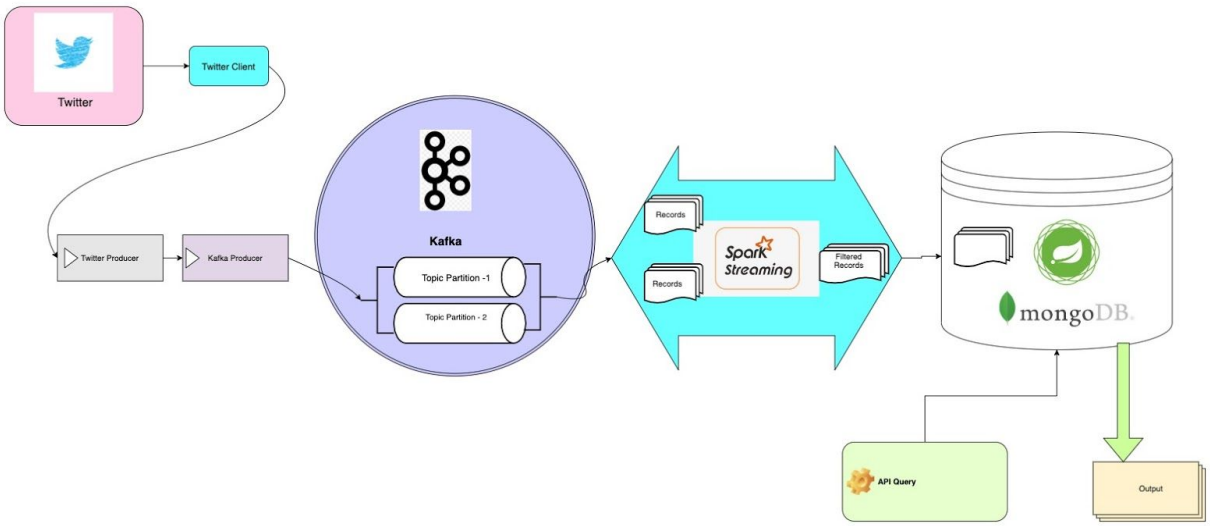
Spark Streaming comes in two flavours
1.DStreams : It's based on the idea of discretized streams or DStreams. Each DStream is represented as a sequence of RDDs, so it's easy to use if you're coming from low-level RDD-backed batch workloads.

2.DataFrames/ Structured Streaming : Spark Structured Streaming was introduced in Spark 2.0 (and became stable in 2.2) as an extension built on top of Spark SQL. Because of that, it takes advantage of Spark SQL code and memory optimizations. Structured Streaming also gives very powerful abstractions like Dataset/DataFrame APIs as well as SQL.



Data stream as an unbounded table

Architecture



Architecture of the Streaming pipeLine.

Below are some useful links to connect Spark to kafka and use structured Streaming
https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html

API query

Part 1
Building API

Why Flask REST API:

1. Simpler Development - Flask's simplicity allows developers to create small applications.
2. Fast development
3. Open source
4. Modularity: Modular code provides a huge number of benefits. With Flask, you have the ability to create multiple Flask applications or servers, distributed across a large network of servers, each with specific purposes. This creates more efficiency, better testability, and better performance.

Other options for REST API:

1. **Bottle:** The advantage of Flask over bottle is that it is easy to use, lightweight, open source. There are only three tools that can be integrated with Bottle but we can integrate more tools in Flask.

2. **Play:** It is built on Akka so it is not easy to handle whereas Flask is easy to handle and builds small applications. Only one tool can be integrated with Play but we can integrate many tools in Flask. Dynamic typing is easy in Flask but not in Play. Flask is more user friendly than Play.Flask is easier to config than Play.

3. **Spring:** Flask supports Python whereas Spring supports Java. Flask is fast as compared to Spring, so application startup time in Flask is very less. Flask supports cross platform whereas Spring supports JVM compatible like Android. Flask is easier to config than Spring. Flask is like writing less in Python but doing more things.

Part 2
Query using **Postman / (Web Browser also)**

The last phase is to query API from the Postman and get the output.