CSP-554-BIG DATA TECHNOLOGIES

PROJECT REPORT

BIG DATA PROCESSING PIPELINE

ILLINOIS INSTITUTE OF TECHNOLOGY

MAY 12, 2021

YASH PATEL	**A20451170**
HARSH VORA	**A20445400**
VISHNU BHARATH	**A20465596**
VARUN VEERLA	**A20458191**

PROF. JOSEPH ROSEN

1. Introduction & Background Information

Data is becoming extremely popular in the industry due to its importance in market analysis, making decisions related to trends, and developing ML and DL models. Therefore, all the operations from data generation to data reception need to be appropriately performed. Any small mistake in mishandling the data can lead to huge problems. In such a scenario, the need for effective and smooth transfer of data is high.

Data pipeline can be described as a complex chain of activities connected in series that manipulates and processes the data. The output of one component acts as input to another and allows a smooth and automated data flow. A data pipeline starts at the element that generates the data and ends at the component that receives the processed data. Although pipelining may not decrease latency, it can surely improve the throughput of the system. Data pipelines can process different data types such as continuous data, batch data, and intermittent data. Data pipelines can also be used to eliminate errors and increase end-to-end throughput. Hence, data pipelines are an absolute necessity for all data-driven companies.

In this study, we are going to show an implementation of how a data pipeline works. We use different software like apache-zookeeper, firebase, Kafka, and spark and utilize Twitter data to create a data pipeline. The main objective of this project is to use big data technologies to build a data processing pipeline that collects data from a source, transforms it, and maintains it in an optimized format such that we can extract insights from real-time data.

In the remainder of this study, we will describe each of the components or software used in the literature survey. In Implementation, we will explain and show how the data pipeline is constructed and worked on, and in the Conclusion section, we will summarize our work.

2. Literature Survey

2.1. Project Goals:

- Ingest data using Twitter's streaming API.
- Capture data using Apache Kafka.
- Process streaming data using Apache Spark.
- Store these processed data using Google Firebase.
- Visualize these processed data using Node.js server and HTML web-page client.

2.2. Twitter:

Twitter is a social media and news platform that allows users to send and receive short messages known as tweets. Tweeting is the method of posting short tweets to people who follow you on Twitter, hoping that your comments will be helpful and interesting to someone in your circle.

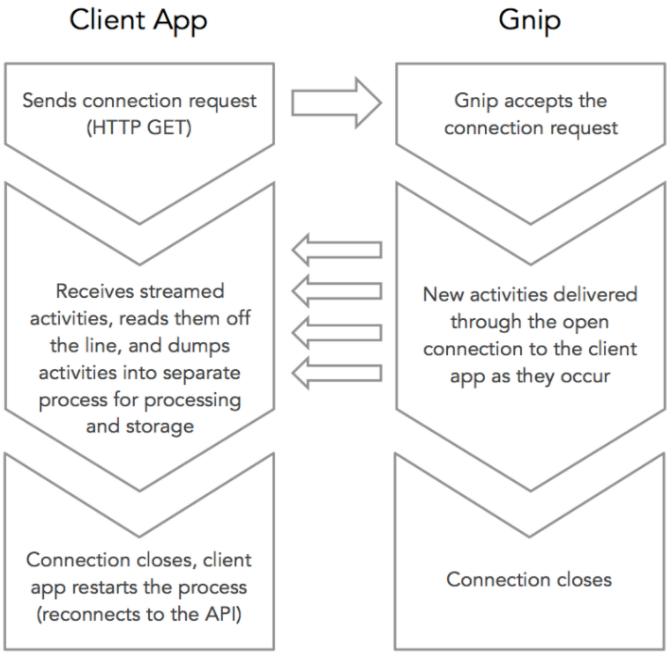
The most significant benefit of Twitter is the ease with which it can be searched. You'll be able to follow hundreds of fascinating Twitter users and read their content efficiently, which is beneficial in our attention-deficit society.

Thousands of people use Twitter to advertise their staffing agencies, consulting firms, and shopping stores, and it works. The strong influence of Twitter on businesses motivates the analysis of Twitter's real-time data to extract actionable insights.

2.3. Filtered Stream API:

Developers may use the filtered stream endpoint community to filter the real-time stream of public Tweets. The functionality of this endpoint category involves several endpoints that allow you to build and maintain rules and apply specific rules to filter a stream of real-time Tweets and return matching public Tweets. This endpoint community enables users to listen in real-time for relevant issues and activities, track the discussion surrounding competitions, learn how patterns evolve in real-time, and even more.

The Streaming HTTP protocol is used by PowerTrack, Volume (e.g., Decahose, Firehose), and Replay streams to deliver data over an open, streaming API connection. Instead of providing data in batches through repetitive requests from your client app, as the REST API would, a single connection is established between your app and the API. New results are transmitted via that connection anytime new matches occur. As a consequence, a low-latency delivery mechanism with high throughput is developed.

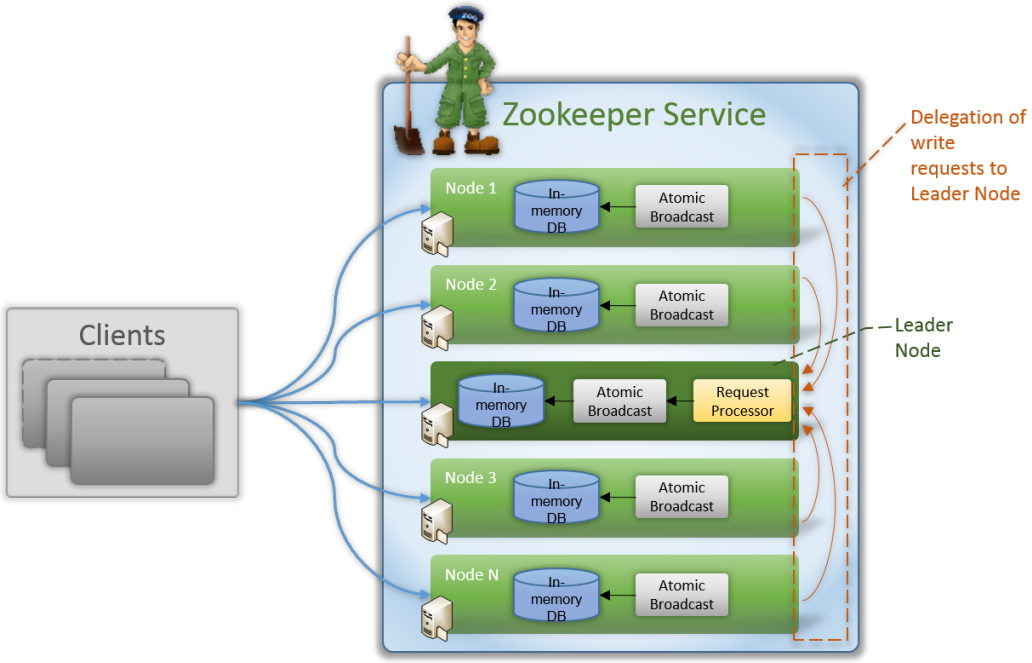


For these streams, the ultimate relationship between your app and Twitter's API is as follows:

2.4. Apache Zookeeper:

Apache Zookeeper is a distributed data storage that is highly concurrent and asynchronous due to network communication. It was developed to provide an open-source high reliable distributed coordination. Zookeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these services are used in some form by distributed applications. Its architecture

supports high availability through redundant services. Zookeeper's prime features are Reliable system, Simple Architecture, Fast processing, and Scalability.

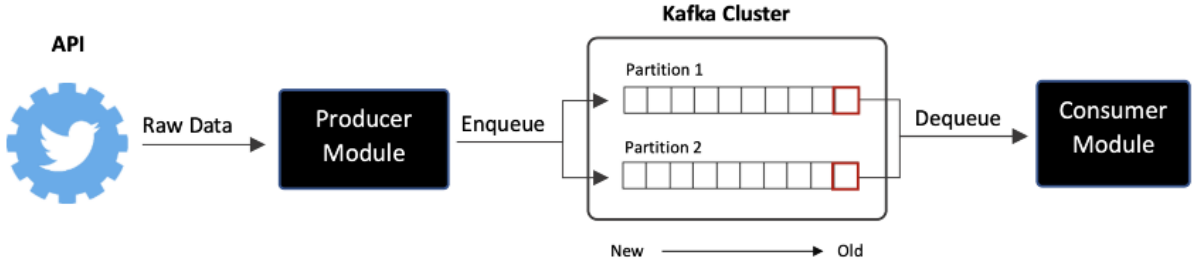


2.5. Apache Kafka:

Apache Kafka is a real-time data management platform that uses event streaming. It offers a unified, high-throughput, low-latency interface for dealing with massive volumes of real-time data streams. Zookeeper serves as the foundation for Apache Kafka. It monitors the health of brokers and serves as a coordinator for Apache brokers and customers.

Initially, Kafka producers deliver the message to a topic, and Kafka brokers assist in storing the messages in partitions of that topic. When the Apache User subscribes to a topic, Kafka offers the current offset of that topic to the consumer and saves the offset into Zookeeper, which aids in monitoring the previous offset. Messages from the topic are sent to the User before it ceases.

Fault resistance is the most important benefit of using Apache Kafka. It comes with two modules, the first of which is the Producer, which collects data from Twitter. The data is then saved as logs in the queue without being processed, and a module called Consumer reads and processes the logs.



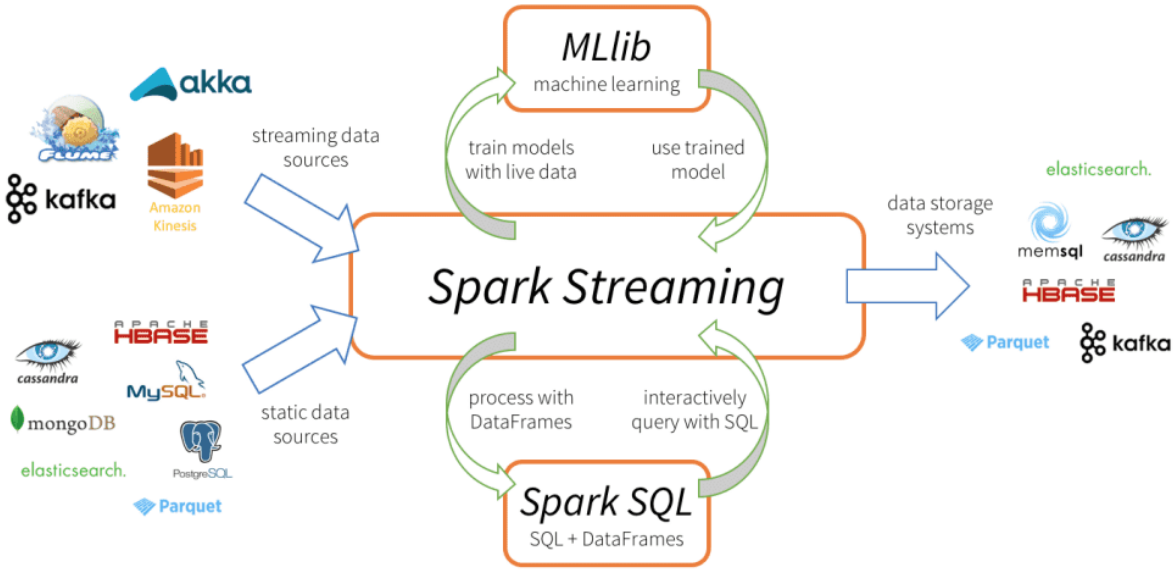
2.6. Apache Spark:

Apache Spark is a Scalable fault-tolerant stream processing that supports both batch and streaming workload. Spark Streaming is an extension of the core Spark API that allows real-time processing of HTTPcrucial data from various sources like Apache Kafka. This transferred data is pushed into file systems, databases, and live dashboards. A Discretized Stream, or DStream, is the primary abstraction, which defines a stream of data separated into small batches. RDDs, Spark's central data abstraction, are the foundation for DStreams. These enable Spark streaming to work in conjunction with other spark elements such as MLlib and Spark SQL effectively.

Since the data is flowing in as a stream, it makes sense to choose a streaming product like Apache Spark Streaming to process it. If you want to write data to disk, this holds data in memory. Streaming info, on the other hand, has meaning when it is live.

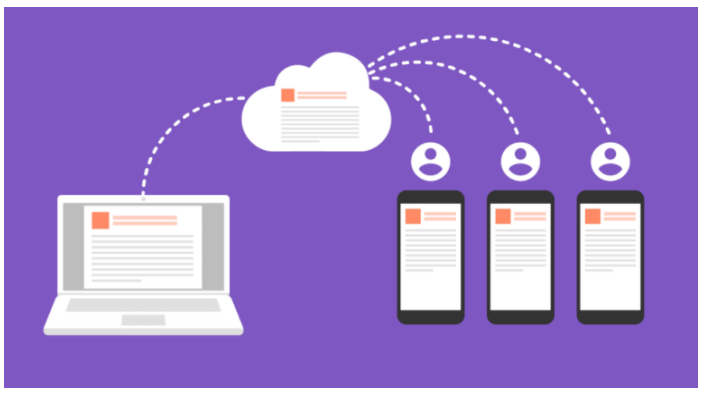
The main benefits of using Apache Spark is:

- Fast recovery from failures and stragglers
- Better load balancing and resource usage

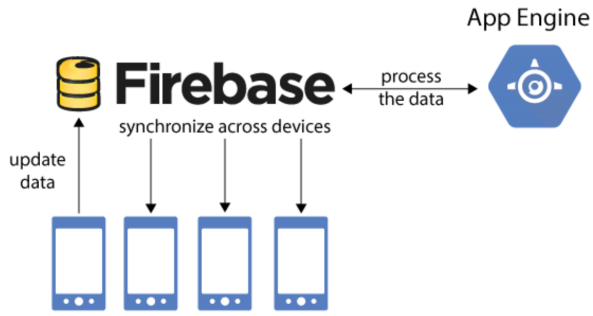


2.7. Firebase Real-time Database:

The Firebase Realtime Database is a NoSQL cloud database that allows you to store, sync and query data in real-time at global scale. The Firebase Realtime Platform is a cloud-based database system. Data is stored in JSON format.



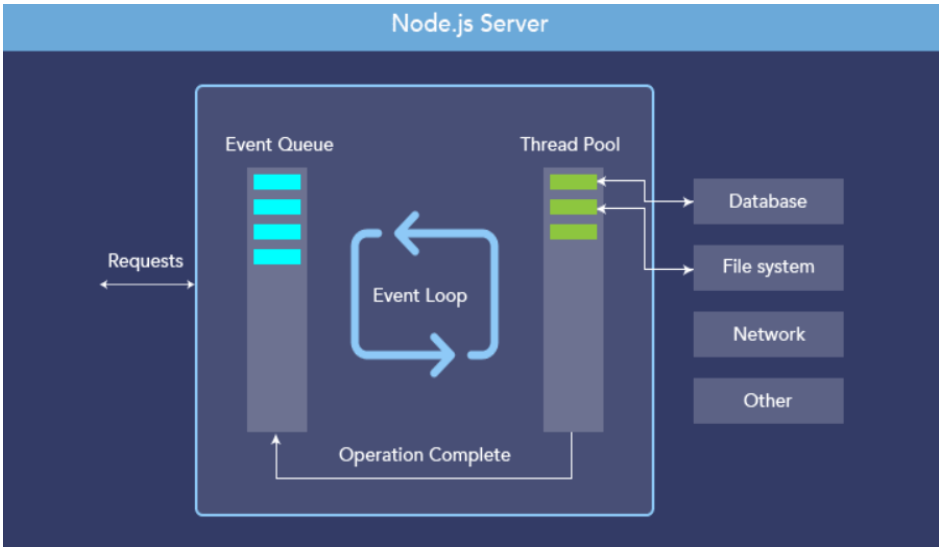
When we use the iOS, Android, and JavaScript SDKs to build cross-platform apps, all of our clients share a single Realtime Database instance and receive automated updates on the most recent results.



Data is synced in real-time through all connected clients. Realtime events continue to fire, giving the end-user a responsive experience.

2.8. Node.js:

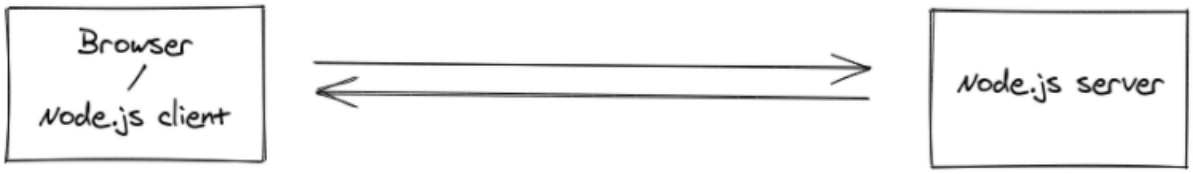
Node.js is a back-end JavaScript runtime environment that is open-source, cross-platform, and runs on the V8 engine. It executes JavaScript code outside of a web browser. Node.js allows developers to use JavaScript to create command-line tools and server-side scripting, which involves running scripts on the server to generate complex web page content before the page is submitted to the user's web browser. As a result, Node.js reflects a "JavaScript everywhere" paradigm, bringing web application development together around a single programming language rather than separate languages for server-side and client-side scripts.



Node.js is a scalable network server runtime that is configured as an asynchronous event-driven JavaScript framework. Many connections can be addressed at the same time. Thread-based networking is inefficient and complex to implement. Furthermore, since there are no locks, Node.js users are not concerned about deadlocking the operation.

2.9. Socket.IO:

Socket.IO is a Javascript library that enables real-time, bidirectional communication between web clients and servers. It consists of 2 things: a Node.js server and a javascript client library for the browser.



The client will try to create a WebSocket connection if possible and fall back on HTTP long polling. So Socket.IO is not a WebSocket Implementation, Although it uses WebSocket as transport when possible. It also adds additional metadata to each packet. You can consider Socket.IO as a “slight” wrapper around the WebSocket API.

Apart from that, Socket.IO provides different features such as Reliability, Automatic Reconnection, Packet Buffering, Multiplexing, Binary Support, and a Simple and Convenient API.

2.10. Comparison Between Apache Kafka And Amazon Kinesis:

Amazon Kinesis makes it simple to capture, process, and analyze real-time, streaming data, allowing you to gain timely insights and react quickly to new data. Amazon Kinesis provides key features for cost-effectively processing streaming data at any size, as well as the freedom to choose the tools that best serve your application's needs. You can absorb real-time data with Amazon Kinesis.

Kinesis Data Streams has the benefits like it’s fully managed, scalable, real-time and elasticity. Kinesis is a completely managed streaming platform that runs your applications without needing you to handle any infrastructure. Amazon kinesis can handle any amount of streaming data and process data from hundreds of thousands of sources with very low latencies. Amazon Kinesis enables you to ingest, buffer, and process streaming data in real-time, so you can derive insights in seconds or minutes instead of hours or days. Scale the stream up or down, so data never loses before they expire.

For real-time data collection streams and big data real-time analytics, Kafka has the following features. Kafka handles a large number of real-time data sources. High throughput is supported for both publishing and subscribing. Kafka has distributed systems that are highly scaled with no downtime in all four dimensions: manufacturers, processors, users, and connectors. Kafka has the fault tolerant property with the masters and databases with zero downtime and zero data loss. Kafka replicates the messages across the cluster to support multiple subscribers.

For real-time data streaming services, both Apache Kafka and AWS Kinesis Data Streams are viable options. Apache Kafka should be your preference if you need to hold

messages for more than 7 days with no limit on message size per blob. Apache Kafka, on the other hand, necessitates additional work to set up, administer, and help. If your company needs Apache Kafka experts and/or human resources, a full-featured solution is a way to go.

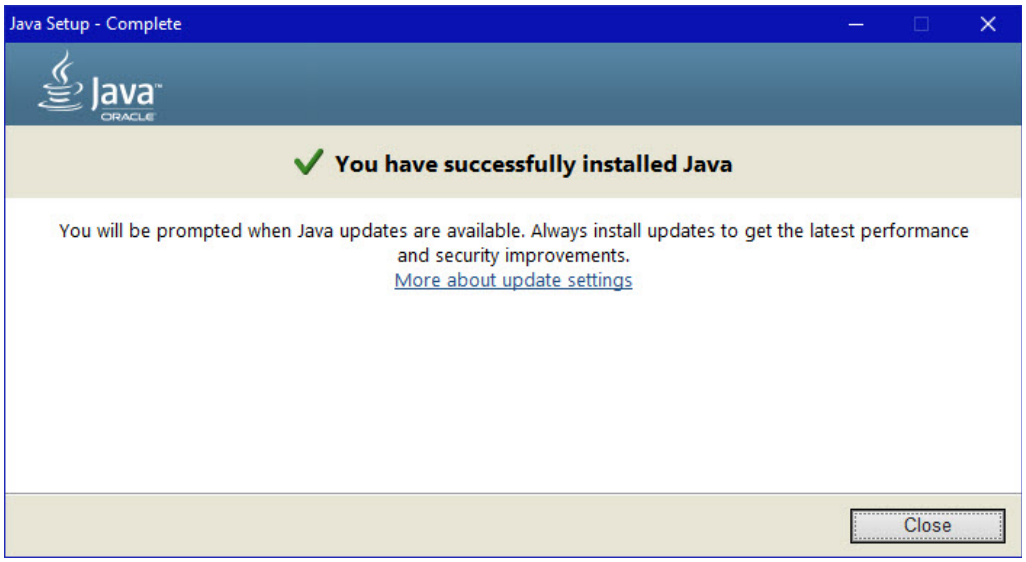
3. Implementation

The implementation for our project includes the installation of the software used. We have performed this project on our local machines, so there are no cloud features used. After starting the servers, we run our project, and the output is visualized on a global map. The visualized result tells us how many tweets were tweeted since the instance started running.

3.1. Software Installation:

3.1.1. JDK Installation:

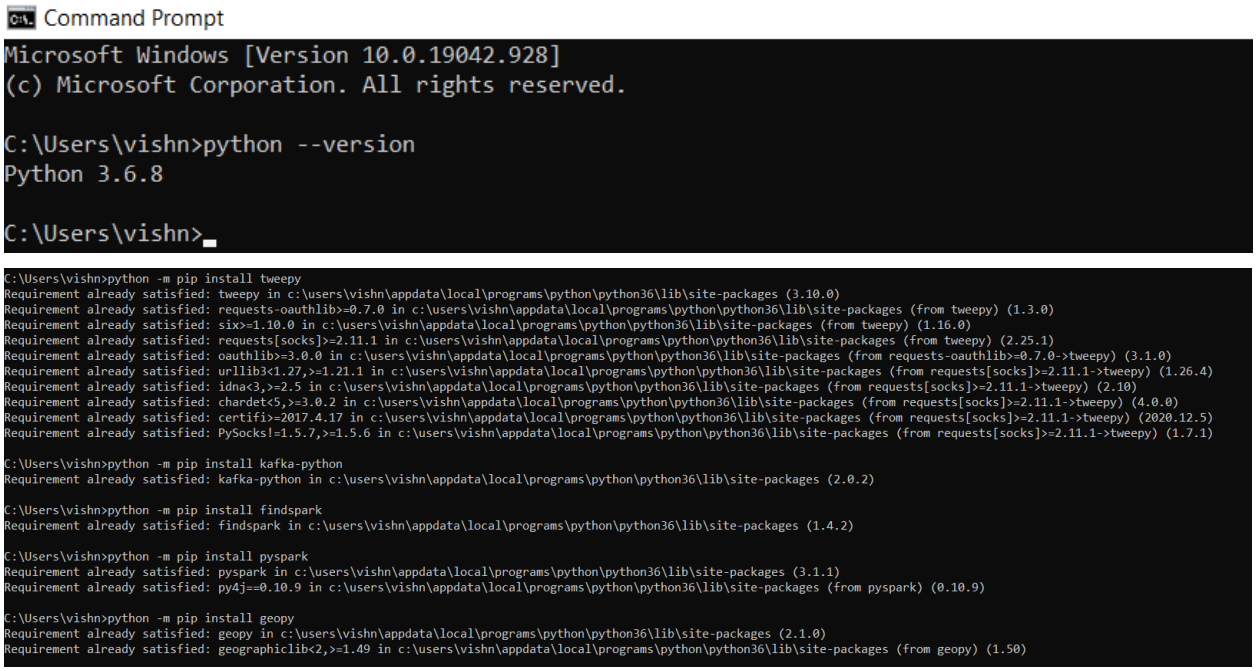
1. Go to https://www.oracle.com/java/technologies/javase-jre8-downloads.html and download the java runtime environment for your operating system.
2. Different versions are available.
3. You will need to create a new account or log in to your existing account to download the file.
4. After downloading the executable file, run the executable file and follow the prompts to install Java runtime environment.
5. If successfully installed, you will get the following.



3.1.2. Python Installation:

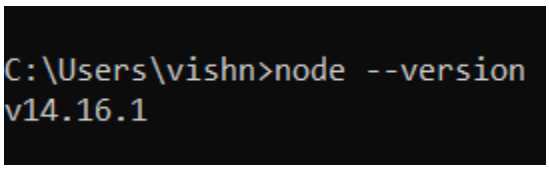
1. Download and python 3.6 installer from https://www.python.org/downloads/
2. Execute the python installer and continue with the prompts and install python.
3. During the installation, select the option where it asks you to set the Path

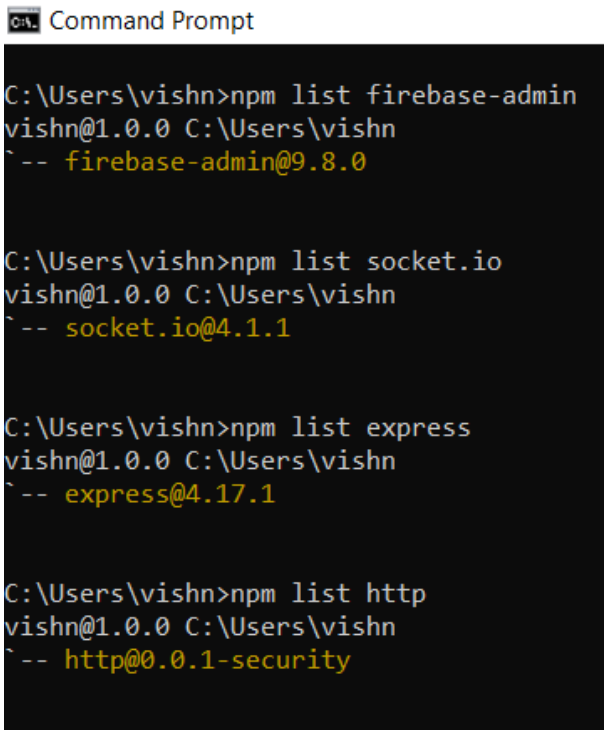
4. Open a command prompt and enter the following commands
    a. pip install tweepy
    b. pip install kafka-python
    c. pip install findspark
    d. pip install pyspark
    e. pip install geopy
    f. pip install firebase admin
5. Additionally, install pycharm from https://www.jetbrains.com/pycharm/.



3.1.3. Node.js Installation:

1. Download and install node js from https://nodejs.org/en/
2. Download and install atom; an open-source text and source code editor. Atom can be download from https://atom.io/
3. After successfully installing atom IDE, open atom and go to file->settings->install and download the required atom packages.
    a. platformio-ide-terminal
    b. Script





3.2. Setting Up The Server:

3.2.1. Apache Zookeeper:

1. Copy zookeeper from project/download directory or download zookeeper from https://zookeeper.apache.org/releases.html
2. Extract apache-zookeeper-3.7.0-bin.tar and open the folder and extract apache-zookeeper-3.7.0-bin folder to project/server directory.
3. Navigate to project/config directory and copy zoo.cfg
4. Replace zoo_sample.cfg file in project/server/apache-zookeeper-3.7.0-bin/conf/ with zoo.cfg file in project/config directory.

3.2.2. Apache Kafka:

1. Copy Kafka from project/download directory or download Kafka from https://kafka.apache.org/downloads.html
2. Extract kafka_2.13-2.7.0 to project/server directory.
3. Replace server.properties file in project/server/kafka_2.13-2.7.0/config/ with server.properties file in project/config directory.

3.2.3. Apache Spark:

1. Copy spark from project/download directory or download spark-2.4.7-bin-hadoop2.7 from https://spark.apache.org/downloads.html

2. Copy spark-streaming-kafka-0-8-assembly.jar from project/download directory or download and copy spark-streaming-kafka-0-8-assembly to project/server/spark/jars from https://mvnrepository.com/artifact/org.apache.spark/spark-streaming-kafka-0-8-assembly
3. This project needs Hadoop; hence we need to set up Hadoop.
4. Create hadoop-2.7/bin directory at project/server/ path (note: match spark's hadoop version)
5. Copy winutils.exe from project/download or download and copy winutils.exe (note: match spark's Hadoop version) in project/server/Hadoop-2.7/bin directory from https://github.com/cdarlint/winutils

3.2.4. Node.js:

1. Go to project/server/node.js path in CMD or open project/server/node.js directory in atom editor tool.
2. Download required node.js libraries using CMD or atom terminal using any of these two methods:
    a. Using pre-setup file(automatic and recommended)
        i. npm install.
    b. Manual setup
        i. npm init
            1. Add {"start": "node index.js"} key-value pair after init.
        ii. npm install express.
        iii. npm install http.
        iv. npm install firebase-admin.
        v. npm install socket.io.

3.3. Setting Up The Twitter:

1. For running this project, a Twitter developer account is needed, which needs to be approved by Twitter.
2. The Twitter developer account can be obtained by applying for a Twitter developer account.
3. After approval, log in to your Twitter developer account and create a project and associated app in the developer portal.
4. Navigate to the applications keys and tokens page, and safely save your app's access token, access token secret, consumer key, consumer secret, and bearer token.

3.4. Setting Up The Firebase Real-Time Database:

1. Log in to your google account, or create a new account if you are new to google mail.
2. Go to the firebase console using https://console.firebase.google.com/
3. Create a new project.

4. Go to settings->service accounts and generate a new private key.
5. Safely store the new private key in the project/server/firebase/service-account/admin-sdk directory.
6. Go to build-> real-time database in firebase console and create a new database.
    a. Choose start in test mode and security rules of setup database and enable start in test mode.
7. If you want no security, then build-> real-time database->rules and edit as below.
    a. {
        "rules": {
            ".read": true,
            ".write": true,
        }
    }
8. Note the database_url, which is available on build->realtime database->data (e.g. https://databaseName.firebaseio.com)

3.5. Setting Up The WebPage Visualization:

1. Log in to your google account and go to the google cloud platform using https://console.cloud.google.com/.
2. Go to Home->Dashboard and create a new project or open an old project created during the setting up firebase real-time database.
3. Go to APIs and Services and enable Maps javascript API in ENABLE APIS AND SERVICES.
4. Go to APIs and Services->Credentials and CREATE CREDENTIALS for API key.
5. RESTRICT KEY by API restrictions and then select and add Maps JavaScript API and then save.
6. Copy the API key and set it as a value of the 'mapsApiKey' key in google.chart.load method's dictionary argument in project/server/node.js/public/index.js file.

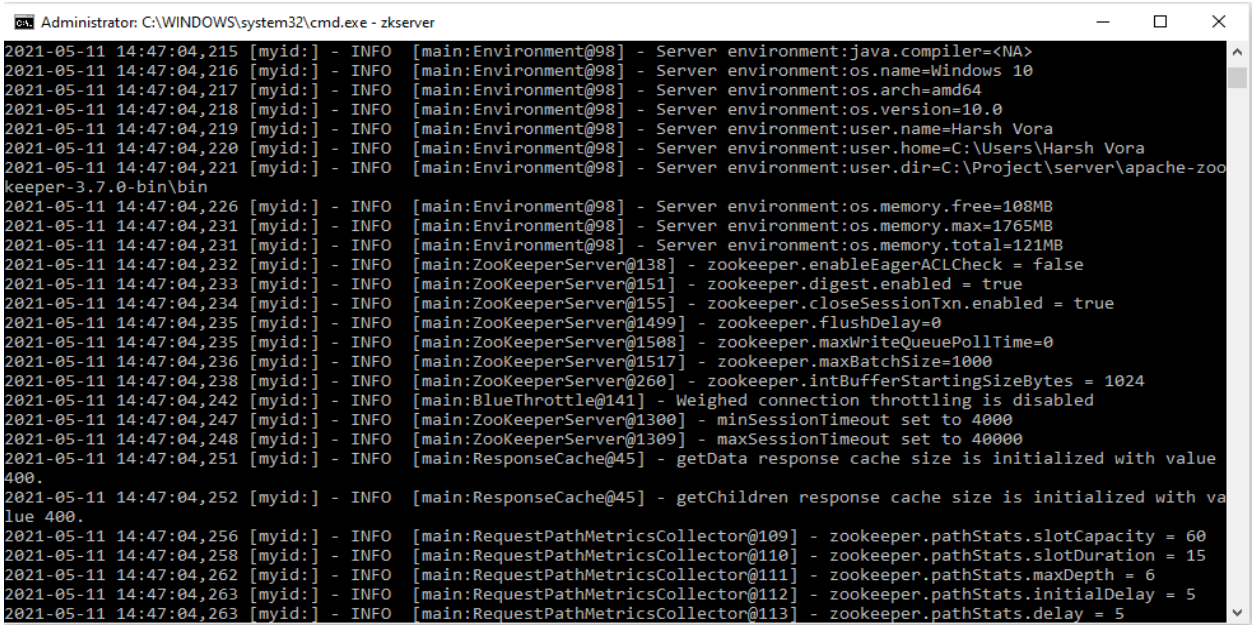
3.6. Setting Up The Environment Variables:

1. Open file setup_env_var.bat in a text editor
2. Change the APP_NAME variable to the name that you want.
3. Change the JAVA_HOME variable according to your JRE path and version.
4. Change TWITTER_ACCESS_TOKEN, TWITTER_ACCESS_TOKEN_SECRET, TWITTER_CONSUMER_KEY, TWITTER_CONSUMER_SECRET, and also TWITTER_BEARER_TOKEN according to your Twitter app.
5. Change TWITTER_FILTER_KEYWORD_LIST that you want to filter in the Twitter stream and write keywords separated by ',' and no extra spaces.
6. Do not change the ZOOKEEPER_HOST_NAME, KAFKA_HOST_NAME, ZOOKEEPER_PORT, and KAFKA_PORT variables.

7. Change KAFKA_TOPIC_NAME to the name that you want.
8. Change the SPARK_HOME variable according to your spark version.
9. Do not change the HADOOP_HOME variable.
10. Change the NODE_JS_WEB_SERVER_PORT variable according to your desired node.js web port you want to listen to.
11. Do not change DATA_DIR_PATH and SPARK_DATA_DIR_PATH variables.
12. Change the GOOGLE_APPLICATION_CREDENTIALS variable according to the name of your private key JSON file.
13. Change FIREBASE_REALTIME_DATABASE_URL according to your database URL, noted in the last step of setting up a firebase real-time database.
14. Do not change the SPARK_DATA_FIREBASE_REALTIME_DATABASE_PATH variable.
15. Do not change the PATH variable.

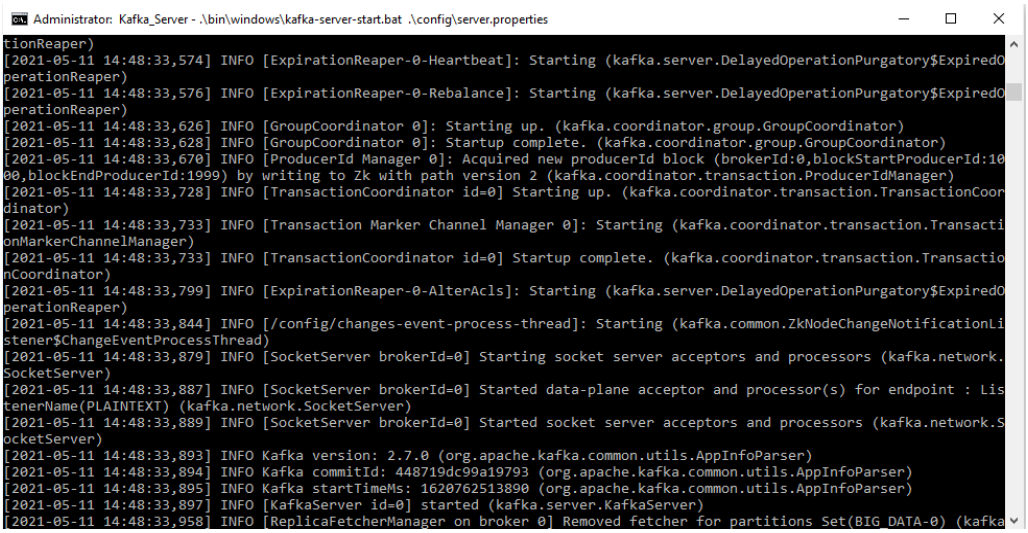
4. Execution Instructions For The Project

1. Open CMD, make sure your current directory is your project directory, and name this CMD by running "title env_var."
2. Run "setup_env_var.bat" in env_var CMD.
3. Open a new CMD with running "start" in env_var CMD and name this CMD by running "title ZooKeeper" in the newly started CMD.
4. In ZooKeeper CMD, go to the project/server/zookeeper/bin directory and run "zkserver" to start zookeeper. Following is the output that you get after executing the previous steps.

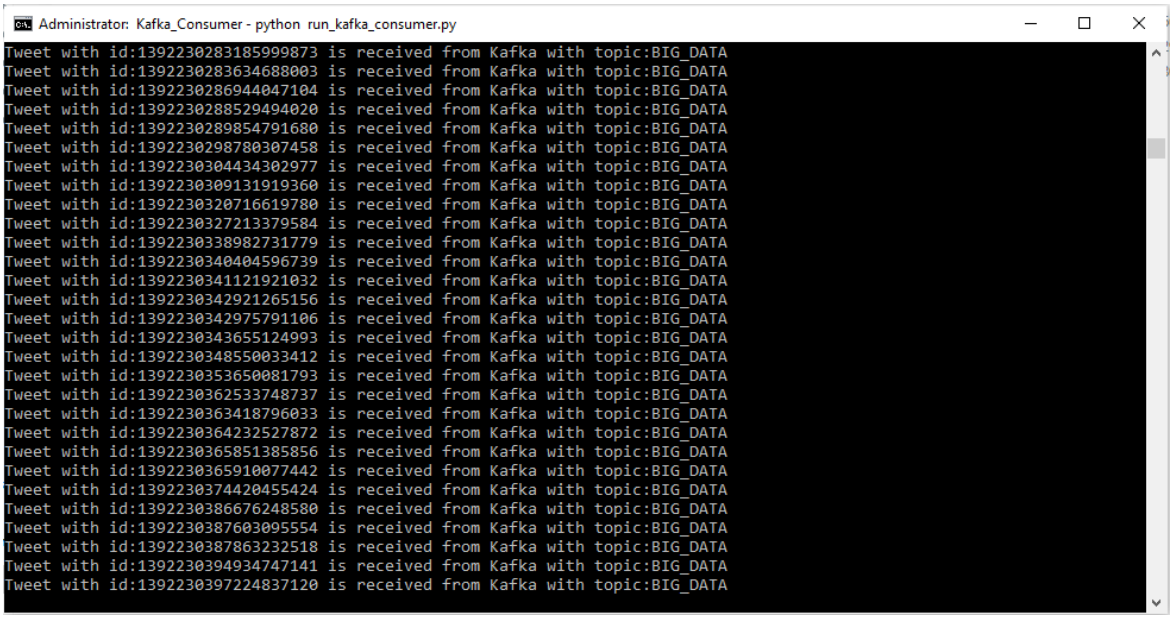


5. Open a new CMD with running "start" in env_var CMD and name this CMD by running "title Kafka_Server" in the newly started CMD.

6. In Kafka_Server CMD, go to the project/server/kafka directory and run ".\bin\windows\kafka-server-start.bat .\config\server.properties" to start kafka.



7. Open a new CMD with running "start" in env_var CMD and name this CMD by running "title Kafka_Topic" in the newly started CMD.
8. In Kafka_Topic CMD, go to project/server/Kafka/bin/windows directory and run "kafka-topics.bat --create --zookeeper %ZOOKEEPER_HOST_NAME%:%ZOOKEEPER_PORT% --replication-factor 1 --partitions 1 --topic %KAFKA_TOPIC_NAME%" to create a kafka topic.
9. Open a new CMD with running "start" in env_var CMD and name this CMD by running "title Kafka_Consumer" in the newly started CMD.
10. In Kafka_Consumer CMD, go to the project/src directory and run "python run_kafka_consumer.py" to start Kafka consumer.

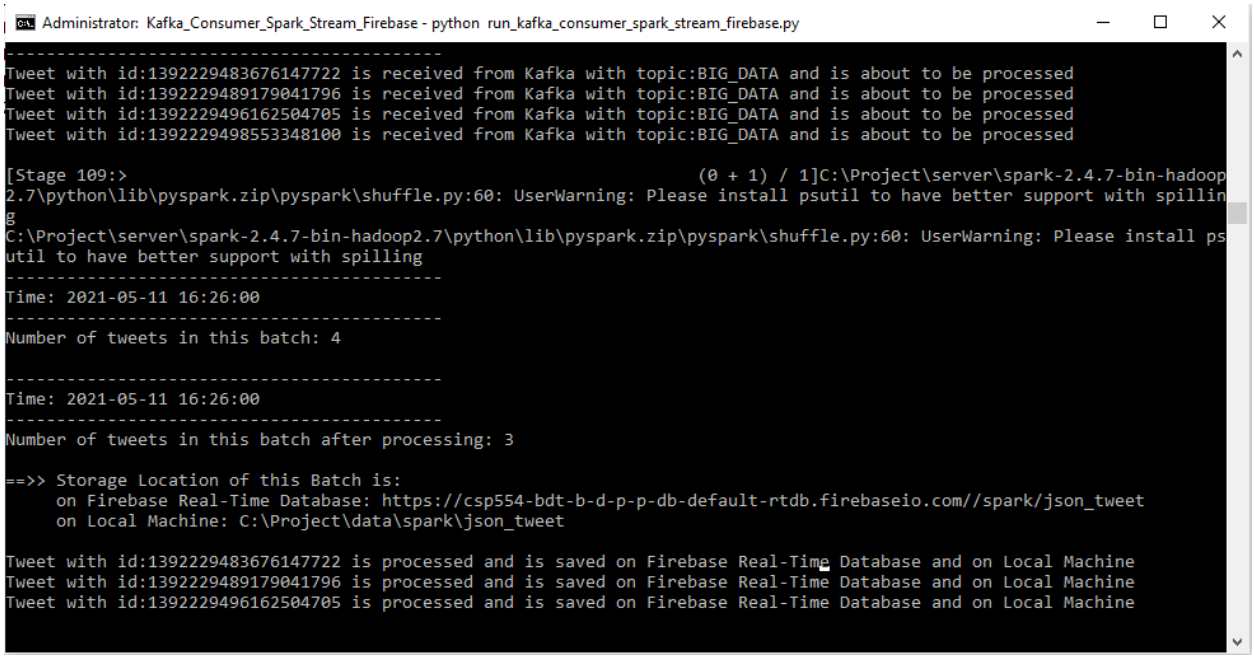


run_kafka_consumer.py:

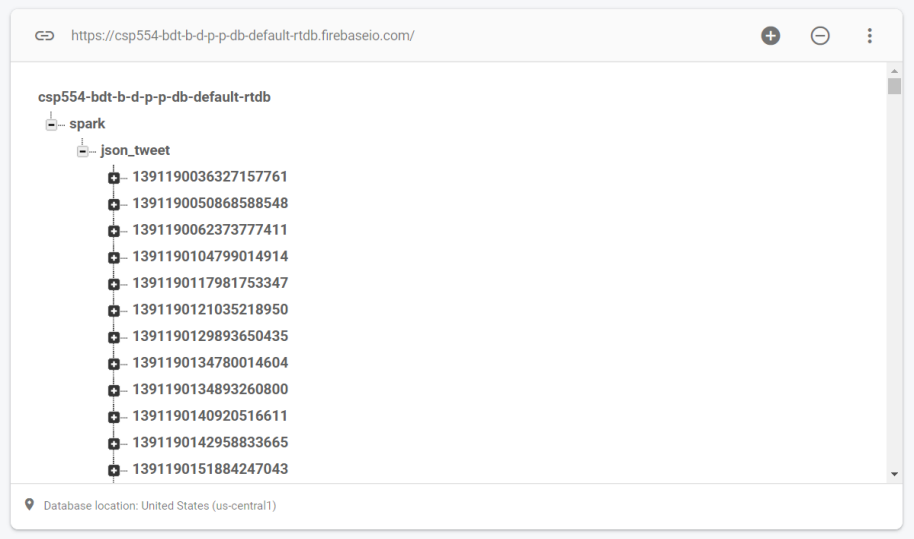
```python
class KafkaTwitterConsumer(object):

    def __init__(self):
        self.KAFKA_HOST_NAME = os.environ.get('KAFKA_HOST_NAME')
        self.KAFKA_PORT = os.environ.get('KAFKA_PORT')
        self.KAFKA_TOPIC_NAME = os.environ.get('KAFKA_TOPIC_NAME')

        self.kafka_twitter_consumer = KafkaConsumer(
            self.KAFKA_TOPIC_NAME,
            bootstrap_servers=f'{self.KAFKA_HOST_NAME}:{self.KAFKA_PORT}'
        )

    def consume_tweets(self):
        for record_tweet in self.kafka_twitter_consumer:
            encoded_tweet = record_tweet.value
            decoded_tweet = encoded_tweet.decode(encoding='utf-8')
            json_tweet = json.loads(decoded_tweet)
            print(f'Tweet with id:{str(json_tweet["id"])} is received from Kafka with topic:{self.KAFKA_TOPIC_NAME}')


def main():
    KafkaTwitterConsumer().consume_tweets()


if __name__ == "__main__":
    main()
```

Above code includes kafka consumer which subscribes or consumes produced tweets from zookeeper on registered kafka host name, port, topic name and print tweet id on terminal.

11. Open a new CMD with running "start" in env_var CMD and name this CMD by running "title Kafka_Consumer_Spark_Stream_Firebase" in the newly started CMD.
12. In Kafka_Consumer_Spark_Stream_Firebase CMD, go to project/src directory and run "python run_kafka_consumer_spark_stream_firebase.py" to start kafka consumer, spark stream with the batch process, and firebase data store.



In the above screenshot we can see that after batch processing like mapping, filtering, etc. the number of tweets counted in a batch is reduced from 4 to 3 because not every tweet has the user's location details and that tweet is dropped.



Above screenshot shows firebase real-time database which contains processed tweet data stored as NoSQL format.

Above screenshot shows one of the NoSQL processed tweets data with fields and values.

Run_kafka_consumer_spark_stream_firebase.py

```
schema = StructType(
    [
        StructField('timestamp_ms', StringType(), True),
        StructField('created_at', StringType(), True),
        StructField('id_str', StringType(), True),
        StructField('text', StringType(), True),
        StructField('user', StructType(
            [
                StructField('id_str', StringType(), True),
                StructField('name', StringType(), True),
                StructField('screen_name', StringType(), True),
                StructField('location', StringType(), True),
                StructField('profile_image_url', StringType(), True)
            ]
        ), True)
    ]
)
```

Above code includes schema to filter required fields and to store filtered fields data in NoSQL data frame.

```
class SparkTwitterStreaming(object):

    def __init__(self):
        self.APP_NAME = os.environ.get('APP_NAME')
        self.KAFKA_HOST_NAME = os.environ.get('KAFKA_HOST_NAME')
        self.KAFKA_PORT = os.environ.get('KAFKA_PORT')
        self.KAFKA_TOPIC_NAME = os.environ.get('KAFKA_TOPIC_NAME')

        self.batch_duration = 10

        self.sc = SparkContext(master='local[*]', appName=self.APP_NAME)
        self.sc.setLogLevel('WARN')
        self.s = SparkSession(self.sc)
        self.ssc = StreamingContext(self.sc, batchDuration=self.batch_duration)

    def stream(self, batch_json_tweet_process=None):

        def batch_process(time=None, rdd=None):
            print('--------------------------------------------')
            print('Time: %s' % time)
            print('--------------------------------------------')
            batch_json_tweet_process(rdd=rdd, s=self.s)
            print('')

        kafka_topic_name = self.KAFKA_TOPIC_NAME

        batch_json_tweet = KafkaUtils.createDirectStream(
            self.ssc,
            [kafka_topic_name],
            {'metadata.broker.list': f'{self.KAFKA_HOST_NAME}:{self.KAFKA_PORT}'}
        ).map(
            lambda tweet: json.loads(tweet[1])
        )

        batch_json_tweet.map(lambda json_tweet: f'Tweet with id:{str(json_tweet["id"])} is received from Kafka
        batch_json_tweet.count().map(lambda count: f'Number of tweets in this batch: {str(count)}').pprint()
        batch_json_tweet.foreachRDD(batch_process)

        self.ssc.start()
        self.ssc.awaitTermination()
```

Above code includes kafka utils which creates and starts a direct stream with zookeeper using kafka host name, port and topic name. After getting tweets from the direct stream it

maps tweets with loaded json tweets, prints count of tweets, tweet id in current batch and batch processes of each rdd.

```
def find_coordinated(record):
    try:
        APP_NAME = os.environ.get('APP_NAME')
        geo_locator = Nominatim(user_agent=APP_NAME)
        json_record = json.loads(record)
        location = geo_locator.geocode(json_record['user']['location'])
        location = location.raw
        json_record['user']['place'] = {}
        json_record['user']['place']['coordinates'] = {}
        json_record['user']['place']['coordinates']['lat'] = location['lat']
        json_record['user']['place']['coordinates']['lon'] = location['lon']
        location = geo_locator.reverse((location['lat'], location['lon']))
        location = location.raw
        address = location.get('address', {})
        json_record['user']['place']['country_code'] = address.get('country_code')
        json_record['user']['place']['country'] = address.get('country')
        json_record['user']['place']['state'] = address.get('state')
        json_record['user']['place']['city'] = address.get('city')
        return json_record
    except BaseException as e:
        return json.loads(record)
```

Above code finds coordinates of tweet owner. It converts tweets to json objects. Using the geopy library it finds latitude, longitude, country code, country, state and city of user location and adds these fields to the json objects.

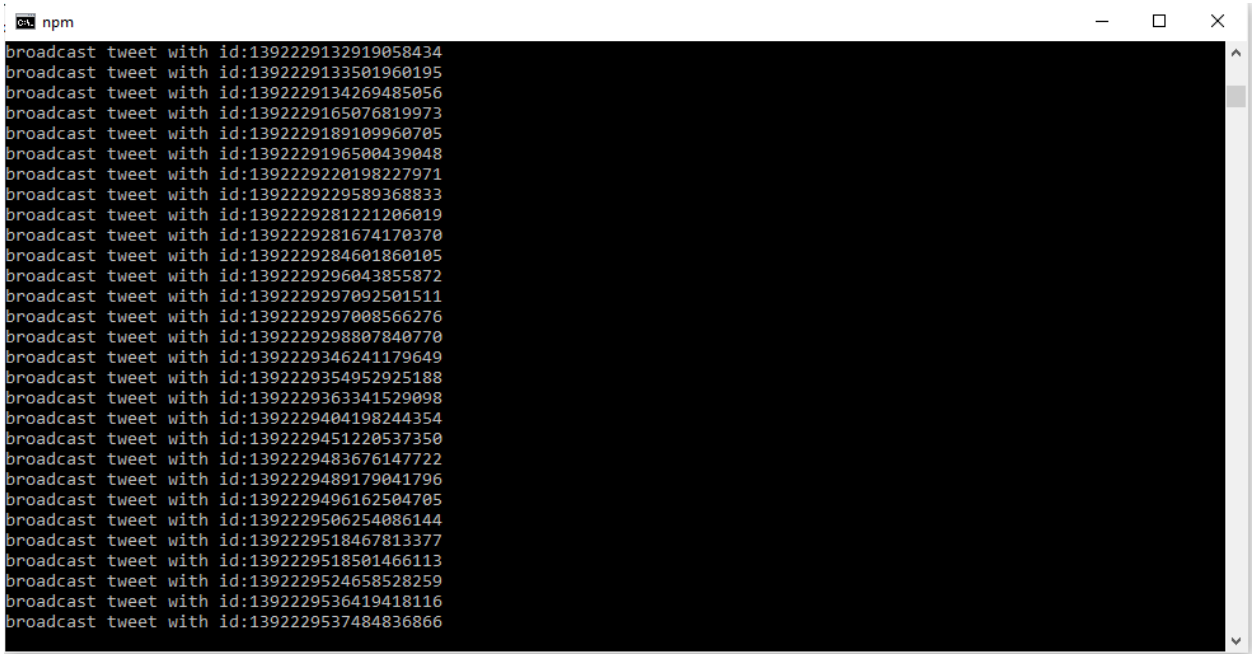
```
df = s.createDataFrame(rdd, schema=schema)
df = df.filter(df.user.isNotNull() & df.user.location.isNotNull())
new_schema = df.schema
new_schema['user'].dataType.add(StructField('place', StructType(
    [
        StructField('coordinates', StructType(
            [
                StructField('lat', StringType(), True),
                StructField('lon', StringType(), True)
            ]
        ), True),
        StructField('country_code', StringType(), True),
        StructField('country', StringType(), True),
        StructField('state', StringType(), True),
        StructField('city', StringType(), True)
    ]
), True))
df = s.createDataFrame(df.toJSON().map(find_coordinated), schema=new_schema)
df = df.filter(
    df.user.place.isNotNull()
    & df.user.place.coordinates.isNotNull()
    & df.user.place.coordinates.lat.isNotNull()
    & df.user.place.coordinates.lon.isNotNull()
    & df.user.place.country_code.isNotNull()
    & df.user.place.country.isNotNull()
)
```

Above code creates NoSQL data frame of tweets' rdds using schema to filter required fields and store them in NoSQL format. In batch processing it creates a new data frame with new schema which includes required location fields for mapped rdd with new location fields in json objects and filters data frame with required fields should not be null condition.

```
df.write.save(
    path=spark_data_dir_local_path,
    format='json',
    mode='append'
)
firebase_app = firebase_admin.initialize_app(
    credential=credentials.Certificate(google_application_credentials),
    options={
        'databaseURL': firebase_realtime_database_url
    },
    name=app_name
)
db_ref = db.reference(
    path=spark_data_firebase_realtime_database_path,
    app=firebase_app,
)
rdd = df.toJSON().map(
    lambda record: json.loads(record)
)
for record in rdd.toLocalIterator():
    db_ref.child(record['id_str']).set(record)
    print(f'Tweet with id:{record["id_str"]} is processed and is saved on Firebase Real-Time Database and on Local Machine')
firebase_admin.delete_app(firebase_app)
```

Above code saves batch processed tweets data frame at registered path on local machine in json format and at registered path on firebase real-time database.

13. Open a new CMD with running "start" in env_var CMD and name this CMD by running "title Node.js_Web_Server" in the newly started CMD.
14. In Node.js_Web_Server CMD, go to project/server/node.js directory and run "npm start" to start the node.js server.



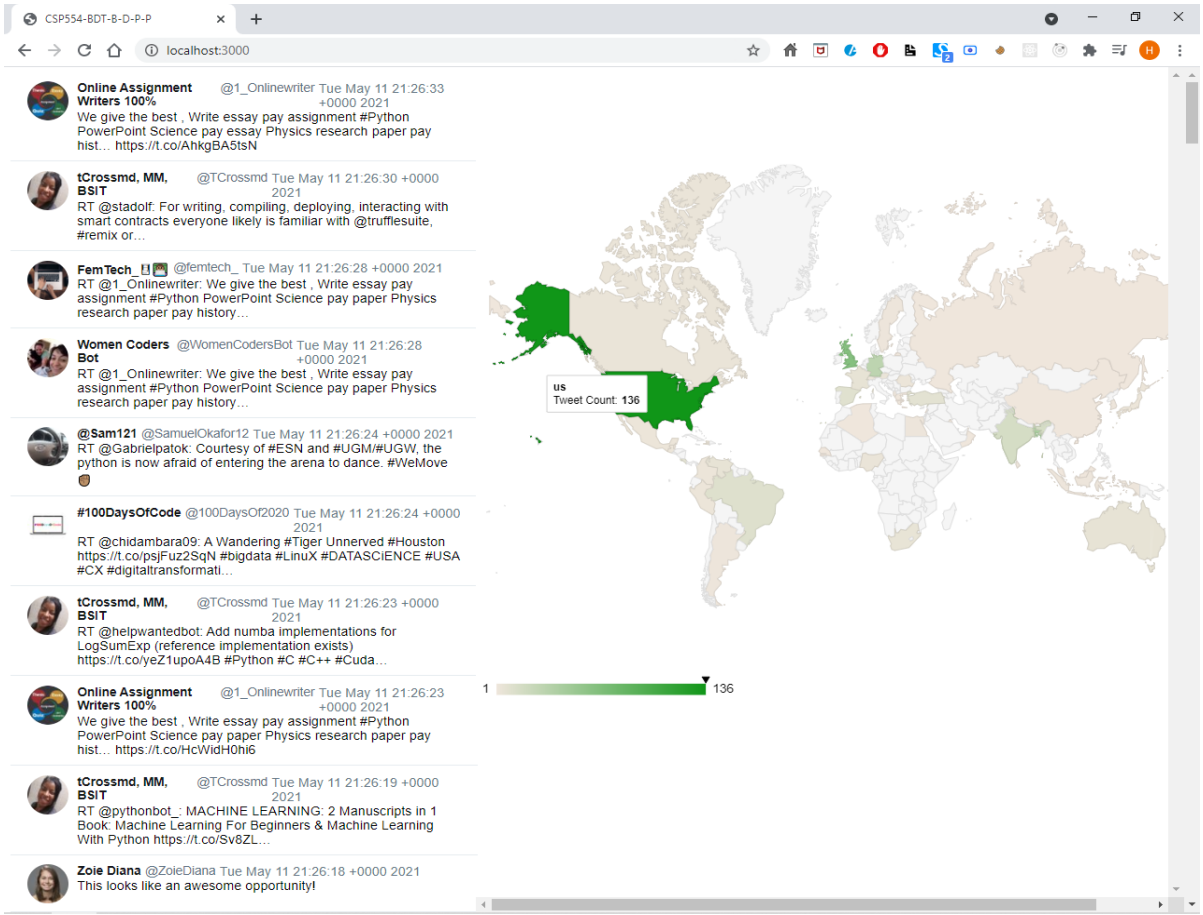
node.js-web-server/index.js

```
function broadcast_map() {
  const tmp_map_data = new Array();
  tmp_map_data.push(['Country Code', 'Tweet Count']);
  for(const [key, value] of map_data.entries()){
    tmp_map_data.push([key, value]);
  }
  io.emit('broadcast_map', tmp_map_data);
}


io.on('connection', (socket) => {
  user_counter++;
  console.log(`a user connected => number of connected users:${user_counter}`);
  broadcast_map();
  socket.on('disconnect', () => {
    user_counter--;
    console.log(`a user disconnected => number of connected users:${user_counter}`);
  });
});


db_ref.on('child_added', (snapshot) => {
  const key = snapshot.key;
  const tweet = snapshot.val();
  console.log(`broadcast tweet with id:${key}`);
  io.emit('broadcast_tweet', snapshot.val());
  country_code = tweet.user.place.country_code;
  if(country_code !== undefined){
    map_data.set(country_code, (map_data.has(country_code) ? map_data.get(country_code) : 0) + 1);
    broadcast_map();
  }
});
```

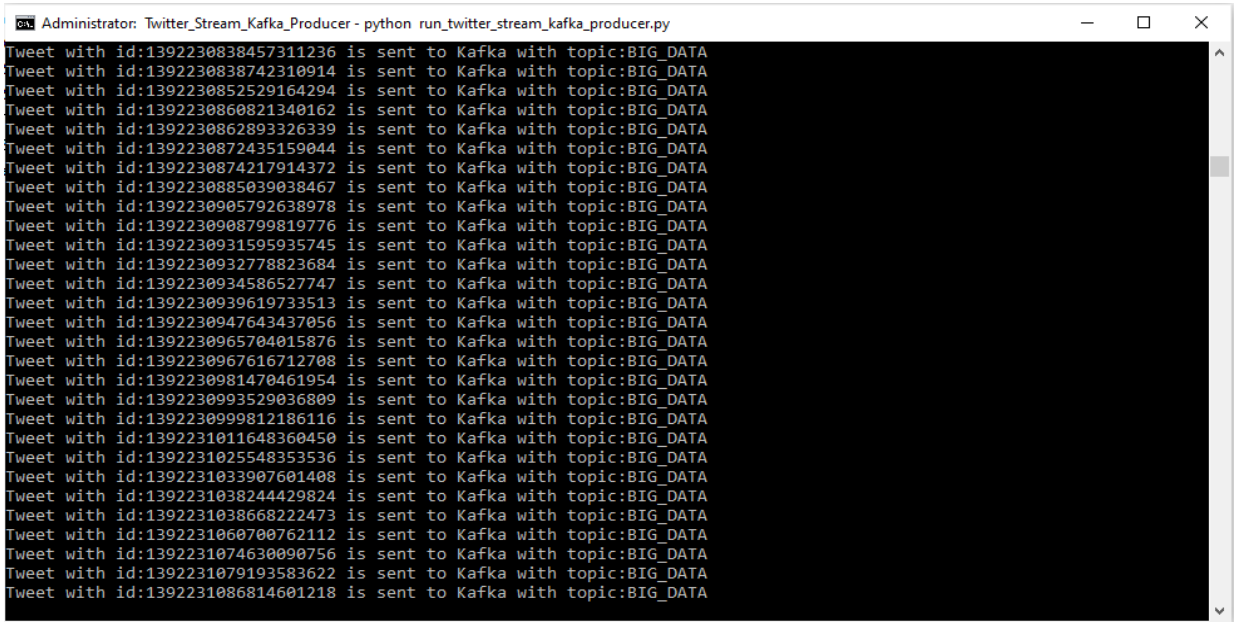
Above code includes three parts. First part is the broadcast_map method which creates an array from a dictionary with a key-value pair of country code, tweet count and broadcasts this array of map data to all connected web clients. Second part is a socket listener of connected and disconnected web clients. It broadcasts map data to every new connected client. And it also prints the number of connected web clients on the terminal after every connection and disconnection. Third part is listening data entry on firebase real-time database. It listens to every new tweet entry in the database, broadcasts new tweet entries, updates a dictionary of map data and broadcasts map data to every connected client to show real-time tweet feed and tweet count for each country from beginning on map.

15. In a web browser, open localhost 127.0.0.1:PORT, where PORT is the NODE_JS_WEB_SERVER_PORT variable that was set in the setup environment variable step (setup_env_var.bat file).

    The following is the visualization done by us on the Twitter stream. For visualization we have divided the screen into 2 parts. In the Left part we are showing tweets of the users which are getting updated as new tweets are emitted from FireBase. In the Right part we are showing the number of tweets coming from different parts of the world. More Tweets make the region darker.



16. Open a new CMD with running "start" in env_var CMD and name this CMD by running "title Twitter_Stream_Kafka_Producer" in the newly started CMD.
17. In Twitter_Stream_Kafka_Producer CMD, go to the project/src directory and run "python run_twitter_stream_kafka_producer.py" to start the Twitter stream and Kafka producer.



run_twitter_stream_kafka_producer.py:

```
class TwitterStreamListener(StreamListener):

    def __init__(self, kafka_producer=None, kafka_topic_name=None):
        super().__init__()
        self.kafka_producer = kafka_producer
        self.kafka_topic_name = kafka_topic_name

    def on_connect(self):
        print(f'Connected to twitter\'s streaming server')
        return

    def on_data(self, tweet):
        encoded_tweet = tweet.encode(encoding='utf-8')
        self.kafka_producer.send(self.kafka_topic_name, encoded_tweet)
        json_tweet = json.loads(tweet)
        print(f'Tweet with id:{str(json_tweet["id"])} is sent to Kafka with topic:{self.kafka_topic_name}')
        return True
```

Above code includes twitter stream listener which listens tweets from twitter filtered stream api, print tweet id on terminal and send these tweets to kafka producer which broadcasts these tweets on zookeeper so that subscribers can get these tweets.

```
class Twitter(object):

    def __init__(self):
        self.TWITTER_ACCESS_TOKEN = os.environ.get('TWITTER_ACCESS_TOKEN')
        self.TWITTER_ACCESS_TOKEN_SECRET = os.environ.get('TWITTER_ACCESS_TOKEN_SECRET')
        self.TWITTER_CONSUMER_KEY = os.environ.get('TWITTER_CONSUMER_KEY')
        self.TWITTER_CONSUMER_SECRET = os.environ.get('TWITTER_CONSUMER_SECRET')
        self.TWITTER_BEARER_TOKEN = os.environ.get('TWITTER_BEARER_TOKEN')
        self.TWITTER_FILTER_KEYWORD_LIST = list(
            [
                key.strip() for key in os.environ.get('TWITTER_FILTER_KEYWORD_LIST').split(',') if key.strip() != ''
            ]
        )

        self.auth = OAuthHandler(
            consumer_key=self.TWITTER_CONSUMER_KEY,
            consumer_secret=self.TWITTER_CONSUMER_SECRET
        )
        self.auth.set_access_token(
            key=self.TWITTER_ACCESS_TOKEN,
            secret=self.TWITTER_ACCESS_TOKEN_SECRET
        )
        self.api = API(
            auth_handler=self.auth
        )

    def stream(self, listener=None):
        Stream(
            auth=self.api.auth,
            listener=listener
        ).filter(
            track=self.TWITTER_FILTER_KEYWORD_LIST,
            is_async=False
        )
```

Above code includes twitter class which reads twitter access token, access token secret, consumer key, consumer key secret and bearer token which is required to authenticate twitter stream api. It also includes a stream method which authenticates and starts twitter filtered streaming and filters tweets according to the filter keyword list which is set in setup_env_var.bat file.

```
class KafkaTwitterProducer(object):

    def __init__(self):
        self.KAFKA_HOST_NAME = os.environ.get('KAFKA_HOST_NAME')
        self.KAFKA_PORT = os.environ.get('KAFKA_PORT')
        self.KAFKA_TOPIC_NAME = os.environ.get('KAFKA_TOPIC_NAME')

        self.kafka_twitter_producer = KafkaProducer(
            bootstrap_servers=f'{self.KAFKA_HOST_NAME}:{self.KAFKA_PORT}'
        )

    def produce_tweets(self):
        Twitter().stream(
            listener=TwitterStreamListener(
                kafka_producer=self.kafka_twitter_producer,
                kafka_topic_name=self.KAFKA_TOPIC_NAME
            )
        )


def main():
    KafkaTwitterProducer().produce_tweets()


if __name__ == "__main__":
    main()
```

Above code includes kafka producer which broadcasts listened or produced tweets on zookeeper to registered host name, port and topic name.

5. Conclusion

We efficiently finished the project by taking real-time Twitter data injected into Kafka Consumer.Then passed on the large amount of data into Spark Streaming,Then the data is then stored into Firebase in real_time. From nodejs we are accessing the data stored in the firebase using Firebase API Finally, Integrating all the components together we are able to visualize the Tweets from all over the world.

6. Contribution Of Individual To The Project

Yash Patel - A20451170

I. Project Proposal
II. Literature Survey - Filtered Stream API, Apache Kafka
III. Setup Twitter, Spark, Node.Js Server
IV. Spark Streaming and Batch Processing
V. Testing
VI. Report

Harsh Vora - A20445400

I. Project Proposal,Draft & Report
II. Literature Survey - Node.JS, Socket.IO
III. Stream Twitter Data into Kafka
IV. Setup Spark, FireBase, Node.js Server
V. Data Loading in FireBase with testing Unit
VI. HTML Web Page for Visualization
VII. Testing
VIII. Report

Vishnu Bharath - A20465596

I. Project Proposal,Draft & Report
II. Literature Survey - Apache Spark, FireBase Realtime Database
III. Setup Twitter, Kafka, FireBase
IV. Data Loading in Firebase with Testing Unit
V. HTML Web Page for Visualization
VI. Testing
VII. Report

Varun Veerla - A20458191

I. Project Proposal,Draft & Report
II. Literature Survey - Twitter
III. Setup Kafka
IV. Stream Twitter Data into Kafka
V. Spark Streaming and Batch Processing
VI. Testing
VII. Report

7. References

[1] **https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/introduction**

[2] **https://developer.twitter.com/en/docs/tutorials/consuming-streaming-data**

[3] **https://dzone.com/articles/running-apache-kafka-on-windows-os**

[4] **http://kafka.apache.org/documentation/**

[5] https://data-flair.training/blogs/kafka-workflow/

[6] https://phoenixnap.com/kb/install-spark-on-windows-10

[7] https://databricks.com/glossary/what-is-spark-streaming

[8] https://spark.apache.org/streaming/

[9] https://firebase.google.com/docs/database

[10] https://pypi.org/project/firebase-admin/

[11] https://firebase.google.com/docs/database

[12] https://firebase.google.com/docs/database/web/start

[13] https://nodejs.org/en/about/

[14] https://www.npmjs.com/package/socket.io

[15] https://socket.io/

[16] https://developers.google.com/chart/interactive/docs/gallery/geochart

[17] https/aws.amazon.com/kinesis/