

# **PROJECT REPORT**

## **DATA ENGINEERING**

Under the Supervision

Prof. Dr. Frank Schulz,

Professor, SRH Hochschule Heidelberg

*Submitted by:*

John Joy Kurian

Avinash Ronanki

Deborah Menezes

Sandeep Krishna Prasad

**--JADS**

Big Data & Business Analytics, SRH Hochschule  
Heidelberg

## Contents

<b>ABSTRACT .....</b>	<b>1</b>
<b>INTRODUCTION .....</b>	<b>2</b>
<b>REQUIREMENT GATHERING .....</b>	<b>3</b>
<b>TWITTER.....</b>	<b>4</b>
<b>PIPELINE CREATION .....</b>	<b>5</b>
<b>1st Stage: High Level Architecture.....</b>	<b>5.1</b>
<b>2<sup>nd</sup> Stage: Docker Stack Architecture .....</b>	<b>5.2</b>
<b>Final Stage: Hosting on Cloud .....</b>	<b>5.3</b>
<b>DEVELOPMENT .....</b>	<b>6</b>
<b>Data Pipeline using Docker containers.....</b>	<b>6.1</b>
<b>1st container → Zookeeper .....</b>	<b>6.1.1</b>
<b>2<sup>nd</sup> container → Kafka .....</b>	<b>6.1.2</b>
<b>3<sup>rd</sup> container → Kafka Producer.....</b>	<b>6.1.3</b>
<b>4<sup>th</sup> container → Kafka Consumer .....</b>	<b>6.1.4</b>
<b>5th container → MongoDB.....</b>	<b>6.1.5</b>
<b>6th container → Notebook .....</b>	<b>6.1.6</b>
<b>Hosting into Cloud.....</b>	<b>6.2</b>
<b>Step 1 — Provisioning the Cluster in Digital Ocean .....</b>	<b>6.2.1</b>
<b>Step 2 — Configuring Firewall Rules to Allow Docker Swarm Traffic .....</b>	<b>6.2.2</b>
<b>Step 3 — Initializing the Cluster Manager.....</b>	<b>6.2.3</b>
<b>Step 4 — Adding worker Nodes to the Cluster .....</b>	<b>6.2.4</b>
<b>Step 5 — Managing the Cluster .....</b>	<b>6.2.5</b>
<b>Step 6 — Running Services in the Docker Swarm .....</b>	<b>6.2.6</b>
<b>Step 7 — Hosting on Cloud .....</b>	<b>6.2.7</b>
<b>HURDLES FACED WHILE BUILDING THE DATA PIPELINE .....</b>	<b>7</b>
<b>FORMULATING BUSINESS STATEMENTS .....</b>	<b>8</b>
<b>Preparation Phase .....</b>	<b>8.1</b>
<b>Formation Phase .....</b>	<b>8.2</b>
<b>Business Questions regarding ‘Hurricane Dorian’. .....</b>	<b>8.2.1</b>
<b>CONCLUSION .....</b>	<b>9</b>
<b>BIBLIOGRAPHY.....</b>	<b>10</b>

## ABSTRACT

Our main purpose is to form a data pipeline to handle live streaming of data. Our main objective is analysing the data and creating a dashboard. In order to achieve this, we have identified few business questions based on previous scientific papers. Creating basic analysis on real time data using the entire data pipeline, we pull the data from different sources (Twitter API), we used Kafka tool to handle the data and push these tweets into MongoDB. We conduct live analysis of data using MongoDB and spark. Finally, they are further pushed onto a dashboard using Tableau for viewing the result. In order to make the environment feasible and flexible to access, we use docker platform. We have created six different containers which individually acts as a service to our data pipeline. Further we try to deploy the docker container to the cloud using Kubernetes.

## INTRODUCTION

In this report, we will design and implement streaming applications for performing basic analytics on real-time data. The data tweets to be analysed will be coming from the different data sources (Twitter). The main objective of the project is to be able to implement data pipeline technology namely Kafka through which real time data from the below mentioned sources. Further an analysis of these tweets is done on the data that is stored in the mongo DB and correspondingly results are tabulated.

### *Software Required:*

- Zookeeper
- Kafka
- MongoDB
- Python
- Jupyter Notebook
- Kubernetes
- Docker Swarm

### *Methodologies:*

- Data Cleansing
- Data Analysis and Visualization

## REQUIREMENT GATHERING

We had couple of brainstorming sessions where we finalized on picking up a topic “TWITTER”. Our purpose to find the right business questions through time investment on R & D where different research papers were examined to accomplish our goal. We selected “*Natural calamities and disaster*” as our theme where we framed some business statements.

### TWITTER

In this project, we have learnt about processing live data streams using streaming APIs and Python. We have performed a basic sentiment analysis of real-time tweets. In addition, we also got a basic introduction to Apache Kafka, which is a queuing service for data streams. Twitter developer platform is an open platform where data from twitter can be streamed in accordance with your application logic. We can download the twitter data from the developer’s platform for maximum 7 days for free. Twitter data is the most comprehensive source of live, public conversation worldwide. We can get insight into audiences, market movements, emerging trends, key topics, breaking news, and much more. One of the first requirements is to get access to the streaming data; in this case, real-time tweets. Twitter provides a very convenient API to fetch tweets in a streaming manner for accessing twitter data, we have to first register our self at the twitters developers’ platform where we have to mention the reason for accessing the twitter data. Verification is done by the twitter and accordingly a consumer access key and access token are given which are specific to one user for accessing the twitter data.

## PIPELINE CREATION

Journey of our pipeline contains three phases in order to fulfil the requirements goals.

### 1st Stage: High Level Architecture

Our goal in this project is to create a high-throughput, scalable, reliable and fault-tolerant data pipeline capable of fetching event-based data and streaming those events to Kafka which will parse their contents to Mongo, all of which will be done in near real-time. We created the pipeline internally, Since we started with a basic idea and took the reference on the ppt of Dr Frank Schulz, we created the entire set up on our local machines (platform) but however we faced problems due to platform independency i.e It very hard to replicate these in all the environment, The actual problem we faced was while installing Kafka on windows system, it failed to do so because of configuration problems, setting up environment variables and visualization in Tableau was not possible due to limitations caused because of SSL certificate, the ports cannot be exposed to an external service from mongodb. We can see our basic outline of our initial architecture shown below.

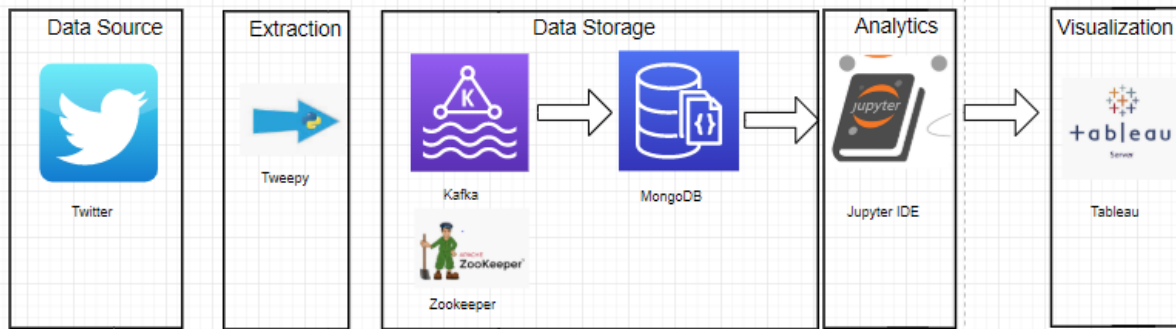


Fig (1): High Level Architecture

## 2<sup>nd</sup> Stage: Docker Stack Architecture

In order to overcome the problem faced in stage 1, We unified the environment using Docker platform as it easily scalable and we can replicate in any environment. Our idea is to build containers for every individual tool, so we created six containers and integrated them into the pipeline. Docker container architecture was initially running on the local machine, Docker environment Integration problems were faced because of port issue but later resolved by going through documentation papers to make it highly scalable we tried to deploy it on cloud environment. We can see our basic outline of our docker stack architecture shown below.

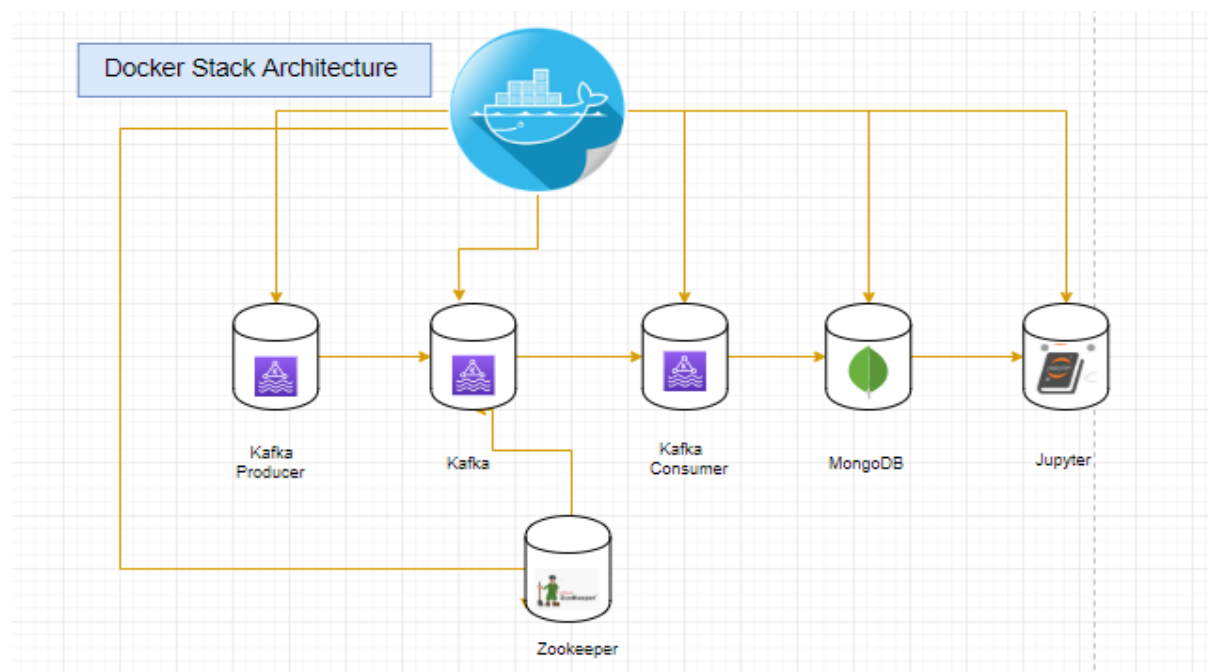


Fig (2): Docker Stack Architecture

### Final Stage: Hosting on Cloud

To host the entire set up onto the cloud platform, initially we tried using Kubernetes because it has high community support, we were successful in creating Kubernetes clusters on the local machine but however we failed to host it because of the complicated architecture. Kubernetes is mainly used in production environment in the presence of numerous docker containers. Alternatively, we opted for Docker Swarm technology where the architecture is comparatively less complex. We can see our basic outline of our initial architecture shown below.

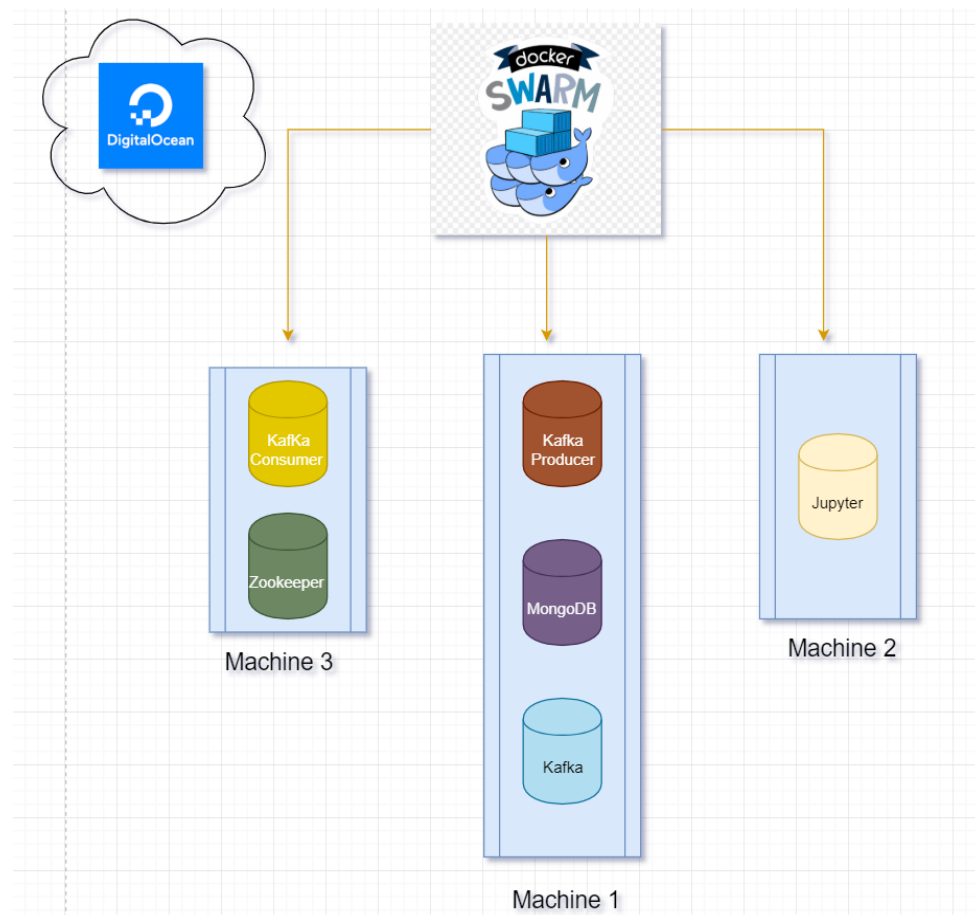


Fig (3): Hosting on Cloud

## DEVELOPMENT

### Data Pipeline using Docker containers

We installed **Docker for Mac** and use **Docker toolbox** on Windows systems

- Docker Toolbox provides a way to use Docker on Windows systems that do not meet minimal system requirements for the Docker for Windows app.

- Docker Toolbox provides *docker-machine*, *docker-compose* and *docker* commands pre-installed in the package.
- It also has an integrated docker terminal, virtual box and Kitematic(GUI Tool)
- After installing of docker toolbox, we pull docker images which are the technologies that are used from docker hub.
- Docker compose yaml file is created to build the containers and link them together.
- In the docker compose file we create the Mongo image taken from docker hub where two environment variables are present “*MONGO\_INITDB\_ROOT\_USERNAME*” and “*MONGO\_INITDB\_ROOT\_PASSWORD*”.
- Port mapping initialized from the local machine to the server (27017:22017)

```
1  version: "3"
2  services:
3
4      mongo:
5          image: mongo
6          environment:
7              MONGO_INITDB_ROOT_USERNAME: root
8              MONGO_INITDB_ROOT_PASSWORD: example
9          ports:
10             - "27017:27017"
11
12      zookeeper:
13          image: wurstmeister/zookeeper
14          ports:
15             - "2181:2181"
16
17      kafka:
18          image: wurstmeister/kafka
19          depends_on:
20             - zookeeper
21          ports:
22             - "9092:9092"
23          environment:
24              KAFKA_ADVERTISED_HOST_NAME: kafka
25              KAFKA_HEAP_OPTS: "-Xmx256m -Xms256M"
26
27              KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
28              KAFKA_CREATE_TOPICS: "tweetstream:1:1"
29
30      producer:
31          image: producer
32          depends_on:
33             - kafka
34          environment:
35              KAFKA_SERVICE: kafka:9092
36
37
38      consumer:
39          image: consumer
40          depends_on:
41             - kafka
42             - mongo
43          environment:
44              KAFKA_SERVICE: kafka:9092
45
46
47
48      notebook:
49          image: johnkurian/ds-notebook:v5
50          depends_on:
51             - mongo
52          ports:
53             - "443:8888"
54          deploy:
55              resources:
56                  reservations:
57                      memory: 3000M
58
```

Fig (4): Docker compose yaml file

### 1st container → Zookeeper

Zookeeper acts as a gate keeper which keeps the track of Kafka. Using zookeeper, we have been keeping track of what all topics have been created.

- Zookeeper is necessary for the installing of Kafka since Kafka depends on it.
- `docker run -d --name zookeeper -p 2181:2181 wurstmeister/zookeeper` → This will basically run the command to create the container from the image pulled from docker hub.
- When Kafka fails to create a node or if it breaks then zookeeper selects a new node by polling within an ensemble.
- Port 2181 is exposed to Kafka.

### 2<sup>nd</sup> container → Kafka

Kafka is a data pipeline technology for system where there is high data input and high through put. It is applied where the data has been continuously changing and according to one's business logic consumption of the data that can be defined. Data remain in the data pipeline and according to one's business needs the consumption can be defined and set. Before applying Kafka in the system, the concepts of broker, topic, partition, producer, and consumer should be introduced.

Creating Kafka producer and consumer:

- Connecting to the twitter database using API.
- We use “Tweepy” library to get the stream of tweets over to the Kafka for processing.
- We created Kafka producer to create a <TOPIC> which is a channel that takes the tweets and puts it into Kafka.
- Kafka will handle the streams of data; Kafka consumer is created to push the data to MongoDB. Kafka-consumer subscribed to the <TOPIC> created by the Kafka-producer to get tweets that are later pushed into the MongoDB.
- In docker compose file, Kafka contains the base image from the docker hub where four environment variables are present.
  - i. `KAFKA_ADVERTISED_HOST_NAME`  
It is an identifier so that the services can refer to the Kafka service, the hostname is “Kafka”
  - ii. `KAFKA_HEAP_OPTS`  
Heap ops restricts the amount of memory that the JVM is using for the Kafka. Kafka by default uses 1GB of memory but due to some restriction we reduce the memory allocation.
  - iii. `KAFKA_ZOOKEEPER_CONNECT`  
Zookeeper listens to Kafka, it specifies the port name of Zookeeper services and hence port mapping.
  - iv. `KAFKA_CREATE_TOPICS`  
Creating a topic with 1 partition and 1 replica  
`<Topic_name>:<Partition>:<Replica>`



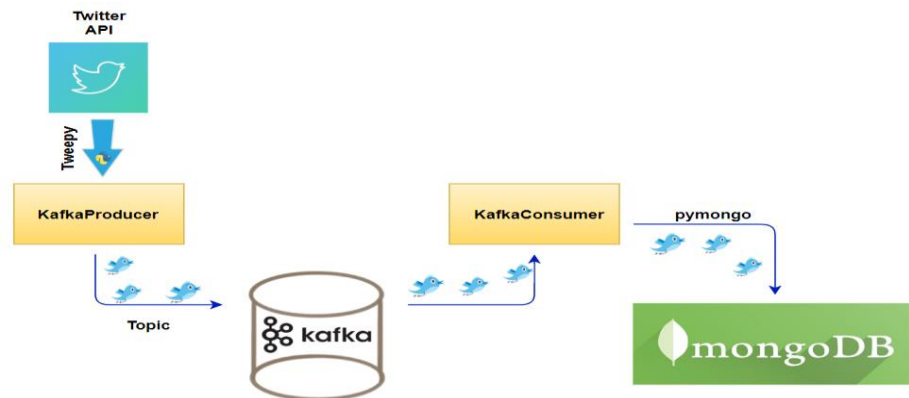


Fig (5): Kafka Producer Consumer working architecture

### 3<sup>rd</sup> container → *Kafka Producer*

A producer publishes messages to Kafka topics.

- It is a custom built docker image.
- The *producer.py* file contains the producer code that connects to the twitter data source as shown in fig (5)
- Producer service depends on the Kafka service. It starts the service when Kafka services has been started.
- It contains one environment variable called `KAFKA_SERVICE` which points to the *Kafka hostname 9092*

### 4<sup>th</sup> container → *Kafka Consumer*

A consumer subscribes to the Kafka topics.

- It is a custom built docker image.
- The *consumer.py* file contains the consumer code that listens to the topic when the twitter streams are coming from and then dumps it into the MongoDB as shown in fig (5)
- The service depends on Kafka and MongoDB.
- It contains one environment variable called `KAFKA_SERVICE` which points to the *Kafka hostname 9092*

### 5<sup>th</sup> container → *MongoDB*

MongoDB is a cross-platform document-oriented database program.

- To handle unstructured data which are being received by the twitter API, we used MongoDB (NoSQL database).
- We created this container from the docker hub.

### 6th container → Notebook

- It is a custom image but we pulled the base image which is “jupyter/scipy-notebook”.
- We created the custom image in order to modify the jupyter.config file and to facilitate WebSocket communication in the cloud. We faced an issue during this process where the WebSocket were unable to communicate i.e unable to view the output in jupyter notebook; We fixed this by adding an SSL certificate to the hosted IP address.

```
$ openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout mykey.key -out mycert.pem
```

- After creating the custom image, we pushed it to docker hub and while deploying, we pull it from docker hub as shown in fig (6).
- The service depends MongoDB.
- Port Mapping from 443:8888 i.e from localhost (8888) to webhost (443); 443 which is a secured connection.
- We can manually customize memory allocation for each container by modifying the docker compose file which helps any memory issues further.

```
/.jupyter/jupyter-notebook-config.py
1 c.NotebookApp.allow_origin = '*'
2 c.NotebookApp.base_url = '/jupyter'
3 c.NotebookApp.certfile = '/absolute/path/to/mycert.pem'
4 c.NotebookApp.ip = 'localhost'
5 c.NotebookApp.keyfile = '/absolute/path/to/mykey.key'
6 c.NotebookApp.open_browser = False
7 c.NotebookApp.password = 'paste_hashed_password_here'
8 c.NotebookApp.trust_xheaders = True
```

Fig (6): Jupyter-notebook-config.py

### Hosting into Cloud

Docker Swarm is the Docker-native solution for deploying a cluster of Docker hosts. We can use it to quickly deploy a cluster of Docker hosts running either on our local machine or on supported cloud platforms.

#### Step 1 — Provisioning the Cluster in Digital Ocean

We need to create several Docker hosts for our cluster. The following command provisions a single Dockerized host, where *\$DOTOKEN* is an environment variable that evaluates to our DigitalOcean API token; We executed this command on our local machine to create three Docker hosts, named machine-1, machine -2, and machine -3:

```
$for I in 1 2 3; do docker- machine create --driver digitalocean \
```

```
$--digitalocean-image ubuntu-16-04-x64 \
```

```
$ docker-machine create --driver digitalocean --digitalocean-image ubuntu-16-04-x64 --digitalocean-access-token $DOTOKEN machine-$i
```

After the command has completed successfully, we can verify that all the machines have been created by visiting your DigitalOcean dashboard, or by typing the following command:

```
$docker-machine ls
```

The output should be similar to the following, and it should serve as a quick reference for looking up the IP address of the nodes:

NAME	ACTIVE	DRIVER	STATE	URL	SWARM	DOCKER	ERRORS
default	-	virtualbox	Saved			Unknown	
machine-1	-	digitalocean	Running	tcp://167.71.164.38:2376		v19.03.2	
machine-2	-	digitalocean	Running	tcp://167.71.100.86:2376		v19.03.2	
machine-3	-	digitalocean	Running	tcp://167.71.165.87:2376		v19.03.2	

Fig (7): Clusters in the Digital Ocean

At this point, all three Dockerized hosts have been created, and we have each host's IP address. In the next steps, we'll configure the firewall rules that will make the nodes to function as members of a cluster, pick one of the nodes and make it the Docker Swarm manager, and configure the rest as Docker Swarm workers.

## Step 2 — Configuring Firewall Rules to Allow Docker Swarm Traffic

A cluster has to have at least one node that serves as manager. For this setup, we pick first node and make it the Swarm manager. The other two nodes will be the worker nodes. Certain network ports must be opened on the nodes that will be part of a cluster for the cluster to function properly. That entails configuring the firewall to allow traffic through those ports. Setting up such a system requires careful manipulation of the Linux firewall. The network ports required for a Docker Swarm to function correctly are:

- TCP port 2376 for secure Docker client communication. This port is required for Docker Machine to work. Docker Machine is used to orchestrate Docker hosts.
- TCP port 2377. This port is used for communication between the nodes of a Docker Swarm or cluster. It only needs to be opened on manager nodes.
- TCP and UDP port 7946 for communication among nodes (container network discovery).
- UDP port 4789 for overlay network traffic (container ingress networking).

We encountered a firewall problem hence followed these steps above. After which we restarted the machine.

## Step 3 — Initializing the Cluster Manager

We've decided that machine-1 will be our cluster manager, so log in to the machine from our local machine:

```
$docker-machine ssh machine-1
```

The command prompt will change to reflect the fact that we are now logged into that particular node. To initialize the swarm network and to assign a manager node, we type the following command:

```
root@machine-1: ~# docker swarm init --advertise-addr 167.71.164.38
```

The output for the above command is shown below:

```
Output
Swarm initialized: current node (a35hhzdzf4g95w0op85tqlow1) is now a
manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-
3k7ighcfs9352hmdfzh31t297fd8tdskg6x6oi8kpzzszznffx-
6kovxm3akca2qe3uaxtu07fj3 \
    167.71.164.38:2377

To add a manager to this swarm, run 'docker swarm join-token manager'
and follow the instructions.
```

So now we have a Docker Swarm with a manager configured and adding the remaining nodes as workers.

#### Step 4 — Adding worker Nodes to the Cluster

We opened another terminal and left the terminal tab or window we used to log into the Swarm manager alone for now. We connect to machine-2 from our local machine:

```
$docker-machine ssh machine-2
```

After the command has been executed successfully, the output is shown below:

**This node joined a swarm as a worker.**

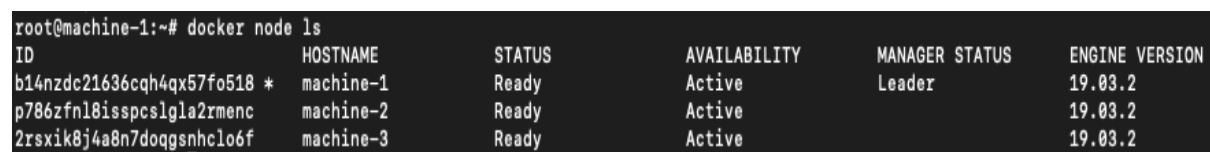
Repeat the same process for machine-3 as well. We have now added two worker nodes to the cluster. If the firewall rules were configured correctly (this is where we faced the problem

since our firewall wasn't configured properly), After which now we have a functioning Docker Swarm, with all the nodes synchronized.

## Step 5 — Managing the Cluster

After the manager and worker nodes have been assigned to the cluster, all Docker Swarm management commands have to be executed on the manager nodes. So, we return to the terminal that we used to add the manager and type in this command to view all members of the cluster:

root@machine-1: ~# docker node ls



ID	HOSTNAME	STATUS	AVAILABILITY	MANAGER STATUS	ENGINE VERSION
b14nzc21636cqh4qx57fo518 *	machine-1	Ready	Active	Leader	19.03.2
p786zfnl8isspcslgla2rmenc	machine-2	Ready	Active		19.03.2
2rsxik8j4a8n7doqgsnhc1o6f	machine-3	Ready	Active		19.03.2

Fig (8): Current nodes present in the cluster

This output shows that we're dealing with a 3-node Docker Swarm and its nodes — a manager and two workers. Specification of the three machines having 1GB memory/ 25 Disk Ubuntu 16 running in the same region of NYC3. Here we faced some memory allocation issue because Jupiter notebook services require more space allocation and hence, we had to upgraded only machine-2 which contains the jupyter services to 4GB memory space which costed us 20\$/month. After which the machine-2 was on shutdown, therefore allowing the jupyter services to jump to machine-3. We had to force manually so that the services run on machine-2 instead of machine-3 using the following command.

`$docker service update --force app_notebook`

## Step 6 — Running Services in the Docker Swarm

The Docker Swarm is up and running. Containers are deployed as Services using the docker service command. And like the docker node command, the docker service command can only be executed on a manager node.

We now deploy then docker stack in the cluster by pulling the files which include the docker compose yaml file and Kafka producer and Kafka consumer from the GitHub link below:

<https://github.com/JohnKurian/twitter-analytics>

We manually built the producer and consumer image and notebook image and then deployed to the stack using the following commands:

`$docker build . -t producer`

`$docker build . -t consumer`

`$docker build . -t ds/notebook`

We need to deploy the stack we executed the following command.

```
$docker stack deploy --compose-file=docker-compose.yml app
```

Now all the services will be up and running as shown below:

```
root@machine-1:~# docker service ls
```

ID	NAME	MODE	REPLICAS	IMAGE	PORTS
0k7j5m8q3l0l	app_consumer	replicated	1/1	consumer:latest	
pormf9b2xsvj	app_kafka	replicated	1/1	wurstmeister/kafka:latest	*:9092->9092/tcp
j99ybg2mnzbc	app_mongo	replicated	1/1	mongo:latest	*:27017->27017/tcp
z8s7lz80yo8i	app_notebook	replicated	1/1	johnkurian/ds-notebook:v5	*:443->8888/tcp
npkenkyx73l3	app_producer	replicated	1/1	producer:latest	
qxp6hqsxht13	app_zookeeper	replicated	1/1	wurstmeister/zookeeper:latest	*:2181->2181/tcp

Fig (9): Services up and running on the swarm

## Step 7 – Hosting on Cloud

The front end of the App is the Jupyter Notebook which we are serving on the port 443. This notebook will be available on the *IP address/Jupyter*

<https://167.71.164.38/Jupyter>

## HURDLES FACED WHILE BUILDING THE DATA PIPELINE

We faced Kafka installation problem on windows system, it failed to do so because of configuration problems, setting up environment variables.

**Solution** → We unified the environment using Docker platform as it is easily scalable and we can replicate in any environment.

We were successful in creating Kubernetes clusters on the local machine but however we failed to host it because of the complicated architecture.

**Solution** → We opted for Docker Swarm technology where the architecture is comparatively less complex.

We faced an issue during the process where the WebSockets were unable to communicate i.e unable to view the output in jupyter notebook.

**Solution** → We fixed this by adding an SSL certificate to the hosted IP address by modifying the Jupyter-notebook-config.py

We encountered a firewall problem on the docker swarm

**Solution** → Configuration of ports were done yet again

We faced some memory allocation issue because Jupyter notebook services require more space allocation

**Solution** → We had to upgrade the machine to allocate jupyter services to 4GB memory space which costed us 20\$/month.

## Formulating Business Statements

### PREPARATION PHASE

The theme/subject that we choose was “disaster monitoring”, during research we tried filtering the stream using relevant keywords hurricane flood storm then we inspected the incoming tweet stream. We found out that a significant fraction of the stream is not relevant to the topic at hand. E.g.: If the keyword is hurricane the filter would fetch the result like “*my life is like a hurricane*” which is not relevant. So, we identified the need to filter the stream even further. So, we decided to create a classifier model that distinguished between relevant and irrelevant tweets.

*Step1:* Obtaining the training dataset.

We found a dataset from a cloud sourcing website which contains 10000 tweet which are marked relevant or irrelevant.

	text	choose_one	class_label
10854	1.3 #Earthquake in 9Km Ssw Of Anza California ...	Relevant	1
10855	Evacuation order lifted for town of Roosevelt:...	Relevant	1
10856	See the 16yr old PKK suicide bomber who detona...	Relevant	1
10857	To conference attendees! The blue line from th...	Relevant	1
10858	The death toll in a #IS-suicide car bombing on...	Relevant	1

*Step2:* Sanitizing the dataset.

This step includes removal of the hashtag (#) sign from the tweets and the links and converting the uppercase to lowercase and other symbols.

	Unnamed: 0	text	choose_one	class_label
10854	10854	1 3 earthquake in 9km ssw of anza california ...	Relevant	1
10855	10855	evacuation order lifted for town of roosevelt	Relevant	1
10856	10856	see the 16yr old pkk suicide bomber who detona...	Relevant	1
10857	10857	to conference attendees! the blue line from th...	Relevant	1
10858	10858	the death toll in a is suicide car bombing on...	Relevant	1

*Step3:* tokenizing the sentences.

It is tokenizing the sentences into a list of words

*Step4:* Separating the dataset.

Separating them into training and test datasets (80:20)

*Step5:* Converting the text into embeddings.

We convert the tokenized list of words to a bag of words representation. “A *bag of words*” just associates an index to each word in our vocabulary, and embeds each sentence as a list of 0s, with a 1 at each index corresponding to a word present in the sentence.

*Step 6:* Visualizing the embeddings.

Analysing and see if we can identify some structure. After creating the embeddings, we used principal component analysis (PCA) to project the 18000-dimensional vectors down to 2 dimensional vectors in order to check any significant degree of separation among the two classes.

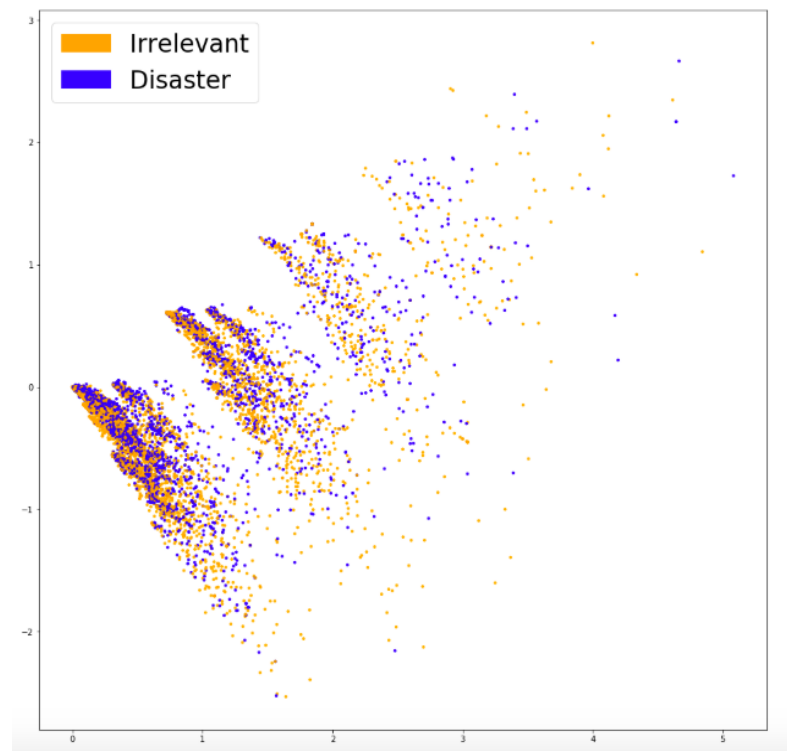


Fig (9): PCA

*Step 7:* Creating a classifier.

We built a simple logistic regression model to predict the relevance of the tweets.

*Step 8:* Model evaluation.

The model achieved an accuracy of 76.1% and F1 score of 75.9%.



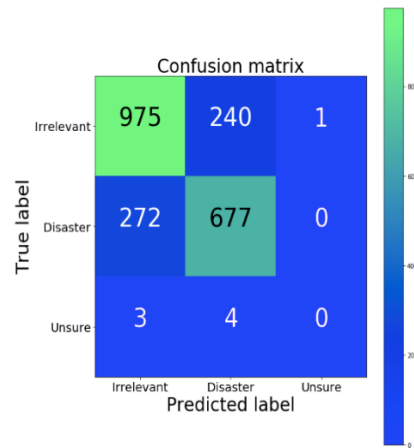


Fig (9): Confusion Model

*Step 9:* After getting the training model we create pipeline which contains the vectorize and the classifier and then we pickel the pipeline and save it and further we load this pipeline in the consumer container located in the docker. As soon as we receive the tweet stream, we filter the stream using this classifier and dump this into the MongoDB. The classifier output is shown below.

```
Flood Warning issued September 14 at 8:12PM CDT until September 21 at 1:00AM CDT by NWS https://t.co/iq8h3hPAH4
[1]

A 24-year-old Togolese migrant viciously attacked an 18yr-old girl passing by him. Then he...
[0]
RT @KDANIEL_SUPPORT: Thai Danity donated goods to flood victims in Thailand under Kang Daniel's name

Thank you @/nowondanielk for telling...
[1]
RT @KDANIEL_SUPPORT: Thai Danity donated goods to flood victims in Thailand under Kang Daniel's name

Thank you @/nowondanielk for telling...
[1]
RT @ZetLorento: Arkadaşlar sizlerden ricam; profilimdeki son flood'a en az 3-4 kişiyi etiketleyin.

Flood şu saate kadar 4 haber sitesi, 2...
[1]
RT @Darakumareel: Art for Benefit. 🎨
This awesome art piece was donated by the artist "Kaperp" (one of the fans of #peckpalit) to bid up fo...
[0]
RT @KDANIEL_SUPPORT: Thai Danity donated goods to flood victims in Thailand under Kang Daniel's name

Thank you @/nowondanielk for telling...
[1]
RT @KDANIEL_SUPPORT: Thai Danity donated goods to flood victims in Thailand under Kang Daniel's name

Thank you @/nowondanielk for telling...
[1]
RT @AndroidAutl This week in Android: Pixel 4 leaks flood in, underwhelming iPhone competition https://t.co/qJFT8Ie300
[0]
```

Fig (10): Classifier Output

## FORMATION PHASE

Our initial analysis was on “*amazon forest fire*” since it was a viral subject and we tried implementing few questions on it as seen as follows:

- What is the geo tagged location using tweets?
- What is the viral hashtag used which were spiked during the timeline?
- Identifying the trend of the tweets which contains the hashtags.

Since we were not able to fetch the tweets from the start date of the wildfire, the analysis becomes irrelevant which has no strong significance. Hence, we tried to figure out the current disaster happening around the world and we came across “*Hurricane Dorian*” happening across the Bahamas at the moment. So, we tried to contemplate some analytical queries on the live data for the past 7 days. i.e 7<sup>th</sup> of sept 2019 to 16<sup>th</sup> of sept 2019. Since there is data unavailability, we found it difficult to find 100% accuracy from the dataset.

### Business Questions regarding ‘Hurricane Dorian’.

- 1) What are the activities in the past 7 days based on the tweet count on the hashtags of hurricane Dorian?

```
enable_plotly_in_cell()data = [go.Bar(  
    x=x.values[:10],  
    y=x.index[:10],  
    orientation = 'h'  
)]  
margin=go.layout.Margin()fig = go.Figure(data=data)plot(fig)
```

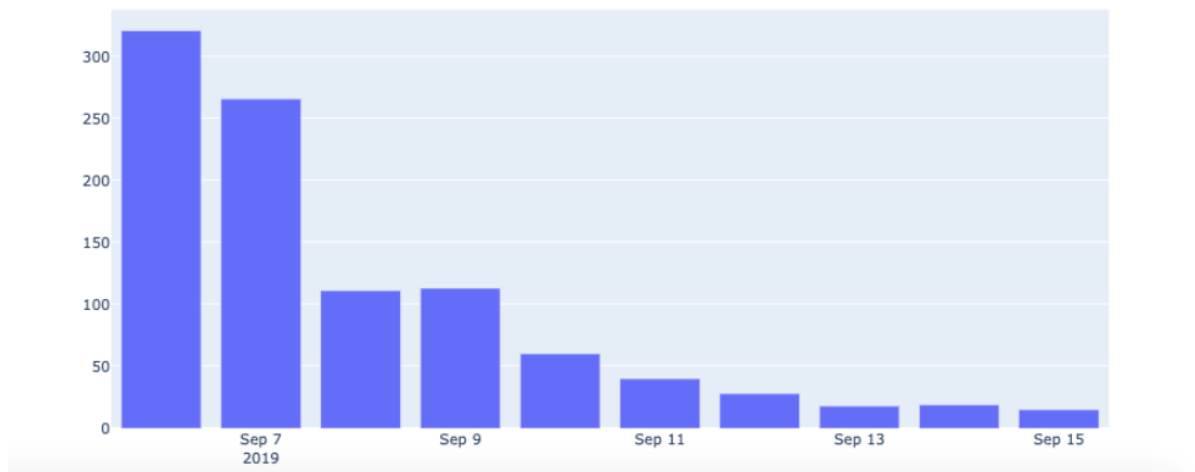


Fig (11): Tweet Count of the hurricane

The above bar graph illustrates about the initial stage of the hurricane, we can see a spike in the tweets and then following a steep decline pattern gradually during the next coming days. This shows that in general that the tweets are high on the day of the incident comparatively to the next day. The maximum number of tweets were 300 on the 7<sup>th</sup> of September 2019

2) Top 10 users that tweeted about the hurricane?

```
enable_plotly_in_cell()
from plotly.offline import iplot
import plotly.graph_objs as go
data = [go.Bar(x=s.index, y=s.values)]
config = {'scrollZoom': True}
iplot(data, config)
```

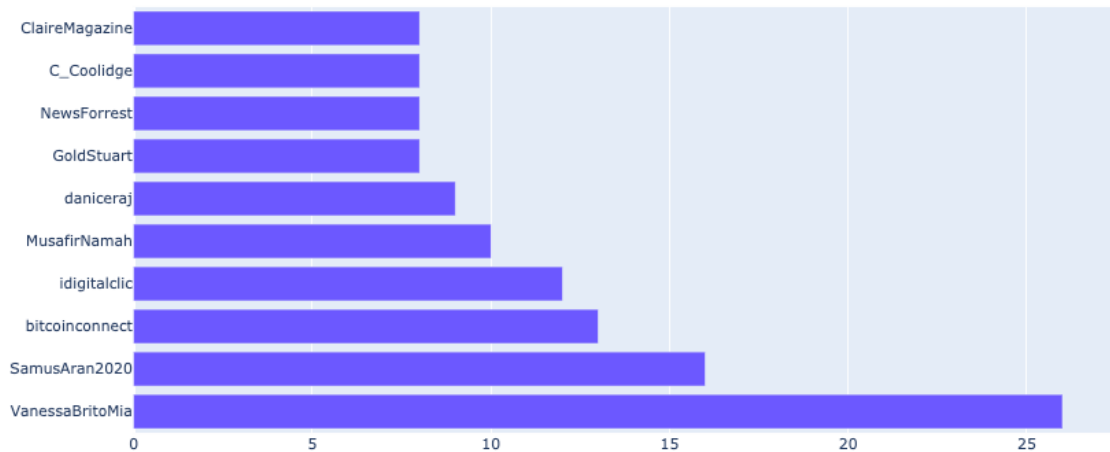


Fig (12): Top 10 Tweet Users

The above bar graph shows the most top ten users who tweeted most number of times about the hurricane, here it shows that “*VanessaBritoMia*” user tweeted 25 times. Knowing the gender as well can play an important role and sentimental analysis can be conducted on this type of dataset.

3) What type of device did the user use to tweet?

```
import plotly.graph_objects as go
source = df.source.value_counts()
labels = source.index[:10]
values = source.values[:10] # Use hole to create a donut-like pie chart
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.3)])
fig.show()
```

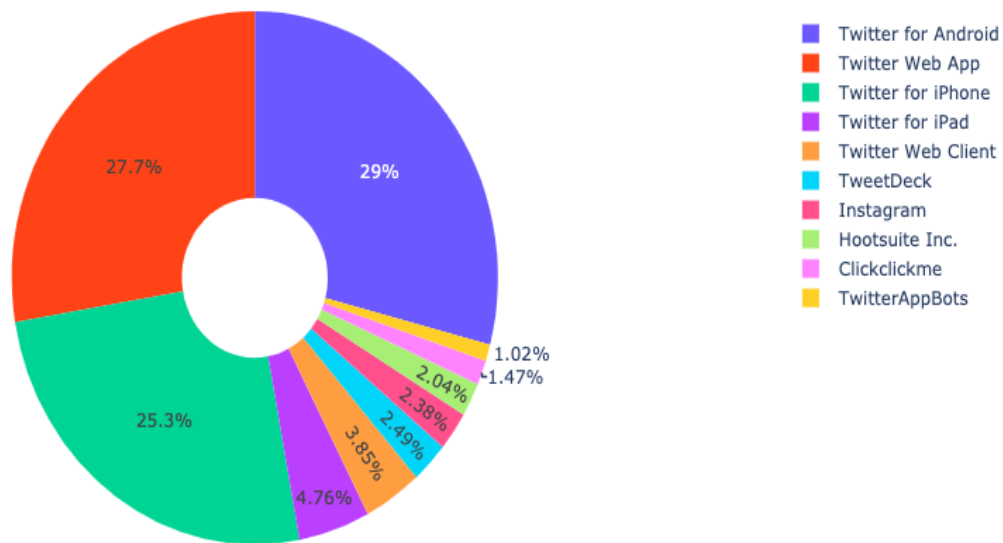


Fig (13): Type of device used for tweeting.

The above graph illustrates that 29% of the tweeter users used android and 27.7% used twitter web app, 25.5% of the tweeter users use iPhone to tweet about the hurricane. From this analysis we can figure out what type of tweeter audience are out there.

#### 4) What is the geo tagged locations using tweets?

```
import plotly.express as px
gapminder = px.data.gapminder().query("year==2007")
fig = px.scatter_geo(country_df, locations="iso_alpha",
                    hover_name="country", size="count",color='country',
                    projection="natural earth")
fig.show()
```

The graph below shows us that the number of tweets were coming from USA and 2<sup>nd</sup> highest would be in Europe. The different colour and size show different countries and the highest number of tweets. Location of the user can be known who tweeted about the hurricane. The conflict here is the population of a particular country and the proximity of the calamity might be a factor. Relatively speaking the hurricane news wasn't explored too much except for USA, UK and Canada.



Fig (14): Geo tagged locations.

*Few more relevant businesses Questions stated below:*

- 5) What is the majority of the gender that tweeted about the hurricane?
- 6) Most effected areas due to the hurricane?
- 7) What is the political influence that the tweet made?
- 8) What would be the hourly death toll because of the hurricane?

## CONCLUSION

The main aim of our project was achieved by implementing a very efficient and robust architecture for analysing live streaming data. The work that we have done will help us in the industry to understand, handle and resolve complex analytical problems. We got a chance to expose ourselves to different technologies and found various approaches to deal with **BIG DATA**.

## BIBLIOGRAPHY

### A. Scientific Papers:

- <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.361.1668&rep=rep1&type=pdf>
- <https://pdfs.semanticscholar.org/62e8/2364b61f06bf1445fa92b76da7b465e62c33.pdf>
- <https://link.springer.com/content/pdf/10.1140%2Fepjds%2Fs13688-019-0193-9.pdf>
- <http://web.stanford.edu/class/cs224n/reports/custom/15785631.pdf>

### B. Website Reference:

- <https://www.codecademy.com/articles/sql-commands>
- <https://spark.apache.org/docs/2.2.0/streaming-programming-guide.html#overview>
- [https://www.cloudkarafka.com/blog/2018-07-04-cloudkarafka\\_what\\_is\\_zookeeper.html](https://www.cloudkarafka.com/blog/2018-07-04-cloudkarafka_what_is_zookeeper.html)
- <https://docs.docker.com/toolbox/overview/>
- <https://spark.apache.org/docs/2.2.0/api/python/index.html>
- [http://192.168.99.100:8888/notebooks/Untitled.ipynb?kernel\\_name=python3](http://192.168.99.100:8888/notebooks/Untitled.ipynb?kernel_name=python3)
- <https://www.geeksforgeeks.org/twitter-sentiment-analysis-using-python/>
- <http://www.met.reading.ac.uk/~sgs02rpa/extreme.html#190913>
- <https://www.digitalocean.com/community/tutorials/how-to-create-a-cluster-of-docker-containers-with-docker-swarm-and-digitalocean-on-ubuntu-16-04>
- <https://blog.couchbase.com/deploy-docker-compose-services-swarm/>
- <https://www.digitalocean.com/community/tutorials/how-to-set-up-a-jupyter-notebook-to-run-ipython-on-ubuntu-16-04>
- <https://www.digitalocean.com/community/tutorials/how-to-set-up-a-jupyter-notebook-to-run-ipython-on-ubuntu-16-04>
- <https://www.linode.com/docs/applications/big-data/install-a-jupyter-notebook-server-on-a-linode-behind-an-apache-reverse-proxy/>
- <https://docs.docker.com/engine/swarm/>
- [https://github.com/hundredblocks/concrete\\_NLP\\_tutorial/blob/master/NLP\\_notebook.ipynb](https://github.com/hundredblocks/concrete_NLP_tutorial/blob/master/NLP_notebook.ipynb)
- <https://www.figure-eight.com/data-for-everyone/>