ĐẠI HỌC QUỐC GIA TP.HCM

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**MÔN: BIG DATA**

**BÁO CÁO CUỐI KỲ**

# ADVERTISING CAMPAIGN ANALYSIS IS BASED ON TWEETS

**Giảng viên: PGS. TS. LÊ ĐÌNH DUY**

Học viên: Lê Nguyễn Sơn Nguyên – CH1702039

Hà Lam – CH1702013

Nguyễn Xuân Đức – CH1702006

TP. Hồ Chí Minh, tháng 10 năm 2019

# MỤC LỤC

# 1. PHẦN CÀI ĐẶT HỆ THỐNG CƠ SỞ – KAFKA VÀ MONGODB

## 1.1. Các bước cài đặt Kafka cluster:

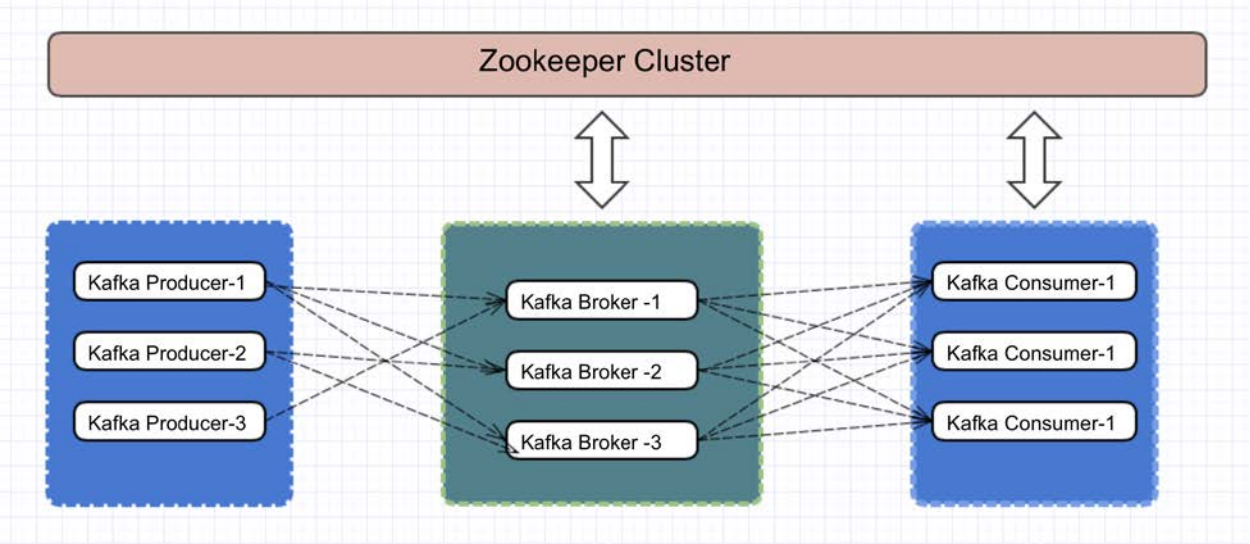
### *1.1.1. Tài nguyên chuẩn bị:*

- 5 máy centos của vlab bao gồm 1 master và 4 slave
- Tạo thư mục kafka trên từng máy để dễ quản lý source code kafka
- Lần lượt thực hiện tải source code kafka lên từng máy vào thư mục kafka và giải nén

Tải source code của kafka có thể sử dụng curl hoặc wget để tải về máy:

```
> cd ~
> mkdir kafka
> cd kafka
> wget http://mirrors.viethosting.com/apache/kafka/2.3.0/kafka_2.12-2.3.0.tgz
> tar -xzf kafka_2.12-2.3.0.tgz
> cd kafka_2.12-2.3.
```

Trên máy master tiến hành cài đặt thông số cho zookeeper, zookeeper giữ nhiệm vụ điều khiển các luồng thông tin từ các producer đến kafka broker xử lý và gửi kết quả đến các kafka consumer.




Tại thư mục kafka_2.12-2.3.0

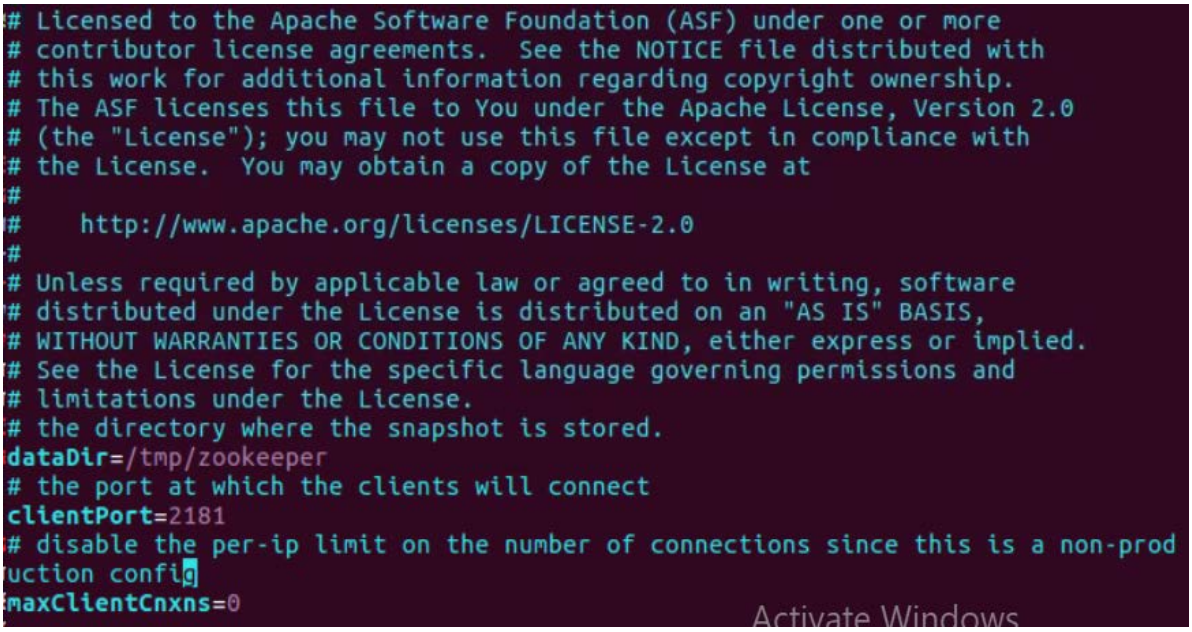
```
> vi config/zookeeper.properties
```

Trong tập tin này có hai thông số quan trọng cần cài đặt:

- dataDir: là thư mục zookeeper sẽ sử dụng để lưu trữ dữ liệu
- clientPort: mặc định là 2181, đây là port zookeeper sử dụng để chạy dịch vụ.

Ta thiết lập hai thông số của zookeeper như sau:

```
# this should ideally be a well maintained and properly backed up directory

dataDir=/tmp/zookeeper

# the port at which the clients will connect

clientPort=2181
```




Khởi động Apache Zookeeper trên máy master bằng cách chạy lệnh tại thư mục kafka_2.12-2.3.0:

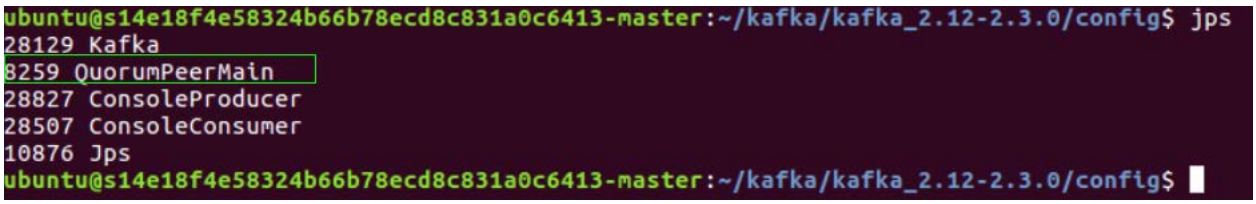
```
>./bin/zookeeper-server-start.sh -daemon
config/zookeeper.properties
```

Để kiểm tra xem dịch vụ Zookeeper đã chạy chưa chạy câu lệnh:

```
> jps
```

Nếu dịch vụ đã chạy bạn sẽ thấy dòng hiển thị dạng như sau:

```
> 8259 QuorumPeerMain
```




### *1.1.2. Cài đặt Apache Kafka*

Chúng ta sẽ cài đặt một cụm kafka với 5 instance (brokers) chạy trên 5 máy vlab đã được cung cấp. Các thông số cơ bản cần chú ý như sau:

- broker.id – mã số định danh của broker, là số kiểu integer: 0, 1, 2 … . Mỗi broker trong cụm cần một số định danh duy nhất.

- log.dirs - Thư mục kafka sử dụng để lưu trữ thông tin, đừng nhầm lẫn với thư mục lưu trữ logs, đây không phải là thư mục lưu trữ logs, thư mục này kafka sử dụng để vận hành.
- port – cổng kafka sử dụng để nhận các kết nối từ producers và consumers.
- zookeeper.connect – danh sách các nodes Zookeeper được phân cách với nhau bởi dấu , VD: hostname1:port1, hostname2:port2. Trong trường hợp này chúng ta thiết lập là localhost:2181

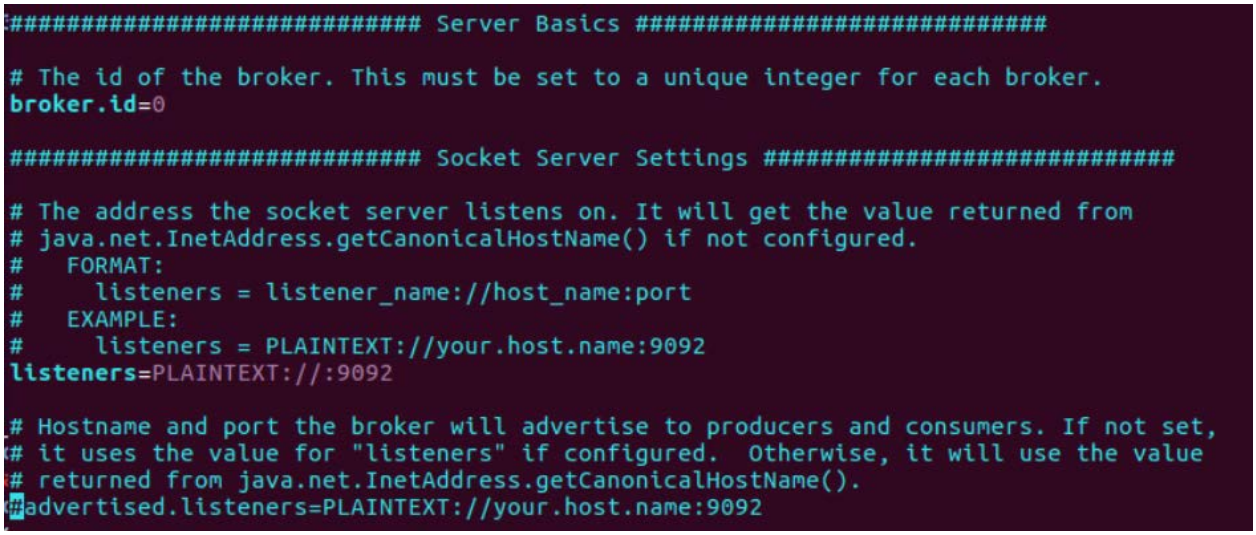
Bước tiếp theo là chúng ta thực hiện cấu hình kafka trên từng máy

Tập tin dùng để cấu hình các thông số của kafka được đặt trong thư mục config, trong thư mục gốc kafka chúng ta đã tải về và giải nén ra. Chúng ta có thể chỉnh sửa các thông số bằng câu lệnh sau:

```
> vi config/server.properties
```

Trên máy master:

```
broker.id=0

# The port the socket server listens on

port=9092

listeners=PLAINTEXT://:9092

# A comma seperated list of directories under which to store
log files

# this should ideally be a well maintained and properly backed
up directory

log.dirs=/tmp/kafka-logs

zookeeper.connect=localhost:2181
```




Trên máy slave 1:

*broker.id=1*

*# The port the socket server listens on*

*port=9092*

*# A comma seperated list of directories under which to store log files*

*# this should ideally be a well maintained and properly backed up directory*

*log.dirs=/tmp/kafka-logs*

*zookeeper.connect=* ***s14e18f4e58324b66b78ecd8c831a0c6413-master.uitlab.com:2181***

*Trên máy slave 2:*

***broker.id=2***

*# The port the socket server listens on*

*port=9092*

*# A comma seperated list of directories under which to store log files*

*# this should ideally be a well maintained and properly backed up directory*

*log.dirs=/tmp/kafka-logs*

*zookeeper.connect=* ***s14e18f4e58324b66b78ecd8c831a0c6413-master.uitlab.com:2181***

Trên máy slave 3:

*broker.id=3*

*# The port the socket server listens on*

*port=9092*

*# A comma seperated list of directories under which to store log files*

*# this should ideally be a well maintained and properly backed up directory*

*log.dirs=/tmp/kafka-logs*

*zookeeper.connect=* ***s14e18f4e58324b66b78ecd8c831a0c6413-master.uitlab.com:2181***

Trên máy slave 4:

*broker.id=4*

*# The port the socket server listens on*

*port=9092*

*# A comma seperated list of directories under which to store log files*

*# this should ideally be a well maintained and properly backed up directory*

*log.dirs=/tmp/kafka-logs*

*zookeeper.connect=* ***s14e18f4e58324b66b78ecd8c831a0c6413-master.uitlab.com:2181***

Bước tiếp theo là khởi động Apache Kafka Cluster

Lần lượt ở thư mục gốc của kafka, chạy lần lượt trên máy master, slave 1, slave 2, slave 3, slave 4 câu lệnh sau:

```
# start broker
./bin/kafka-server-start.sh -daemon config/server.properties
```

Chạy lệnh jps để kiểm tra xem kafka broker đã chạy hay chưa

Kiểm tra lại vận hành của kafka cluster

Chúng ta thực hiện các bước để thiết lập kafka cluster vận hành trên 5 máy.

Sau đây là các bước để kiểm tra xem kafka cluster chúng ta đã cài đặt có vận hành đúng như mong đợi không?

Để kiểm tra, chúng ta thực hiện công việc đơn giản là tạo một topic, gửi đi thông điệp và nhận các thông điệp.

Chúng ta tiến hành tạo một topic trên bất kỳ máy nào (master, slave 1, slave 2, slave 3 hoặc slave 4. Ví dụ ta tạo chủ đề là “test-topic”

```
>./bin/kafka-topics.sh --create --zookeeper
s14e18f4e58324b66b78ecd8c831a0c6413-master.uitlab.com:2181 --topic test-
topic --partitions 1
```

Bạn sẽ nhận được phản hồi như sau nếu topic tạo thành công

```
Created topic "test-topic".
```

Topic đã được tạo, bây giờ ta thử gửi thông điệp vào topic này:

```
>./bin/kafka-console-producer.sh --broker-list localhost:9092
--topic test-topic
```

Nhập vào hai thông điệp sau bằng cách typing và enter trên dòng lệnh

```
First test message

Second test message
```

Bạn đã gửi thông điệp vào topic, bây giờ bạn thử nhận thông điệp bằng câu lệnh sau:

```
>./bin/kafka-console-consumer.sh --zookeeper
s14e18f4e58324b66b78ecd8c831a0c6413-master.uitlab.com:2181 --
topic test-topic --from-beginning
```

Hai thông điệp đã gửi vào test-topic sẽ hiển thị

```
First test message

Second test message
```

Bạn đã thiết lập Apache Kafka Cluter thành công!

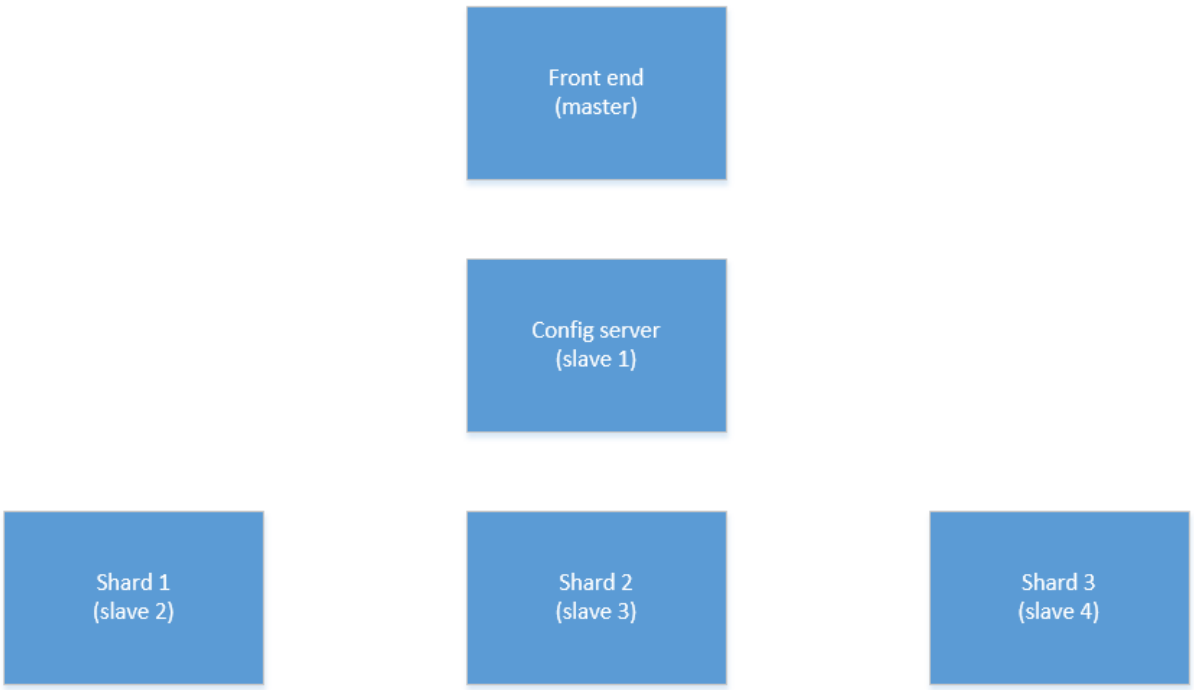
Nguồn tham khảo: https://www.allprogrammingtutorials.com/tutorials/setting-up-distributed-apache-kafka-cluster.php

### *1.1.3. Cài đặt mongodb cluster*

Kế hoạch chuẩn bị

Mongdb cluster có các thành phần cơ bản sau:

- Nhóm máy front-end: dùng để nhận kết nối từ client, trong vlab này nhóm dùng máy master để làm front-end
- Nhóm máy config-server: mongo cluster dùng để vận hành việc phân bố dữ liệu, trong vlab này nhóm dùng slave-1 để làm config-server
- Nhóm máy shard để lưu dữ liệu phân tán: trong vlab này nhóm sử dụng slave-2, slave-3, slave-4 lần lượt tương ứng với shard-1, shard-2, shard-3




Các bước cài đặt

Cài đặt mongodb lần lượt trên từng máy master, slave-1, slave-2, slave-3, slave-4:

Trên mỗi máy đều thực hiện các thao tác sau:

```
sudo yum install -y mongodb-org
```

```
sudo yum install -y mongodb-org-3.2.22 mongodb-org-server-3.2.22 mongodb-org-shell-3.2.22 mongodb-org-mongos-3.2.22 mongodb-org-tools-3.2.22
```

Theo mặc định, MongoDB chạy bằng user mongod và sử dụng các thư mục mặc định sau:

- /var/lib/mongo (the data directory)
- /var/log/mongodb (the log directory)

Trên mỗi máy shard (slave-2 = shard-1, slave-3 = shard-2, slave-4 = shard-3) ta tiến hành khởi động dịch vụ mongod theo cú pháp sau:

Tạo thư mục lưu trữ dữ liệu:

```
sudo mkdir -p /data/db
```

```
sudo chmod -R 777 /data/db
```

Thực hiện kill mongod nếu dịch vụ này đang chạy

```
killall -q mongod (nếu không kill được bằng user đang sử dụng thì dùng quyền sudo để kill process)
```

Trên máy slave-2:

```
mongod --shardsvr --replSet rs1 --dbpath /data/db --fork --logpath=/data/db/log.txt
```

Trên máy slave-3:

```
mongod --shardsvr --replSet rs2 --dbpath /data/db --fork --logpath=/data/db/log.txt
```

Trên máy slave-4:

```
mongod --shardsvr --replSet rs3 --dbpath /data/db --fork --logpath=/data/db/log.txt
```

Tên **rs1, rs2, rs3** là duy nhất trong cluster, nếu ta có thêm shard thì chúng ta cứ thêm vào theo cấu trúc như trên.

Khởi động tập các replication trên từng máy slave-2, slave-3, slave-4 bằng cách chạy câu lệnh sau trên từng máy:

```
mongo localhost:27018 --eval "JSON.stringify(rs.initiate())"
```

Chạy dịch vụ mongod trên máy cấu hình (config-server, ở đây sử dụng slave-1 làm config-server)

Tạo thư mục để lưu trữ config

```
sudo mkdir -p /data/configdb
```

```
sudo chmod -R 777 /data/configdb
```

kill process mongod nếu dịch vụ này đang chạy

```
killall -q mongod (nếu không kill được bằng user đang sử dụng thì dùng quyền sudo để kill process)
```

Chạy mongod trên config server bằng câu lệnh sau:

```
mongod --configsvr --replSet c1 --enableMajorityReadConcern --fork --logpath=/data/configdb/log.txt
```

Khởi động tập relication trên máy config bằng câu lệnh sau:

```
mongo localhost:27019 --eval "JSON.stringify(rs.initiate())"
```

Khởi động dịch vụ mongos trên máy tiền trạm (front-end)

Chạy câu lệnh sau trên máy master và giữ màn hình terminal:

```
mongos --configdb c1/s14e18f4e58324b66b78ecd8c831a0c6413-slave1.uitlab.com
```

Cấu hình cluster trên máy tiền trạm

Mở một màn hình terminal khác và chạy câu lệnh để kết nối đến mongos:

```
mongo localhost:27017
```

Tại màn hình terminal này, ta tiến hành thêm shards bằng câu lệnh sau:

```
sh.addShard("rs1/s14e18f4e58324b66b78ecd8c831a0c6413-slave2.uitlab.com:27018")
```

```
sh.addShard("rs2/ s14e18f4e58324b66b78ecd8c831a0c6413-slave3.uitlab.com:27018")
```

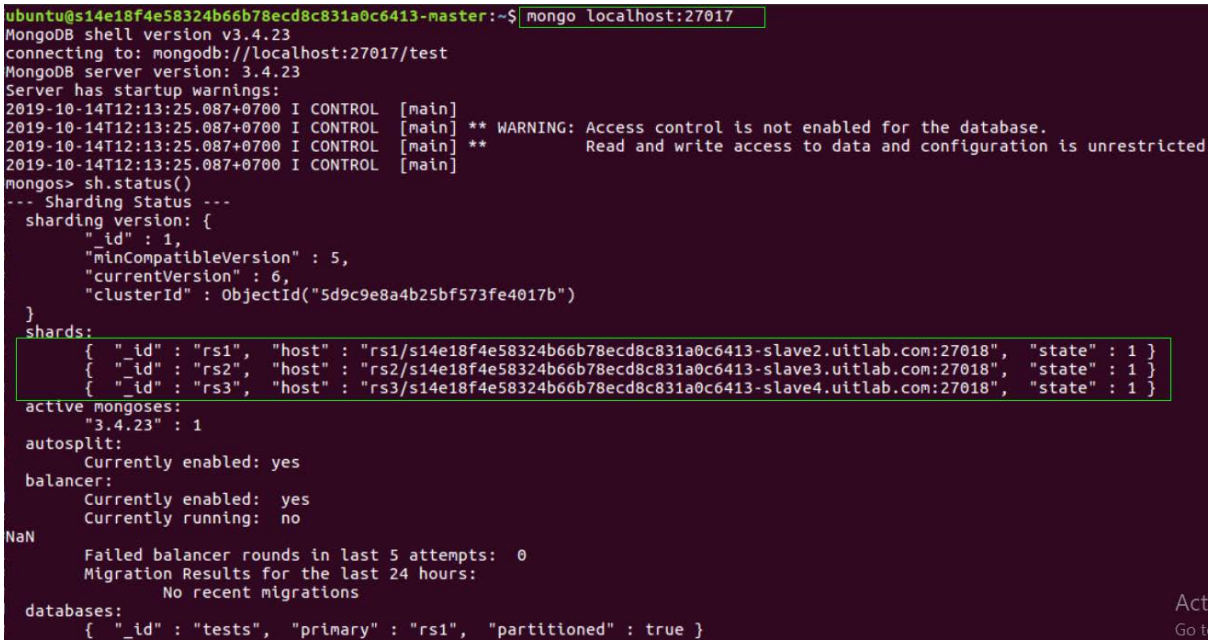
```
sh.addShard("rs3/ s14e18f4e58324b66b78ecd8c831a0c6413-slave4.uitlab.com:27018")
```

Tiến hành enable sharding cho database ta muốn phân tán, ví dụ trong trường hợp này là database “tests”

```
sh.enableSharding("tests")
```

Để kiểm tra việc cài đặt shards có thành công không ta dùng lệnh sau:

```
sh.status()
```


ubuntu@s14e18f4e58324b66b78ecd8c831a0c6413-master:~$ mongo localhost:27017
MongoDB shell version v3.4.23
connecting to: mongodb://localhost:27017/test
MongoDB server version: 3.4.23
Server has startup warnings:
2019-10-14T12:13:25.087+0700 I CONTROL [main]
2019-10-14T12:13:25.087+0700 I CONTROL [main] ** WARNING: Access control is not enabled for the database.
2019-10-14T12:13:25.087+0700 I CONTROL [main] ** Read and write access to data and configuration is unrestricted
2019-10-14T12:13:25.087+0700 I CONTROL [main]
mongos> sh.status()
--- Sharding Status ---
sharding version: {
"_id" : 1,
"minCompatibleVersion" : 5,
"currentVersion" : 6,
"clusterId" : ObjectId("5d9c9e8a4b25bf573fe4017b")
}
shards:
{ "_id" : "rs1", "host" : "rs1/s14e18f4e58324b66b78ecd8c831a0c6413-slave2.uitlab.com:27018", "state" : 1 }
{ "_id" : "rs2", "host" : "rs2/s14e18f4e58324b66b78ecd8c831a0c6413-slave3.uitlab.com:27018", "state" : 1 }
{ "_id" : "rs3", "host" : "rs3/s14e18f4e58324b66b78ecd8c831a0c6413-slave4.uitlab.com:27018", "state" : 1 }
active mongoses:
"3.4.23" : 1
autosplit:
Currently enabled: yes
balancer:
Currently enabled: yes
Currently running: no
NaN
Failed balancer rounds in last 5 attempts: 0
Migration Results for the last 24 hours:
No recent migrations
databases:
{ "_id" : "tests", "primary" : "rs1", "partitioned" : true }

# 2. HỆ THỐNG PHÂN TÍCH CHIẾN DỊCH QUẢNG CÁO

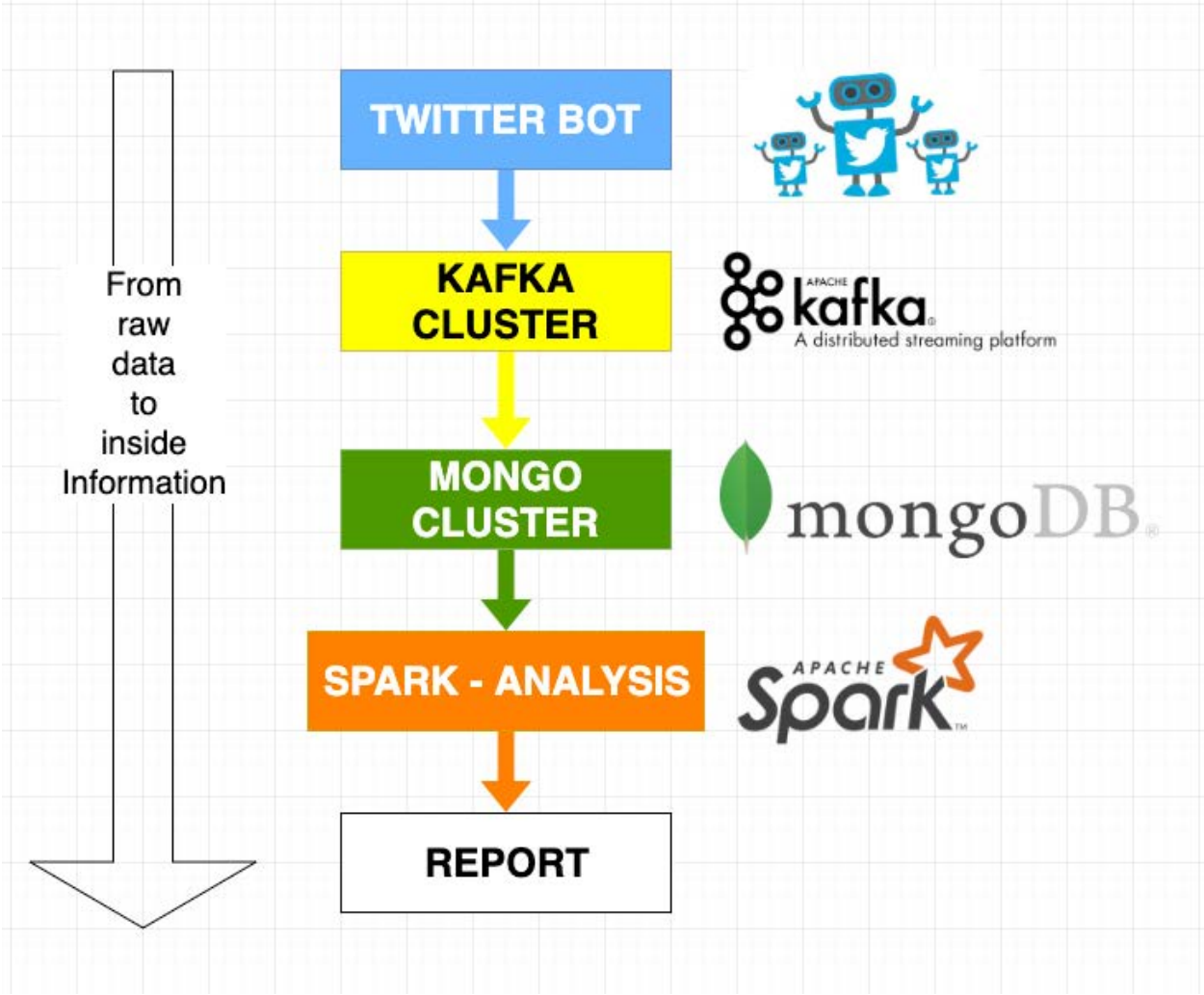
## 2.1. Bài toán đặt ra

Xây dựng một hệ thống phân tích thời gian thực tình hình của một chiến lược quảng cáo, marketing cho một đơn vị Agency bất kỳ.

Yêu cầu của hệ thống:

- Cần phải thu thập được tất các tweet đang nói về chiến dịch quảng cáo
- Dữ liệu cần được sẵn sàng với độ trễ thấp nhất có thể
- Hệ thống phải có khả năng chịu lỗi cũng như tránh mất mát dữ liệu

## 2.2. Kiến trúc tổng quát




## 2.3. Chi tiết

### *2.3.1. Twitter bot*

Sử dụng twint để lấy tweet

Twint là một thư viện python được quảng cáo là lấy được tweet của người dùng không cần thông qua Twitter API. Ưu thế:

- Lấy được toàn bộ Tweets. So với Twitter API bị giới hạn chỉ 3200 Tweets
- Không cần account Twitter
- Không có giới hạn
- Cài đặt nhanh đơn giản thông qua pip

Ví dụ một đoạn code đơn giản để lấy tất cả các tweet có hashtag #BTS

```
c = twint.Config()
c.Search = '#BTS'
c.Store_json = True
c.Store_object = True
c.Output = "tweets.json"
twint.run.Search(c)
```

### *2.3.2. Kafka*

Kafka là một nền tảng streaming phân tán, mã nguồn mở.

Từ những yêu cầu của bài toán nhóm quyết định sử dụng kafka để streaming tweets. Lý do:

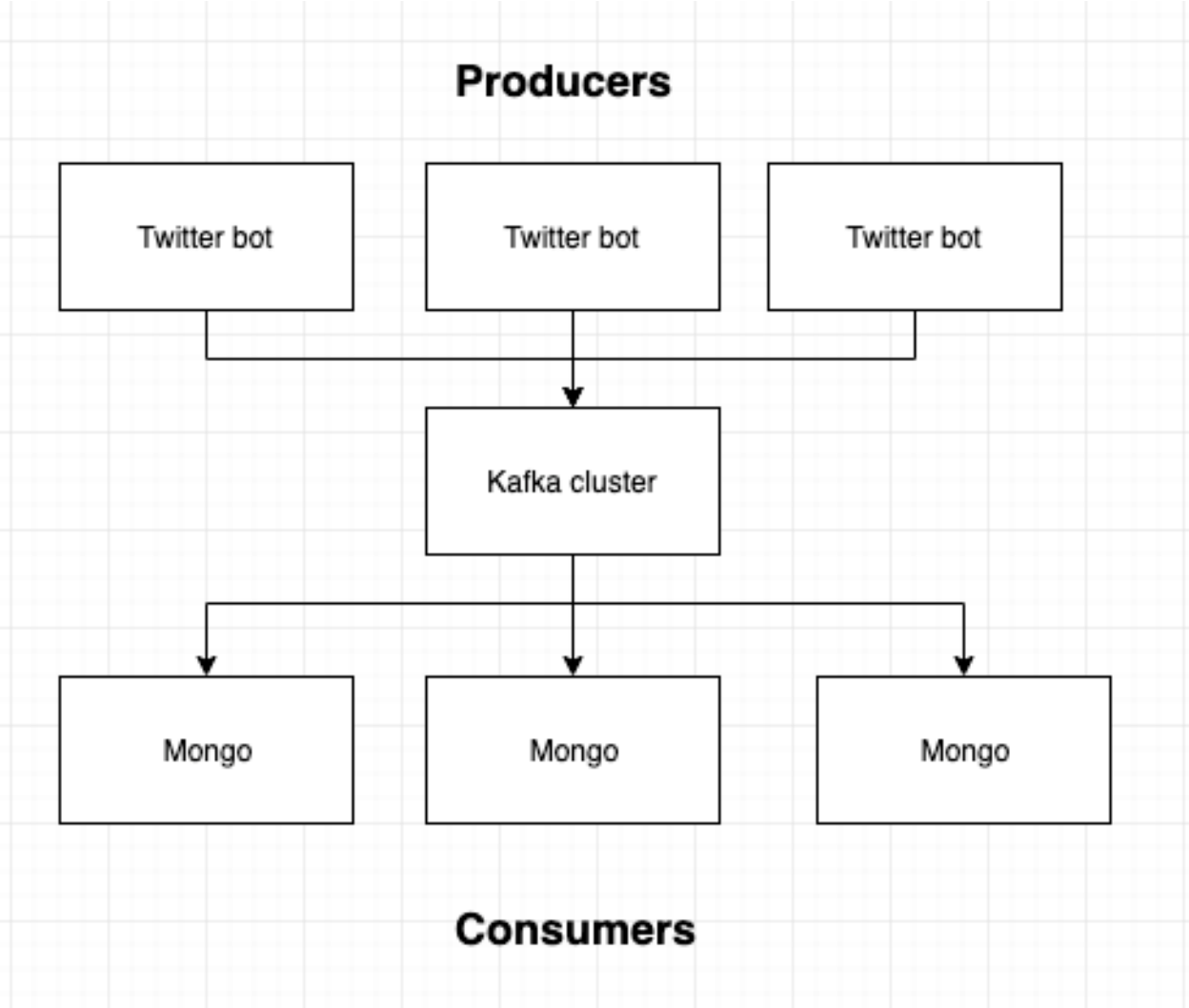
- Số lượng tweet về chủ đề quảng cáo đến dồn dập (Có thể lên tới 200,000 messages). Kafka được xây dựng để có thể số lượng lên đến hàng triệu messages.
- Kafka được triển khai trên hệ thống cluster gồm 5 máy (đã được nói rõ ở phần cài đặt). Giúp đảm bảo tính chịu lỗi của hệ thống (một máy trong cụm bị hư hệ thống vẫn hoạt động). Hơn thế nữa, các messages được nhân bản (replicate) trong cụm cluster đảm bảo việc toàn vẹn của dữ liệu. Messages được lưu trên ổ cứng theo batch để tối ưu việc đọc/ghi.
- Kafka có tính cluster nên hệ thống dễ dàng được mở rộng theo chiều ngang (việc thêm các máy vào cluster được thiết kế rất dễ dàng). Kafka được thiết kế để xây

dựng các ứng dụng có yêu cầu realtime. Kiểu dữ liệu đơn thuần là byte array khiến cho kafka linh hoạt xử lý được nhiều dạng messages (JSON, ProtoBuf, Avro hoặc các kiểu tự định nghĩa).

- So với mô hình truyền thống như cơ sở dữ liệu quan hệ, chúng ta phải thiết kế một vòng loop để liên tục kiểm tra dữ liệu mới. Với việc ra đời của kafka công việc trở nên dễ dàng hơn trước rất nhiều. Dữ liệu mới sẽ được tự động cập nhập cho các consumers.

Thiết kế:

Các producers là các twitter bots có nhiệm vụ thu thập tweet, các consumers là mongoDB có nhiệm vụ lưu trữ tweet




### *2.3.3. Mongo*

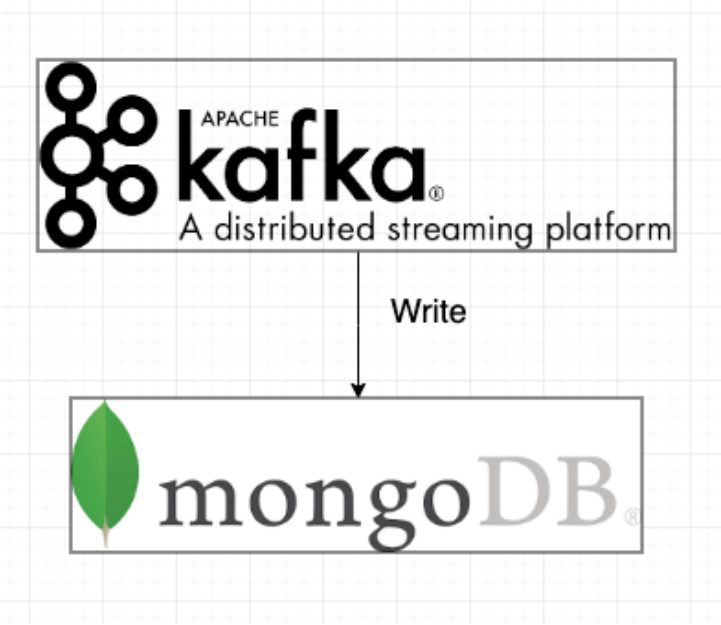
MongoDB là một hệ cơ sở dữ liệu phi quan hệ (no sql).

Lý do nhóm chọn mongoDB để lưu trữ tweet:

- Tweet có dạng json, hiện tại không có cấu trúc xác định cho tất cả các tweet để có thể ánh xạ thành các trường và bảng như trong cơ sở dữ liệu quan hệ. Mongo được thiết kế để lưu trữ dữ liệu không có định dạng, dễ dàng thêm xoá sửa thông qua key-value
- MongoDB có thể lưu trữ ảnh và video dưới dạng GridFS, rất tiện lợi cho việc mở rộng hệ thống sau này. Hơn thể MongoDB hỗ trợ lưu trữ mặc định (không cần cấu hình) nhiều loại ngôn ngữ như tiếng Việt, tiếng Anh, tiếng Thái …
- MongoDB được nhóm triển khai dạng cluster gồm 5 máy (sẽ được nói rõ ở phần cài đặt). Điều này giúp cho MongoDB có khả năng mở rộng theo chiều ngang (gắn thêm máy mới), dữ liệu được nhân bản giữa các máy theo cơ chế sharding để tránh mất mát.
- Do hệ thống yêu cầu về thời gian thực (realtime) nên tần suất đọc/ghi sẽ rất lớn. Tốc độ đọc/ghi của MongoDB nhanh hơn rất nhiều so với MySQL. Đây chính là ưu điểm của MongoDB so với hệ cơ sở dữ liệu quan hệ.

Thiết kế:

Dữ liệu tweet dạng JSON sẽ được Kafka chuyển tới MongoDB



### *2.3.4. Báo cáo*

Để hiển thị báo cáo nhóm sử dụng thư viện python có tên Dash.

Dash hỗ trợ nhiều loại biểu đồ khác nhau như: cột, đường, tròn … Ngoài ra thư viện còn hỗ trợ vẽ theo thời gian thực phù hợp với yêu cầu của đồ án.

## 2.4.Cài đặt phần mềm

Nhóm muốn theo dõi hiệu xuất của hệ thống, dữ liệu có thật sự đổ về dồn dập ở hệ thống kafka và tần xuất đọc ghi của MongoDB nên nhóm quyết định sử dụng 2 phần mềm sau:  Grafana, prometheus

### *2.4.1. Prometheus*

Download phiên bản prometheus

https://prometheus.io/download/

Phiên bản nhóm sử dụng:

prometheus-2.13.1.linux-amd64.tar.gz

Giải nén bằng câu lệnh sau:

tar xvfz prometheus-*.tar.gz

Truy cập vào thư mục vừa giải nén, sửa lại prometheus.ym như sau

```
# my global config
global:
  scrape_interval:  15s # Set the scrape interval to every 15
seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds.
The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
```

```
  - static_configs:
      - targets:
      # - alertmanager:9093

# Load rules once and periodically evaluate them according to
the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to
scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any
timeseries scraped from this config.
  - job_name: 'kafka'

      # metrics_path defaults to '/metrics'
      # scheme defaults to 'http'.

      static_configs:
      - targets: ['localhost:9090', 'localhost:7071',
'localhost:9216']
```

Trong đó, ta cần **lưu ý** dòng

*- targets: ['localhost:9090', 'localhost:7071', 'localhost:9216']*

*localhost:9090 là địa chỉ của prometheus*

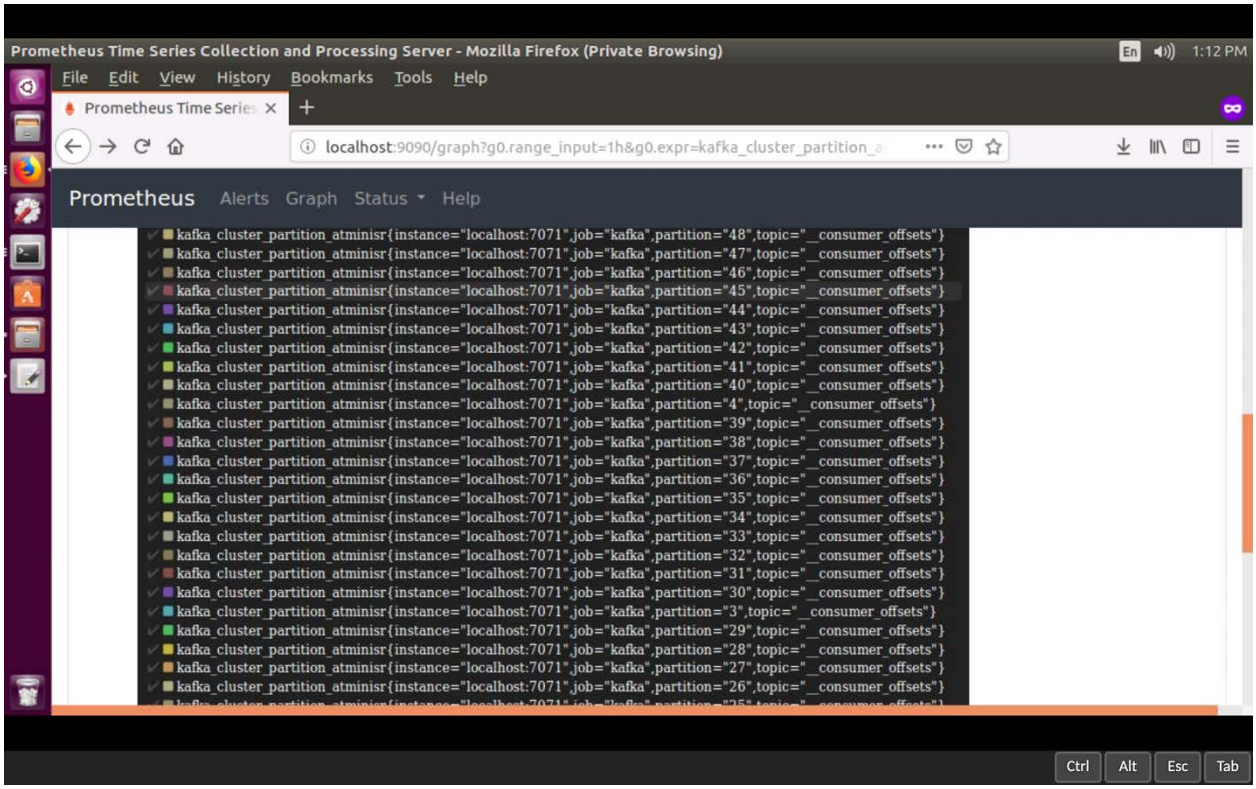
*localhost:7071 là địa chỉ của JXM exporter (sẽ được giải thích ở phần sau)*

*localhost:9216 là địa chỉ của mongo db*

Khởi động prometheus bằng câu lệnh sau:

```
./prometheus --config.file=prometheus.yml
```

Truy cập vào địa chỉ localhost:9090 để xem giao diện Prometheus




### *2.4.2. Grafana*

Do giao diện Prometheus quá hạn chế ta cần một giao diện quản lý mới, hỗ trợ nhiều loại biểu đồ khác nhau. Grafana là ứng cử viên sáng giá cho nhiệm vụ này.

Chúng ta tiến hành download Grafana bằng câu lệnh sau

```
wget https://dl.grafana.com/oss/release/grafana-6.4.3.linux-amd64.tar.gz
```

Sau đó tiến hành giải nén

```
tar -zxvf grafana-6.4.3.linux-amd64.tar.gz
```

Khởi động Grafana bằng câu lệnh sau

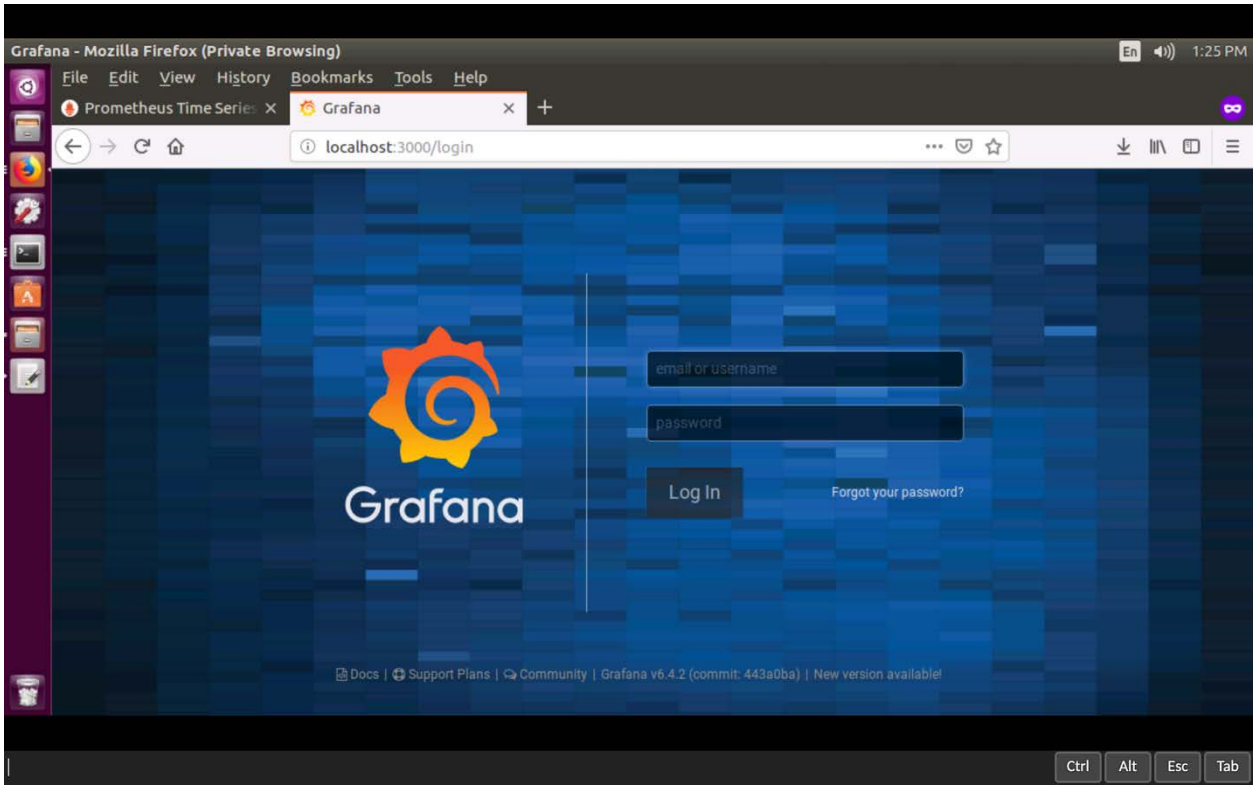
```
./bin/grafana-server web
```

Truy cập vào địa chỉ: localhost:3000

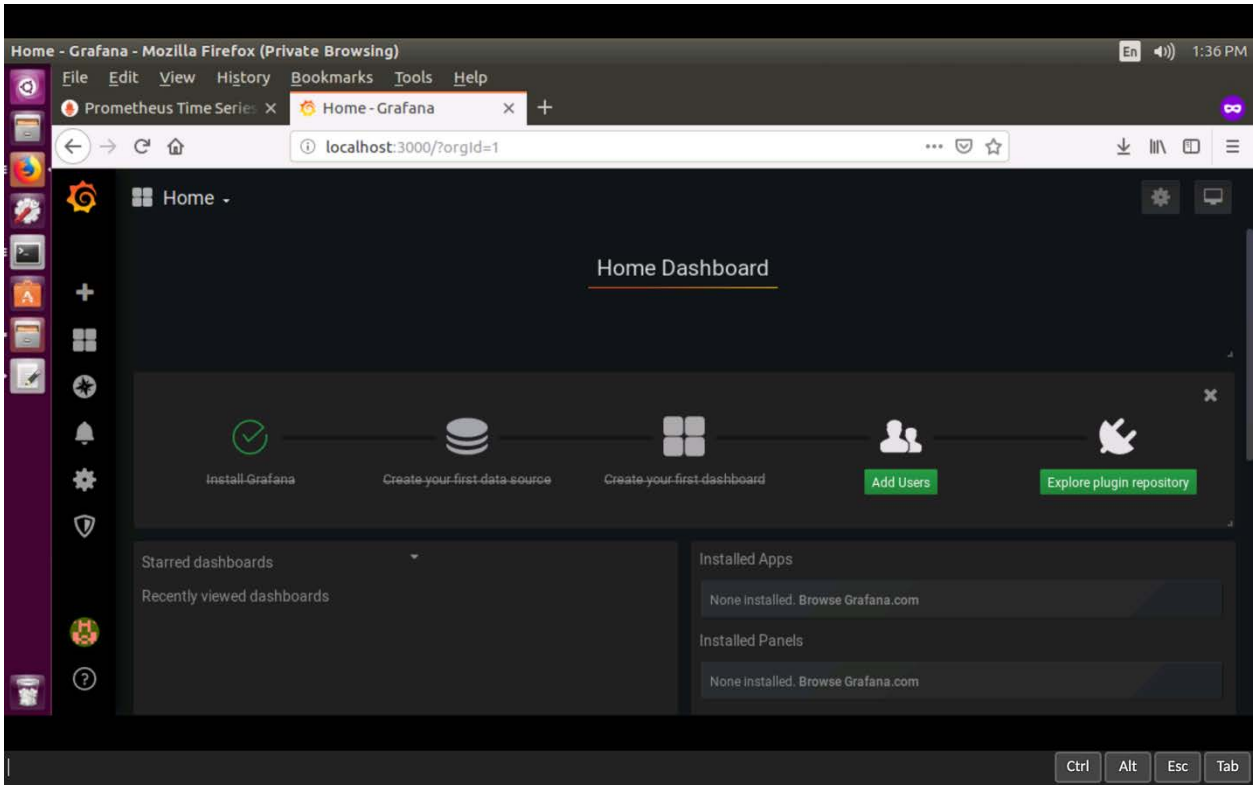
Tài khoản mặc định là: admin/admin

Bạn sẽ được yêu cầu đổi mật khẩu mới

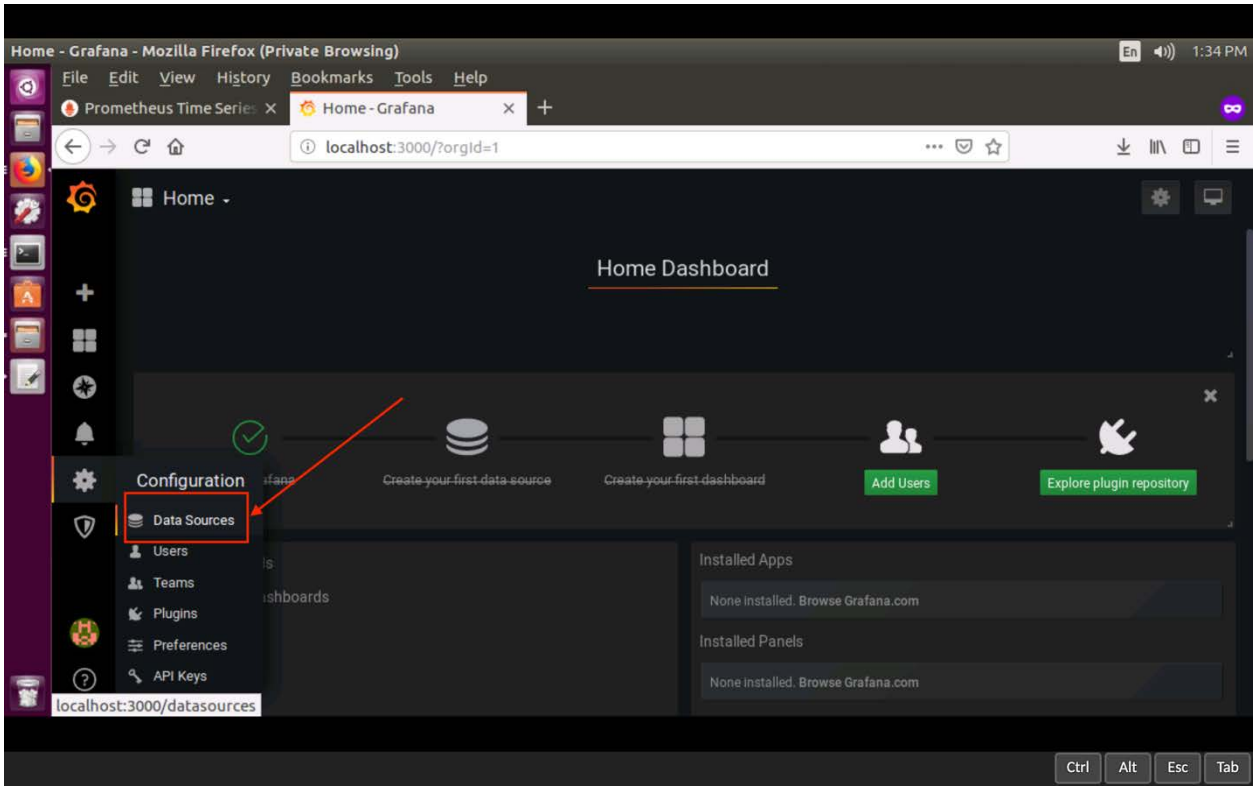
Giao diện Grafana

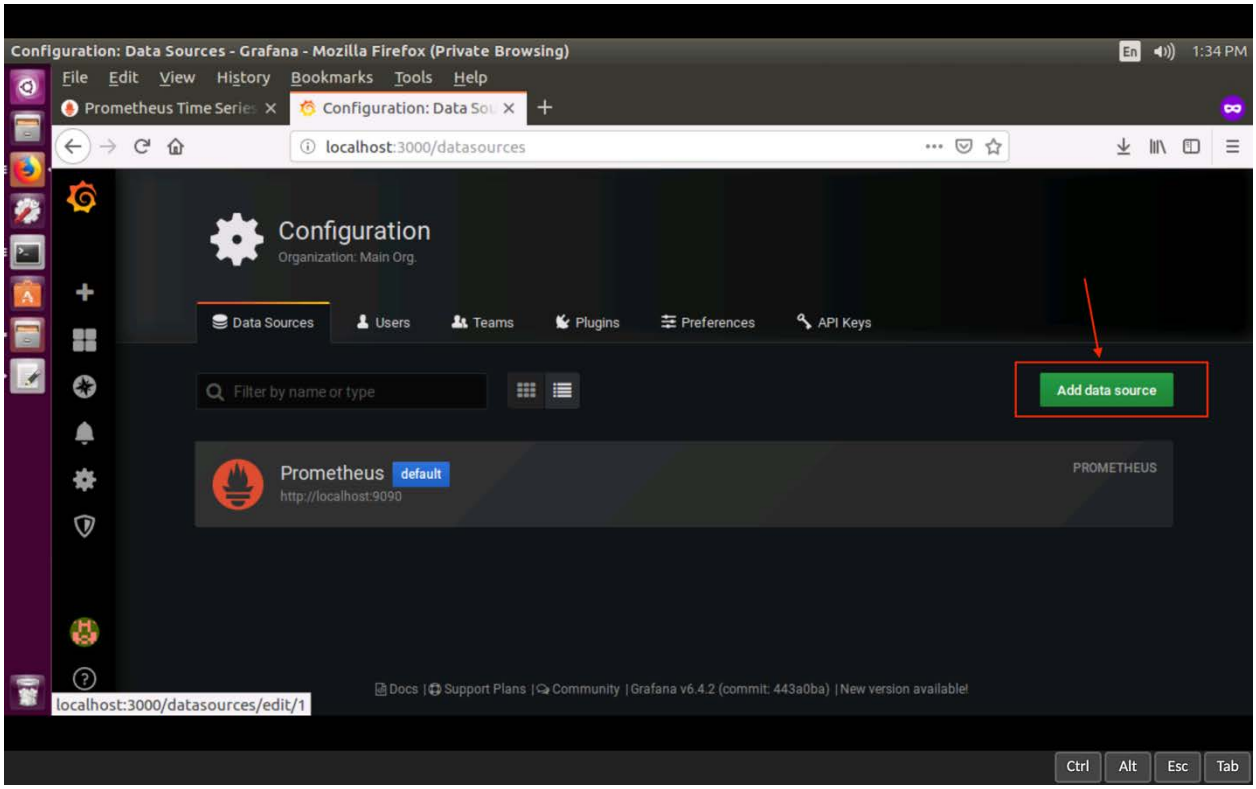



Sau khi đăng nhập

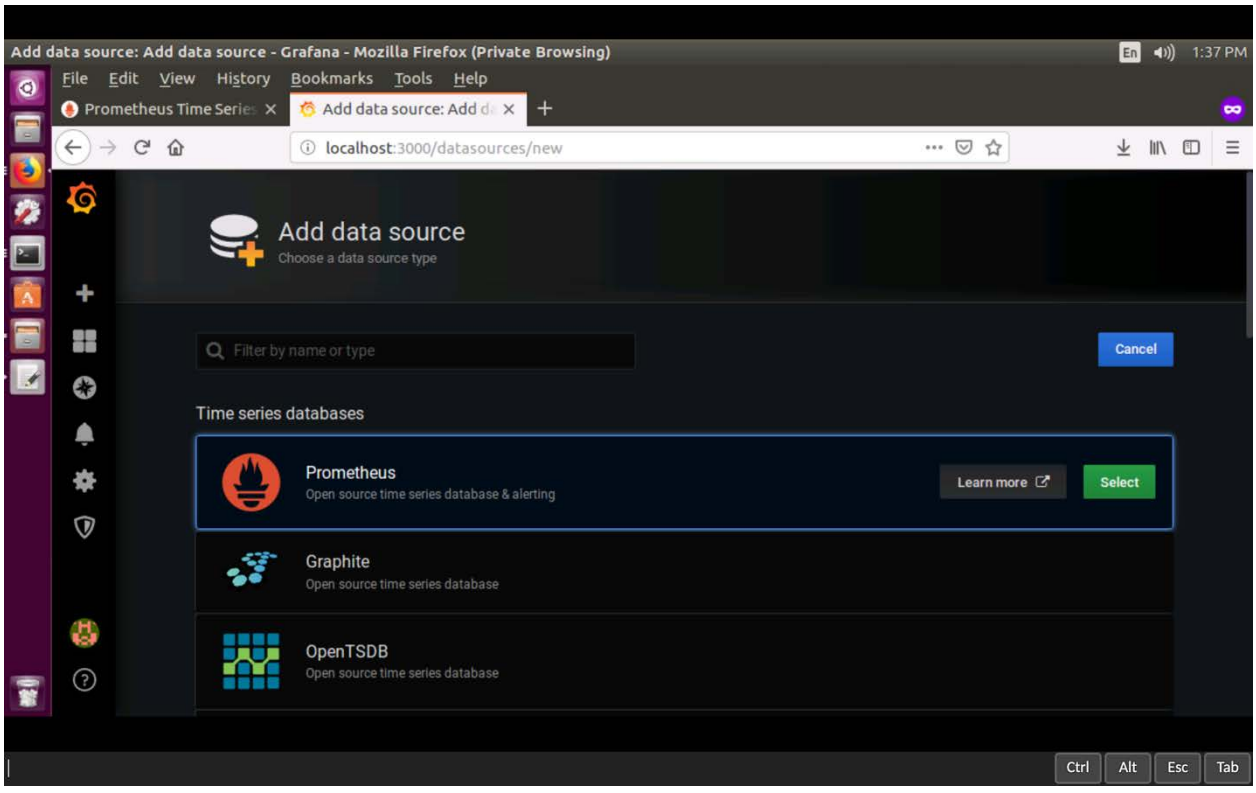



Ta chọn DataSource -> Add data source



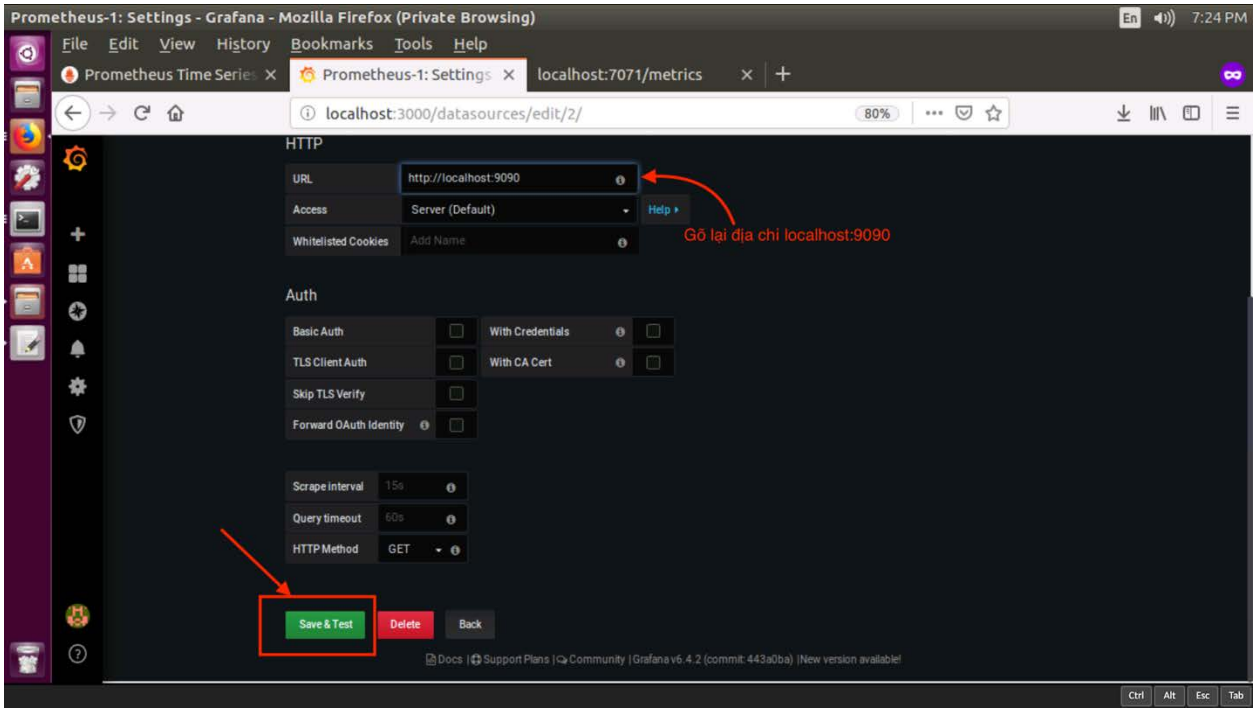



Chọn Prometheus




Các thông số kia để mặc định NHƯNG lưu ý phần URL, cần phải được gõ bằng tay url của prometheus là: localhost:9090

Sau đó ấn Save & Test




### *2.4.3. Cấu hình quản lý kafka*

Sau kết nối prometheus và Grafana, để quản lý Kafka ta cần cấu hình để prometheus cũng như Grafana có thể đọc dữ liệu từ Kafka

Ta cần cài JMX exporter

Download JMX exporter tại địa chỉ sau:

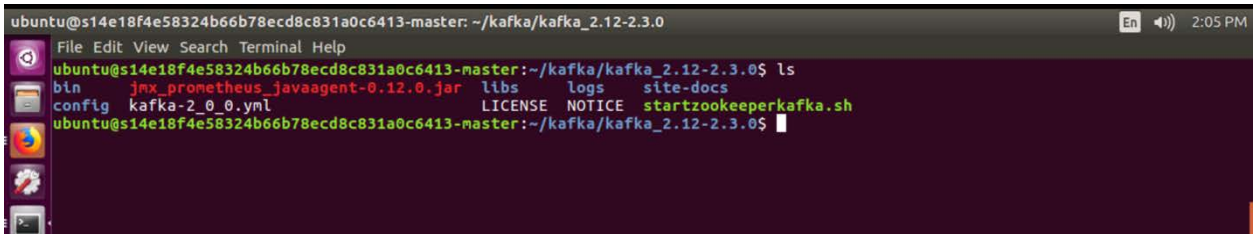
https://repo1.maven.org/maven2/io/prometheus/jmx/jmx_prometheus_javaagent/0.12.0/jmx_prometheus_javaagent-0.12.0.jar

Download file config JXM exporter tại địa chỉ sau:

https://github.com/prometheus/jmx_exporter/tree/master/example_configs

Chọn đúng phiên bản kafka cài đặt ở đây nhóm chọn kafka-2_0_0.yml

Sau đó tiến hành copy file jar này và file config vào đường dẫn cài đặt kafka tương ứng

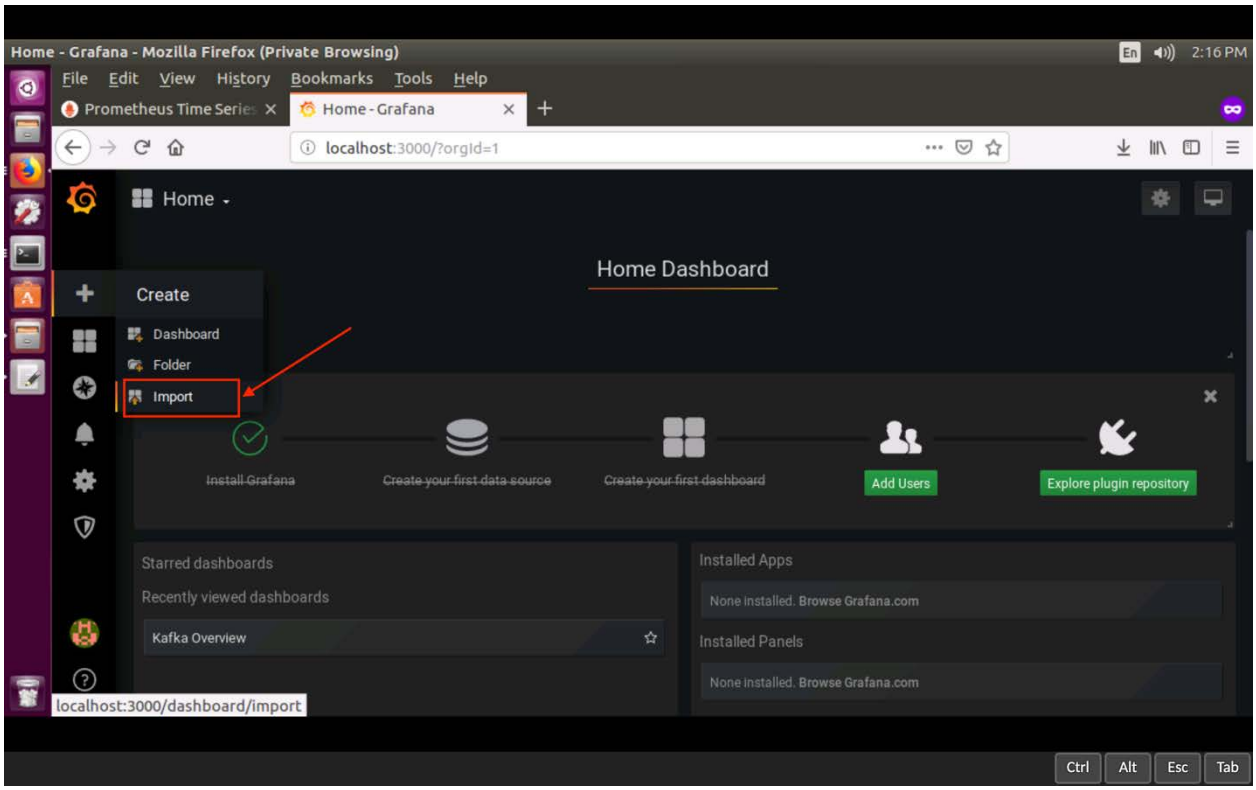



Tiến hành khởi động lại kafka theo câu lệnh sau:

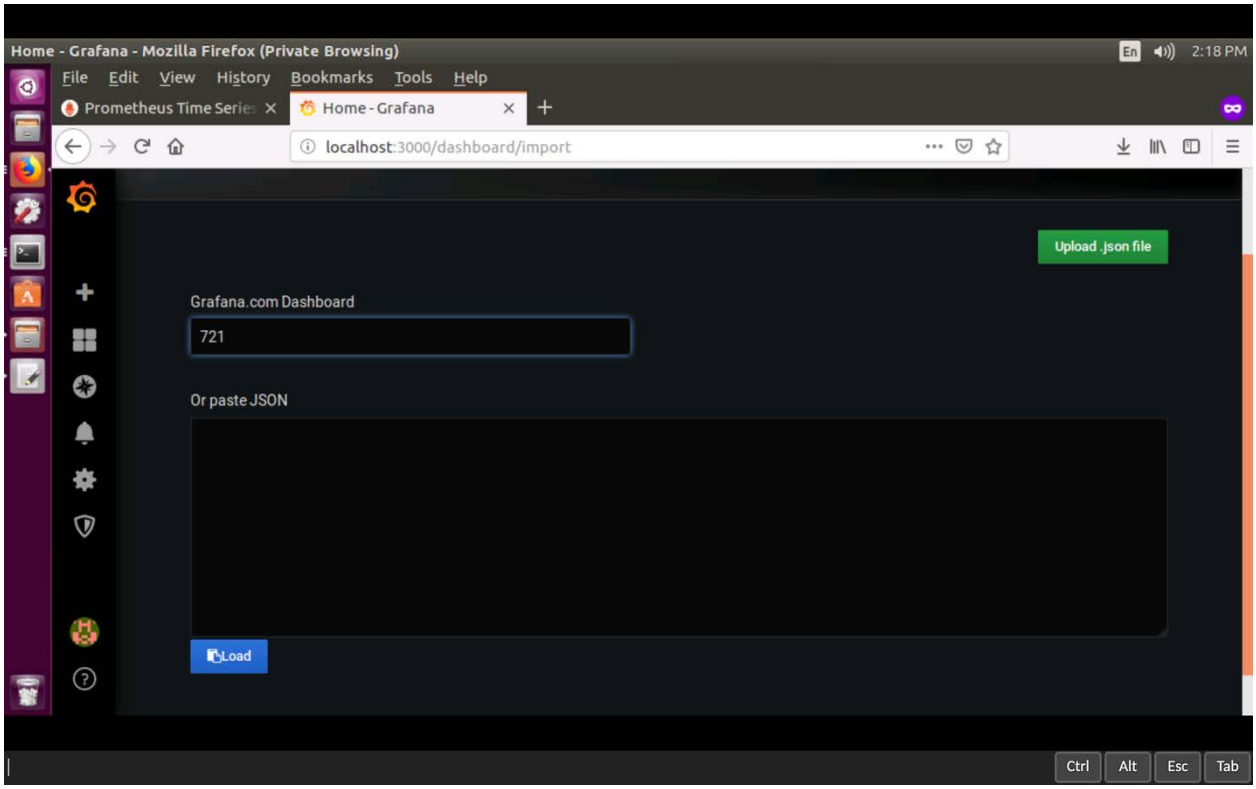
```
./bin/zookeeper-server-start.sh config/zookeeper.properties &

KAFKA_OPTS="$KAFKA_OPTS -
javaagent:$PWD/jmx_prometheus_javaagent-
0.12.0.jar=7071:$PWD/kafka-2_0_0.yml" ./bin/kafka-server-
start.sh config/server.properties
```

Hiện thị Kafka dashboard

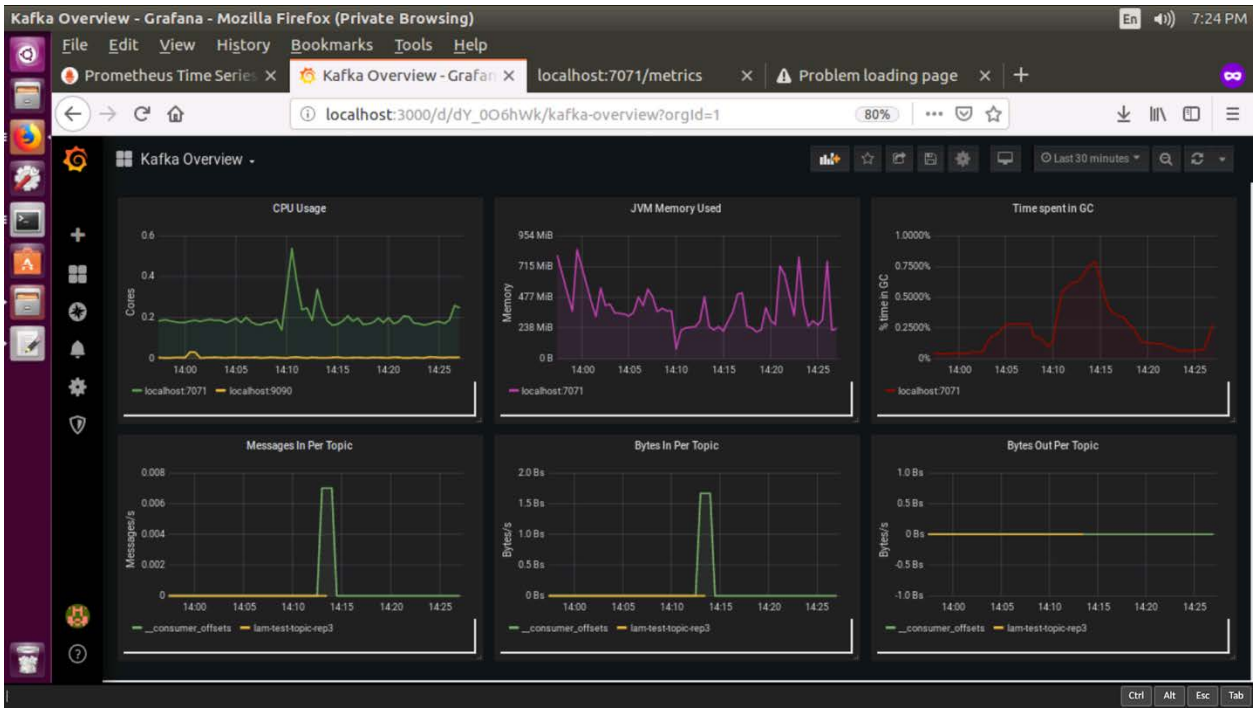
Ở trang chủ Grafana (localhost:3000). Ta chọn Dashboard -> Import




Nhập 721 vào Grafana.com dashboard và chọn Load




Đây là giao diện cuối cùng




### *2.4.4. Cấu hình quản lý MongoDB*

Ta cũng tiến hình cấu hình để prometheus và Grafana có thể đọc được dữ liệu từ MongoDB phục vụ công việc quản lý

Ta cần cài Mongo Exporter

https://github.com/percona/mongodb_exporter/releases

Nhóm chọn mongodb_exporter-0.10.0.linux-amd64.tar.gz

Tiến hành giải nén

```
tar xvzf mongodb_exporter-0.10.0.linux-amd64.tar.gz
```

Tiến hành tạo một tài khoản MongoDB với quyền hạn admin

Kết nối tới MongoDB bằng câu lệnh

```
mongo
```

Tạo tài khoản

```
use admin
db.createUser(
  {
    user: "nhom_13",
    pwd: "123456",
    roles: [
        { role: "clusterMonitor", db: "admin" },
        { role: "read", db: "local" }
    ]
  }
)
```

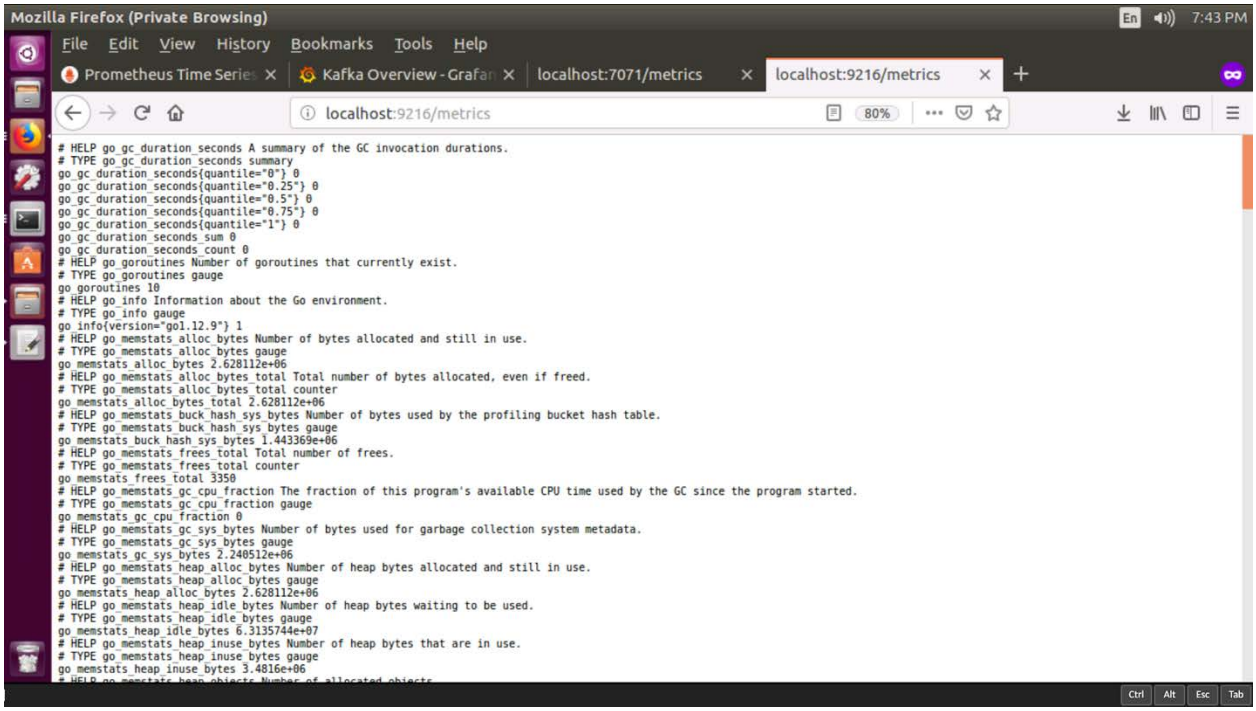
Gõ exit để thoát khỏi mongo

**Lưu ý:** Ở đây ta không cần phải tạo service cũng như thiết lập biến môi trường như các hướng dẫn trên mạng

Chạy Mongo Exporter với tài khoản vừa tạo trên bằng câu lệnh sau ()

```
./mongodb_exporter --
mongodb.uri=mongodb://nhom_13:123456@localhost:27017
```

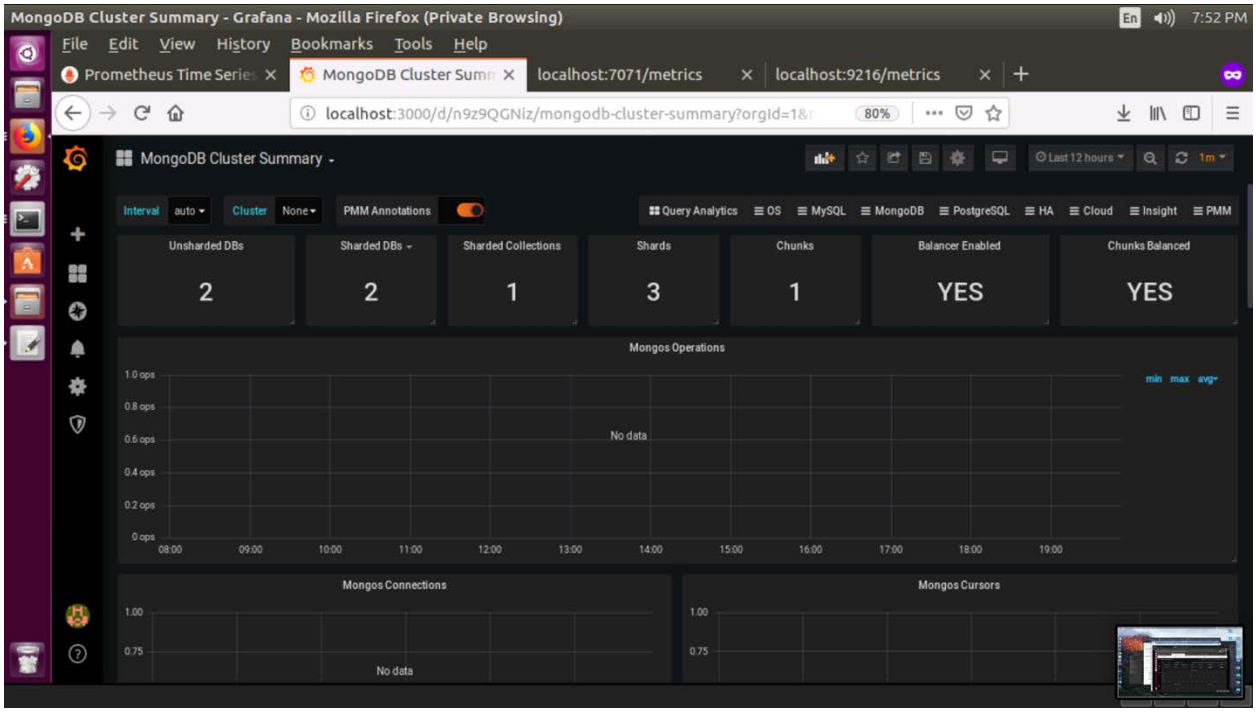
Truy cập địa chỉ localhost:9216/metric để xem mongo exporter




Hiện thị MongoDB dashboard

Ở trang chủ Grafana (localhost:3000). Ta chọn Dashboard -> Import. Nhập 7353 vào Grafana.com dashboard và chọn Load. Tương tự như ở Kafka dashboard.

Giao diện MongoDB dashboard




## 2.5. Cài đặt

Ta tiến hành chạy script sau để thu thập tweet

```
python scrape_tweet.py
```

**scrape_tweet.py** có nhiệm vụ lấy tweet về chủ đề được cho trước và đẩy vào một topic kafka có tên tweets

Ta tiến hành chạy script sau để lưu trữ tweet vào MongoDB

```
python store_tweet.py
```

**store_tweet.py** có nhiệm vụ đọc topic tweets, lấy ra các tweet dưới dạng JSON và lưu trữ xuống MongoDB

Ta tiến hành phân tích tình thái của tweet bằng script. Chi tiết phần phân tích này đã được trình bày ở phần spark

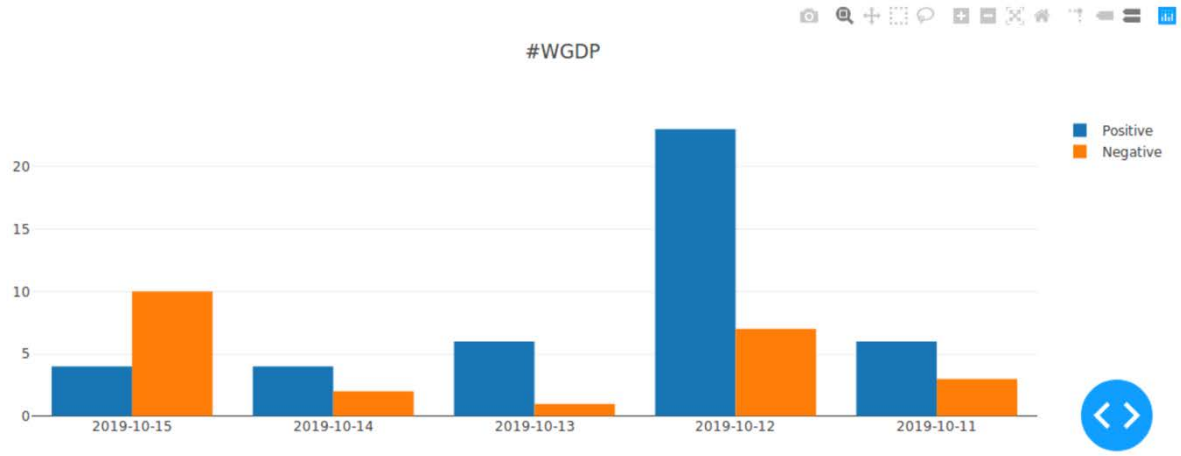
```
python analyze_tweet.py
```

**analyze_tweet.py** có nhiệm vụ gán nhãn do tweet ở MongoDB là tích cực hay tiêu cực (positive/negative)

Để hiện thị thành dạng báo cáo chúng ta dùng script sau:

```
python dashboard.py
```

**dashboard.py** có nhiệm vụ hiện thị biểu đồ cột số lượng tweet mang giá trị tích cực/tiêu cực được nhóm theo ngày. Dưới đây là kết quả các tweet thuộc về hashtag: #WDC trong năm này từ 11 đến 15 tháng 10

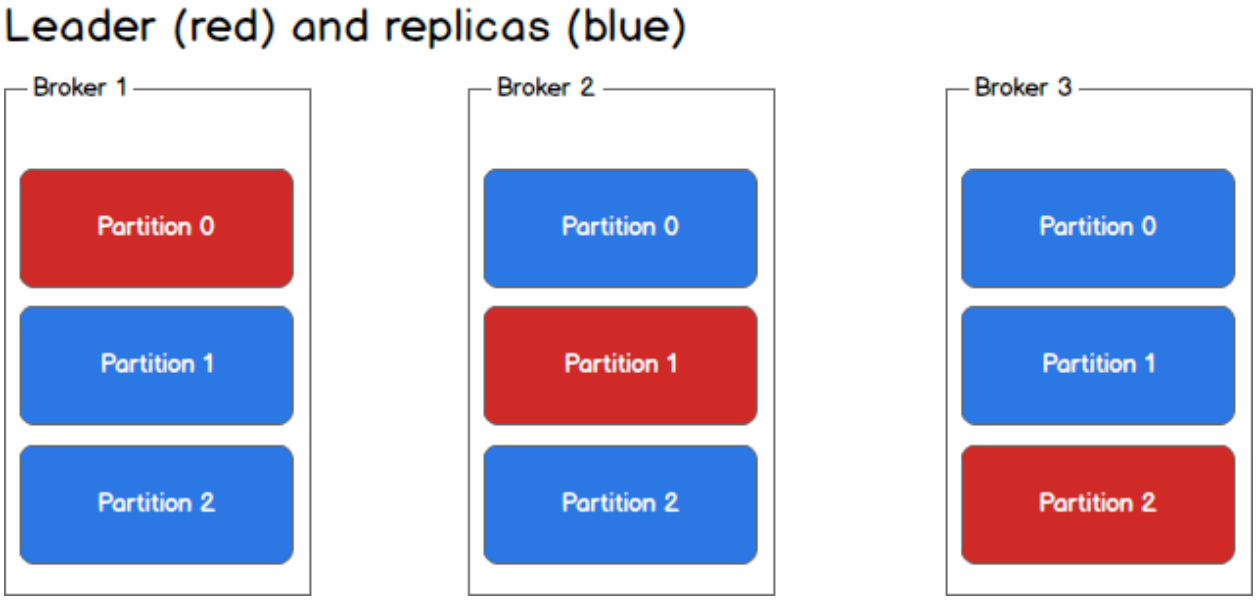



## 2.6.Mở rộng

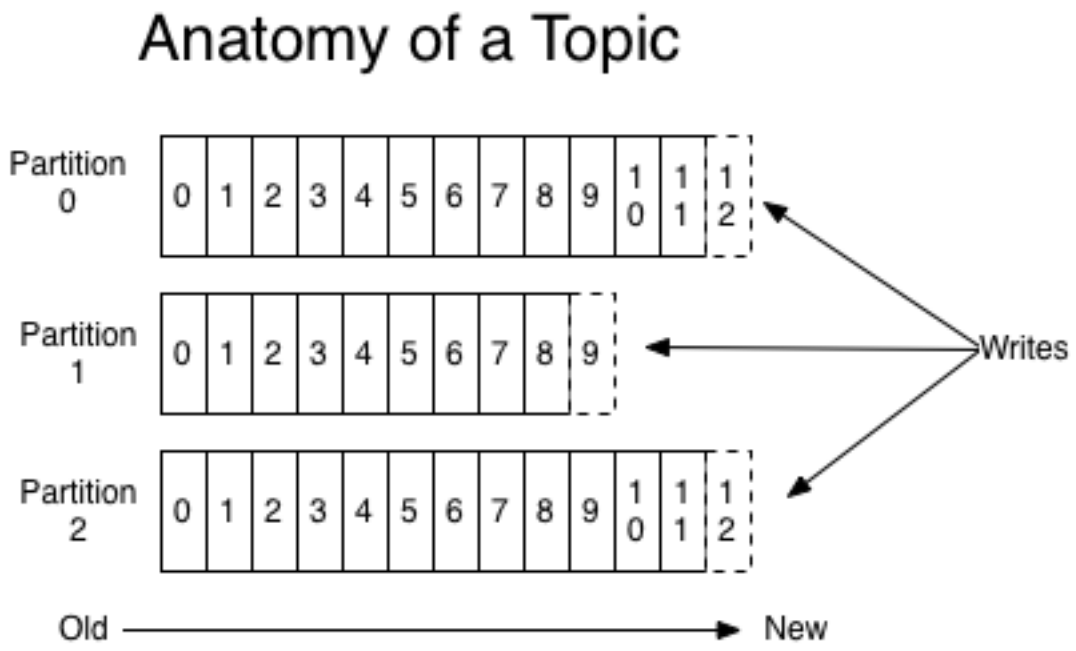
### *2.6.1. Một số lưu ý khi làm việc với Kafka*

Kafka có 2 thành phần cần chú ý và chúng có mối liên hệ mật thiết với nhau: Topic và consumer.

Topic: topic có thể được chia thành nhiều partitions. Nếu có 3 partitions thì sẽ có 1 làm leader và 2 làm replicas. Mỗi messages được ghi tại một offset

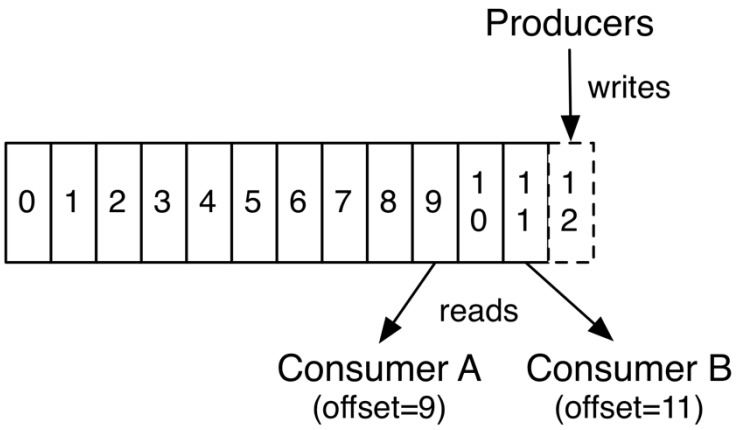



Hình topic với 3 partitions. Nguồn link




Hình topic với 3 partitions. Nguồn [link]

- Consumer: Đọc messages từ topic bằng cách subscribe vô topic đó




Hình đọc messages từ topic. Nguồn [link]

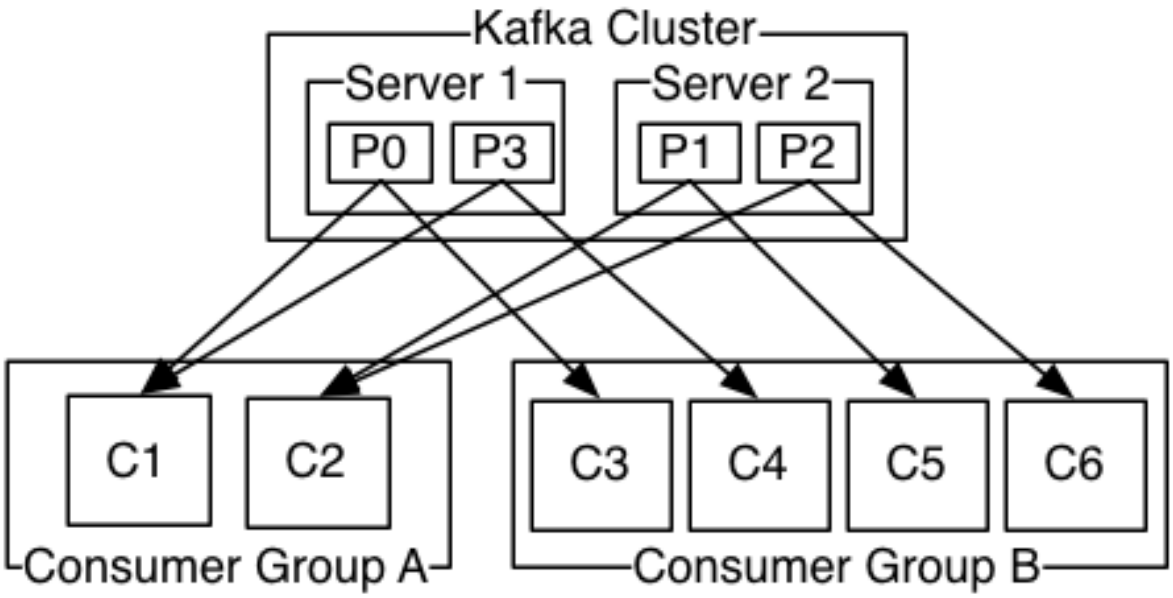
**Lưu ý:**

Khi làm việc ta cần chú ý kỹ số lượng partition và số lượng consumer, để tránh tình trạng bị mất messages, hệ thống chạy không đúng như mong đợi.

Xét trường hợp một topic và các consumer thuộc về cùng 1 group-id

Nếu số lượng consumers lớn hơn số lượng partitions thì sẽ có một số consumers ở trạng thái idle (đứng yên) vì không có đủ partition để đọc. Nếu số lượng partitions lớn hơn consumers thì mỗi consumers sẽ nhận được messages từ nhiều hơn 1 partitions.

Do đó khi cài đặt để đảm bảo số lượng messages luôn nhận được đủ ở các consumers thì ta phải khởi tạo chúng với các group-id khác nhau.




Hình consumer group. Nguồn link P: viết tắt chữ Partition

**Thực tế**

Câu lệnh tạo topic

```
bin/kafka-topics.sh --create --bootstrap-server
localhost:9092 --replication-factor 3 --partitions 4 --topic
13-group-topic
```

Ta tạo topic tên 13-group-topic với partition = 4, replicate = 3

Code python tạo Kafka consumer

```
from kafka import KafkaConsumer
```

```
scrape_consumer = KafkaConsumer('13-group-topic',
group_id='scrape-tweet')
analysis_consumer = KafkaConsumer('13-group-topic',
group_id='analyze-tweet')
```

Tạo 1 consumer lắng nghe topic tên 13-group-topic với group-id = scrape-tweet

Do ứng dụng của nhóm gồm tập hợp nhiều consumer lắng nghe (subscribe) trên một topic. Các consumers này đảm nhiệm các nhiệm vụ khác nhau như lưu trữ, phân tích, vẽ biểu đồ … Nên nhóm gán mỗi consumer với một group-id khác nhau, để đảm bảo không bỏ xót một tweet nào.

# 3. SPARK – PHẦN TÍNH TOÁN PHÂN TÁN TRONG HỆ THỐNG

## 3.1. Giới thiệu tổng quan

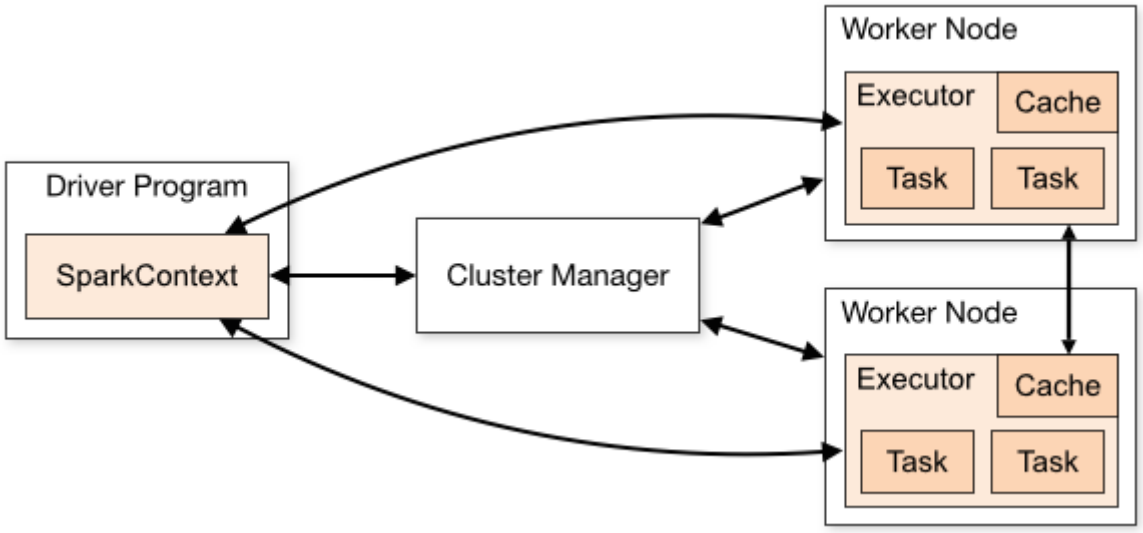
Apache Spark là một nền tảng tính toán theo cụm (cluster), đa mục đích. Spark cung cấp các API bằng Java, Scala, Python và R và được tối ưu hóa hỗ trợ việc tính toán phân tán. Spark hỗ trợ một bộ công cụ cấp cao bao gồm Spark SQL xử lý dữ liệu có cấu trúc, MLlib cho máy học, GraphX để xử lý đồ thị và Spark Streaming.

Spark chạy trên cả hệ thống Windows và UNIX (ví dụ: Linux, Mac OS). Spark dễ dàng chạy cục bộ trên một máy - cần cài đặt java trên hệ thống PATH hoặc biến môi trường JAVA_HOME trỏ đến nơi cài đặt Java.

Spark chạy trên Java 8, Python 2.7 + / 3.4 + và R 3.1+. Đối với API Scala, Spark 2.4.4 sử dụng Scala 2.12.

## 3.2. Mô hình Spark Cluster

Các ứng dụng Spark chạy dưới dạng các process độc lập trên một cụm, được điều phối bởi đối tượng SparkContext trong chương trình chính (được gọi là driver program). Cụ thể, để chạy trên một cụm, SparkContext kết nối với một số loại trình quản lý cụm (có thể là trình quản lý cụm độc lập của riêng Spark, Mesos hoặc YARN) nhằm phân bổ tài nguyên của các máy trong cluster đó. Sau khi kết nối, Spark tạo ra các executor, đó là các process để chạy các tính toán và lưu trữ dữ liệu cho ứng dụng. Tiếp theo, Spark sẽ gửi mã ứng dụng (được xác định bởi các tệp JAR hoặc file Python được truyền cho SparkContext) cho executor. Cuối cùng, SparkContext gửi các tác vụ (tasks) cho executors để chạy.




*Về kiến trúc này:*

Mỗi ứng dụng có các executor process riêng, tương thích với toàn bộ ứng dụng và chạy các tác vụ trong nhiều luồng (multiple threads). Điều này có lợi trong việc cô lập các ứng dụng với nhau, trên cả hai mặt lập lịch (mỗi trình điều khiển lên lịch các tác vụ riêng) và mặt thực thi (các tác vụ từ các ứng dụng khác nhau chạy trong các JVM khác nhau). Tuy nhiên, điều đó cũng có nghĩa là dữ liệu không thể được chia sẻ trên các ứng dụng Spark khác nhau (các phiên bản của SparkContext) mà không ghi nó vào hệ thống lưu trữ ngoài.

Spark không quan tâm đến loại cluster mannager, có thể triển khai Spark trên trình quản lý cụm khác nhau (ví dụ Mesos / YARN) tương đối dễ dàng.

Driver program phải lắng nghe và chấp nhận các kết nối đến từ các texecutor trong suốt quãng thời gian tồn tại. Do vậy, Driver program phải biết được địa chỉ mạng của các worker.

Bởi vì Driver lập lịch các tác vụ trên cụm, nên nó cần được chạy gần các nút worker, tốt nhất là trên cùng một mạng cục bộ.

### 3.3. Spark cluster Standalone

Có thể bắt đầu một máy chủ chính độc lập bằng cách thực thi:

```
./sbin/start-master.sh
```

Tương tự, có thể start một hoặc nhiều worker và kết nối các worker với master bằng câu lệnh:

```
./sbin/start-slave.sh <master-spark-URL>
```

Một số thông số, biến môi trường cần khai báo:

| Environment Variable | Meaning |
| --- | --- |
| SPARK_MASTER_HOST | Bind the master to a specific hostname or IP address, for example a public one. |
| SPARK_MASTER_PORT | Start the master on a different port (default: 7077). |
| SPARK_MASTER_WEBUI_PORT | Port for the master web UI (default: 8080). |
| SPARK_MASTER_OPTS | Configuration properties that apply only to the master in the form "-Dx=y" (default: none). See below for a list of possible options. |
| SPARK_LOCAL_DIRS | Directory to use for "scratch" space in Spark, including map output files and RDDs that get stored on disk. This should be on a fast, local disk in your system. It can also be a comma-separated list of multiple directories on different disks. |
| SPARK_WORKER_CORES | Total number of cores to allow Spark applications to use on the machine (default: all available cores). |
| SPARK_WORKER_MEMORY | Total amount of memory to allow Spark applications to use on the machine, e.g. `1000m`, `2g` (default: total memory minus 1 GB); note that each application's *individual* memory is configured using its `spark.executor.memory` property. |
| SPARK_WORKER_PORT | Start the Spark worker on a specific port (default: random). |
| SPARK_WORKER_WEBUI_PORT | Port for the worker web UI (default: 8081). |
| SPARK_WORKER_DIR | Directory to run applications in, which will include both logs and scratch space (default: SPARK_HOME/work). |
| SPARK_WORKER_OPTS | Configuration properties that apply only to the worker in the form "-Dx=y" (default: none). See below for a list of possible options. |
| SPARK_DAEMON_MEMORY | Memory to allocate to the Spark master and worker daemons themselves (default: 1g). |
| SPARK_DAEMON_JAVA_OPTS | JVM options for the Spark master and worker daemons themselves in the form "-Dx=y" (default: none). |
| SPARK_DAEMON_CLASSPATH | Classpath for the Spark master and worker daemons themselves (default: none). |
| SPARK_PUBLIC_DNS | The public DNS name of the Spark master and workers (default: none). |

SPARK_MASTER_OPTS supports the following system properties:

| Property Name | Default | Meaning |
|---|---|---|
| `spark.deploy.retainedApplications` | 200 | The maximum number of completed applications to display. Older applications will be dropped from the UI to maintain this limit. |
| `spark.deploy.retainedDrivers` | 200 | The maximum number of completed drivers to display. Older drivers will be dropped from the UI to maintain this limit. |
| `spark.deploy.spreadOut` | true | Whether the standalone cluster manager should spread applications out across nodes or try to consolidate them onto as few nodes as possible. Spreading out is usually better for data locality in HDFS, but consolidating is more efficient for compute-intensive workloads. |
| `spark.deploy.defaultCores` | (infinite) | Default number of cores to give to applications in Spark's standalone mode if they don't set `spark.cores.max`. If not set, applications always get all available cores unless they configure `spark.cores.max` themselves. Set this lower on a shared cluster to prevent users from grabbing the whole cluster by default. |
| `spark.deploy.maxExecutorRetries` | 10 | Limit on the maximum number of back-to-back executor failures that can occur before the standalone cluster manager removes a faulty application. An application will never be removed if it has any running executors. If an application experiences more than `spark.deploy.maxExecutorRetries` failures in a row, no executors successfully start running in between those failures, and the application has no running executors then the standalone cluster manager will remove the application and mark it as failed. To disable this automatic removal, set `spark.deploy.maxExecutorRetries` to `-1`. |
| `spark.worker.timeout` | 60 | Number of seconds after which the standalone deploy master considers a worker lost if it receives no heartbeats. |

SPARK_WORKER_OPTS supports the following system properties:

| Property Name | Default | Meaning |
|---|---|---|
| spark.worker.cleanup.enabled | false | Enable periodic cleanup of worker / application directories. Note that this only affects standalone mode, as YARN works differently. Only the directories of stopped applications are cleaned up. |
| spark.worker.cleanup.interval | 1800 (30 minutes) | Controls the interval, in seconds, at which the worker cleans up old application work dirs on the local machine. |
| spark.worker.cleanup.appDataTtl | 604800 (7 days, 7 * 24 * 3600) | The number of seconds to retain application work directories on each worker. This is a Time To Live and should depend on the amount of available disk space you have. Application logs and jars are downloaded to each application work dir. Over time, the work dirs can quickly fill up disk space, especially if you run jobs very frequently. |
| spark.storage.cleanupFilesAfterExecutorExit | true | Enable cleanup non-shuffle files(such as temp. shuffle blocks, cached RDD/broadcast blocks, spill files, etc) of worker directories following executor exits. Note that this doesn't overlap with `spark.worker.cleanup.enabled`, as this enables cleanup of non-shuffle files in local directories of a dead executor, while `spark.worker.cleanup.enabled` enables cleanup of all files/subdirectories of a stopped and timeout application. This only affects Standalone mode, support of other cluster manangers can be added in the future. |
| spark.worker.ui.compressedLogFileLengthCacheSize | 100 | For compressed log files, the uncompressed file can only be computed by uncompressing the files. Spark caches the uncompressed file size of compressed log files. This property controls the cache size. |

Để chạy shell Spark tương tác với cluster, chạy lệnh:

```
./bin/spark-shell --master spark://IP:PORT
```

### 3.4. Submitting Applications

Khi ứng dụng đã được đóng gói, nó có thể được khởi chạy bằng cách sử dụng câu lệnh bin / spark-submit. Lệnh này đảm nhiệm việc thiết lập đường dẫn lớp, các tham số phụ thuộc và các chế độ triển khai khác nhau mà Spark hỗ trợ:

```
./bin/spark-submit \
  --class <main-class> \
  --master <master-url> \
  --deploy-mode <deploy-mode> \
  --conf <key>=<value> \
  ... # other options
  <application-jar> \
  [application-arguments]
```

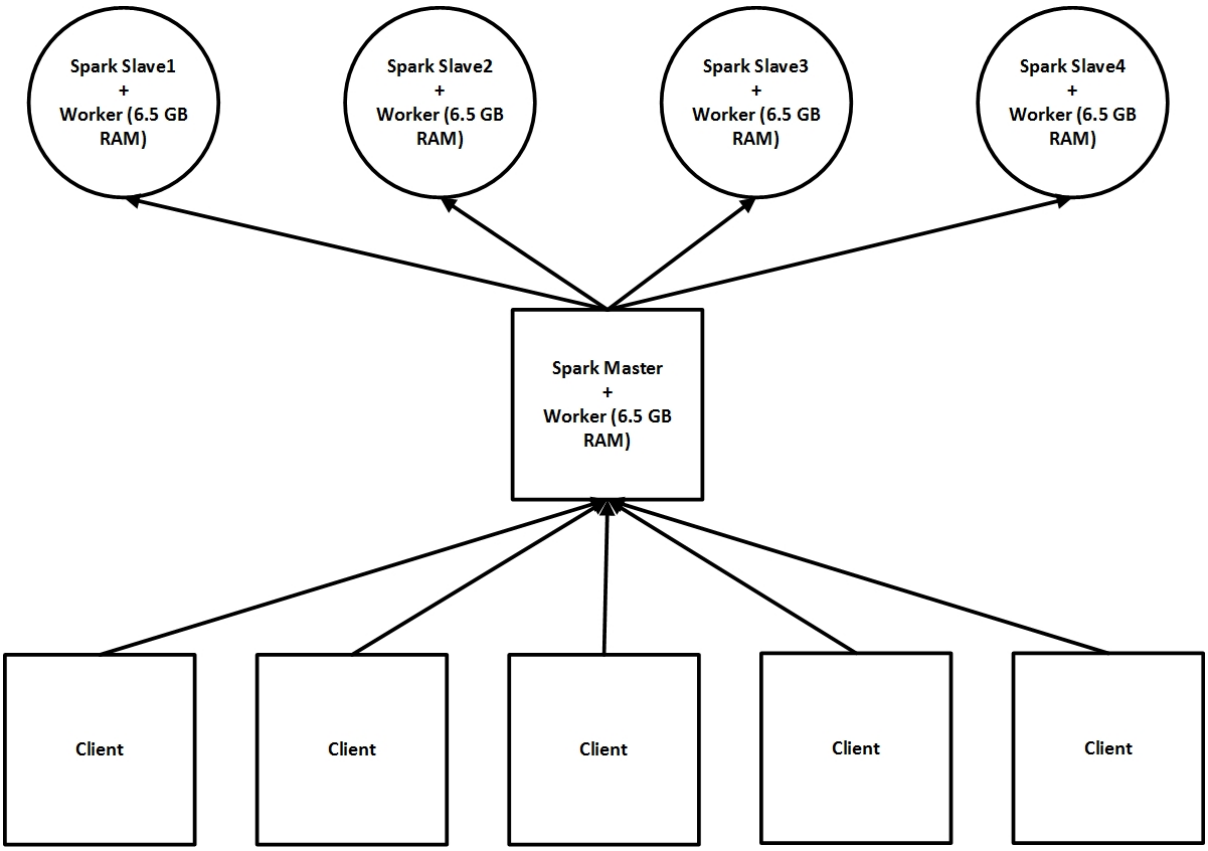
- `--class`: The entry point for your application (e.g. `org.apache.spark.examples.SparkPi`)
- `--master`: The master URL for the cluster (e.g. `spark://23.195.26.187:7077`)
- `--deploy-mode`: Whether to deploy your driver on the worker nodes (`cluster`) or locally as an external client (`client`) (default: `client`) †
- `--conf`: Arbitrary Spark configuration property in key=value format. For values that contain spaces wrap "key=value" in quotes (as shown).
- `application-jar`: Path to a bundled jar including your application and all dependencies. The URL must be globally visible inside of your cluster, for instance, an `hdfs://` path or a `file://` path that is present on all nodes.
- `application-arguments`: Arguments passed to the main method of your main class, if any

Dưới đây là một vài ví dụ về các tùy chọn phổ biến:

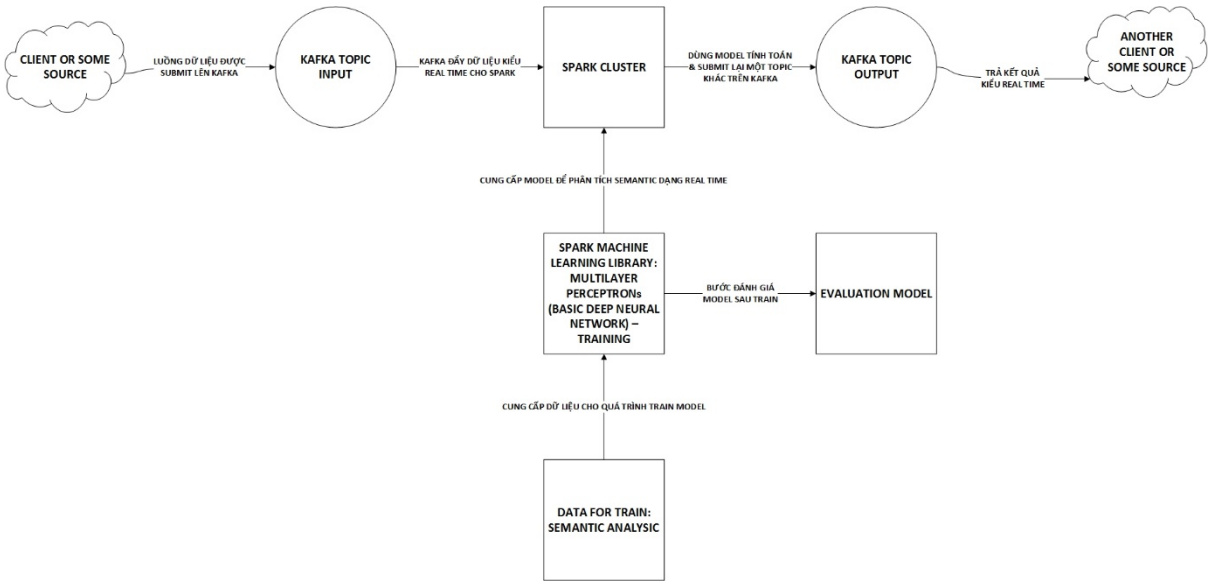
```
# Run application locally on 8 cores
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
  /path/to/examples.jar \
  100

# Run on a Spark standalone cluster in client deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000

# Run on a Spark standalone cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000

# Run on a YARN cluster
export HADOOP_CONF_DIR=XXX
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn \
  --deploy-mode cluster \  # can be client for client mode
  --executor-memory 20G \
  --num-executors 50 \
  /path/to/examples.jar \
  1000

# Run a Python application on a Spark standalone cluster
./bin/spark-submit \
  --master spark://207.184.161.138:7077 \
  examples/src/main/python/pi.py \
  1000

# Run on a Mesos cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master mesos://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  http://path/to/examples.jar \
  1000

# Run on a Kubernetes cluster in cluster deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master k8s://xx.yy.zz.ww:443 \
  --deploy-mode cluster \
  --executor-memory 20G \
  --num-executors 50 \
  http://path/to/examples.jar \
  1000
```

Tập lệnh spark-submit load các giá trị cấu hình Spark mặc định từ tệp thuộc tính và chuyển chúng vào ứng dụng. Theo mặc định, spark-submit sẽ đọc các tùy chọn từ conf / spark-defaults.conf trong thư mục Spark. Load cấu hình Spark mặc định theo cách này có thể làm giảm việc khai báo trên dòng lệnh, các giá trị cấu hình được đặt rõ ràng trên SparkConf được ưu tiên cao nhất, sau đó là các tham số trên Spark-submit, và cuối cùng là các giá trị trong tệp mặc định.

## 3.5. Cấu trúc cài đặt cluster Spark trên Vlab




## 3.6. Mô hình Spark cho Semantic Analysic




Mô hình Multilayer perceptron classifier là thuật toán mới, được tích hợp vào Spark 2.4.4.

MLP được triển khai dựa theo feedforward artificial neural network (https://en.wikipedia.org/wiki/Feedforward_neural_network)

Tham khảo thêm cách sử dụng MLP trong Spark tại địa chỉ:

https://spark.apache.org/docs/latest/ml-classification-regression.html#multilayer-perceptron-classifier

Bộ dữ liệu được lấy tại:

http://ai.stanford.edu/~amaas/data/sentiment/

Đây là một bộ dữ liệu để phân loại ngữ nghĩa (tích cực hay tiêu cực) chứa nhiều dữ liệu hơn so với các bộ dữ liệu chuẩn trước đó, cung cấp một bộ 25.000 tweet đánh giá dùng để training và 25.000 tweet cho testing.

Dùng bộ dữ liệu này cho việc train model MLP trong Spark và đánh giá model sau khi train.

*Lưu ý: khi tăng số lượng Layer lên càng nhiều, thì sẽ càng tiêu tốn nhiều tài nguyên tính toán, cần cân nhắc số layer phù hợp với resource phần cứng đang có.*

### 3.7.Kết quả và mã nguồn

Link Notebook github cho quá trình training data:

https://nbviewer.jupyter.org/github/lenguyensonnguyen/bigdata-ch1702039-2019/blob/master/semantic_analysis_train-model_notebook1.ipynb

Độ chính xác của model MLP trong Spark thử nghiệm với bộ dữ liệu Stanford đạt được khoản 0.691

Link notebook cho việc đánh giá hệ thống MLP trong Spark:

https://nbviewer.jupyter.org/github/lenguyensonnguyen/bigdata-ch1702039-2019/blob/master/semantic_analysis_test-model_notebook1.ipynb

## 3.8.Phần bổ sung

### *3.8.1. Giới thiệu BigDL*

BigDL là một thư viện deep learning phân tán cho Apache Spark, có thể chạy trực tiếp trên các cụm Spark hoặc Hadoop hiện có, giúp dễ dàng xây dựng các ứng dụng Spark deep learning cao cấp, cung cấp cơ chế pipeline cho việc phân tích từ đầu đến cuối.

BigDL sử dụng Intel MKL và lập trình đa luồng trong mỗi tác vụ Spark. Do đó, nó cho kết quả training nhanh hơn so với Caffe, Torch hoặc TensorFlow chạy trên một máy đơn.

### *3.8.2. Tích hợp BigDL vào Spark*

Chủ yếu là việc khai báo các tham số để Spark nhận được các gói thư viện của BigDL khi khởi chạy PySpark.

Cách khai báo chi tiết tham khảo thêm tại:

https://bigdl-project.github.io/0.9.0/#PythonUserGuide/install-without-pip/

https://bigdl-project.github.io/0.9.0/#PythonUserGuide/run-without-pip/

### *3.8.3. Cách tạo model Deep Learning trong BigDL*

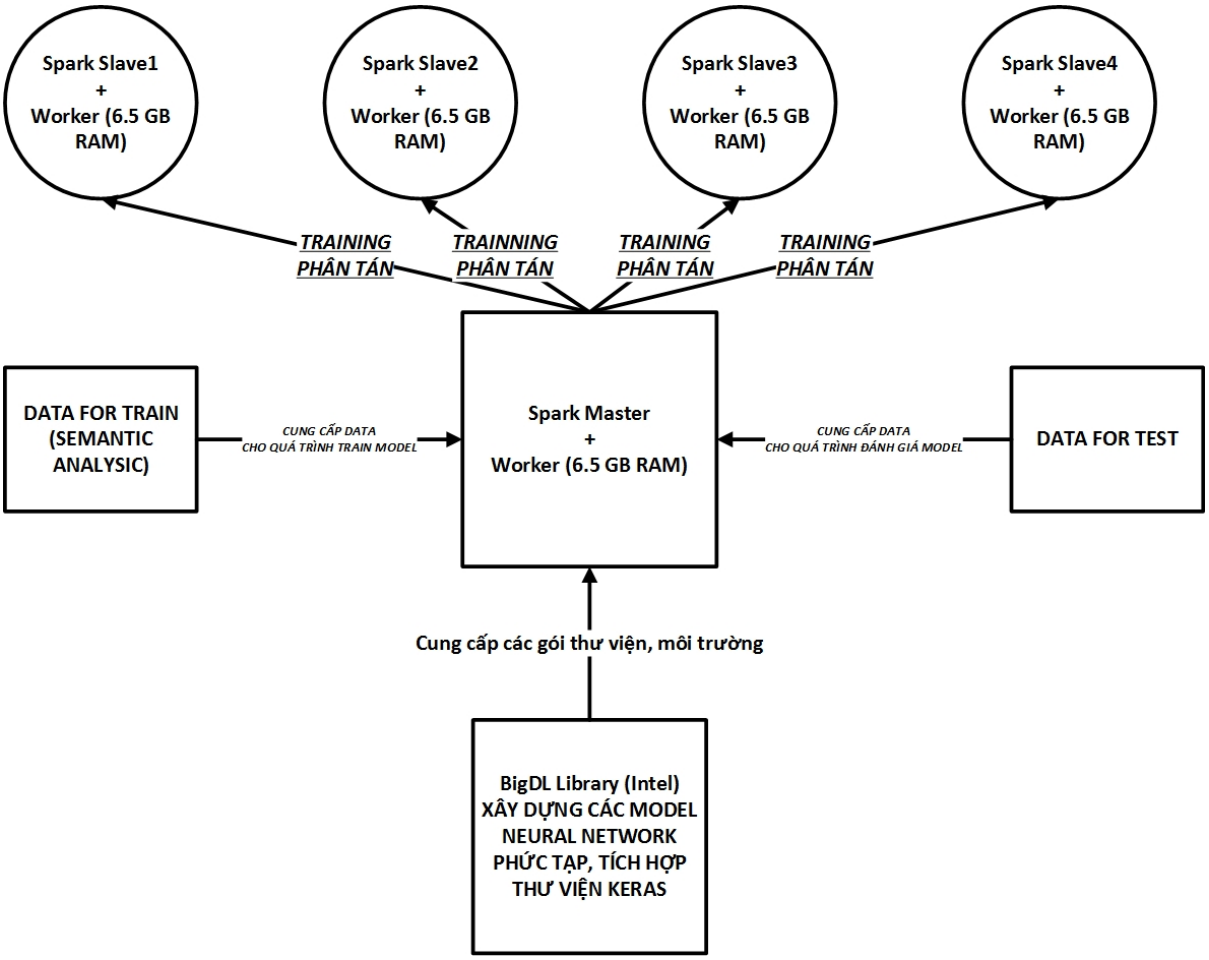
Về cách khai báo cũng như lập trình model deep learning trong BigDL hoàn toàn tương tự như cách khai báo và lập trình trong Keras. Trong khuôn khổ môn học nên không trình bày cách lập trình deep learning bằng Keras.

Chi tiết việc khai báo, định nghĩa các lớp (layer), mô hình hóa một mạng neural cho việc training dữ liệu có thể tham khảo tại đây:

https://bigdl-project.github.io/0.9.0/#ProgrammingGuide/Model/Sequential/

Hoặc có thể tham khảo trực tiếp file Jupyter Notebook cụ thể bên dưới.

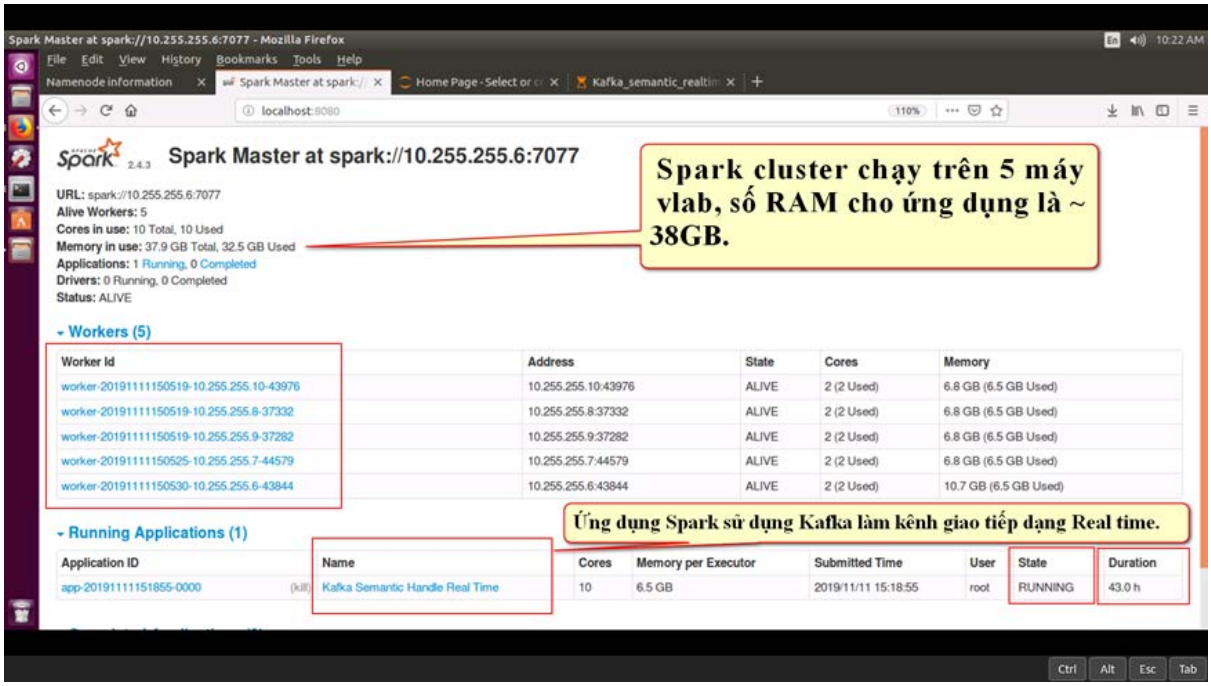
### *3.8.4. Mô hình của BigDL*

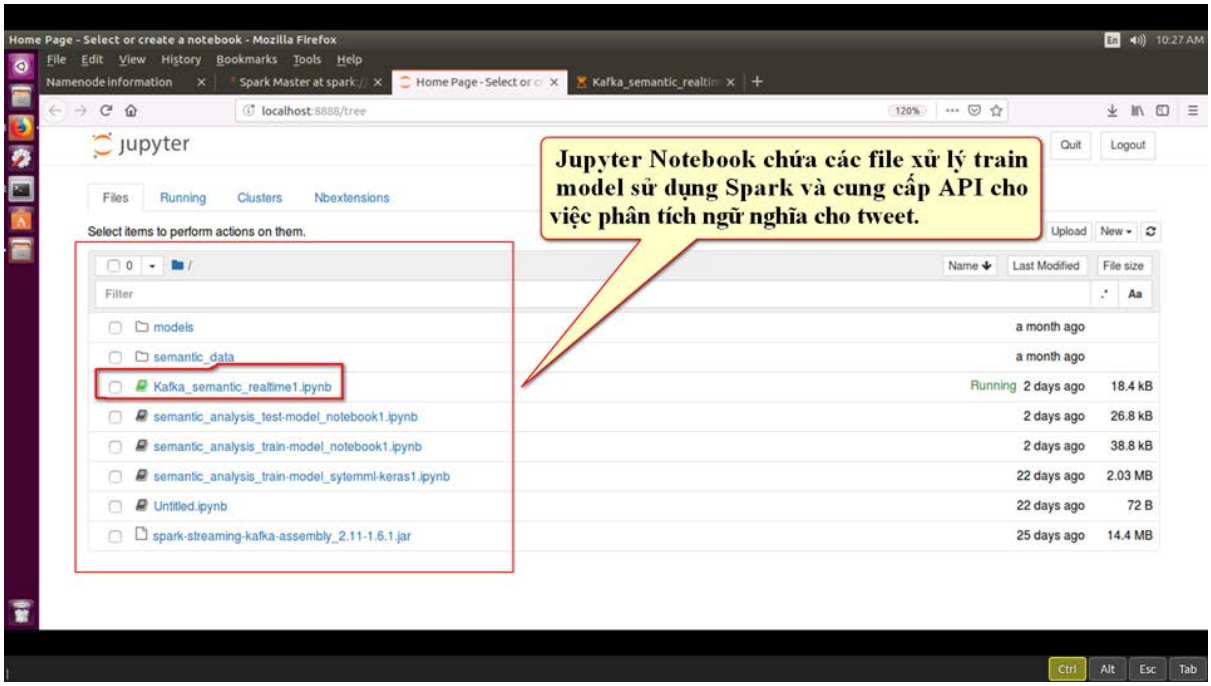


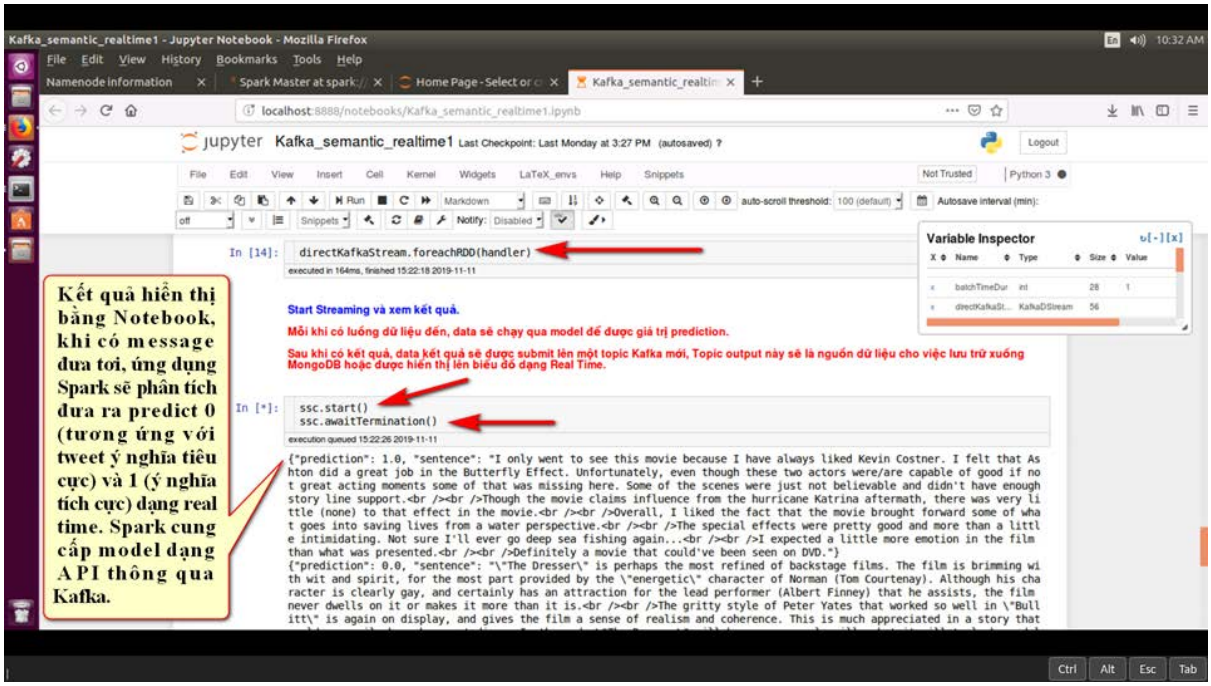

### *3.8.5. Kết quả và mã nguồn*

Cách sử dụng và chi tiết xem tại:

https://nbviewer.jupyter.org/github/lenguyensonnguyen/bigdata-ch1702039-2019/blob/master/semantic_analysis_train-evaluation-model_BIGDL-notebook1.ipynb


Spark Master at spark://10.255.255.6:7077 - Mozilla Firefox
Spark Master at spark://10.255.255.6:7077
URL: spark://10.255.255.6:7077
Alive Workers: 5
Cores in use: 10 Total, 10 Used
Memory in use: 37.9 GB Total, 32.5 GB Used
Applications: 1 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE
Spark cluster chạy trên 5 máy vlab, số RAM cho ứng dụng là ~ 38GB.
Workers (5)
Worker Id
Address
State
Cores
Memory
worker-20191111150519-10.255.255.10-43976
10.255.255.10:43976
ALIVE
2 (2 Used)
6.8 GB (6.5 GB Used)
worker-20191111150519-10.255.255.8-37332
10.255.255.8:37332
ALIVE
2 (2 Used)
6.8 GB (6.5 GB Used)
worker-20191111150519-10.255.255.9-37282
10.255.255.9:37282
ALIVE
2 (2 Used)
6.8 GB (6.5 GB Used)
worker-20191111150525-10.255.255.7-44579
10.255.255.7:44579
ALIVE
2 (2 Used)
6.8 GB (6.5 GB Used)
worker-20191111150530-10.255.255.6-43844
10.255.255.6:43844
ALIVE
2 (2 Used)
10.7 GB (6.5 GB Used)
Ứng dụng Spark sử dụng Kafka làm kênh giao tiếp dạng Real time.
Running Applications (1)
Application ID
Name
Cores
Memory per Executor
Submitted Time
User
State
Duration
app-20191111151855-0000
Kafka Semantic Handle Real Time
10
6.5 GB
2019/11/11 15:18:55
root
RUNNING
43.0 h


Home Page - Select or create a notebook - Mozilla Firefox
jupyter
Jupyter Notebook chứa các file xử lý train model sử dụng Spark và cung cấp API cho việc phân tích ngữ nghĩa cho tweet.
Files
Running
Clusters
Nbextensions
Select items to perform actions on them.
Upload
New
Quit
Logout
Name
Last Modified
File size
models
a month ago
semantic_data
a month ago
Kafka_semantic_realtime1.ipynb
Running 2 days ago
18.4 kB
semantic_analysis_test-model_notebook1.ipynb
2 days ago
26.8 kB
semantic_analysis_train-model_notebook1.ipynb
2 days ago
38.8 kB
semantic_analysis_train-model_sytemml-keras1.ipynb
22 days ago
2.03 MB
Untitled.ipynb
22 days ago
72 B
spark-streaming-kafka-assembly_2.11-1.6.1.jar
25 days ago
14.4 MB




Đường link gitHub cho toàn bộ file Jupyter Notebook:

https://github.com/lenguyensonnguyen/bigdata-ch1702039-2019

***Lưu ý: trên Github có file Readme chứa các đường link dẫn đến các file notebook được format dưới dạng full định dạng (nên sử dụng link này), do gitHub không hỗ trợ render HTML + CSS.***

# TÀI LIỆU THAM KHẢO


[1]. https://blog.florimond.dev/building-a-streaming-fraud-detection-system-with-kafka-and-python

[2]. https://markhneedham.com/blog/2019/05/29/loading-tweets-twint-kafka-neo4j/

[3]. https://github.com/twintproject/twint

[4]. https://kafka.apache.org/quickstart

[5]. https://community.plot.ly/t/python-in-real-time/20533/2

[6]. https://spark.apache.org/docs/latest

[7]. https://bigdl-project.github.io/0.9.0