**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**BỘ MÔN KỸ THUẬT MÁY TÍNH**

---------------o0o---------------



**ĐỒ ÁN MÔN HỌC**

**IT4931 - Lưu trữ và xử lý dữ liệu lớn**

**Đề tài: Lưu trữ và xử lý dữ liệu cổ phiếu**

**Giảng viên hướng dẫn : TS. Trần Việt Trung**

**Nhóm : 21**

**Phạm Thành Phong 20173297**

**Dương Xuân Hoàn 20173125**

**Dương Kim Trường 20173428**

**Chử Việt Hoàng 20173135**

**Dương Đức Điệp 20173013**

*Hà Nội, 12/2020*

MỤC LỤC

CHƯƠNG 1 : Giới thiệu về đề tài………………………………………………………………………3

CHƯƠNG 2 : Mô tả hệ thống……………………………………………………………………………3

CHƯƠNG 3 : Triển khai và cài đặt…………………………………………………………………….5

CHƯƠNG 4 : Triển khai mô hình……………………………………………………………………..18

**CHƯƠNG 1 : Giới thiệu về đề tài**

1. Giới thiệu

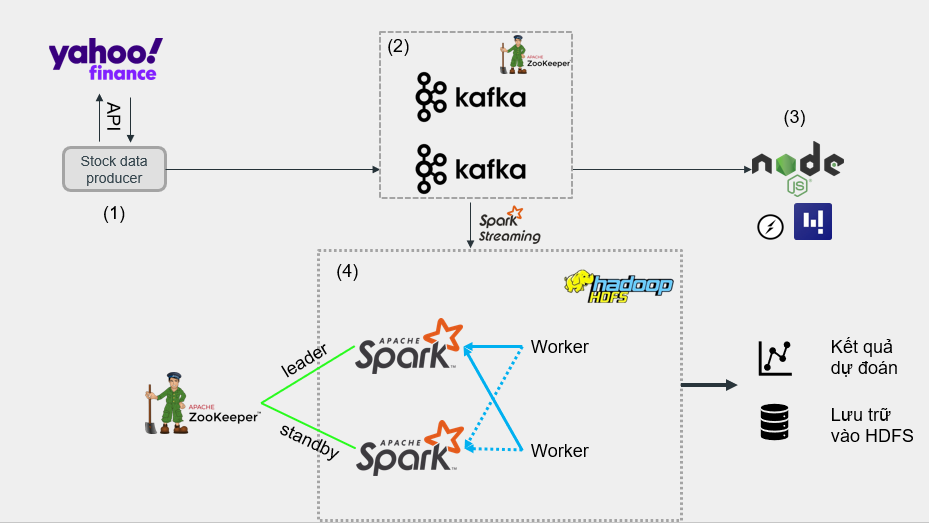
* Thị trường chứng khoán sinh ra phục vụ cho nền kinh tế rộng lớn. Nó giúp cho các cá nhân kiếm được lợi nhuận khi đầu tư, đồng thời giúp cho các công ty phân tán rủi ro và có được kết quả kinh doanh tốt
* Bên cạnh đó, nó cũng đóng một vai trò quan trọng trong nền kinh tế của một quốc gia về mặt chi tiêu và đầu tư. Các công ty quan tâm đến giá trị chứng khoán của mình đều vì sức khỏe tài chính, minh chứng cho năng lực quản lý, và tạo hình ảnh tích cực cho công ty trên diện truyền thông, báo chí
* Do đó, việc lưu trữ các dữ liệu cổ phiếu để có thể xử lý và dự đoán nó trong tương lai là điều cực kỳ cần thiết đối với mỗi công ty

1. Mục đích

* Đề tài dựa vào điều trên để xây dựng một hệ thống có khả năng đáp ứng được các tiêu chí như lưu trữ một lượng lớn dữ liệu, có khả năng phân tán và chịu lỗi, đồng thời có thể xử lý nó theo thời gian thực như là hiển thị biểu đồ và xây dựng mô hình dự đoán giá trị trong tương lai. Qua đó, nắm chắc hơn các kiến thức được học trên lớp

**CHƯƠNG 2 : Mô tả hệ thống**

* Mô hình tổng quan của hệ thống được xây dựng như sau :



* Các thành phần chính bao gồm :

1. Là một chương trình Producer sẽ đóng vai trò thu thập dữ liệu thời gian thực thông qua lấy các API từ trên Yahoo Finance và sau đó đẩy luồng dữ liệu đó liên tục vào Kafka.
2. Cụm Kafka được quản lý bởi một Zookeeper riêng đóng vai trò là Broker chuyển tiếp dữ liệu giữa Producer và Consumer. Cụm Kafka sẽ luôn có 1 Kafka được để ở chế độ Active trong khi các Kafka còn lại sẽ ở chế độ Standby phòng hờ. Khi Kafka hiện tại không thể hoạt động ổn định thì Zookeeper sẽ lựa chọn 1 Kafka trong những Kafka còn lại vào chế độ Active để phục vụ luồng hoạt động.
3. Đây là một Web server được xây dựng bằng NodeJS và đồng thời là một Consumer luôn lắng nghe dữ liệu từ Kafka để hiển thị thời gian thực lên trên giao diện Web cho người dùng theo dõi.
4. Cụm Spark xử lý dữ liệu theo dòng lấy từ Kafka được đặt trong một kho dữ liệu phân tán HDFS và xử lý theo yêu cầu như là lưu trữ các giá trị cổ phiếu với hệ số nhân bản là 3 cho trường hợp chịu lỗi. Khi dữ liệu đủ lớn, admin có thể yêu cầu cụm Spark chạy chương trình dự đoán giá trị cổ phiếu trong tương lai bằng mô hình được xây dụng trước. Nếu hệ thống đủ khả năng đáp ứng, mô hình có thể chạy theo thời gian thực với định kỳ số ngày xác định.

**CHƯƠNG 3 : Cấu hình và cài đặt**

Yêu cầu phần mềm trước khi cài đặt:

* Spark 3.0.1
* Hadoop 2.7.7
* Zookeeper 3.4.13
* Kafka 2.4.1
* Python 3.8+
* Java 8+
* Scala 2.12

1. Thu thập dữ liệu

* Dữ liệu được thu thập thông qua API của Yahoo Finance
* Tuy nhiên vì điều kiện không cho phép nên dữ liệu ko thể thu thập theo thời gian thực mà sẽ mô phỏng thời gian thực bằng cách lấy trước một bộ dữ liệu quá khứ của 5 công ty và sẽ đẩy dần dần vào Kafka để mô phỏng thời gian thực
* Một số thư viện Maven bổ sung

<dependency>

<groupId>com.yahoofinance-api</groupId>

<artifactId>YahooFinanceAPI</artifactId>

<version>3.15.0</version>

</dependency>

<dependency>

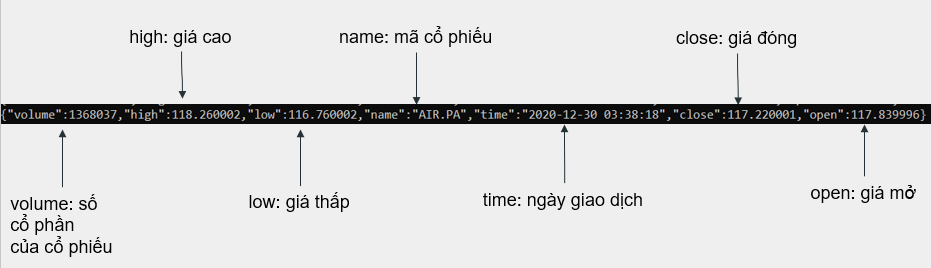
<groupId>com.googlecode.json-simple</groupId>

<artifactId>json-simple</artifactId>

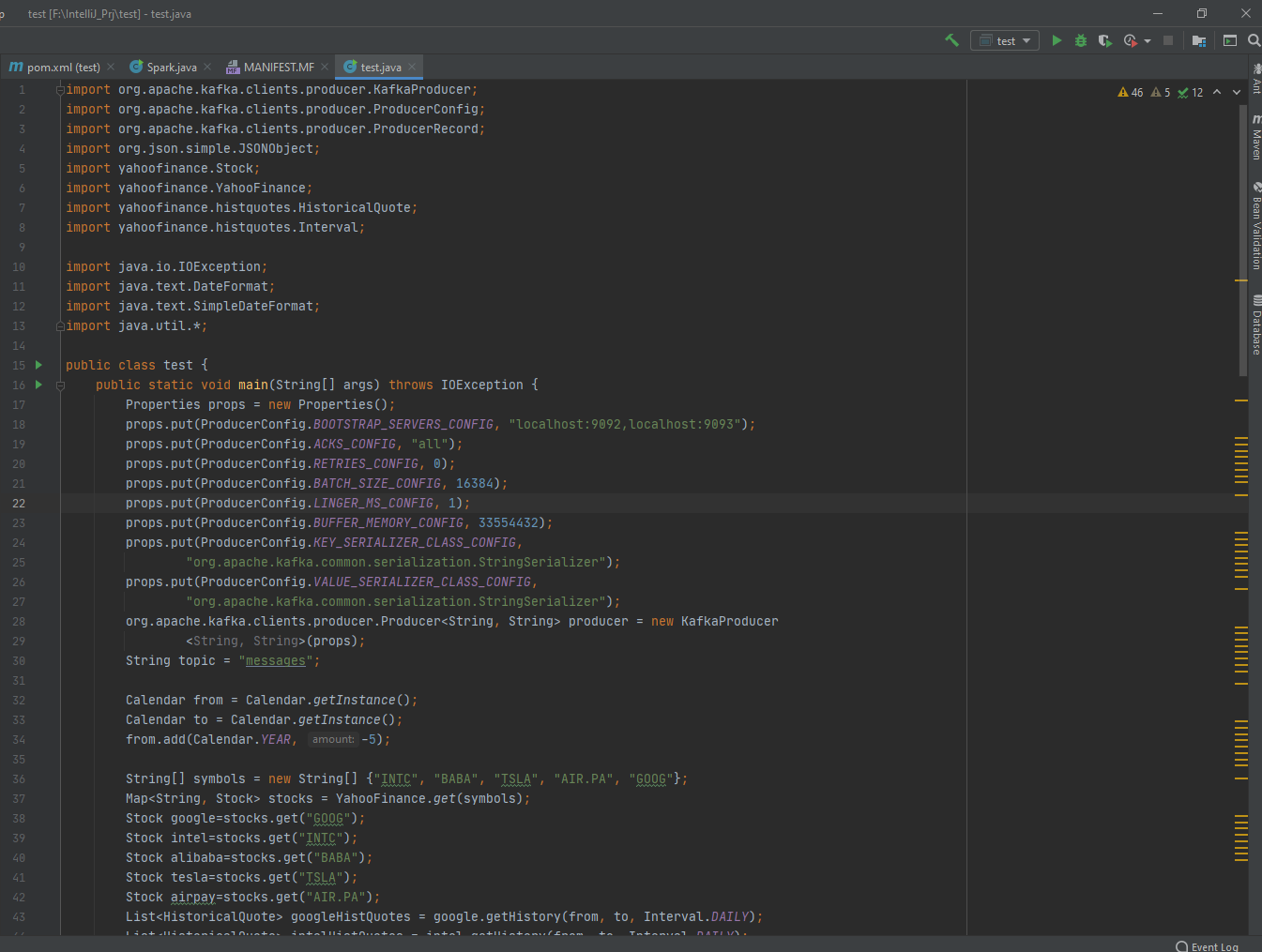
<version>1.1.1</version>

</dependency>

* Dữ liệu thu được sẽ gồm các trường sau



* Đoạn mã chương trình Producer viết bằng Java



* Trước tiên, ta tạo Kafka Producer với các thông số config được lưu trong Properties, do Kafka và chương trình Producer trên cùng một máy nên mình cấu hình BOOTSTRAP\_SERVERS\_CONFIG là localhost, cổng mặc định là 9092, các thông số khác có thể xem trên Document để lựa chọn các tùy chọn khác

Properties props = new Properties();  
props.put(ProducerConfig.*BOOTSTRAP\_SERVERS\_CONFIG*, "localhost:9092");  
props.put(ProducerConfig.*ACKS\_CONFIG*, "all");  
props.put(ProducerConfig.*RETRIES\_CONFIG*, 0);  
props.put(ProducerConfig.*BATCH\_SIZE\_CONFIG*, 16384);  
props.put(ProducerConfig.*LINGER\_MS\_CONFIG*, 1);  
props.put(ProducerConfig.*BUFFER\_MEMORY\_CONFIG*, 33554432);  
props.put(ProducerConfig.*KEY\_SERIALIZER\_CLASS\_CONFIG*, "org.apache.kafka.common.serialization.StringSerializer");  
props.put(ProducerConfig.*VALUE\_SERIALIZER\_CLASS\_CONFIG*,  
"org.apache.kafka.common.serialization.StringSerializer");  
org.apache.kafka.clients.producer.Producer<String, String> producer = new KafkaProducer  
 <String, String>(props);

* Lấy giá trị nhận được của 5 năm gần đây của 5 loại cổ phiếu GOOG, TSLA, BABA, INTC, AIR.PA qua từng ngày, lưu vào List<HistoricalQuote>

Calendar from = Calendar.*getInstance*();

Calendar to = Calendar.*getInstance*();

from.add(Calendar.*YEAR*, -5);

String[] symbols = new String[] {"INTC", "BABA", "TSLA", "AIR.PA", "GOOG"};

Map<String, Stock> stocks = YahooFinance.*get*(symbols);

Stock google=stocks.get("GOOG");

Stock intel=stocks.get("INTC");

Stock alibaba=stocks.get("BABA");

Stock tesla=stocks.get("TSLA");

Stock airpay=stocks.get("AIR.PA");

List<HistoricalQuote> googleHistQuotes = google.getHistory(from, to, Interval.*DAILY*);

List<HistoricalQuote> intelHistQuotes = intel.getHistory(from, to, Interval.*DAILY*);

List<HistoricalQuote> alibabaHistQuotes = alibaba.getHistory(from, to, Interval.*DAILY*);

List<HistoricalQuote> teslaHistQuotes = tesla.getHistory(from, to, Interval.*DAILY*);

List<HistoricalQuote> airpayHistQuotes = airpay.getHistory(from, to, Interval.*DAILY*);

* Lần lượt đẩy JSON chứa dữ liệu của từng bản ghi rồi đẩy vào Kafka, mỗi record gồm 7 trường: name, open, high, low, close,volume, time. Sau đó ta thu được 1 bộ dữ liệu tĩnh của 5 năm:

String topic = "messages";

for (int i = 0; i < googleHistQuotes.size(); i++) {

Date date = googleHistQuotes.get(i).getDate().getTime();

DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

String now = dateFormat.format(date);

JSONObject gooJSON=new JSONObject();

gooJSON.put("name","GOOG"); gooJSON.put("open",googleHistQuotes.get(i).getOpen()); gooJSON.put("high",googleHistQuotes.get(i).getHigh()); gooJSON.put("low",googleHistQuotes.get(i).getLow()); gooJSON.put("close",googleHistQuotes.get(i).getClose());  
gooJSON.put("volume",googleHistQuotes.get(i).getVolume());

gooJSON.put("time",now);

producer.send(new ProducerRecord<String, String>(topic, gooJSON.toJSONString()));

}

for (int i = 0; i < intelHistQuotes.size(); i++) {

Date date = intelHistQuotes.get(i).getDate().getTime();

DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

String now = dateFormat.format(date);

JSONObject intelJSON=new JSONObject();

intelJSON.put("name","INTC");

intelJSON.put("open",intelHistQuotes.get(i).getOpen()); intelJSON.put("high",intelHistQuotes.get(i).getHigh()); intelJSON.put("low",intelHistQuotes.get(i).getLow()); intelJSON.put("close",intelHistQuotes.get(i).getClose()); intelJSON.put("volume",intelHistQuotes.get(i).getVolume());

intelJSON.put("time",now);

producer.send(new ProducerRecord<String, String>(topic, intelJSON.toJSONString()));

}

for (int i = 0; i < alibabaHistQuotes.size(); i++) {

Date date = alibabaHistQuotes.get(i).getDate().getTime();

DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

String now = dateFormat.format(date);

JSONObject alibabaJSON=new JSONObject();

alibabaJSON.put("name","BABA"); alibabaJSON.put("open",alibabaHistQuotes.get(i).getOpen());  
alibabaJSON.put("high",alibabaHistQuotes.get(i).getHigh());

alibabaJSON.put("volume",alibabaHistQuotes.get(i).getVolume());

alibabaJSON.put("time",now);

producer.send(new ProducerRecord<String, String>(topic, alibabaJSON.toJSONString()));

}

for (int i = 0; i < teslaHistQuotes.size(); i++) {

Date date = teslaHistQuotes.get(i).getDate().getTime();

DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

String now = dateFormat.format(date);

JSONObject teslaJSON=new JSONObject();

teslaJSON.put("name","TSLA");  
teslaJSON.put("open",teslaHistQuotes.get(i).getOpen()); teslaJSON.put("high",teslaHistQuotes.get(i).getHigh());  
teslaJSON.put("low",teslaHistQuotes.get(i).getLow());  
teslaJSON.put("close",teslaHistQuotes.get(i).getClose());  
teslaJSON.put("volume",teslaHistQuotes.get(i).getVolume());

teslaJSON.put("time",now);

producer.send(new ProducerRecord<String, String>(topic, teslaJSON.toJSONString()));

}

for (int i = 0; i < airpayHistQuotes.size(); i++) {

Date date = airpayHistQuotes.get(i).getDate().getTime();

DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

String now = dateFormat.format(date);

JSONObject airpayJSON=new JSONObject();

airpayJSON.put("name","AIR.PA"); airpayJSON.put("open",airpayHistQuotes.get(i).getOpen());  
airpayJSON.put("high",airpayHistQuotes.get(i).getHigh());  
airpayJSON.put("low",airpayHistQuotes.get(i).getLow());  
airpayJSON.put("close",airpayHistQuotes.get(i).getClose());  
airpayJSON.put("volume",airpayHistQuotes.get(i).getVolume());

airpayJSON.put("time",now);

producer.send(new ProducerRecord<String, String>(topic, airpayJSON.toJSONString()));

}

* Để demo dữ liệu real time, do Yahoo Finance ko cung cấp API real-time miễn phí nên em đã fake data bằng cách thay đổi tần suất gửi 1 record/s.

1. Cụm Kafka trung chuyển dữ liệu

* Cụm Kafka được sử dụng trong phạm vi đề tài là hai Kafka được quản lý bởi một Zookeeper và đều nằm trên cùng một máy
* Cách cấu hình cho chúng như sau :
  + Từ tệp server.properties trong $KAFKA\_HOME/config, ta tạo hai tệp khác lấy tên là server1.properties cho Kafka thứ nhất và server2.properties cho Kafka thứ hai
  + Cấu hình cho server1.properties

*log.dirs=/home/hadoopuser/kafka/kafka-logs1*

*listeners=LISTENER\_LAN://0.0.0.0:9092*

*advertised.listeners=LISTENER\_LAN://192.168.99.100:9092*

*listener.security.protocol.map=LISTENER\_LAN:PLAINTEXT*

*inter.broker.listener.name=LISTENER\_LAN*

* + Cấu hình cho server2.properties với cổng khác

*log.dirs=/home/hadoopuser/kafka/kafka-logs2*

*listeners=LISTENER\_LAN://0.0.0.0:9093*

*advertised.listeners=LISTENER\_LAN://192.168.99.100:9093*

*listener.security.protocol.map=LISTENER\_LAN:PLAINTEXT*

*inter.broker.listener.name=LISTENER\_LAN*

* + Bên cạnh đó cũng cần tạo thư mục chứa log cho hai Kafka tương tự như tham số log.dirs ở trên

1. Hiển thị dữ liệu trên giao diện Web

* Web server được xây dựng bằng NodeJS sẽ đóng vai trò là Consumer sẽ subscribe để topic “messages” của Kafka
* Thư viện được sử dụng là socket.io để kết nối với front-end và FusionChart để vẽ biểu đồ
* Mã nguồn của Web server
* Với mỗi luồng dữ liệu nhận được, phía backend sẽ gửi trường value của messages đến frontend để cập nhật biểu đồ CandleStick thời gian thực

//import các thư viện cần thiết, tạo đối tượng kafka, config các tham số về broker, topic, id

//...

const run = async () => {

  await consumer.connect()

  await consumer.subscribe({ topic, fromBeginning: true })

  await consumer.run({

    eachMessage: async ({ topic, partition, message }) => {

      const prefix = `${topic}[${partition} | ${message.offset}] / ${message.timestamp}`

      console.log(`- ${prefix} ${message.key}#${message.value}`)

      io.emit("aggregator-message", message.value.toString())

    },

  })

}

app.get('/', function(req, res){

    res.sendFile('web/index.html', {root: \_\_dirname});

});

* Mã nguồn Frontend : Với mỗi dữ liệu nhận được từ Backend, kiểm tra xem trường name (mã cổ phiếu) của dữ liệu là gì, dùng hàm feedData() để chèn dữ liệu mới vào đối tượng biểu đồ phù hợp

//Khởi tạo các đối tượng biểu đồ thuộc thư viện FusionChart, các đối tượng biểu đồ được lưu vào mảng realTimeChartArray

//...

var socket = io();

socket.on('aggregator-message', function(data){

    if(typeof(data) !== "object"){

        data = JSON.parse(data);

    }

    console.log("====================");

    console.log(data);

    console.log("--------------------")

    for (let [key, value] of stockMap.entries()) {

        if (data.name === key) {

            var dataElement = ["" + fd(new Date(data.time))

                              , data.open

                              , data.high

                              , data.low

                              , data.close

                              , data.volume

                              ];

            var tickerCode = stockTickerArray.indexOf(key);

            realtimeChartArray[tickerCode].feedData([dataElement]);

            break;

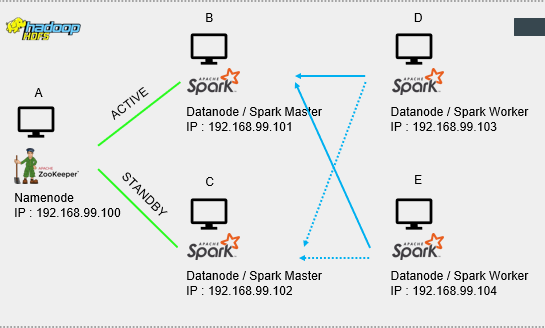
        }

    }

});

1. Xử lý dữ liệu bằng Spark
   1. Cấu hình cho cụm spark

* Do cơ sở vật chất không cho phép nên cụm Spark sẽ được bố trí cơ bản như sau :



* Máy A sẽ vừa đóng vai trò là Namenode và cùng là Zookeeper server để các máy B và C là hai máy Spark Master kết nối tới. Để đạt được điều này cần phải có cấu hình trong các tệp hệ thống của các máy như sau :
* Thêm vào cấu hình vào tệp zoo.cfg trong $ZOOKEEPER\_HOME/conf đoạn sau :

*dataDir=/home/hadoopuser/zookeeper/data*

*clientPort=2181*

*server.1=192.168.99.100:2888:3888*

* Và tạo một thư mục data trong $ZOOKEEPER\_HOME và một tệp tên là myid có nội dung là 1 bên trong để định danh cho Zookeeper Server và đồng thời thư mục này sẽ là nơi lưu trữ log cho cả 2 Spark Master để có thể phục hồi dữ liệu khi cần
* Thêm cấu hình vào tệp spark-env.sh trong SPARK\_HOME/conf của cả 2 máy Spark tham số sau :

*SPARK\_DAEMON\_JAVA\_OPTS="-Dspark.deploy.recoveryMode=ZOOKEEPER -Dspark.deploy.zookeeper.url=192.168.99.100:2181 -Dspark.deploy.zookeeper.dir= /home/hadoopuser/zookeeper/data"*

* Trong đó zookeeper.url là địa chỉ IP và cổng còn zookeeper.dir là thư mục trỏ đến thư mục data của Zookeeper Server ở trên
* Còn bên phía Worker, các Worker phải đều kết nối tới cả hai Master nói trên nên cần phải cài đặt cấu hình trong tệp spark-default.sh như sau :

*spark.master spark://192.168.99.101:7077,192.168.99.102:7077*

* 1. Nhận và lưu trữ dữ liệu theo dòng
* Thư viện Maven bổ sung

<!-- https://mvnrepository.com/artifact/org.apache.spark/spark-core -->

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-core\_2.12</artifactId>

<version>2.4.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.spark/spark-sql -->

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-sql\_2.12</artifactId>

<version>2.4.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.spark/spark-streaming -->

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming\_2.12</artifactId>

<version>2.4.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.spark/spark-streaming-kafka-0-10 -->

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-streaming-kafka-0-10\_2.12</artifactId>

<version>2.4.1</version>

</dependency>

<dependency>

<groupId>org.apache.spark</groupId>

<artifactId>spark-sql-kafka-0-10\_2.12</artifactId>

<version>2.4.1</version>

</dependency>

* Khởi tạo SparkSession, JavaSparkContext từ SparkSession

SparkSession spark = SparkSession

.*builder*()

.appName("StockStreaming")

.getOrCreate();

JavaSparkContext jsc = new JavaSparkContext(spark.sparkContext());

* Tạo SparkStreamingContext để nhận dữ liệu từ Kafka, ở đây để thuận tiện cho việc demo, em cấu hình 10s để tạo thành 1 RDD và xử lý nên SparkStreaming xử lý 10s 1 lần với tập dữ liệu thu nhận được

JavaStreamingContext streamingContext = new JavaStreamingContext(jsc, Durations.*seconds*(10));

Map<String, Object> kafkaParams = new HashMap<>();

kafkaParams.put("bootstrap.servers", "192.168.43.218:9092");

kafkaParams.put("key.deserializer", StringDeserializer.class);

kafkaParams.put("value.deserializer", StringDeserializer.class);

kafkaParams.put("group.id", "use\_a\_separate\_group\_id\_for\_each\_stream");

kafkaParams.put("auto.offset.reset", "latest");

kafkaParams.put("enable.auto.commit", false);

Collection<String> topics = Arrays.*asList*("messages");

JavaInputDStream<ConsumerRecord<String, String>> messages =

KafkaUtils.*createDirectStream*(

streamingContext,

LocationStrategies.*PreferConsistent*(),

ConsumerStrategies.<String, String> *Subscribe*(topics, kafkaParams));

* Với mỗi RDD nhận được sẽ được tạo thành dataset dạng json, dataset sẽ chia partion theo “name” để lưu vào hdfs:

JavaDStream<String> data = messages.map(v -> {

return v.value(); // mapping to convert into spark D-Stream

});

data.foreachRDD(s->{

if(!s.isEmpty()){

Dataset<Row> dataset=spark.read().json(s); dataset.write().partitionBy("name").format("org.apache.spark.sql.execution.datasources.json.JsonFileFormat").mode(SaveMode.*Append*).save("hdfs://masternode:9000/data");

}

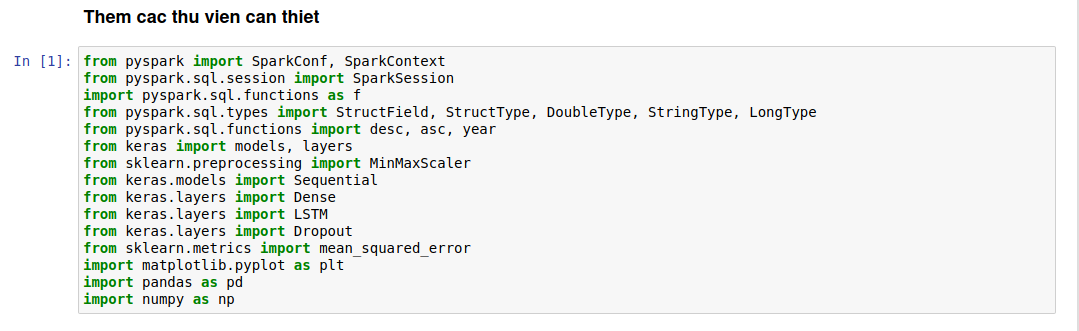
});

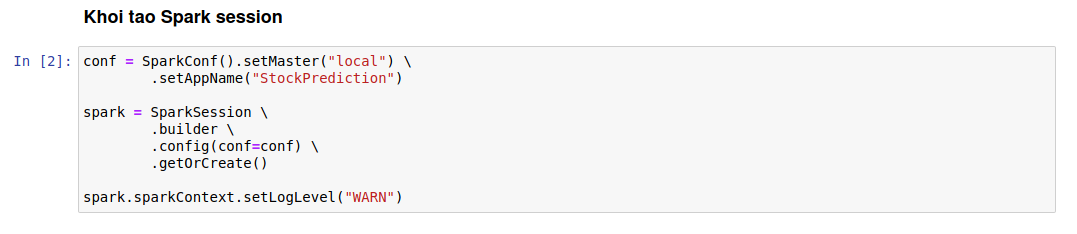
* Bắt đầu truyền dòng dữ liệu:

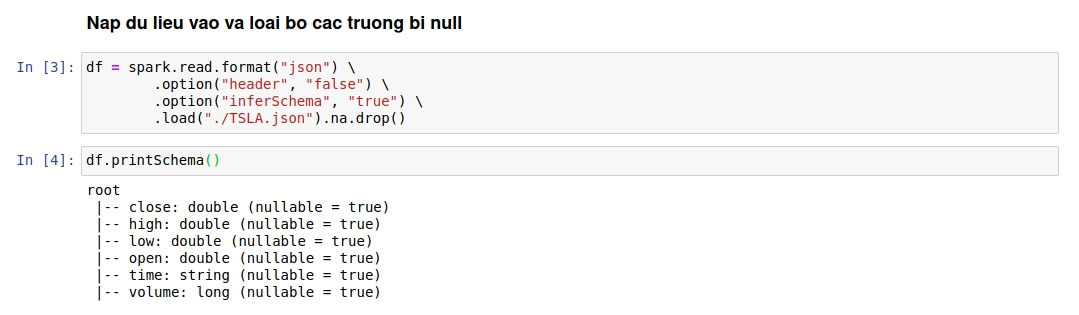
streamingContext.start();

streamingContext.awaitTermination();

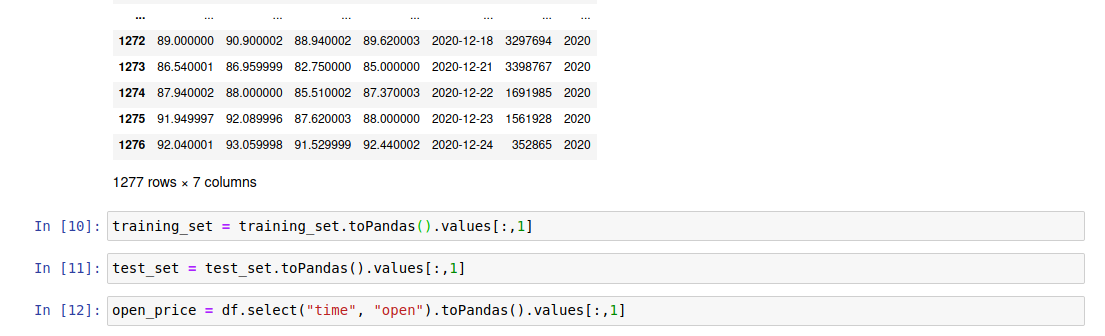
* 1. Xây dựng mô hình dự đoán giá trị cổ phiếu bằng LSTm

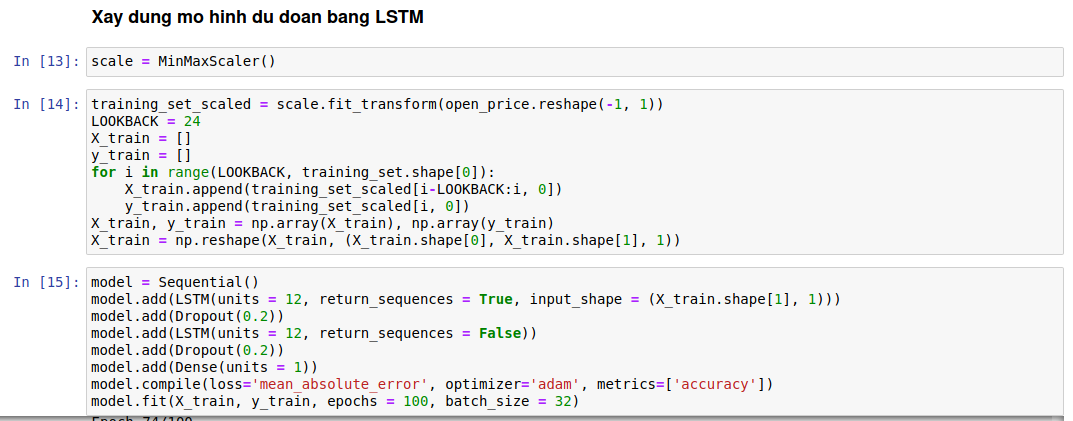


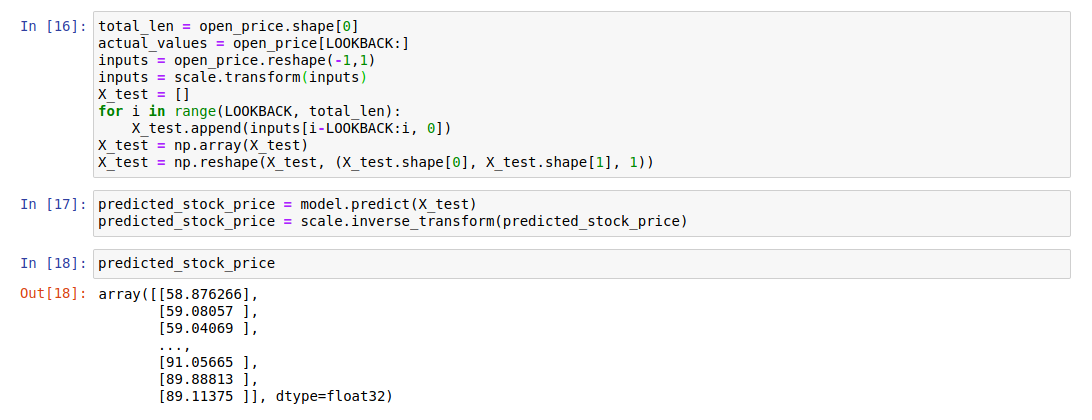


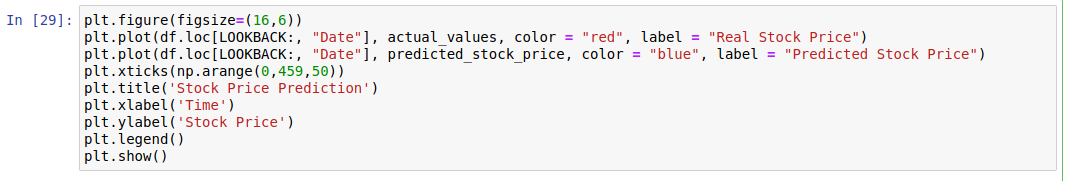


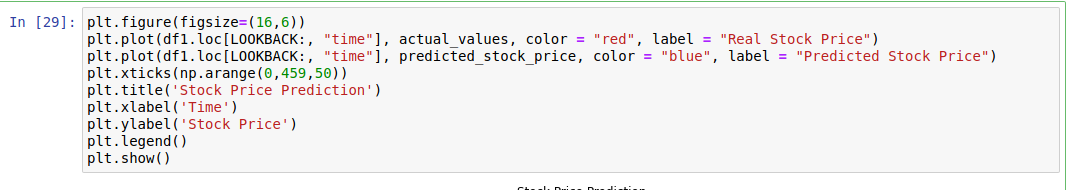


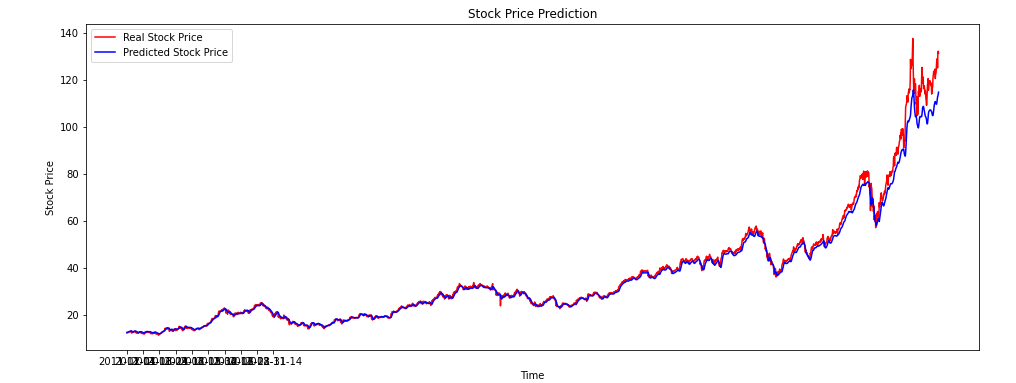








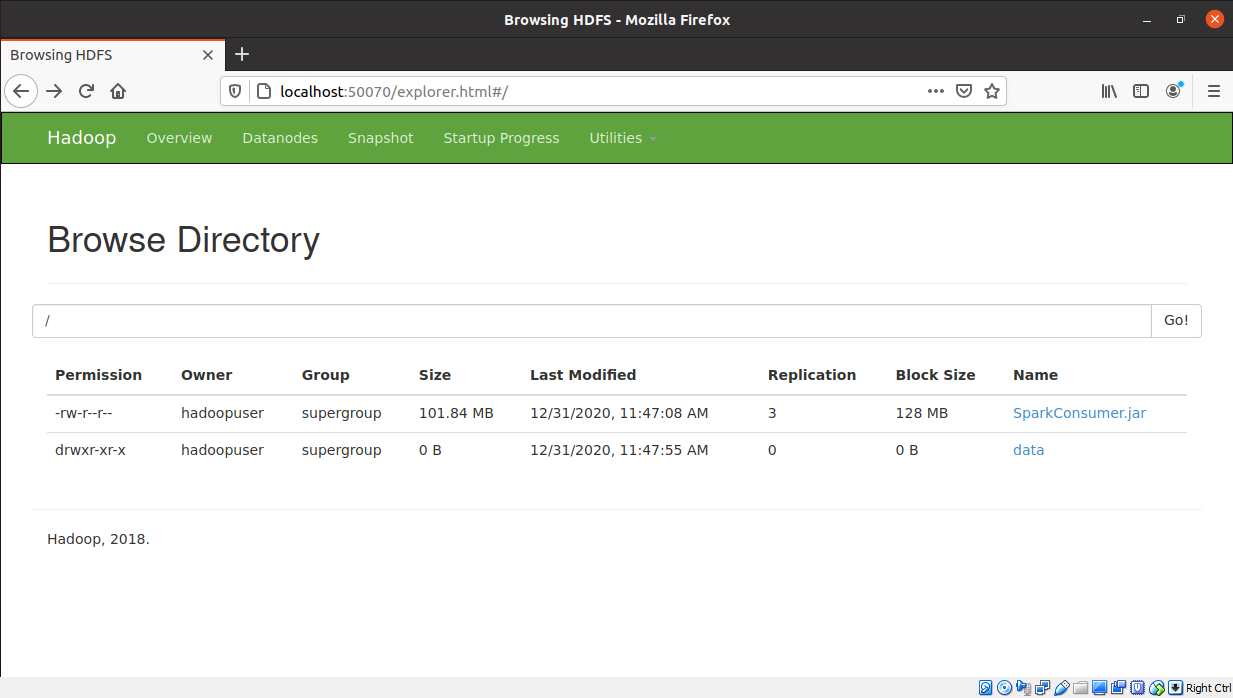




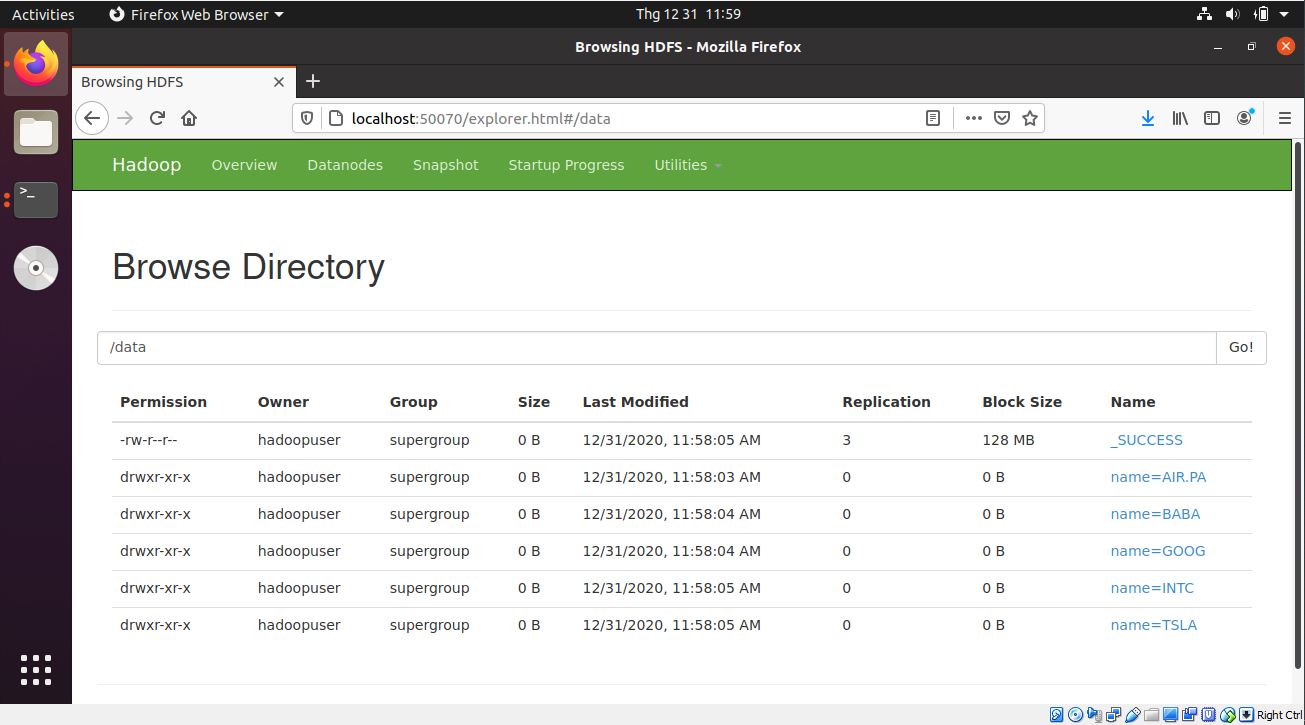


**CHƯƠNG 4 : Triển khai mô hình**

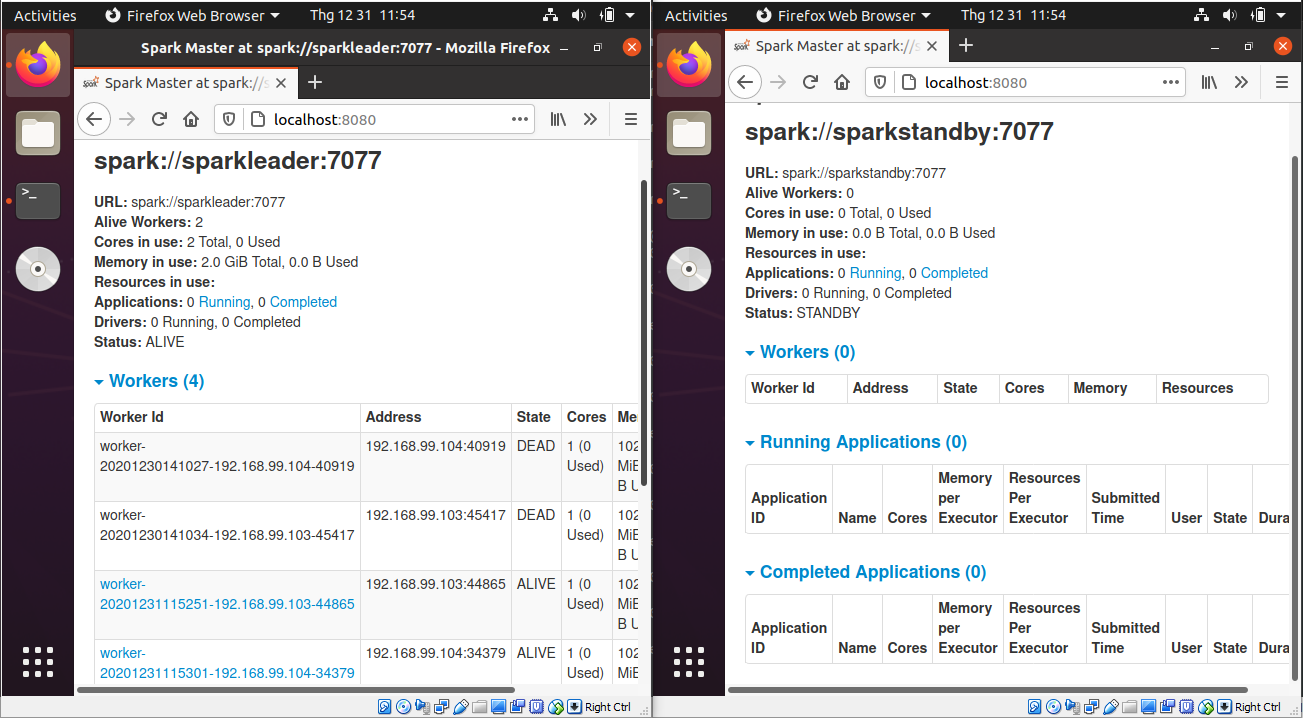
* + - 1. Triển khai HDFS
* Chạy câu lệnh khởi động HDFS : *start-dfs.sh*



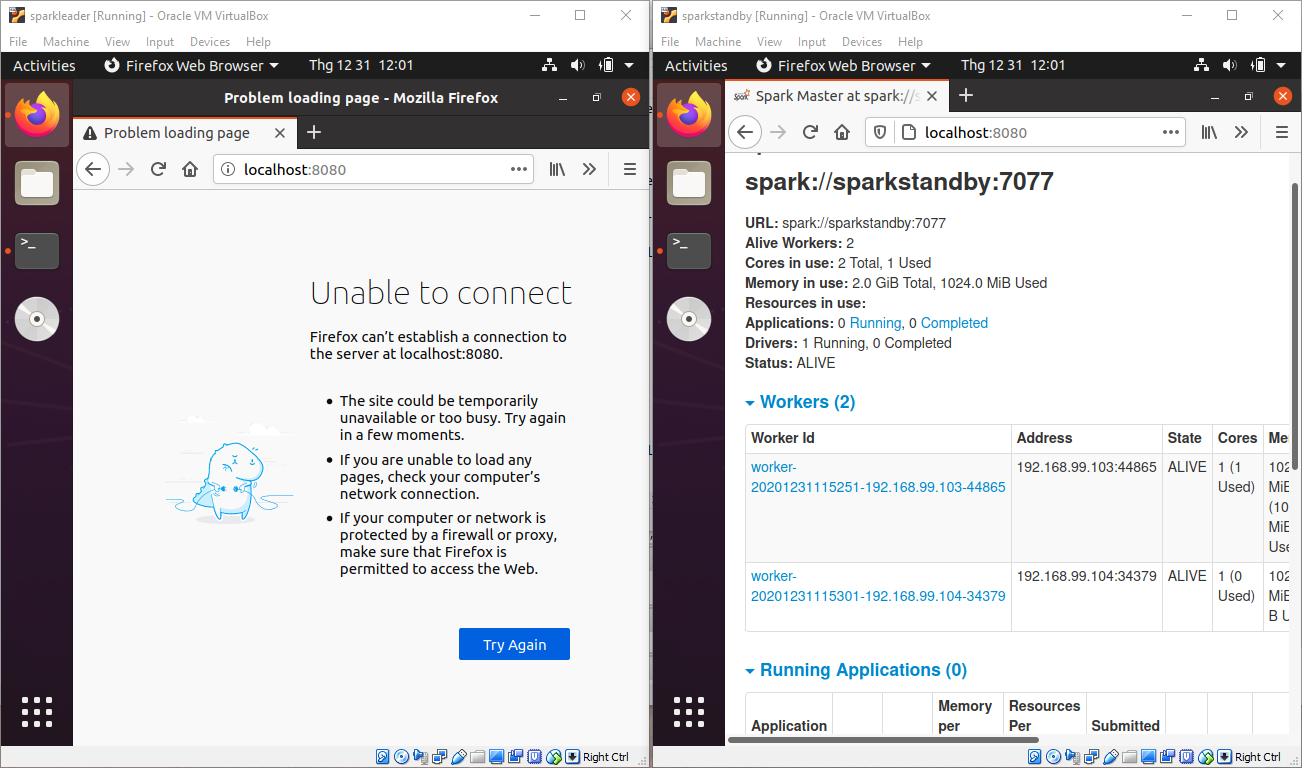
* Sau khi dữ liệu nhận được từ Kafka sẽ được lưu trữ vào trong HDFS với mỗi thư mục là dữ liệu cổ phiếu của mỗi công ty

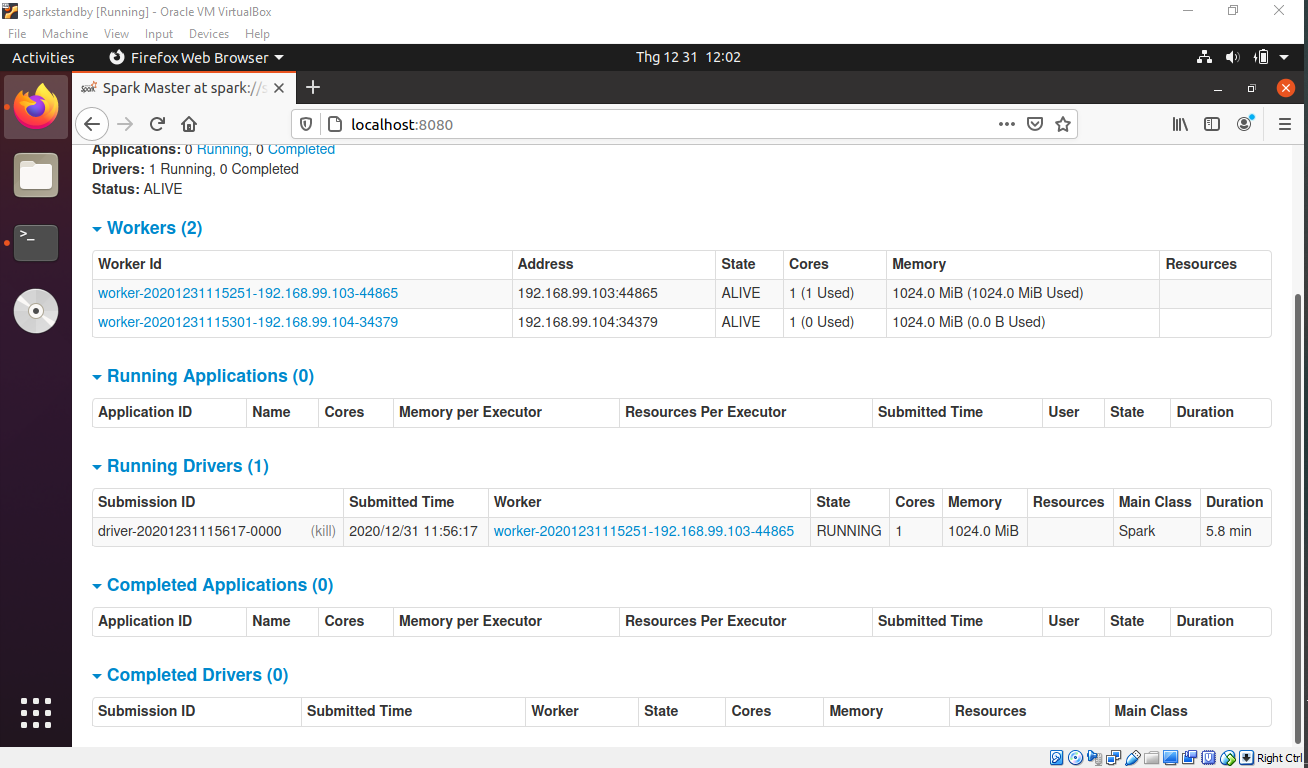


* + - 1. Giao diện Spark
* Khởi động Zookeeper Server trên máy namenode: *zkServer.sh start-foreground conf/zoo.cfg*
* Khởi động Spark Master : *start-master.sh*
* Khởi động Spark Worker : *start-slave.sh spark://192.168.99.101:7077,192.168.99.102:7077*
* Sau đó 2 chương trình Spark Master sẽ có trạng thái như sau

**

* Nếu trong trường hợp Spark Leader bị mất kết nối, Spark Standby sẽ thay thế nó và tiếp tục chạy chương trình

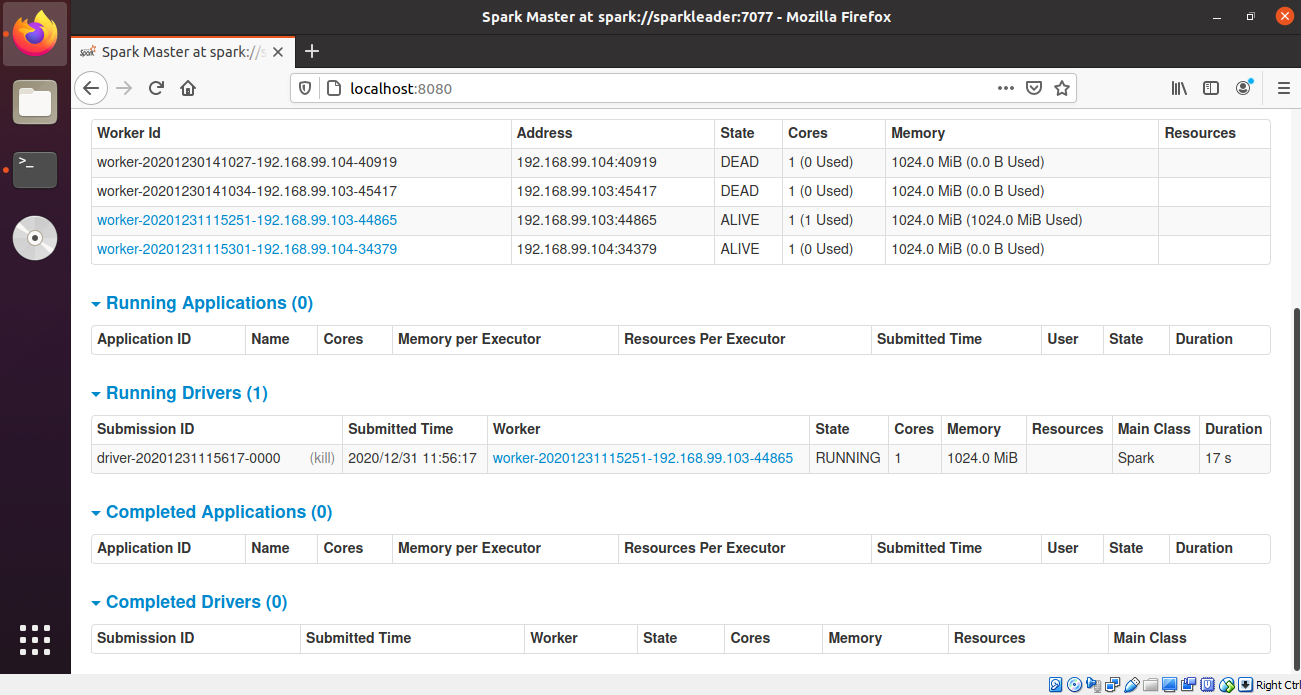
**



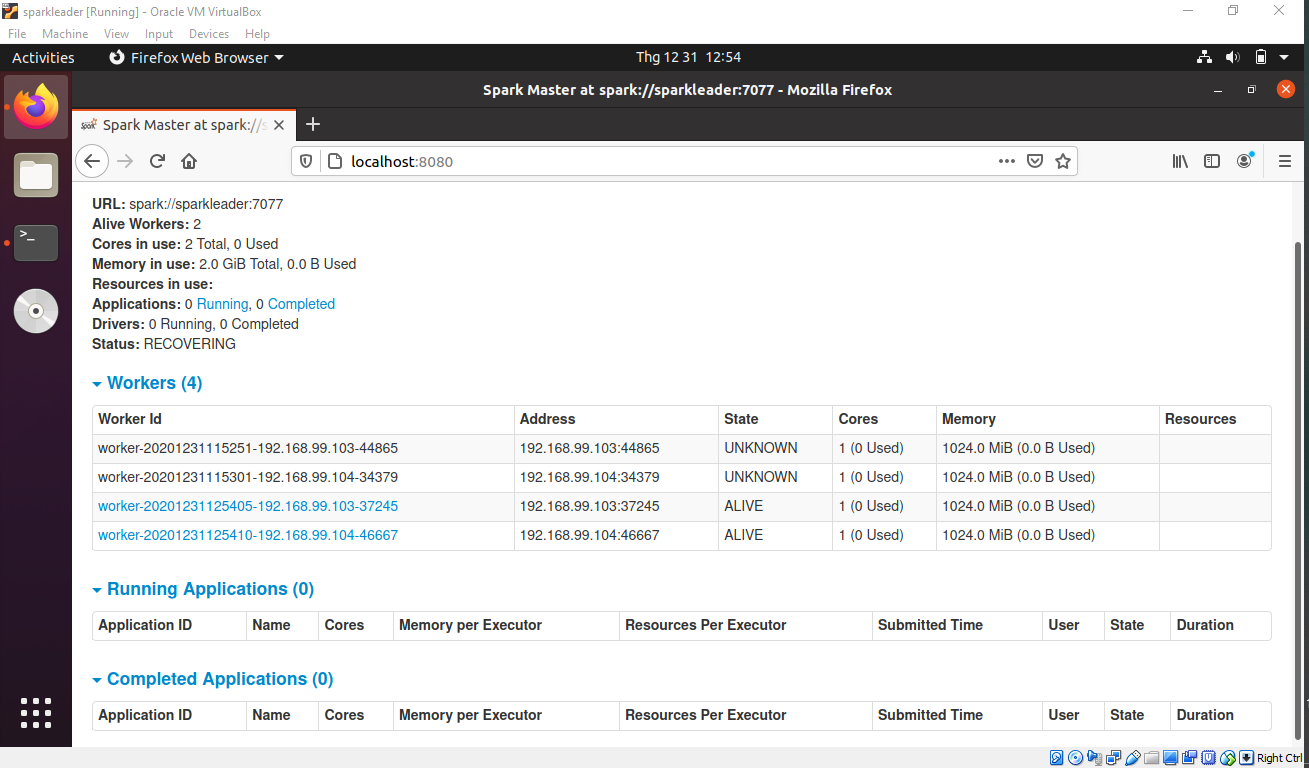
* Submit chương trình StockConsumer.jar ở trên máy chứa Zookeeper Server đẩy sang cho Spark

*Spark-submit –master spark://192.168.99.101:7077,192.168.99.102:7077 –deploy-mode cluster –supervise hdfs://masternode:9000/SparkConsumer.jar*

Chương trình Spark sẽ được chạy ở chế độ cluster với một tham số là bổ sung là –supervise cho phép Driver(Spark Master) tự khởi động lại trong trường hợp gặp lỗi trả về là -1



* Trong một số trường hợp, Spark đang trong quá trình thực thi mà bị ngắt kết nối đột ngột, sau đó được khởi động lại thì nó dựa vào log được đặt trong thư mục data của $ZOOKEEPER\_HOME để khôi phục lại các tính toán còn đang dang dở



* Submit chương trình StockPredict.py sẽ có khác một chút là buộc phải chạy ở chế độ client do Spark ở chế độ này không hỗ trợ Python

*Spark-submit –master spark://192.168.99.101:7077,192.168.99.102:7077 hdfs://masternode:9000/StockPredict.py*

* Chương trình sau khi chạy sẽ sinh ra một tệp là result.csv chứa kết quả dự đoán
  + - 1. Giao diện bên phía Web

