A black and yellow sign with white text

Description automatically generated

|  |
| --- |
| **THE 2nd SEMESTER FINAL PROJECT**  ***“Developing Automatic System for Data Warehouse & Analysis”*** |

**Developed by Kieu Thi Huyen Trang**

**Members:**

|  |  |  |
| --- | --- | --- |
| **No.** | **Student Name** | **Student ID** |
| 1 | Kieu Thi Huyen Trang | Student1465625 |

* **Class No.:** DT2304L
* **Start Date:** Aug 5th, 2024.
* **End Date:** Aug 27th, 2024.
* **Name of the Coordinator:** MR. Huynh Nam

**Date of Submission:** Aug 27th**,** 2024

**Table of Contents**

[I. DATA INTRODUCTION 3](#_Toc175840745)

[1. About the website 3](#_Toc175840746)

[2. Data structure 3](#_Toc175840747)

[2.1. Columns and rows of numerous details 3](#_Toc175840748)

[2.2. Columns and rows of news related to each stock: 5](#_Toc175840749)

[II. THE AUTOMATIC SYSTEM OF DATA PROCESSING 7](#_Toc175840750)

[1. System overview 7](#_Toc175840751)

[2. Component details 8](#_Toc175840752)

[2.1. Docker system 8](#_Toc175840753)

[2.2. Microsoft ecosystem: 9](#_Toc175840754)

[2.3. Airflow system 10](#_Toc175840755)

[III. PROGRAMMING 10](#_Toc175840756)

[A – BUILDING DOCKER CONTAINERS 10](#_Toc175840757)

[1. Scrapy 10](#_Toc175840758)

[2. MongoDB & Kafka 12](#_Toc175840759)

[3. Spark 15](#_Toc175840760)

[4. Postgres & Adminer 17](#_Toc175840761)

[5. Ubuntu 22.04 18](#_Toc175840762)

[B – BUILDING CONTAINER APPLICATIONS 20](#_Toc175840763)

[1. Crawling raw data and inserting crawled data into MongoDB 20](#_Toc175840764)

[1.1. Numerous data: 20](#_Toc175840765)

[1.2. Text data: 25](#_Toc175840766)

[2. Creating and registering Kafka topics from MongoDB 29](#_Toc175840767)

[3. Retrieving data from Kafka topics followed by transforming and cleaning data using PySpark 31](#_Toc175840768)

[4. Creating database for data warehouse using SQL commands from Docker container in Linux 39](#_Toc175840769)

[5. Removing duplicates, creating data frames and inserting data frames into SQL Server database using PySpark 40](#_Toc175840770)

[6. Creating cube database using MS Visual Studio with SSAS plugin 45](#_Toc175840771)

[7. Creating Airflow tasks for automatic process 45](#_Toc175840772)

[IV. FINAL PRODUCTS 59](#_Toc175840773)

[1. A galaxy schema data warehouse: 59](#_Toc175840774)

[2. Dashboards and analysis 60](#_Toc175840775)

[2.1. Overview of 10 stocks 60](#_Toc175840776)

[2.2. Relationship between news and stock price 61](#_Toc175840777)

[2.3. Relationship between stock price and volume 61](#_Toc175840778)

# I. DATA INTRODUCTION

## 1. About the website

There are many websites that provide public data of stocks, from inside or outside Vietnam such as vietstock.vn, yahoo.com, investing.com, cafef.vn … Among these, I have had good experience working with vietstock.vn the most, so it seems to be the most trust-worthy source for me to process. Moreover, Yahoo or Investing can have at least 10-year-time for the data of foreign stocks but that for Vietnamese stocks are more limited than Vietstock’s.

However, Vietstock just publicizes very few information of stock trading transactions. For examples, they give one-month data for non-account users and one-year for free-account ones. Due to the requirement of the project that we need to analyze the relation of prices, volumes and news over a period a year for each of 10 chosen stocks, I decided to agree with a one-year span for our results.

The primary requirement of the project is that I need to get information of 10 listed stocks from the vietstock.vn website. Therefore, it is necessary to build a system that is able to crawl data and push it into databases to clean it before continue to write it into a data warehouse connecting to power BI for visualization.

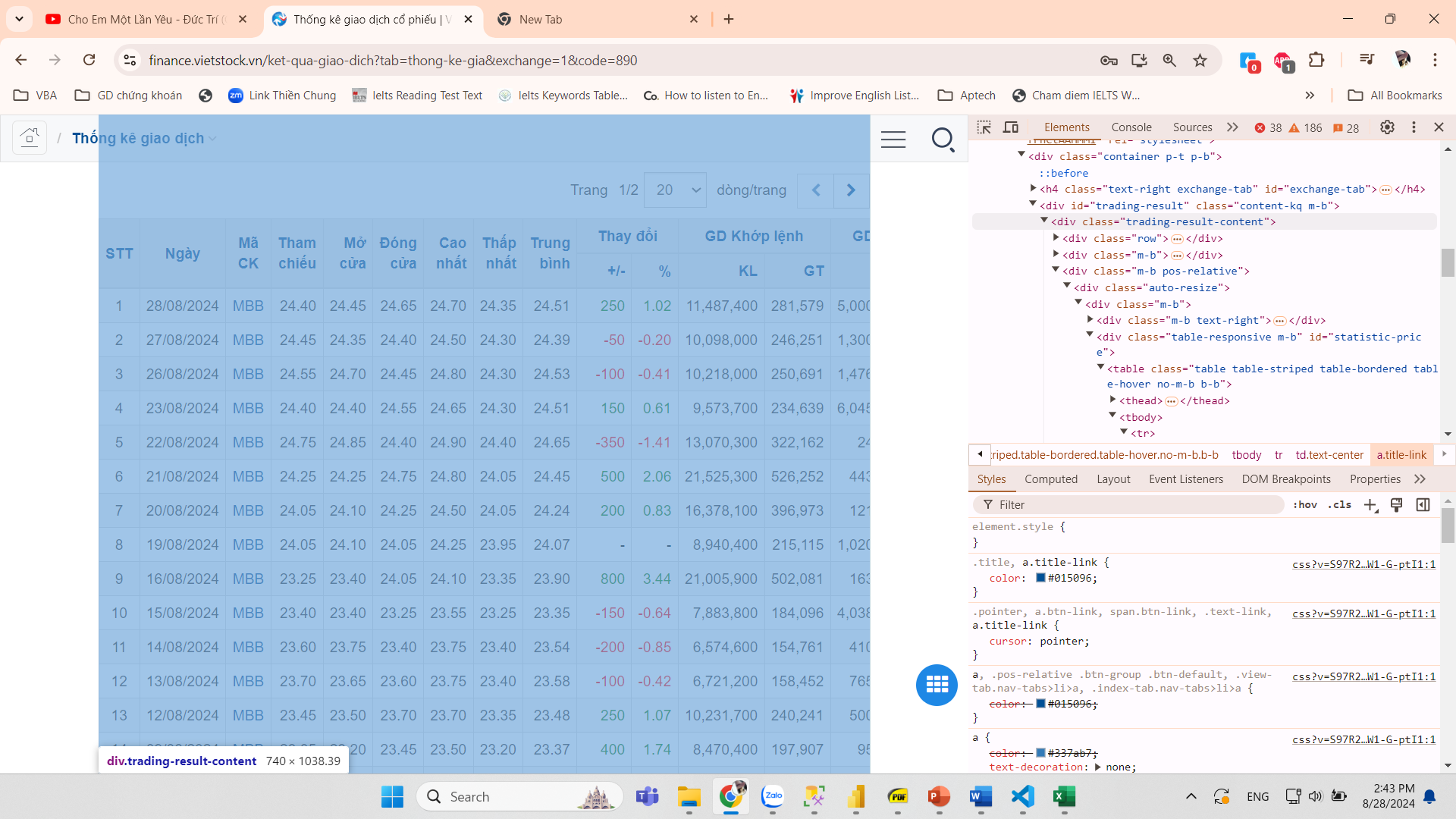
## 2. Data structure

### 2.1. Columns and rows of numerous details

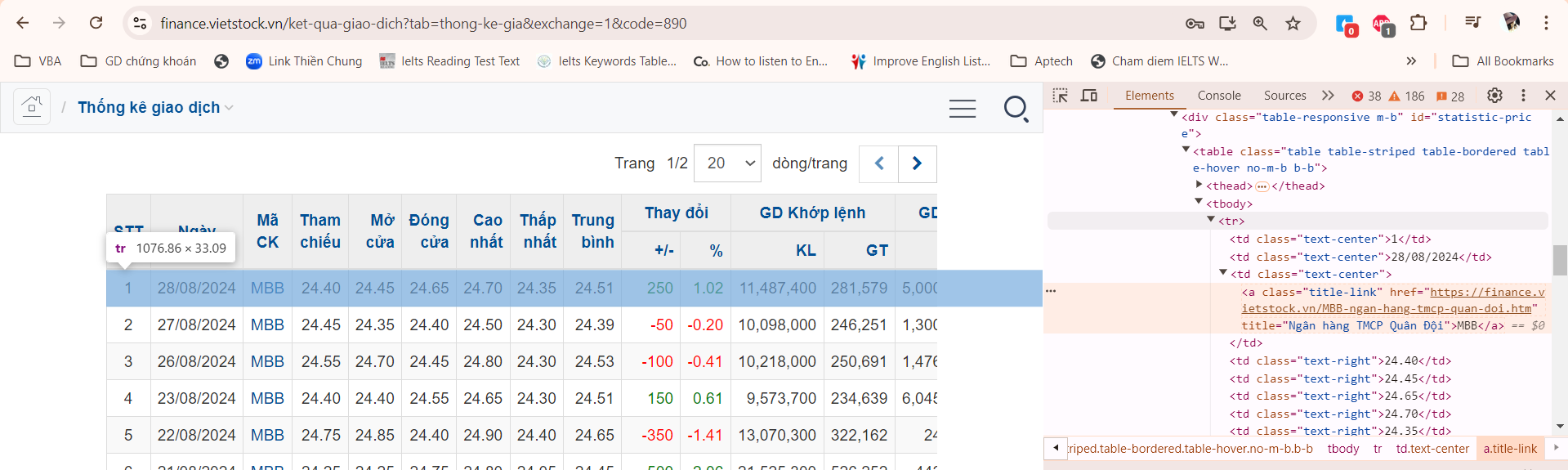
#### 2.1.1. URL link structure:

Vietstock has the same structure in the url link for each stock which starts with <https://finance.vietstock.vn/ket-qua-giao-dich?tab=thong-ke-gia> string and appended by "&exchange=" + exchangelist1[i] + "&code=" + linkcode1[i]” string. Here, “exchangelist1[i]” is the number code, for example: 1 for “HOSE” and “2” for “HNX” and so on while linkcode1[i] is the number code of each stock. For example, the one for MBB stock is 890, that of TCB is 13240.

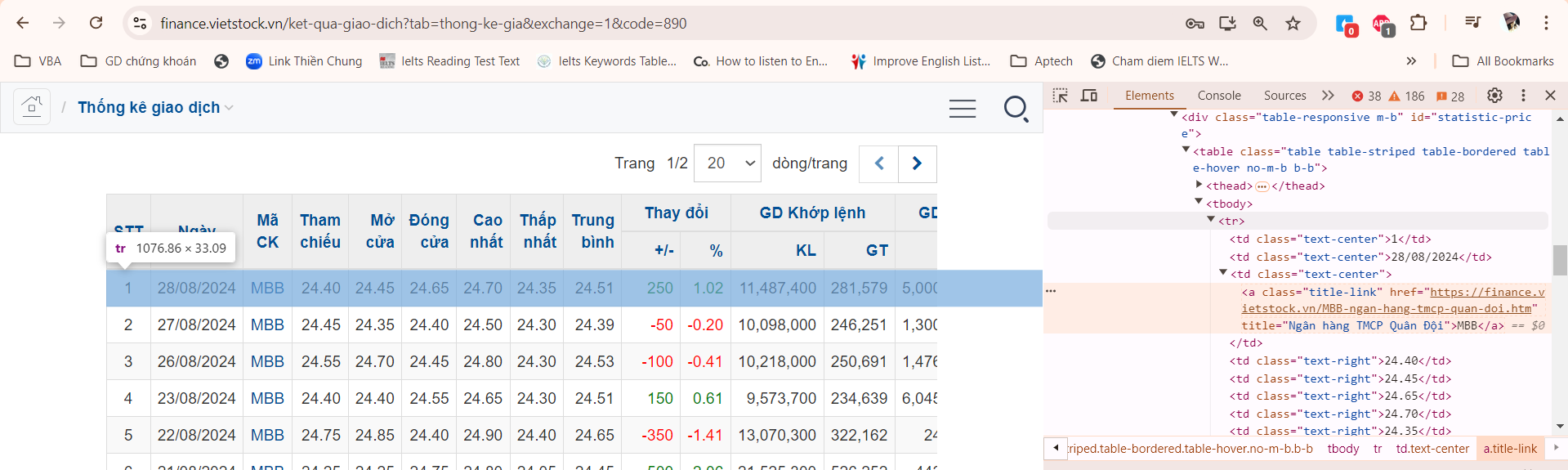
#### 2.1.2. Page source:



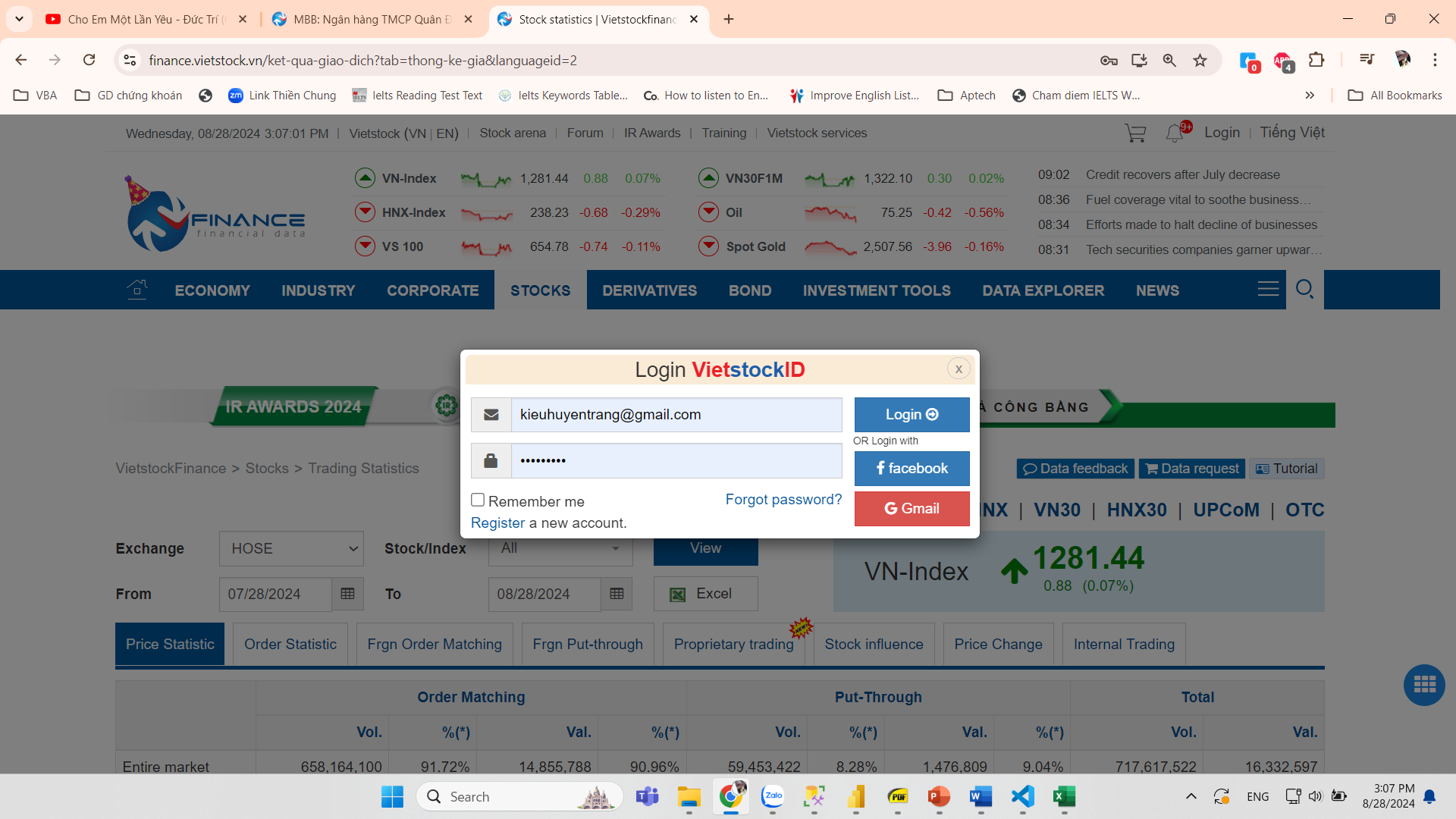
The whole table is included in “trading-result” class divided into 20 separate pages. This table is dynamically loaded content that there are 20 rows and 18 columns in each page. We need to click on the Next button for loading details of the next page.



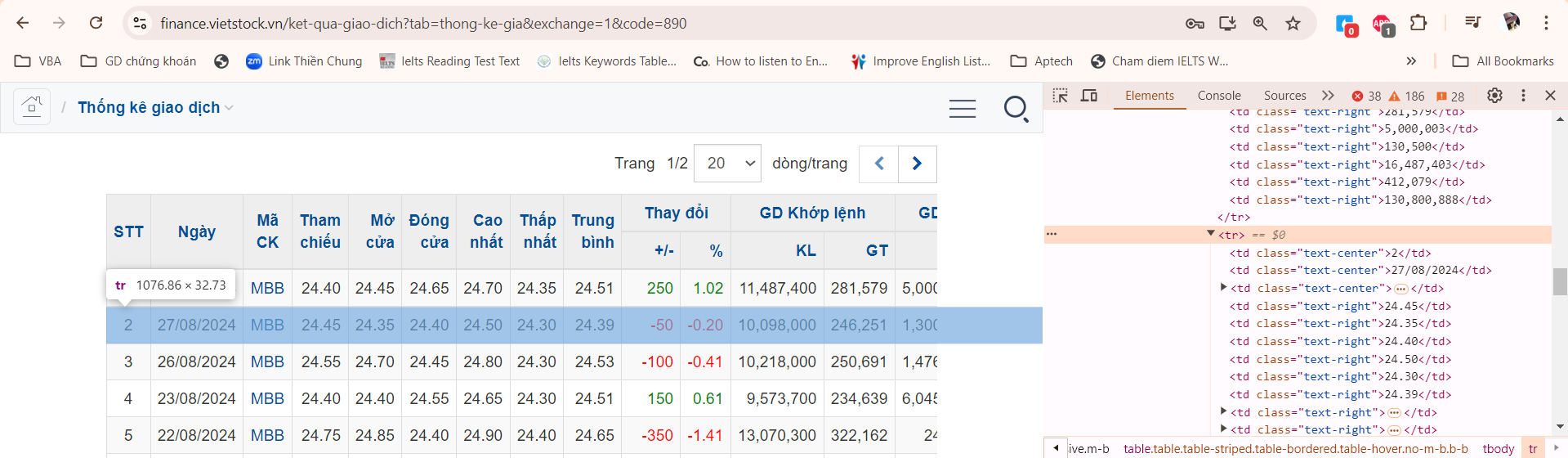
Next button

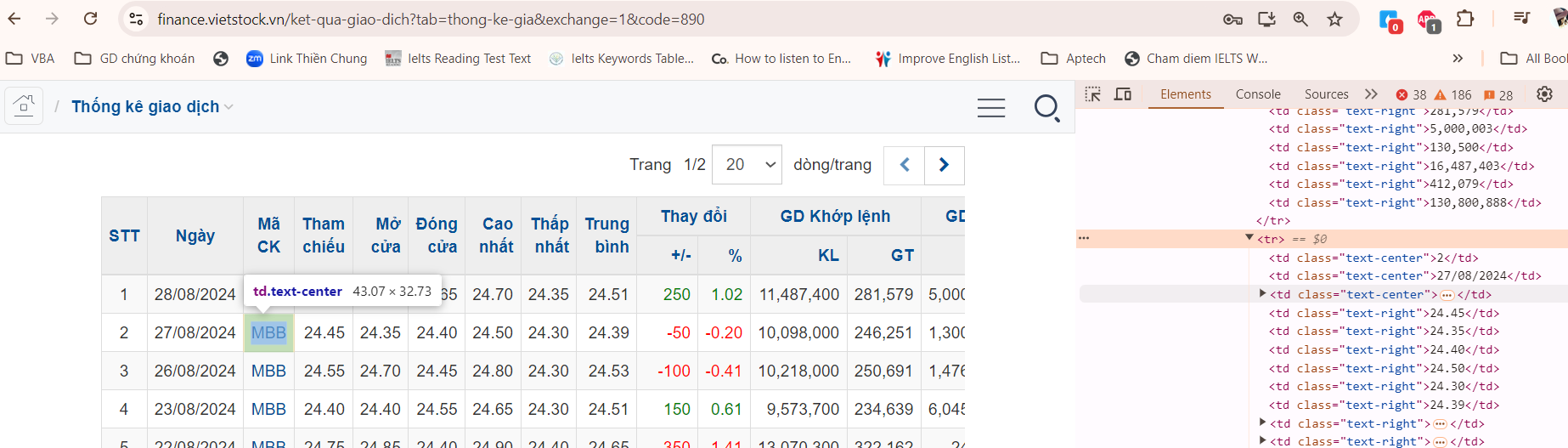


Noticeably, unregistered user can see only the first page of the table, which means that the result is shown in one-month period only. In contrast, registered user can have the latest one-year information. Therefore, I need to log in to the website before scrawling data.



Specifically, each row is wrapped in a “tr” class under “trading-result”, and each cell of the row contained in a “td” class as shown in the below pics:





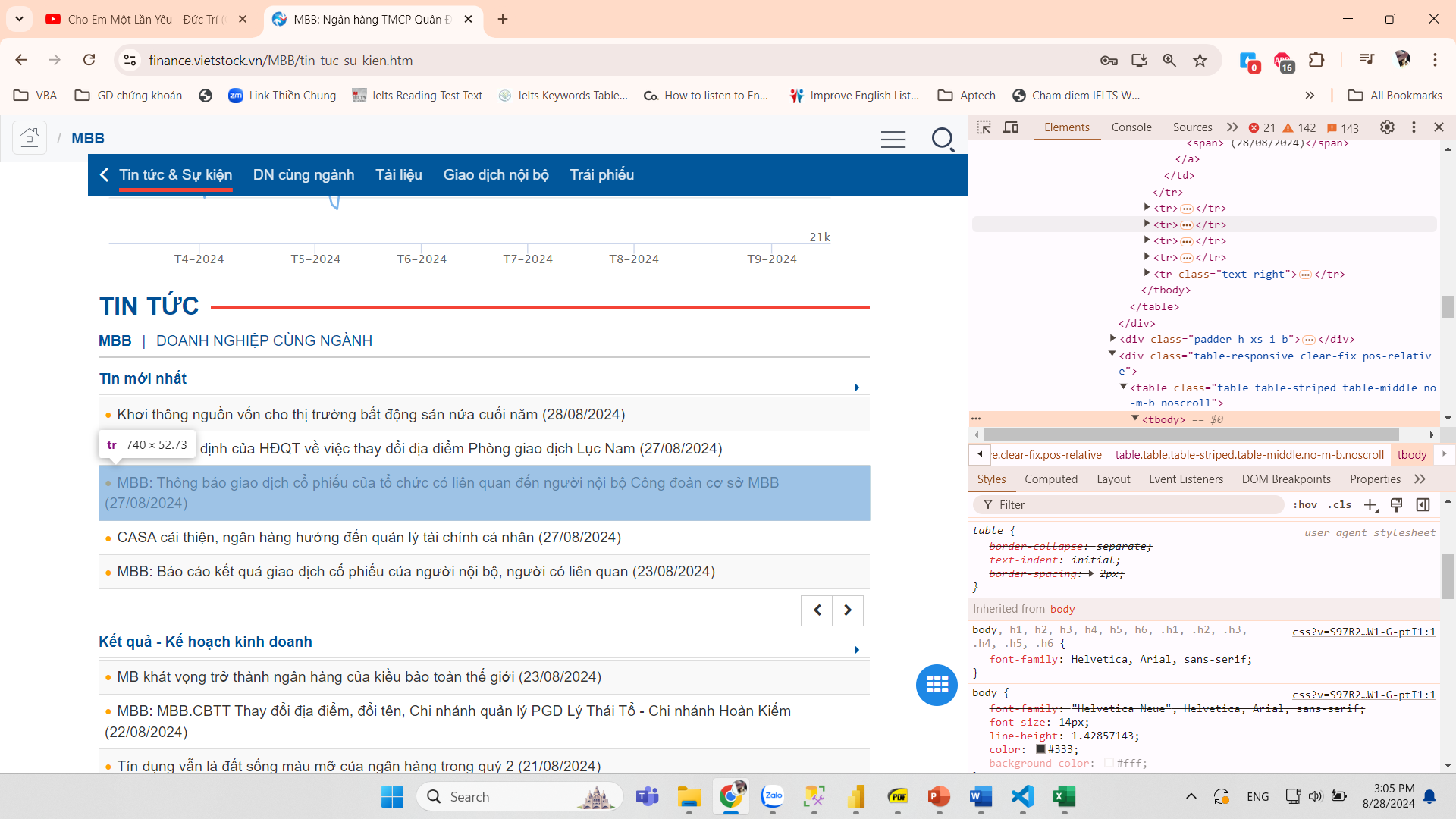
### 2.2. Columns and rows of news related to each stock:

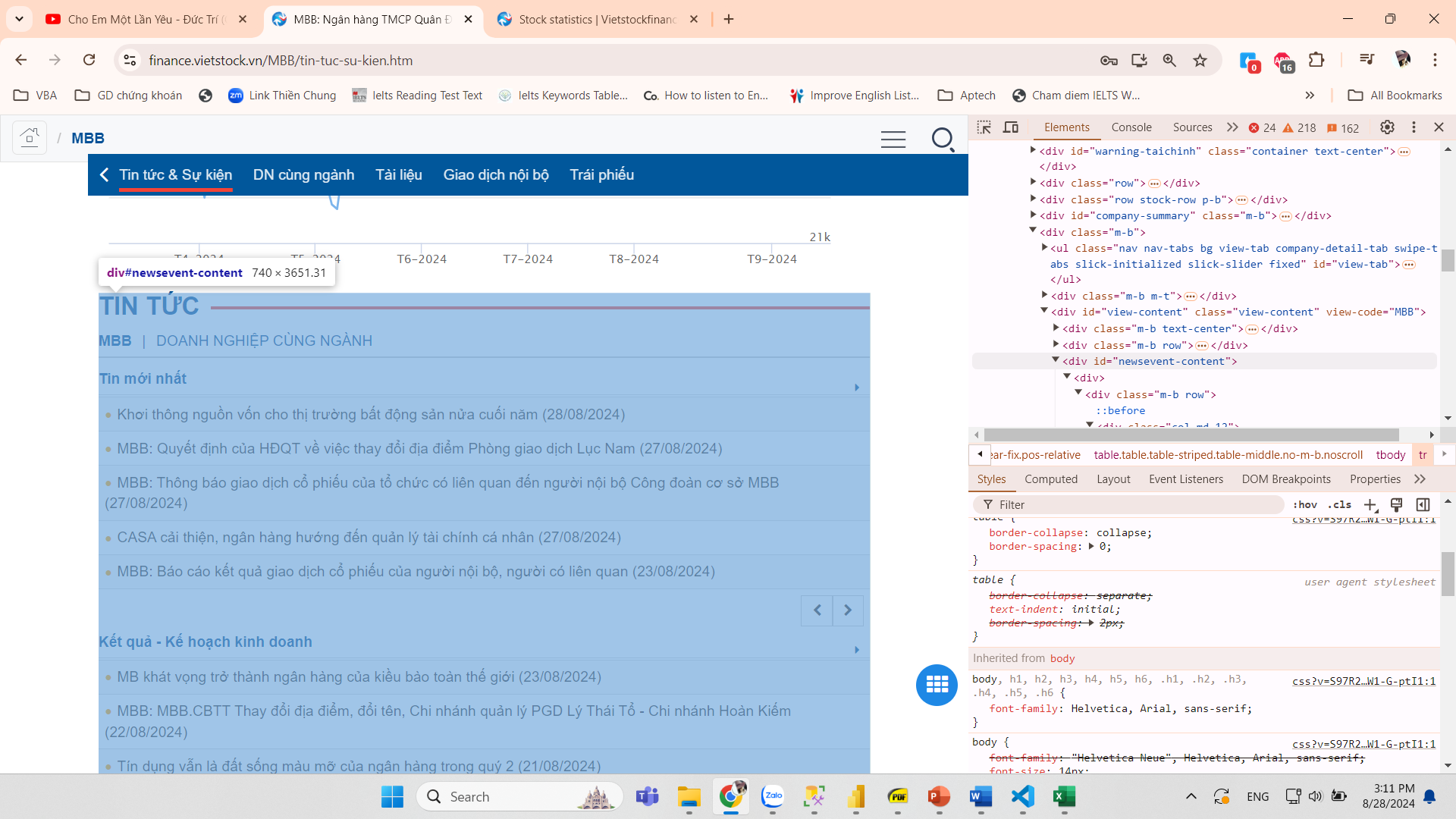
#### 2.1.1. URL link structure:

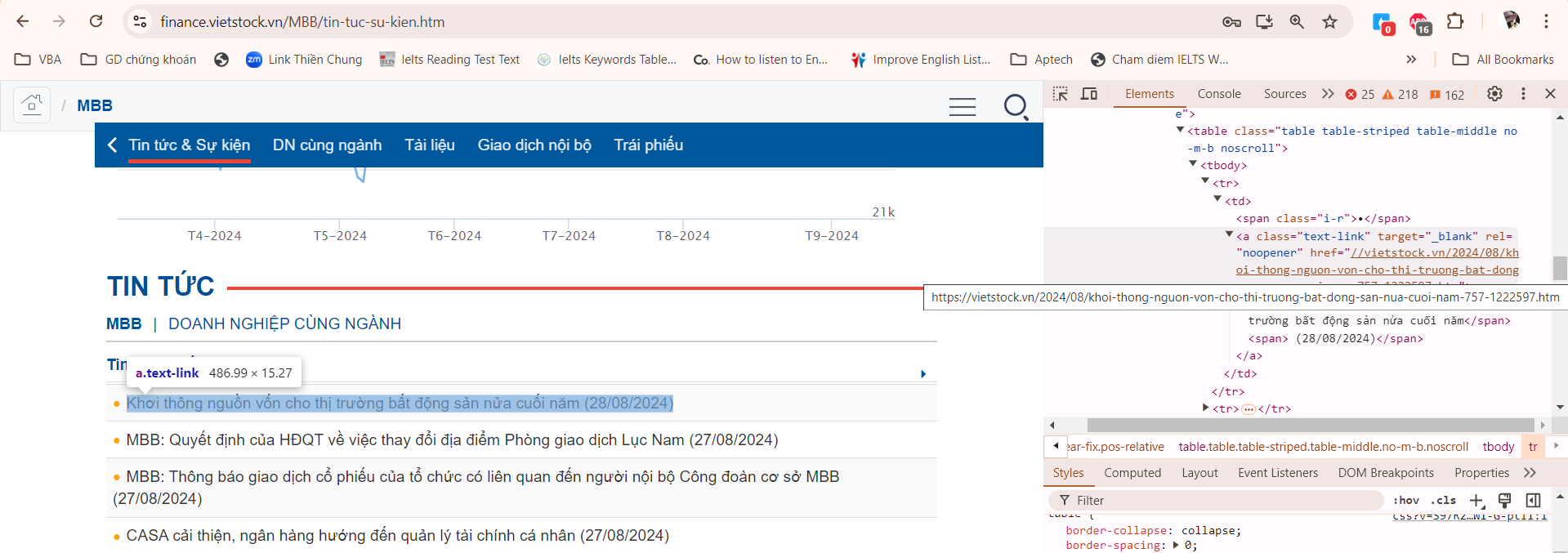
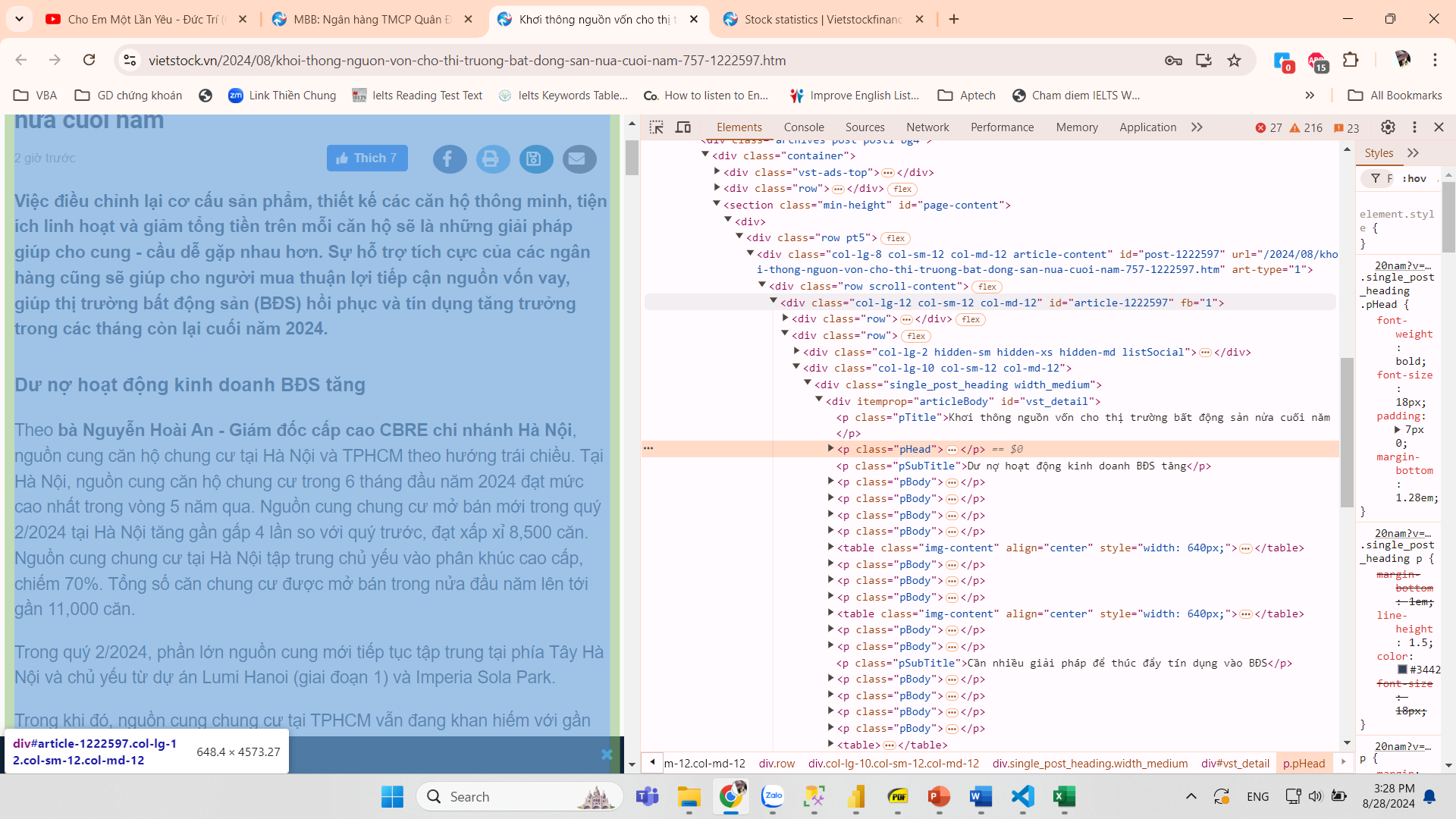
Each stock has its own URL link where all latest news is contained. The same structure of the URL for each stock is “https://finance.vietstock.vn/"+stocklist1[i]+"/tin-tuc-su-kien.htm”. Here, stocklist1[i] is the stock code such as MBB, TCB and so on. Thus, what I need to do is just replacing stocklist1[i] with each stock code I need to analyze.

#### 2.1.2. Page source:

Differing to numerous data, news URLs are all contained in “a” class/tag under the element with the “newsevent-content” ID:





In detail, each URL link is wrapped in “href” attribute of the element with “a” tag:  
Finally, what I need to extract is the news’s body text inside all elements of the “div.col-lg-10.col-sm-12.col-md-12” class for each URL link:  


# II. THE AUTOMATIC SYSTEM OF DATA PROCESSING

## 1. System overview

This diagram shows the organization and process of the system, starting with crawling data from the website using Scrapy and ending with visualizing my analysis using power BI dashboards:



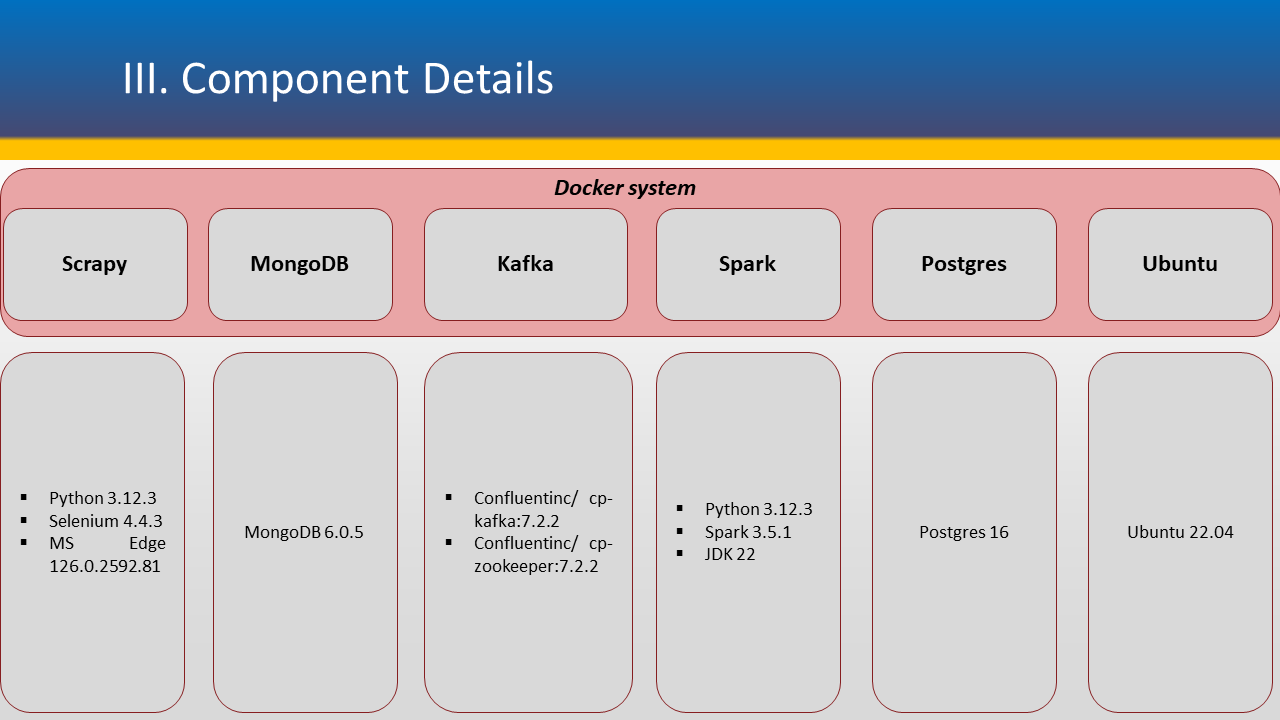
The process is divided into three main parts:

* The first one is to scraping raw data from the Vietstock website before transforming, cleaning and storing it into a Postgres database while the second phrase involves in building a data warehouse and a set of dashboards.
* When the first stage requests for numbers of Docker containers installed in Linux operating system, the second one relates to Microsoft ecosystem. Next, I will explain the components in details.
* The master controller of all the tasks is Airflow. It runs the process automatically, starting from scraping data and ending with inserting data into data warehouse.

## 2. Component details

### 2.1. Docker system

Docker is a software platform that allows us to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run, including libraries, system tools, code, and runtime. By using Docker, we can quickly deploy and scale applications into any environment and be confident that our code will run. In my case, there are six biggest set of containers participated in the back-end process to get raw data and process it as shown in the picture:

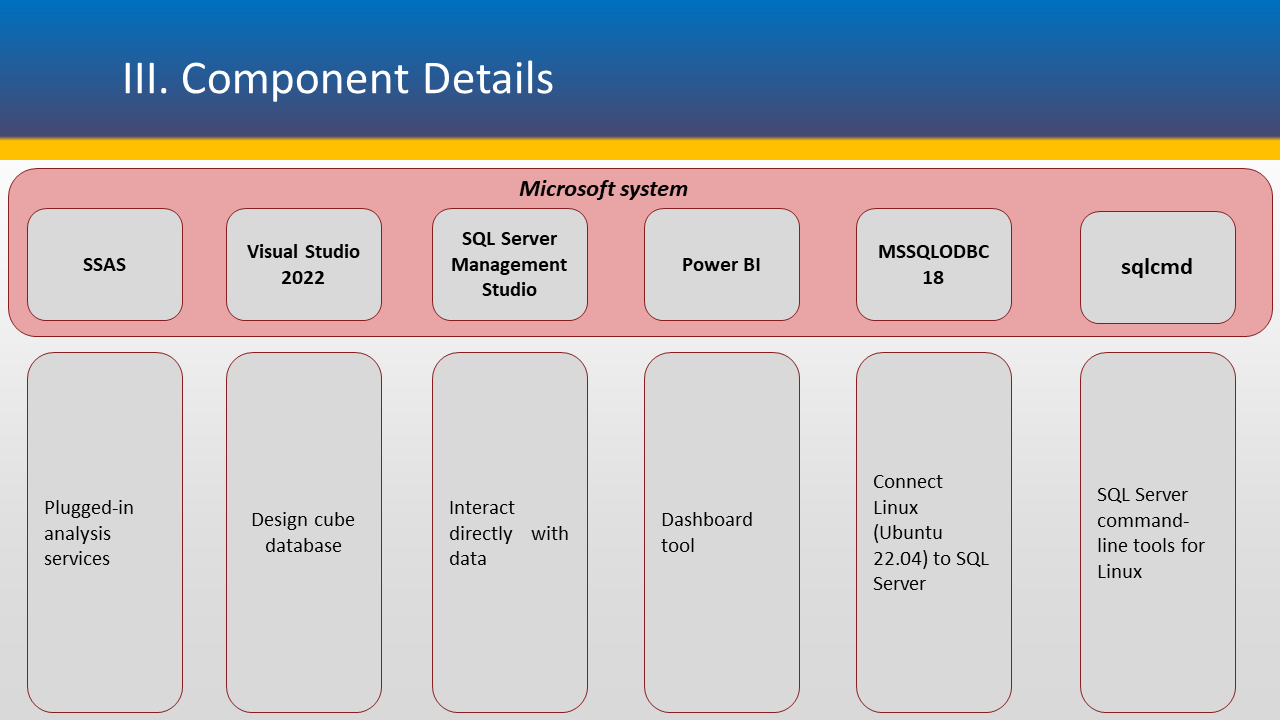


* **Scrapy:** Built from Python 3.12.3 base image, it is installed some more necessary packages such as Selenium 4.4.3, Microsoft Edge 126.0.2592.81 browser and driver for scraping data directly from the Vietstock website.
* **MongoDB:** Built from MongoDB 6.0.16 base image in replication mode according to the introduction from MongoDB official website. MongoDB does not support Kafka streaming connector to stand-alone mode, therefore I need to set up a replication mode based on their tutorials. This container helps storing raw data, both structured and unstructured ones at high speed. Besides, it remains and creates most essential information of data such as timestamp, which is important for real-time aggregation
* **Kafka:** Built from a set of five various services that are zookeeper, broker, connect, rest-proxy and schema-registry provided by MongoDB. While zookeeper is built from “confluentinc/cp-zookeeper:7.2.2” image, broker is built from “confluentinc/cp-kafka:7.2.2” image, connect is set up from “confluentinc/cp-kafka-connect:7.2.5” image, rest-proxy is based on “confluentinc/cp-kafka-rest:7.2.2” image, and schema-resigtry is based on “confluentinc/cp-schema-registry:7.2.2” image. Kafka helps boosting the speed of processing data, especially for real-time streaming data. I choose to connect Kafka between MongoDB and PySpark for two reasons:
* Kafka can keep raw data from MongoDB in topics so that avoid the data loss when streaming and flowing inside the pipeline.
* Kafka can boost up the speed of processing streamed data nearly to immediately due to the distribution mechanism of partitions that divide and distribute data into smaller batches to process in parallel.
* **Spark:** Built from Python 3.12.3 base image. The supplementing packages to be installed are Spark 3.5.1, JDK 22, postgresql-42.7.3.jar (Postgres driver to help connect Spark and Postgres). Most of the data processing tasks including cleaning, transforming, dealing with null data, writing cleaned data into different external storages in various operating system such as Linux and Windows are handled by Spark container. Therefore, it is vital to install correct packages as well as acceptable drivers for connecting those applications.
* **Postgres:** Built from Postgres 16 image.
* **Ubuntu:** Built from Ubuntu 22.04 image, this is the environment for installing and running MS SQL ODBC 18 and SQL CMD (mentioned in the next content).

Above is a distinguishment of a set of containers based on the difference of their features and roles in the whole system. In fact, we can totally reduce the number of containers by building a bigger container based on Ubuntu 22.04 and install all needed packages inside it such as Python, Spark, JDK …, and other drivers.

### 2.2. Microsoft ecosystem:

When Linux in association with Docker is a back-end system with its strength in machine resource utility and its flexibility to deal with the difference in various versions of applications, Microsoft with Windows operating system takes advantage of its user-friendly interface which is the good choice for dash board presentation. In other words, Power BI is quite a common software for visualizing which means that choosing it leads me to an ecosystem of Microsoft with supported applications for building data warehouse such as MS Visual Studio with SQL Server Analysis Services (SSAS, in short) plugin as shown in the diagram:



* **MS SQL ODBC 18:** This is a Linux driver developed by Microsoft and considered as the key component to connect Linux and Windows. Because I plan to work with Power BI, I need to apply Microsoft driver for the compatibility between two systems. However, there is one issue that Microsoft only supports some listed versions of Ubuntu (like 18.04, 20.04, 22.04) which exclude my Ubuntu 23.10 virtual machine’s. My solution for this problem is to build my own Ubuntu container based on ubuntu image version 22.04.
* **SQL CMD:** This is the command-line tool inside MS SQL ODBC 18 package, which is used for executing SQL commands on Linux machine for remotely manipulating databases in SQL Server Management Studio in Windows machine.
* **SQL Server Management Studio:** This is a front-end application for interacting with database. From here, I can see the database created by SQL commands from Linux machine as well as its tables with complete data. Also, it helps connect with SQL Server Analysis Services in Microsoft Visual Studio 2022 for working with cube database. My practical version is the latest release, 20.2.
* **Microsoft Visual Studio:** The practical version of my machine is Community 2022 which is used for developers.
* **SQL Server Analysis Services (SSAS):** This is a plugin installed inside Microsoft Visual Studio for designing cube database. It helps making some calculated columns and measures that pre-calculate the results of formulas for better performance of Power BI queries.
* **Power BI:** This is a common toolof Microsoft for visualizing. It can connect to SQL Server in import or live connect modes.

### 2.3. Airflow system

Airflow 2.9.3 is installed directly to my Ubuntu 23.10 virtual machine using “pip” installation under a Python virtual environment to avoid conflicts with built-in Python packages of Ubuntu 23.10. It is not necessary to create one more Docker container for Airflow due to the machine resource utility. I need to take advantage of all the installed packages inside my virtual machine. Moreover, the purpose of Airflow is running all the tasks automatically no matter what they are inside or outside my Docker system.

# III. PROGRAMMING

# A – BUILDING DOCKER CONTAINERS

## 1. Scrapy

* The Dockerfile used to build scrapy container:

FROM python:latest

# Install necessary packages

RUN apt-get update && apt-get install -y

RUN apt-get update && apt-get install -y gnupg2

RUN apt-get update && apt-get install -y gnupg

RUN apt-get -y install apt-transport-https

RUN apt-get -y install software-properties-common

RUN apt-get -y install libglib2.0-0

RUN apt-get -y install libnss3

RUN apt-get -y install libgconf-2-4

RUN apt-get -y install libfontconfig1

RUN apt-get -y install libxss1

RUN apt-get -y install libappindicator1

RUN apt-get -y install unzip

RUN apt-get -y install nano

RUN apt-get -y install vim

# Bổ sung để fix lỗi MS Edge crashed:

RUN apt-get -y install libatk-bridge2.0-0

RUN apt-get -y install libatspi2.0-0

RUN apt-get -y install libdrm2

RUN apt-get -y install libgbm1

RUN apt-get -y install libgtk-3-0

RUN apt-get -y install libu2f-udev

RUN apt-get -y install libvulkan1

RUN apt-get -y install libxkbcommon0

RUN apt-get -y install xdg-utils

RUN apt-get update && apt-get install -y \

    wget \

    xvfb \

    fonts-liberation \

    libappindicator3-1 \

    libgbm-dev

RUN rm -rf /var/lib/apt/lists/\*

# Update the package list with the new repository

RUN apt-get update

# Copy the Microsoft Edge Browser setup file into the Docker image

COPY microsoft-edge-stable\_126.0.2592.81-1\_amd64.deb /tmp/

# Install Microsoft Edge Browser from the .deb file

RUN dpkg -i /tmp/microsoft-edge-stable\_126.0.2592.81-1\_amd64.deb || apt-get install -f -y \

    && rm /tmp/microsoft-edge-stable\_126.0.2592.81-1\_amd64.deb

# Copy MS Edge Vdriver zip file from local machine vào image:

COPY msedgedriver /usr/local/bin

# Set execute permissions for msedgedriver

RUN chmod +x /usr/local/bin/msedgedriver

RUN chmod 777 /usr/local/bin/msedgedriver

# Export the path of EdgeDriver to the PATH environment variable

ENV PATH="/usr/local/bin/msedgedriver:${PATH}"

# Export the path of chromedriver to the PATH environment variable

ENV PATH="/usr/local/bin/chromedriver-linux64:${PATH}"

# set display port to avoid crash

ENV DISPLAY=:99

# Copy requirements.txt into the Docker image

COPY requirements.txt /tmp/

# Install Python packages from requirements.txt

RUN pip install --upgrade pip \

    && pip install -r /tmp/requirements.txt

# Cleanup

RUN rm /tmp/requirements.txt

# Set the working directory

WORKDIR /

# Create a logs directory

RUN mkdir -p /log/

# Set the environment variable for logs directory

ENV LOG\_DIR=/log/

# Copy the application files

COPY Ticker\_table-xvfb.xlsx /

COPY dbstockcrawl.py /

COPY contentpubver3.py /

COPY start\_service.sh /

COPY startpubver3.sh /

RUN chmod 777 /start\_service.sh

RUN chmod +x /startpubver3.sh

# Set command

CMD ["bash"]

* “start\_service.sh”:

#!/bin/bash

# Set up the timezone for the Docker container

ln -snf /usr/share/zoneinfo/Asia/Ho\_Chi\_Minh /etc/localtime

echo "Asia/Ho\_Chi\_Minh" > /etc/timezone

# Start Xvfb and capture its PID

Xvfb :99 -screen 0 1920x1080x24 & python "dbstockcrawl.py"

* “startpubver3.sh”:

#!/bin/bash

# Set up the timezone for the Docker container

ln -snf /usr/share/zoneinfo/Asia/Ho\_Chi\_Minh /etc/localtime

echo "Asia/Ho\_Chi\_Minh" > /etc/timezone

# Start Xvfb in the background and capture its PID

Xvfb :99 -screen 0 1920x1080x24 & python "contentpubver3.py"

## 2. MongoDB & Kafka

The docker compose file to build a set of MongoDB and Kafka service containers:

services:

  zookeeper: #1

    image: confluentinc/cp-zookeeper:7.2.2

    hostname: zookeeper

    container\_name: zookeeper

    networks:

      - localnet

    environment:

      ZOOKEEPER\_CLIENT\_PORT: 2181

      ZOOKEEPER\_TICK\_TIME: 2000

  broker: #2

    image: confluentinc/cp-kafka:7.2.2

    hostname: broker

    container\_name: broker

    depends\_on:

      - zookeeper

    networks:

      - localnet

    environment:

      KAFKA\_BROKER\_ID: 1

      KAFKA\_ZOOKEEPER\_CONNECT: "zookeeper:2181"

      KAFKA\_LISTENERS: LISTENER\_1://broker:29092,LISTENER\_2://broker:9092

      KAFKA\_ADVERTISED\_LISTENERS: LISTENER\_1://broker:29092,LISTENER\_2://localhost:9092

      KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: LISTENER\_1:PLAINTEXT,LISTENER\_2:PLAINTEXT

      KAFKA\_INTER\_BROKER\_LISTENER\_NAME: LISTENER\_1

      KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

      KAFKA\_GROUP\_INITIAL\_REBALANCE\_DELAY\_MS: 0

      CONFLUENT\_SUPPORT\_CUSTOMER\_ID: "anonymous"

      KAFKA\_DELETE\_TOPIC\_ENABLE: "true"

  connect: #5

    build:

      context: .

      dockerfile: connect.Dockerfile

    ports:

      - "35000:35000"

    hostname: connect

    container\_name: connect

    depends\_on:

      - zookeeper

      - broker

    networks:

      - localnet

    environment:

      KAFKA\_JMX\_PORT: 35000

      KAFKA\_JMX\_HOSTNAME: localhost

      CONNECT\_BOOTSTRAP\_SERVERS: "broker:29092"

      CONNECT\_REST\_ADVERTISED\_HOST\_NAME: connect

      CONNECT\_REST\_PORT: 8083

      CONNECT\_GROUP\_ID: connect-cluster-group

      CONNECT\_CONFIG\_STORAGE\_TOPIC: docker-connect-configs

      CONNECT\_CONFIG\_STORAGE\_REPLICATION\_FACTOR: 1

      CONNECT\_OFFSET\_FLUSH\_INTERVAL\_MS: 10000

      CONNECT\_OFFSET\_STORAGE\_TOPIC: docker-connect-offsets

      CONNECT\_OFFSET\_STORAGE\_REPLICATION\_FACTOR: 1

      CONNECT\_STATUS\_STORAGE\_TOPIC: docker-connect-status

      CONNECT\_STATUS\_STORAGE\_REPLICATION\_FACTOR: 1

      CONNECT\_ZOOKEEPER\_CONNECT: "zookeeper:2181"

      CONNECT\_PLUGIN\_PATH: "/usr/share/java,/usr/share/confluent-hub-components"

      CONNECT\_CONNECTIONS\_MAX\_IDLE\_MS: 180000

      CONNECT\_METADATA\_MAX\_AGE\_MS: 180000

      CONNECT\_AUTO\_CREATE\_TOPICS\_ENABLE: "true"

      CONNECT\_KEY\_CONVERTER: "org.apache.kafka.connect.json.JsonConverter"

      CONNECT\_VALUE\_CONVERTER: "org.apache.kafka.connect.json.JsonConverter"

  rest-proxy: #4

    image: confluentinc/cp-kafka-rest:7.2.2

    depends\_on:

      - zookeeper

      - broker

      - schema-registry

    hostname: rest-proxy

    container\_name: rest-proxy

    networks:

      - localnet

    environment:

      KAFKA\_REST\_HOST\_NAME: rest-proxy

      KAFKA\_REST\_BOOTSTRAP\_SERVERS: "broker:29092"

      KAFKA\_REST\_LISTENERS: "http://0.0.0.0:8082"

      KAFKA\_REST\_SCHEMA\_REGISTRY\_URL: "http://schema\_registry:8081"

  schema-registry: #3

    image: confluentinc/cp-schema-registry:7.2.2

    hostname: schema-registry

    container\_name: schema-registry

    depends\_on:

      - broker

    networks:

      - localnet

    environment:

      SCHEMA\_REGISTRY\_HOST\_NAME: schema-registry

      SCHEMA\_REGISTRY\_KAFKASTORE\_BOOTSTRAP\_SERVERS: "broker:29092"

      SCHEMA\_REGISTRY\_KAFKASTORE\_CONNECTION\_URL: "zookeeper:2181"

      SCHEMA\_REGISTRY\_LISTENERS: http://0.0.0.0:8081

  mongo1:

    image: "mongodb-kafka-base-mongod:1.0"

    container\_name: mongo1

    ports:

      - "35001:27017"

    build:

      context: .

      dockerfile: mongo.Dockerfile

    command: --replSet rs0 --oplogSize 128

    depends\_on:

      - zookeeper

      - broker

      - connect

    networks:

      - localnet

      - smnet2 #

    restart: always

    volumes: #

      - mongo\_data:/data/db

      - /home/osboxes/mongo1backup:/data/backup #

  mongo1-setup:

    image: "mongodb-kafka-base-setup-mongod:1.0"

    container\_name: mongo1-setup

    build:

      context: .

      dockerfile: mongo.Dockerfile

    depends\_on:

      - mongo1

    networks:

      - localnet

    entrypoint:

      [

        "bash",

        "-c",

        "sleep 10 && mongosh --host mongo1:27017 config-replica.js && sleep 10",

      ]

    restart: "no"

networks:

  smnet2:

  localnet:

    attachable: true

volumes:

  mongo\_data:

## 3. Spark

* The Dockerfile to build spark container:

FROM python:3.12.3

# Upgrade pip

RUN pip install --no-cache-dir --upgrade pip

# Install OpenJDK (required for Spark) and other dependencies

RUN apt-get update && apt-get install -y

RUN apt-get -y update

RUN apt-get -y upgrade

RUN apt-get update

RUN apt-get install -y curl

RUN apt-get install -y python3-pip

RUN apt-get install -y nano

RUN apt-get install vim -y

RUN apt-get install -y iputils-ping

RUN apt-get install -y kafkacat

RUN apt-get install -y git

# Copy the local JDK tar.gz file into the Docker image

COPY jdk-22\_linux-x64\_bin.tar.gz /tmp/jdk-22\_linux-x64\_bin.tar.gz

# Install OpenJDK 22 from the local file

RUN apt-get update && apt-get install -y \

    wget && \

    tar -xzvf /tmp/jdk-22\_linux-x64\_bin.tar.gz -C /opt && \

    rm /tmp/jdk-22\_linux-x64\_bin.tar.gz

# Copy the Apache Spark tar.gz file from the local machine to the Docker image

COPY spark-3.5.1-bin-hadoop3.tgz /tmp/spark-3.5.1-bin-hadoop3.tgz

# Install Apache Spark 3.5.1

RUN tar -xzf /tmp/spark-3.5.1-bin-hadoop3.tgz -C /opt && \

    rm /tmp/spark-3.5.1-bin-hadoop3.tgz

# Set environment variables

ENV PYSPARK\_PYTHON=python3

ENV PYSPARK\_DRIVER\_PYTHON=python3

ENV PYSPARK\_VERSION=3.5.1

ENV SPARK\_HOME=/opt/spark-3.5.1-bin-hadoop3

ENV JAVA\_HOME=/opt/jdk-22

ENV PATH=$JAVA\_HOME/bin:$SPARK\_HOME/bin:$PATH

# Copy the PostgreSQL JDBC driver into the Docker image

COPY postgresql-42.7.3.jar ${SPARK\_HOME}/jars/

# Set working directory

WORKDIR /app

# Copy Python script and requirements file

COPY kafka-clients-3.8.0.jar /app/

COPY pyspark-requirements.txt /app/pyspark-requirements.txt

# Install Python dependencies

RUN pip install -r pyspark-requirements.txt

CMD ["bash"]

After building the spark container, we can write more entrypoint scripts and copy them into spark container.

* “entrypoint.sh”:

#!/bin/bash

# Set up the timezone for the Docker container

ln -snf /usr/share/zoneinfo/Asia/Ho\_Chi\_Minh /etc/localtime

echo "Asia/Ho\_Chi\_Minh" > /etc/timezone

# Start PySpark with the required packages

pyspark --packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.5.1 <<EOF

exec(open("/app/dbstockread.py").read())

EOFO

* “entrypoint\_content.sh”:

#!/bin/bash

# Set up the timezone for the Docker container

ln -snf /usr/share/zoneinfo/Asia/Ho\_Chi\_Minh /etc/localtime

echo "Asia/Ho\_Chi\_Minh" > /etc/timezone

# Start PySpark with the required packages

pyspark --packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.5.1 <<EOF

exec(open("/app/dbcontent-15.py").read())

EOFO

* “post2ssms.sh”:

#!/bin/bash

# Set up the timezone for the Docker container

ln -snf /usr/share/zoneinfo/Asia/Ho\_Chi\_Minh /etc/localtime

echo "Asia/Ho\_Chi\_Minh" > /etc/timezone

# Start PySpark with the required packages

pyspark --packages org.apache.spark:spark-sql-kafka-0-10\_2.12:3.5.1 --jars /app/sqljdbc\_12.8/enu/jars/mssql-jdbc-12.8.0.jre11.jar \

--conf "spark.driver.extraJavaOptions=-Duser.timezone=Asia/Ho\_Chi\_Minh" --conf "spark.executor.extraJavaOptions=-Duser.timezone=Asia/Ho\_Chi\_Minh" <<EOF

exec(open("/app/ssms5tables2.py").read())

EOF

## 4. Postgres & Adminer

I build a docker-compose file to create postgres along with adminer services in order to interact with Postgres database using Adminer Web UI. This is the content of the “adminerpostgres.yml” file:

version: "3.8"

services:

    postgres:

        image: postgres

        container\_name: postgres

        restart: always

        environment:

            - POSTGRES\_USER=admin

            - POSTGRES\_PASSWORD=admin

            - POSTGRES\_HOST=ppnet #

            - POSTGRES\_DB=db249

        volumes:

            - datavol:/var/lib/postgresql/data

        ports:

            - "5432:5432"

        networks:

            - ppnet

            - panet # ket noi adminer

    adminer:

        image: adminer # uses an image from a registry

        container\_name: adminer

        restart: always

        environment:

            - PGUSER=admin

            - PGPASSWORD=admin

            - PGDATABASE=db249

            - PGHOST=postgres # ppnet

            - PGPORT=5432

        ports:

            - "8080:8080"

        networks:

            - panet

networks:

    panet:

    ppnet:

volumes:

    datavol:

## 5. Ubuntu 22.04

I build the container from an ubuntu 22.04 image, then I install MS SQL ODBC package inside it using bash shell commands.

* The Dockerfile:

# Use the latest Ubuntu 24.04 image as the base

FROM ubuntu:22.04

# Set environment variables to non-interactive to avoid prompts during build

ENV DEBIAN\_FRONTEND=noninteractive

# Update and install required packages

RUN apt-get update && \

    apt-get upgrade -y && \

    apt-get install -y \

    python3 \

    python3-pip \

    curl \

    gnupg \

    apt-transport-https \

    unixodbc-dev \

    && rm -rf /var/lib/apt/lists/\*

# Create and activate a Python virtual environment, then install packages

RUN python3 -m venv /opt/venv && \

    /opt/venv/bin/pip install --upgrade pip && \

    /opt/venv/bin/pip install pyspark pyodbc pandas kafka

# Copy the local JDK tar.gz file into the Docker image

COPY jdk-22\_linux-x64\_bin.tar.gz /tmp/jdk-22\_linux-x64\_bin.tar.gz

# Install OpenJDK 22 from the local file

RUN apt-get update && apt-get install -y \

    wget && \

    tar -xzvf /tmp/jdk-22\_linux-x64\_bin.tar.gz -C /opt && \

    rm /tmp/jdk-22\_linux-x64\_bin.tar.gz

# Set environment variables

ENV PYSPARK\_PYTHON=python3

ENV PYSPARK\_DRIVER\_PYTHON=python3

ENV JAVA\_HOME=/opt/jdk-22

ENV PATH=$JAVA\_HOME/bin:$SPARK\_HOME/bin:$PATH

# Set working directory

WORKDIR /

# Command to run when the container starts

CMD ["/bin/bash"]

* Bash shell commands to install MS SQL ODBC 18:

curl https://packages.microsoft.com/keys/microsoft.asc | tee /etc/apt/trusted.gpg.d/microsoft.asc

curl https://packages.microsoft.com/keys/microsoft.asc | tee /etc/apt/trusted.gpg.d/microsoft.asc

curl https://packages.microsoft.com/config/ubuntu/22.04/prod.list | tee /etc/apt/sources.list.d/mssql-release.list

apt-get update

apt-get install mssql-tools18 unixodbc-dev

apt-get install -y unixodbc unixodbc-dev

apt --fix-broken install

ACCEPT\_EULA=Y apt-get install -y msodbcsql18

* “entrypoint.sh”:

#!/bin/bash

# Set up the timezone for the Docker container

ln -snf /usr/share/zoneinfo/Asia/Ho\_Chi\_Minh /etc/localtime

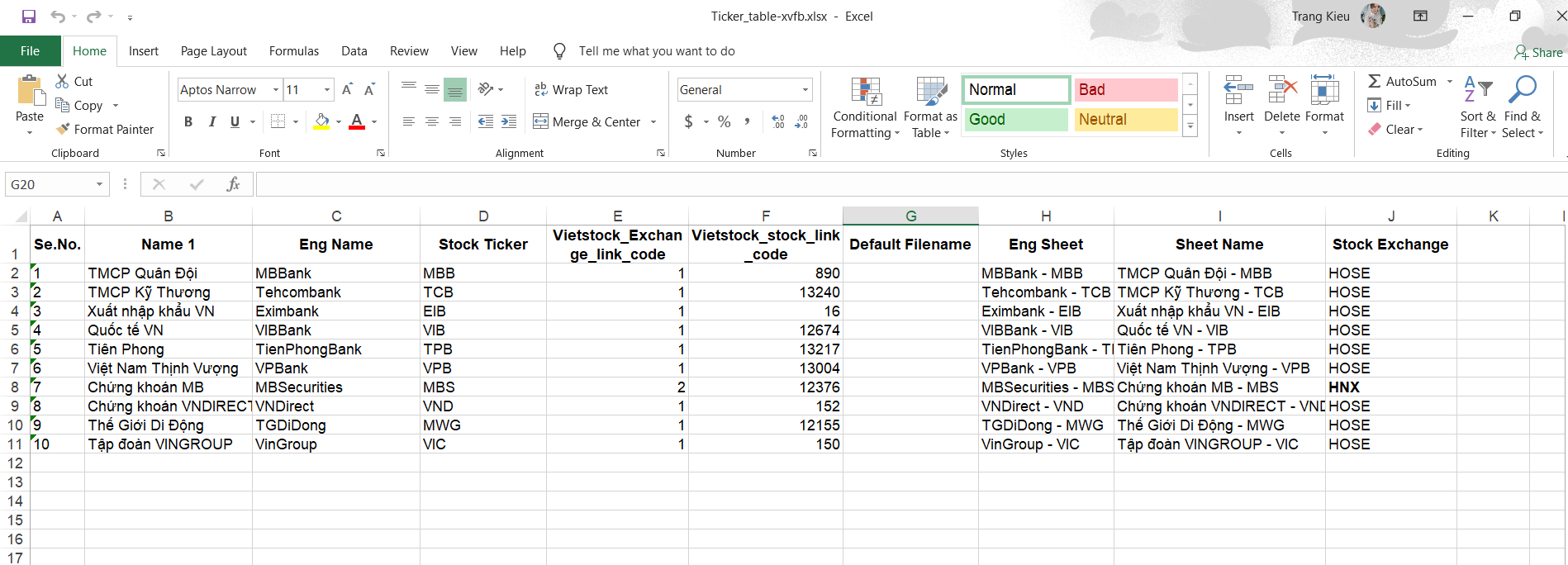
echo "Asia/Ho\_Chi\_Minh" > /etc/timezone

# Create the new database and tables

/opt/mssql-tools18/bin/sqlcmd -S 192.168.10.100 -U trangkieu4 -P 123456 -N -C -i /createstockdt.sql

# B – BUILDING CONTAINER APPLICATIONS

## 1. Crawling raw data and inserting crawled data into MongoDB

I create an Excel file named “Ticker\_table-xvfb.xlsx” which contains basic info of 10 stocks in my portfolio including variables used in my source codes :  


### 1.1. Numerous data:

For each stock code, the URL link has a rule for its string structure. Variables in the string are from “Vietstock\_exchange\_link\_code” and “Vietstock\_stock\_link\_code” columns of the Excel file. I create a “for” loop to iterate through all stock lists and scrape data in each loop. Additionally, I use Selenium for simulating the actions of human when suffering the internet in order to avoid crawling prevention from the website administrator. It does not only help me input the log-in information by clicking on “Log in” button but can also help load more dynamically loaded data by clicking on the Next button of each page and wait for some seconds until the element is visible. Thus, each page is displayed with full html sources and all the XPath of elements are available for scraping. I create another “for” loop to ietrate through all pages of the table to scrape data. This is the content of the source code file – “dbstockcrawl.py”:

print("Let's start crawling!")

import scrapy

import selenium

import pandas as pd

import time

from selenium import webdriver

from selenium.webdriver.edge.service import Service

from selenium.webdriver.edge.options import Options

from selenium.webdriver.common.desired\_capabilities import DesiredCapabilities

from selenium.webdriver.edge.service import Service

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from scrapy.selector import Selector

import pymongo

from pymongo import MongoClient

# Delete import-to-mongo def

#class Spider9(scrapy.Spider):

name = "spider9"

url1 = "https://finance.vietstock.vn/ket-qua-giao-dich?tab=thong-ke-gia"

import pandas as pd

#tickertable = pd.read\_excel("D:\\1\_Docker-compose\\scrapy-con - 100 - stocks\\Ticker\_table-100.xlsx")

tickertable = pd.read\_excel("/Ticker\_table-xvfb.xlsx")

#tickertable = pd.read\_excel("C:\\stockenv\\env\\Scripts\\crawl-con\\python-3-12-slim-continue\\Ticker\_table.xlsx")

stocklist = tickertable['Stock Ticker'].tolist()

stocklist1 = [str(stock) for stock in stocklist]

linkcode = tickertable['Vietstock\_stock\_link\_code'].tolist()

linkcode1 = [str(link) for link in linkcode]

exchangelist = tickertable['Vietstock\_Exchange\_link\_code'].tolist()

exchangelist1 = [str(exchange) for exchange in exchangelist]

data = []

for i in range (len(stocklist)):

    url = url1 + "&exchange=" + exchangelist1[i] + "&code=" + linkcode1[i]

    # Selenium webdriver setup

    service = Service("/usr/local/bin/msedgedriver")

    options = webdriver.EdgeOptions()

    options.add\_argument("--window-size=1920,1080")

    options.add\_argument("headless")  # Optional: Run in headless mode

    options.add\_argument("--no-sandbox")

    options.add\_argument("--disable-dev-shm-usage")

    options.add\_argument("--disable-gpu")

    # avoid timeout error:

    #prefs = {

    #"profile.managed\_default\_content\_settings.images": 2,

    #"profile.managed\_default\_content\_settings.stylesheets": 2,

    #"profile.managed\_default\_content\_settings.javascript": 2

    #}

    #options.add\_experimental\_option("prefs", prefs)

    driver = webdriver.Edge(service=service, options=options)

    #driver.set\_page\_load\_timeout(600) # avoid timeout crash

    driver.get(url)

    try:

        driver.maximize\_window()

    except:

        print("can not maximize the window")

    # Click on Excel download button:

    # Find the button by its class name and click it

    try:

        button\_container = WebDriverWait(driver, 4).until(

            EC.visibility\_of\_element\_located((By.XPATH, "//div[@class='col-md-6']"))

        )

        # Now find the ExCEL Download button within the div:

        # Find the <a> element

        a\_element = WebDriverWait(driver, 4).until(

            EC.visibility\_of\_element\_located((By.XPATH, "//a[@title='Export Excel']"))

        )

    except Exception as e:

        print("Can not click the button: ", e)

    try:

        a\_element.click()

    except Exception:

        print("Can not click Xem button")

    # Log in:

    # Find the container:

    login\_container = driver.find\_element(By.XPATH,"//a[@class='title-link btnlogin-link']")

    # Click on the log in container to activate it

    #login\_container.click()

    # Log in:

    # Find the email input field by its ID

    email\_input = driver.find\_element(By.ID,"txtEmailLogin")

    # Input your email into the field

    email\_input.send\_keys("kieuhuyentrang@gmail.com")

    # Find the password input field by its ID

    password = driver.find\_element(By.ID,"txtPassword")

    # Input your password into the field

    password.send\_keys("767122119")

    # Optionally, you can submit the form after inputting the email

    login\_button = driver.find\_element(By.ID, "btnLoginAccount")

    login\_button.click()

    # Wait for the page to stabilize after login

    WebDriverWait(driver, 10).until(EC.staleness\_of(button\_container))

    # Input values:

    # Wait for the input field container to be visible

    input\_container1 = WebDriverWait(driver, 4).until(

        EC.visibility\_of\_element\_located((By.XPATH, "//\*[@id='txtFromDate']"))

    )

    # Get the current date

    end\_date = pd.Timestamp.today()

    # Calculate the start date as 2 years (365\*2 days) before the end date

    start\_date = end\_date - pd.Timedelta(days=365\*2)

    # Format the dates as text in the format dd/mm/yyyy

    end\_date\_text = end\_date.strftime('%d/%m/%Y')

    start\_date\_text = start\_date.strftime('%d/%m/%Y')

    # Click on the input container to activate it

    input\_container1.click()

    # Now, locate and input the value into the field

    input\_field1 = WebDriverWait(driver, 4).until(

        EC.visibility\_of\_element\_located((By.XPATH, "//div[@id='txtFromDate']/input"))

    )

    input\_field1.clear()  # Clear existing value if any

    input\_field1.send\_keys(start\_date\_text) #  "01/04/2023"

    # Similar to ToDate:

    # Wait for the input field container to be visible

    input\_container2 = WebDriverWait(driver, 4).until(

        EC.visibility\_of\_element\_located((By.ID, "txtToDate"))

    )

    # Click on the input container to activate it

    input\_container2.click()

    # Now, locate and input the value into the field

    input\_field2 = WebDriverWait(driver, 4).until(

        EC.visibility\_of\_element\_located((By.XPATH, "//div[@id='txtToDate']/input"))

    )

    input\_field2.clear()  # Clear existing value if any

    input\_field2.send\_keys(end\_date\_text) # "23/03/2024"

    # Find the button Contains XEM button

    button\_container = WebDriverWait(driver, 4).until(

        EC.visibility\_of\_element\_located((By.XPATH, "//div[@class='col-md-6']"))

    )

    # Wait for the page to stabilize after login

    # WebDriverWait(driver, 10).until(EC.staleness\_of(button\_container))

    Xembt = WebDriverWait(driver, 10).until(

        EC.element\_to\_be\_clickable((By.XPATH, "//button[text()='Xem']"))

    )

    Xembt = WebDriverWait(driver, 4).until(EC.element\_to\_be\_clickable((By.XPATH, "//\*[@id='trading-result']/div/div[1]/div[1]/div/div[2]/button")))

    #Click the button

    Xembt.click()

    # Select 20 displayed lines for each page

    select = WebDriverWait(driver, 4).until(

        EC.visibility\_of\_element\_located((By.XPATH, '//\*[@id="trading-result"]/div/div[3]/div/div/div[1]/div/select'))

    )

    select.find\_element(By.XPATH, "//option[text()='20']").click()

    time.sleep(1)

    # thêm tên cột:

    columns = ['STT', 'Date', 'Stock', 'Basic', 'Open', 'Close', 'High', 'Low', 'Average', 'PriceChange', 'PriceChangeRate', 'OrdMatVol', 'OrdMatVal', 'PutThrVol', 'PutThrVal', 'TotalTranVol', 'TotalTranVal', 'MarCap']

    # Scraping each page:

    for p in range(1, 14): # for each page after "Next" clicking

        #tbody = response.meta['tbody']

        # Locate the parent <div> element

        #parent\_div = driver.find\_element(By.XPATH, "//div[@class='parent-div-class']") # wrong xpath

        parent\_div = driver.find\_element(By.XPATH, '//\*[@id="trading-result"]/div/div[3]') # try another XPath level

        # Use the parent <div> to locate all <tr> elements representing rows in the updated content

        new\_rows = parent\_div.find\_elements(By.XPATH, './/tr')

        time.sleep(3) #

        client = pymongo.MongoClient('mongodb://mongo1:27017/') # thay mongodb = mongo1

        db = client['dbstock']

        collection = db['tbl10stocks']

        # Process the new rows as needed

        for row in new\_rows:

            #item = Spider007Item()

            row\_selector = Selector(text=row.get\_attribute("outerHTML"))

            cols = row\_selector.xpath('.//td')

            row\_data = []

            for col in cols:

                row\_data.append(col.xpath('.//text()').get().strip())

                # Ensure that row\_data has the same length as columns

            if len(row\_data) == len(columns):

                document = {columns[i]: row\_data[i] for i in range(len(columns))}

                ## IMPORT VÀO MONGODB:

                try:

                    collection.insert\_one(document)

                    print("Row index {row} of the stock {stocklist1[i]} inserted successfully into MongoDB")

                except Exception as e:

                    print(f"An error occurred: {e}")

            else:

                print(f"Data length mismatch: Expected {len(columns)}, got {len(row\_data)}")

                print(f"Row data: {row\_data}")

        time.sleep(3)

        #df = pd.DataFrame(data)

        #time.sleep(5)

        try:

            # Wait for the button to be clickable, handling potential dynamic loading

            nextbutton = WebDriverWait(driver, 6).until(

            EC.element\_to\_be\_clickable((By.XPATH, "//\*[@id='btn-page-next']/i"))

            )

            nextbutton.click()

            time.sleep(3) #

            continue

        except:

            break

    #print("One stock has been scraped and inserted into MongoDB successfully!")

    print(f"The stock {stocklist1[i]} with index {i} has been scraped and inserted into MongoDB")

driver.quit()

print("CRAWLING PROSCESS HAS JUST FINISHED!")

For each row scraped from the website, I immediately insert it into MongoDB. If it fails, pass it and continue with the next one with try – except block.

### 1.2. Text data:

Different from numerous data crawling, crawling content is much simpler. The only different part of the main URL link of each stock is indeed the stock code itself. I can get all the URL links of latest articles displayed in the main URL link through “find\_elements” method of Scrapy. Since then, I can extract all needed text. This is the content of the source code file – “contentpubver3.py”:

print("Content Crawling Has Just Begun ...")

import scrapy

import selenium

import pandas as pd

import time

from selenium import webdriver

from selenium.webdriver.edge.service import Service

from selenium.webdriver.edge.options import Options

from selenium.webdriver.common.desired\_capabilities import DesiredCapabilities

from selenium.webdriver.edge.service import Service

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

from scrapy.selector import Selector

import numpy as np

import pymongo

from selenium.common.exceptions import NoSuchElementException, TimeoutException

import random

## INITIALIZE STOCK CODES, LISTS TO SAVE COLUMN VALUES

#tickertable = pd.read\_excel("D:\\1\_Docker-compose\\scrapy-con - 100 - stocks\\Ticker\_table-100.xlsx")

tickertable = pd.read\_excel("/Ticker\_table-xvfb.xlsx")

#tickertable = pd.read\_excel("C:\\stockenv\\env\\Scripts\\crawl-con\\python-3-12-slim-continue\\Ticker\_table.xlsx")

stocklist = tickertable['Stock Ticker'].tolist()

stocklist1 = [str(stock) for stock in stocklist]

linkcode = tickertable['Vietstock\_stock\_link\_code'].tolist()

linkcode1 = [str(link) for link in linkcode]

exchangelist = tickertable['Vietstock\_Exchange\_link\_code'].tolist()

exchangelist1 = [str(exchange) for exchange in exchangelist]

stock = []

link = []

content = []

## IN THE LOOP OF EACH STOCK CODE:

links = []

contents = []

codes = []

import pymongo

client = pymongo.MongoClient('mongodb://mongo1:27017/')

db = client['dbstock']

collection = db['tbcontent']

for i in range (len(stocklist1)):

    # Open news link in headless mode

    url = "https://finance.vietstock.vn/"+stocklist1[i]+"/tin-tuc-su-kien.htm"

    service = Service("/usr/local/bin/msedgedriver")

    options = webdriver.EdgeOptions()

    options.add\_argument("--headless")

    options.add\_argument("--window-size=1920,1080")

    options.add\_argument("--no-sandbox")

    options.add\_argument("--disable-dev-shm-usage")

    options.add\_argument("--disable-gpu")

    driver = webdriver.Edge(service=service, options=options)

    driver.get(url)

    # Log in:

    # Wait for the login button and click it if necessary

    WebDriverWait(driver, 10).until(EC.element\_to\_be\_clickable((By.XPATH, "//a[@class='title-link btnlogin-link']"))).click()

    login\_container = driver.find\_element(By.XPATH,"//a[@class='title-link btnlogin-link']")

    time.sleep(3)

    email\_input = driver.find\_element(By.ID,"txtEmailLogin")

    email\_input.send\_keys("kieuhuyentrang@gmail.com")

    password = driver.find\_element(By.ID,"txtPassword")

    password.send\_keys("767122119")

    login\_button = driver.find\_element(By.ID, "btnLoginAccount")

    login\_button.click()

    WebDriverWait(driver, 10).until(EC.staleness\_of(login\_button))

    # Get all article links:

    ## Find the element with the id 'newsevent-content'

    newsevent\_content = driver.find\_element(By.ID, 'newsevent-content') # The top level contains the href and is common to all stock codes.

    # Find all <a> elements within 'newsevent-content'

    a\_tags = newsevent\_content.find\_elements(By.TAG\_NAME,'a') # the tag that contains all hrefs

    hrefs = []

    for a\_tag in a\_tags:

        href = a\_tag.get\_attribute('href') # Each href is a link to an article.

        hrefs.append(href)

    print(hrefs)

    # Go to each link to scrape all text::

    #links = []

    #contents = []

    for href in hrefs: # for each article link

        if href != None:

            links.append(href) # them urlcontent vao links

            codes.append(stocklist1[i])

            driver.get(href)

            try:

                driver.maximize\_window()

            except:

                print("can not maximize the window")

            wait = WebDriverWait(driver, 3)

            # Generate a random number of lines to scroll down (between 5 and 30) - mimik human actions

            lines\_to\_scroll = random.randint(5, 30)

            # Assuming each line is approximately 20 pixels in height

            pixels\_to\_scroll = lines\_to\_scroll \* 20

            # Scroll down by the calculated number of pixels

            driver.execute\_script(f"window.scrollBy(0, {pixels\_to\_scroll});")

            # Pause to see the effect (optional)

            sec = random.uniform(1,3)

            time.sleep(sec)

            # Find all elements with the specified class name

            elements = driver.find\_elements(By.CSS\_SELECTOR, 'div.col-lg-10.col-sm-12.col-md-12')

            # Initialize date\_text to ensure it's defined

            date\_text = None

            try:

                # Wait until the elements are visible

                element = WebDriverWait(driver, 10).until(

                    EC.presence\_of\_element\_located((By.XPATH, "/html/body/div[14]/div[5]/div[1]/div/section/div[1]/div/div[1]/div/div/div[1]/div[2]/div[2]/div/div[1]/div/span"))

                )

                # Get the text of the elements:

                date\_text = element.text

            except (NoSuchElementException, TimeoutException):

                # Handle the exception and print "Error"

                print("Error: Can't find PublishDate")

                date\_text = "Unknown Date"

            title\_text = None

            try:

                title\_element = WebDriverWait(driver, 10).until(

                    EC.presence\_of\_element\_located((By.XPATH,"//h1[contains(@class,'title')]")))

                # Get the text of the elements:

                title\_text = title\_element.text

            except (NoSuchElementException, TimeoutException):

                # Handle the exception and print "Error"

                print("Error: Can't find Title")

                title\_text = "Unknown Title"

            text\_contents = []

            for element in elements:

                text\_contents.append(element.text)

            ## IMSERT DATA INTO MONGODB:

            # Create a dictionary representing the MongoDB document

            document = {

                "Stock": stocklist1[i],  # Renamed variable for valid field name

                "Link": href,

                "Contents": text\_contents,

                "PublishDate" : date\_text,

                "Title" : title\_text

            }

            try:

                result = collection.insert\_one(document)

                print("Data inserted successfully")

                print(document)

            except Exception as e:

                print(f"An error occurred: {e}")

        second = random.uniform(1, 4)

        time.sleep(second)

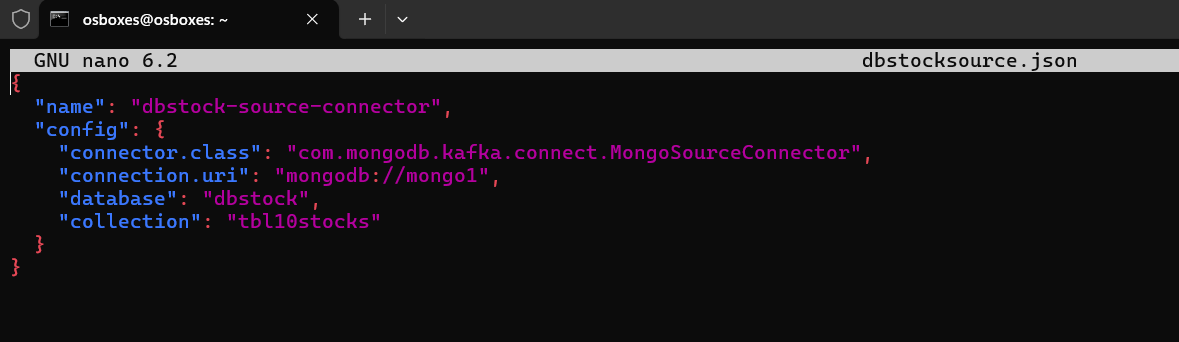
driver.quit()

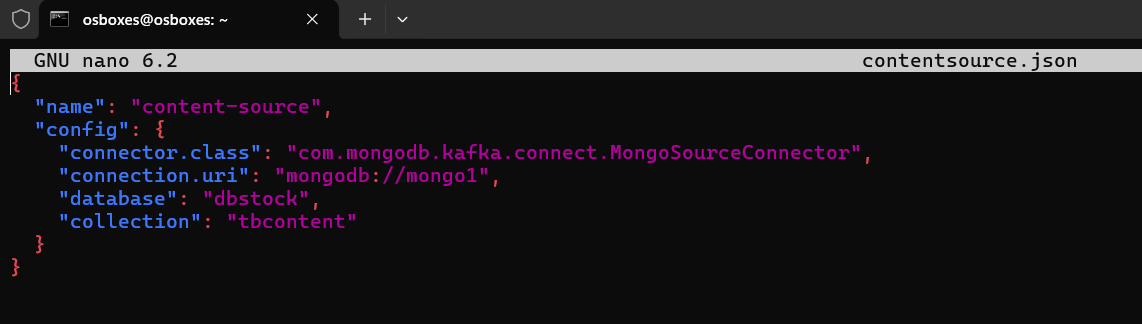
print("\nContent Crawling has just finished!")

## 2. Creating and registering Kafka topics from MongoDB

Before creating a Kafka topic, be sure that all Kafka services are running in healthy status. Then, follow these steps:

* Firstly, a topic based on the name of database and its collection are created inside MongoDB container. The json file which contains information about this topic is called a “source connector”. Below are the files named “dbstocksource.json” and “contentsource.json” to create “dbstock.tbl10stocks” and “dbstock.tbcontent” topics respectively:



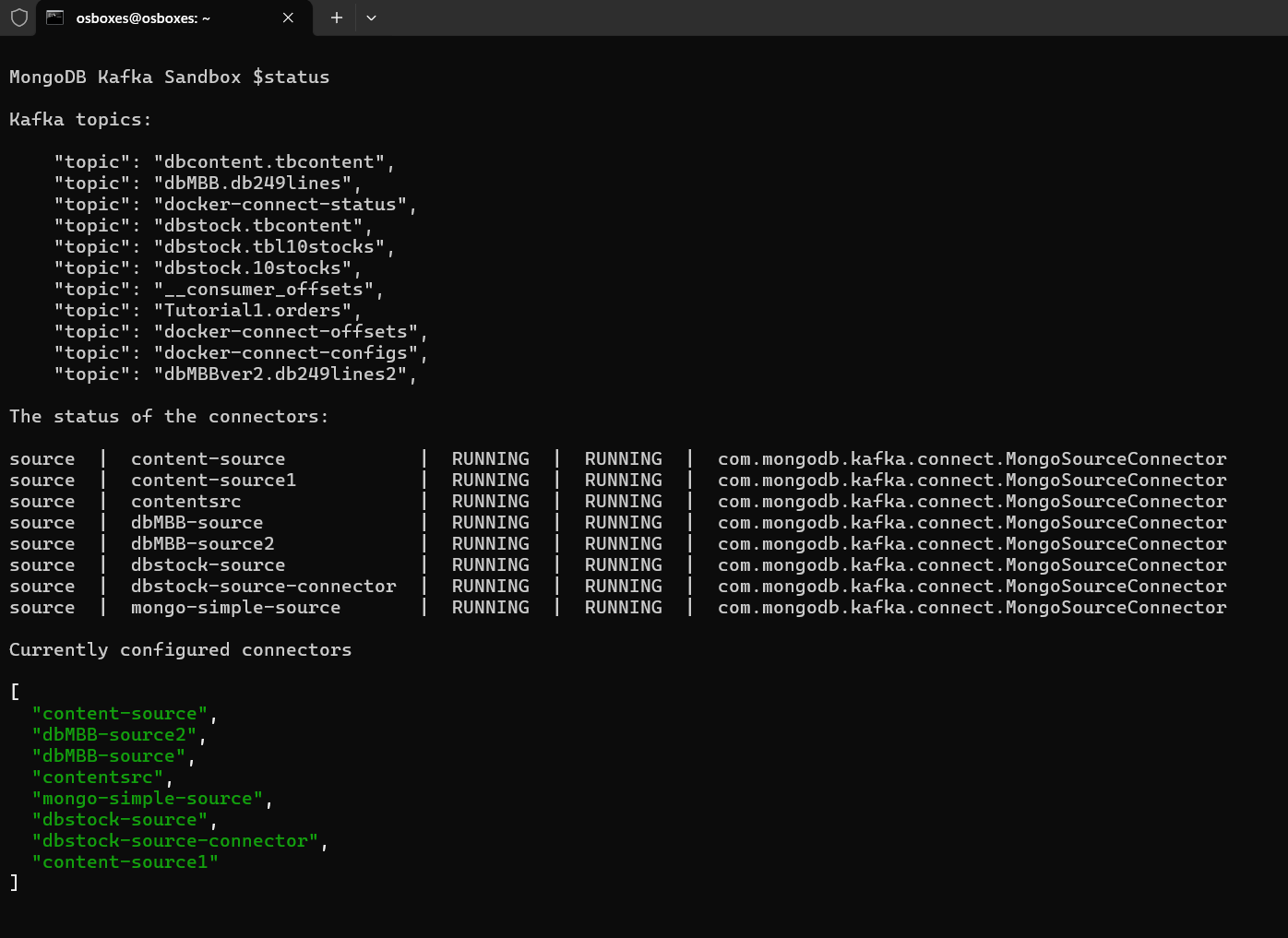


* Then, these connectors are registered to Kafka using POST method using bash shell commands:

curl -X POST -H "Content-Type: application/json" -d @ dbstocksource.json http://connect:8083/connectors -w "\n"

curl -X POST -H "Content-Type: application/json" -d @ contentsource.json http://connect:8083/connectors -w "\n"

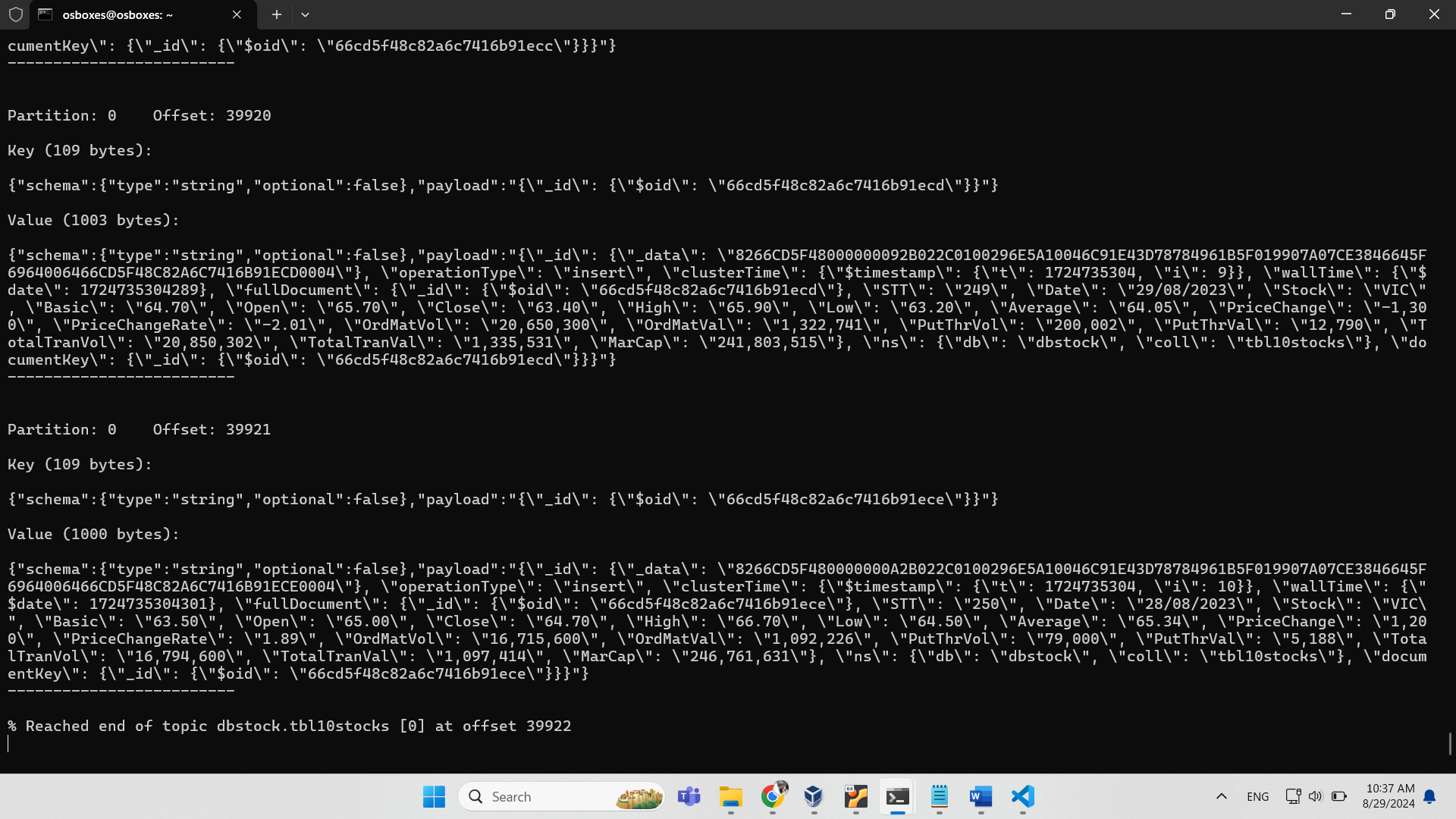
* Once it’s done, check the status of topics:

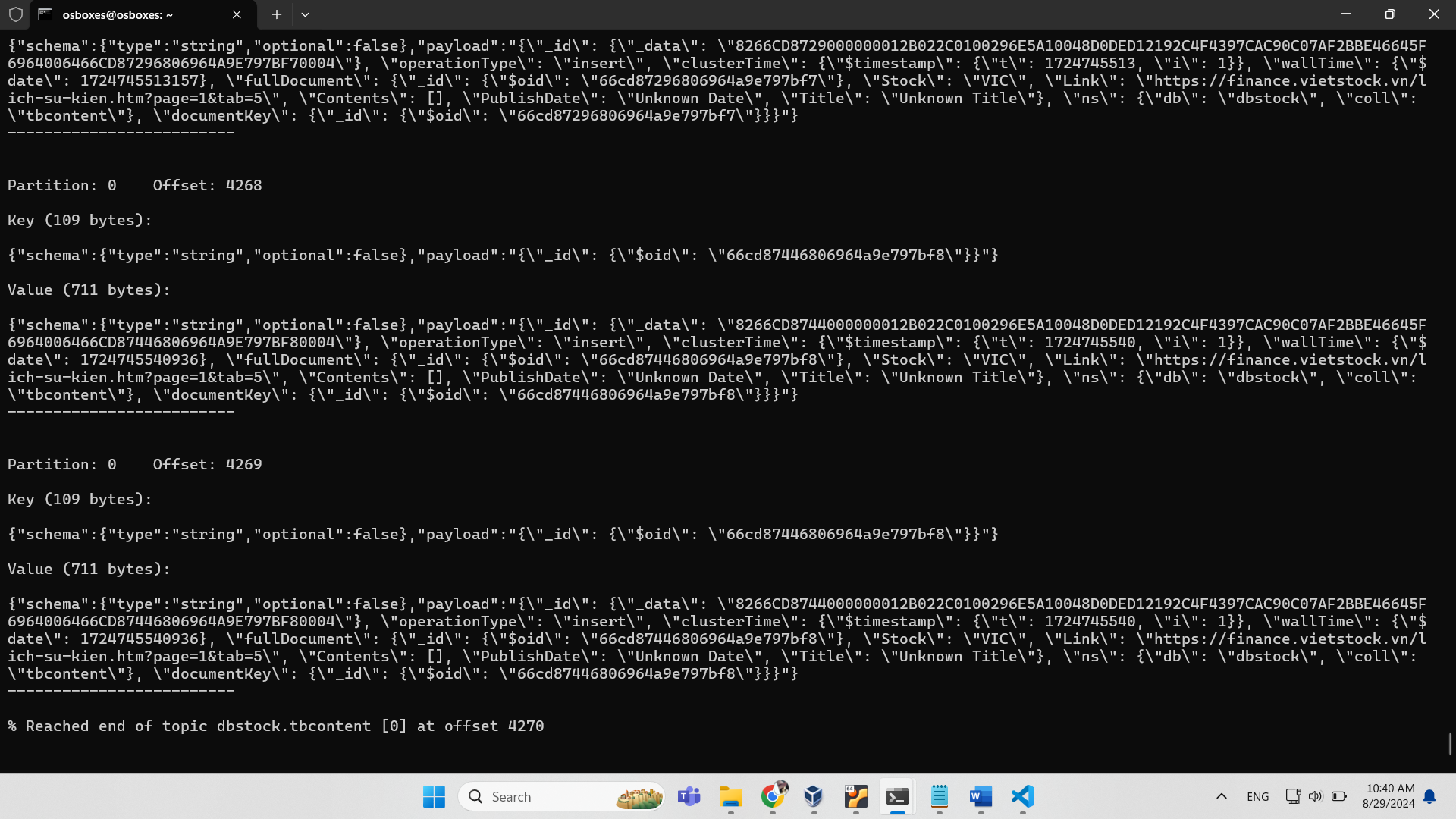


As we can see from the output, the two topics are active.

* Display data from the topic into the screen to see streamed data inserted into MongoDB by Scrapy application in real time. Also, a full structure schema of each item/document from MongoDB is shown at the same moment.





Such things are similar to “dbstock.tbcontent” topic:

Note that “status” and “kc” commands are user-defined functions provided by MongoDB. For more details, please view the full source codes attached.

## 3. Retrieving data from Kafka topics followed by transforming and cleaning data using PySpark

* The steps involved in this task are:
* Creating a spark session.
* Connecting Kafka server and reading a specific topic.
* Assign a correct schema structure of data from Kafka which originally created by MongoDB.
* Deserializing json structure of data and selecting needed columns.
* Transforming data in columns into correct data type (string to date, integer, float types) using pyspark.sql.functions.
* Writing to Postgres in writeStream mode.
* "dbstockread.py" source codes:

from pyspark.sql import SparkSession

from pyspark.sql.functions import from\_json, col, regexp\_replace, date\_format, to\_date, when

from pyspark.sql.types import StructType, StructField, StringType, LongType, IntegerType, FloatType

# Create Spark session

spark = SparkSession.builder \

    .master("local") \

    .appName("KafkaSparkExample") \

    .config("spark.sql.streaming.kafka.consumer.cache.enabled", "false") \

    .config("spark.sql.streaming.kafka.consumer.pollTimeoutMs", "512") \

    .getOrCreate()

# Set the logging level for Kafka to ERROR to suppress warnings

spark.sparkContext.setLogLevel("ERROR")

# Define the Kafka bootstrap servers and topic

kafka\_bootstrap\_servers = "broker:29092"

kafka\_topic = "dbstock.tbl10stocks"

# Define schema for the JSON content inside the 'payload' field

# 1

value\_schema = StructType([

                    StructField("schema", StructType([

                        StructField("type", StringType(), True),

                        StructField("optional", StringType(), False)

                    ]),True),

                    StructField("payload", StringType(), True)

                    ])

df = spark.readStream \

    .format("kafka") \

    .option("kafka.bootstrap.servers", kafka\_bootstrap\_servers) \

    .option("subscribe", kafka\_topic) \

    .load()

# 2

deserialized\_df = df.select(

    from\_json(col("value").cast("string"), value\_schema).alias("parsed\_value")

).select(

    from\_json(col("parsed\_value.payload").cast("string"),

              StructType([

                  StructField("\_id", StructType([

                      StructField("\_data", StringType(), True)

                  ]), True),

                  StructField("operationType", StringType(), True),

                  StructField("clusterTime", StructType([

                      StructField("$timestamp", StructType([

                          StructField("t", StringType(), True),

                          StructField("i", StringType(), True)

                      ]), True)

                  ]), True),

                  StructField("wallTime", StructType([

                      StructField("$date", StringType(),True)

                  ]), True),

                  StructField("fullDocument", StructType([

                      StructField("\_id", StructType([

                          StructField("$oid", StringType(), True)

                      ]), True),

                      StructField("STT", StringType(), True),

                      StructField("Date", StringType(), True),

                      StructField("Stock", StringType(), True),

                      StructField("Basic", StringType(), True),

                      StructField("Open", StringType(), True),

                      StructField("Close", StringType(), True),

                      StructField("High", StringType(), True),

                      StructField("Low", StringType(), True),

                      StructField("Average", StringType(), True),

                      StructField("PriceChange", StringType(), True),

                      StructField("PriceChangeRate", StringType(), True),

                      StructField("OrdMatVol", StringType(), True),

                      StructField("OrdMatVal", StringType(), True),

                      StructField("PutThrVol", StringType(), True),

                      StructField("PutThrVal", StringType(), True),

                      StructField("TotalTranVol", StringType(), True),

                      StructField("TotalTranVal", StringType(), True),

                      StructField("MarCap", StringType(), True)

                  ]), True),

                  StructField("ns", StructType([

                      StructField("db", StringType(), True),

                      StructField("coll", StringType(), True)

                  ]), True),

                  StructField("documentKey", StructType([

                      StructField("\_id", StructType([

                          StructField("$oid", StringType(), True)

                      ]), True)

                  ]), True)

              ])

    ).alias("payload\_json")

)

full\_document\_df = deserialized\_df.select(

    col("payload\_json.fullDocument.\*")

)

# 3

# Remove commas as decimal separators from specific fields

cleaned\_df = full\_document\_df.select(

    col("STT"),

    col("Date"),

    col("Stock"),

    regexp\_replace(col("Basic"), ",", "").alias("Basic"),

    regexp\_replace(col("Open"), ",", "").alias("Open"),

    regexp\_replace(col("Close"), ",", "").alias("Close"),

    regexp\_replace(col("High"), ",", "").alias("High"),

    regexp\_replace(col("Low"), ",", "").alias("Low"),

    regexp\_replace(col("Average"), ",", "").alias("Average"),

    regexp\_replace(col("PriceChange"), ",", "").alias("PriceChange"),

    regexp\_replace(col("PriceChangeRate"), ",", "").alias("PriceChangeRate"),

    regexp\_replace(col("OrdMatVol"), ",", "").alias("OrdMatVol"),

    regexp\_replace(col("OrdMatVal"), ",", "").alias("OrdMatVal"),

    regexp\_replace(col("PutThrVol"), ",", "").alias("PutThrVol"),

    regexp\_replace(col("PutThrVal"), ",", "").alias("PutThrVal"),

    regexp\_replace(col("TotalTranVol"), ",", "").alias("TotalTranVol"),

    regexp\_replace(col("TotalTranVal"), ",", "").alias("TotalTranVal"),

    regexp\_replace(col("MarCap"), ",", "").alias("MarCap")

)

# Convert dd/MM/yyyy to MM/dd/yyyy format

df\_transformed = cleaned\_df.withColumn(

    "DateChange",

    date\_format(to\_date(col("Date"), "dd/MM/yyyy"), "MM/dd/yyyy")

)

# Convert specific columns from string to integer

integer\_columns = ['OrdMatVol', 'OrdMatVal', 'PutThrVol', 'PutThrVal', 'TotalTranVol', 'TotalTranVal', 'MarCap']

for col\_name in integer\_columns:

    df\_transformed = df\_transformed.withColumn(col\_name, col(col\_name).cast("integer"))

# Convert specific columns from string to float

float\_columns = ['PriceChange', 'PriceChangeRate', 'Basic', 'Open', 'Close', 'High', 'Low', 'Average']

for col\_name in float\_columns:

    df\_transformed = df\_transformed.withColumn(col\_name, when(col(col\_name) != "-", col(col\_name).cast("float"))

                                  .otherwise(None))

# Define PostgreSQL JDBC URL and properties

postgres\_url = "jdbc:postgresql://postgres:5432/dbstock"

postgres\_properties = {

    "user": "admin",

    "password": "admin",

    "driver": "org.postgresql.Driver"

}

# Write DataFrame to PostgreSQL

def write\_to\_postgres(df, epoch\_id):

    df.write.jdbc(

        url=postgres\_url,

        table="tbl10stocks",

        mode="append",

        properties=postgres\_properties

    )

# Write DataFrame to PostgreSQL in streaming mode

query1 = df\_transformed.writeStream \

    .foreachBatch(write\_to\_postgres) \

    .outputMode("append") \

    .start()

# Write DataFrame to console for debugging

query2 = df\_transformed.writeStream \

    .outputMode("append") \

    .format("console") \

    .option("truncate", "false") \

    .start()

# Keep the streaming query running

query1.awaitTermination()

query2.awaitTermination()

spark.stop()

* "dbcontent-15.py" source codes:

from pyspark.sql import SparkSession, Window

from pyspark.sql.functions import udf, from\_json, col, regexp\_replace, date\_format, to\_date, when, expr, current\_timestamp, to\_timestamp

from pyspark.sql.functions import explode, split, col, sum, when, row\_number, desc, max, window

from pyspark.sql.types import StructType, StructField, StringType, TimestampType

import pyspark.sql.functions as F

import re

from datetime import datetime, timedelta

# Create Spark session

spark = SparkSession.builder \

    .master("local") \

    .appName("KafkaSparkExample") \

    .config("spark.sql.streaming.kafka.consumer.cache.enabled", "false") \

    .config("spark.sql.streaming.kafka.consumer.pollTimeoutMs", "512") \

    .getOrCreate()

# Set the logging level for Kafka to ERROR to suppress warnings

spark.sparkContext.setLogLevel("ERROR")

# Define the Kafka bootstrap servers and topic

kafka\_bootstrap\_servers = "broker:29092"

kafka\_topic = "dbstock.tbcontent"

# Define schema for the JSON content inside the 'payload' field

# 1

value\_schema = StructType([

                    StructField("schema", StructType([

                        StructField("type", StringType(), True),

                        StructField("optional", StringType(), False)

                    ]),True),

                    StructField("payload", StringType(), True)

                    ])

df = spark.readStream \

    .format("kafka") \

    .option("kafka.bootstrap.servers", kafka\_bootstrap\_servers) \

    .option("subscribe", kafka\_topic) \

    .load()

# 2

deserialized\_df = df.select(

    from\_json(col("value").cast("string"), value\_schema).alias("parsed\_value")

).select(

    from\_json(col("parsed\_value.payload").cast("string"),

              StructType([

                  StructField("\_id", StructType([

                      StructField("\_data", StringType(), True)

                  ]), True),

                  StructField("operationType", StringType(), True),

                  StructField("clusterTime", StructType([

                      StructField("$timestamp", StructType([

                          StructField("t", StringType(), True),

                          StructField("i", StringType(), True)

                      ]), True)

                  ]), True),

                  StructField("wallTime", StructType([

                      StructField("$date", StringType(),True)

                  ]), True),

                  StructField("fullDocument", StructType([

                      StructField("\_id", StructType([

                          StructField("$oid", StringType(), True)

                      ]), True),

                      StructField("Stock", StringType(), True),

                      StructField("Link", StringType(), True),

                      StructField("Contents", StringType(), True),

                      StructField("PublishDate", StringType(), True),

                      StructField("Title", StringType(), True)

                  ]), True)

              ])

    ).alias("payload\_json")

)

df3 = deserialized\_df.select(

    col("payload\_json.fullDocument.\*"),

    col("payload\_json.clusterTime.$timestamp.t").cast(TimestampType()).alias("event\_time")

)

# Select columns: Stock, Link, Contents, Title, NewsDate:

df5= df3.select(col("Stock"),col("Link"),col("Contents"),col("PublishDate"),col("Title"))

##Cleaning:

### Filter out where "PublishDate" = "Unknown Date":

# Filter out rows where PublishDate is 'Unknown Date'

df4 = df5.filter(col("PublishDate") != "Unknown Date")

# Handling the strings "n hours ago", "n minutes ago" in the "PublishDate" column:

from datetime import datetime, timedelta

import re

def parse\_relative\_time(relative\_time\_str):

    # Define the date format

    date\_format = '%m/%d/%Y'

    if relative\_time\_str == 'Unknown Date':

        return relative\_time\_str

    # Attempt to extract hours from the string

    match = re.match(r'(\d+) giờ trước', relative\_time\_str)

    if match:

        # Extract number of hours

        hours\_ago = int(match.group(1))

        # Calculate the date by subtracting the hours from the current time

        past\_time = datetime.now() - timedelta(hours=hours\_ago)

        return past\_time.strftime(date\_format)

    # Attempt to extract minutes from the string

    match\_minutes = re.match(r'(\d+) phút trước', relative\_time\_str)

    if match\_minutes:

        # Extract number of minutes

        minutes\_ago = int(match\_minutes.group(1))

        # Calculate the date by subtracting the minutes from the current time

        past\_time = datetime.now() - timedelta(minutes=minutes\_ago)

        return past\_time.strftime(date\_format)

    try:

        # If the format does not match, use the provided date

        date\_obj = datetime.strptime(relative\_time\_str, '%d/%m/%Y %H:%M')

        return date\_obj.strftime(date\_format)

    except ValueError:

        # Handle any errors in date parsing

        return relative\_time\_str

    else:

        try:

            # If the format does not match, use the provided date

            date\_obj = datetime.strptime(relative\_time\_str, '%d/%m/%Y %H:%M')

            return date\_obj.strftime(date\_format)

        except ValueError:

            # Handle any errors in date parsing

            return relative\_time\_str

    # Return the date in the specified format

    #return past\_time.strftime(date\_format)

# Register the function as a UDF

parse\_relative\_time\_udf = udf(parse\_relative\_time, StringType())

# Apply the UDF to transform the relative time

df\_transformed = df4.withColumn("formatted\_date", parse\_relative\_time\_udf(col("PublishDate")))

# Convert the formatted date strings to date type

df5 = df\_transformed.withColumn("DateChange", to\_date(col("formatted\_date"), "MM/dd/yyyy"))

df6 = df5.select(col("Stock"),col("Link"),col("Contents"),col("PublishDate"),col("Datechange"),col("formatted\_date"),col("Title"))

# Define PostgreSQL JDBC URL and properties

postgres\_url = "jdbc:postgresql://postgres:5432/dbstock"

postgres\_properties = {

    "user": "admin",

    "password": "admin",

    "driver": "org.postgresql.Driver"

}

# Write DataFrame to PostgreSQL

def write\_to\_postgres(df, epoch\_id):

    df.write.jdbc(

        url=postgres\_url,

        table="tbcontent",

        mode="append",

        properties=postgres\_properties

    )

# Write DataFrame to PostgreSQL in streaming mode

query1 = df6.writeStream \

    .foreachBatch(write\_to\_postgres) \

    .outputMode("append") \

    .start()

# Write DataFrame to console for debugging

query2 = df6.writeStream \

    .outputMode("append") \

    .format("console") \

    .option("truncate", "false") \

    .start()

# Keep the streaming query running

query1.awaitTermination()

query2.awaitTermination()

spark.stop()

## 4. Creating database for data warehouse using SQL commands from Docker container in Linux

* Write an SQL script file named “createstockdt.sql” containing SQL commands:

-- Declare and initialize the database name variable

DECLARE @DatabaseName NVARCHAR(128) = 'dbstock4';

-- Create the new database

DECLARE @Sql NVARCHAR(MAX);

SET @Sql = 'CREATE DATABASE [' + @DatabaseName + '];';

EXEC sp\_executesql @Sql;

-- Grant CONTROL permission to the user on the specified database

DECLARE @Sql1 NVARCHAR(MAX);

SET @Sql1 = 'USE [' + @DatabaseName + ']; GRANT CONTROL TO [' + 'trangkieu4' + '];';

EXEC sp\_executesql @Sql;

-- Set the new database to multi-user mode (directly on the database name)

SET @Sql = 'ALTER DATABASE [' + @DatabaseName + '] SET MULTI\_USER;';

EXEC sp\_executesql @Sql;

-- Create tables directly in the new database without using USE

SET @Sql = '

CREATE TABLE [' + @DatabaseName + '].dbo.DataTime

(

DateChange Date,

Year int,

Month int,

Day int,

PRIMARY KEY (DateChange)

);

CREATE TABLE [' + @DatabaseName + '].dbo.DataStock

(

Stock VARCHAR (5),

PRIMARY KEY (Stock)

);

CREATE TABLE [' + @DatabaseName + '].dbo.Fact

(

Stock VARCHAR (5),

DateChange Date,

[Open] float,

[Close] float,

[Low] float,

[High] float,

PutThrVol Integer,

OrdMatVol Integer,

MarCap float,

PRIMARY KEY (Stock, DateChange),

FOREIGN KEY (Stock) REFERENCES [' + @DatabaseName + '].dbo.DataStock ([Stock]),

FOREIGN KEY (DateChange) REFERENCES [' + @DatabaseName + '].dbo.DataTime ([DateChange])

);

CREATE TABLE [' + @DatabaseName + '].dbo.LinkDim

(

Link\_ID VARCHAR (40),

[Link] VARCHAR (300),

Title NVARCHAR (4000),

Body NVARCHAR (MAX),

PRIMARY KEY (Link\_ID)

);

CREATE TABLE [' + @DatabaseName + '].dbo.LinkFact

(

Stock VARCHAR (5),

DateChange Date,

Link\_ID VARCHAR (40),

Link VARCHAR (300),

Title NVARCHAR (4000),

Body NVARCHAR (MAX),

Count\_of\_phat Integer,

Count\_of\_tra Integer,

PRIMARY KEY (Stock, DateChange, Link\_ID),

FOREIGN KEY (Stock) REFERENCES [' + @DatabaseName + '].dbo.DataStock ([Stock]),

FOREIGN KEY (DateChange) REFERENCES [' + @DatabaseName + '].dbo.DataTime ([DateChange]),

FOREIGN KEY (Link\_ID) REFERENCES [' + @DatabaseName + '].dbo.LinkDim ([Link\_ID])

);

';

EXEC sp\_executesql @Sql;

## 5. Removing duplicates, creating data frames and inserting data frames into SQL Server database using PySpark

from pyspark.sql import SparkSession

from pyspark.sql import Row

from pyspark.sql import DataFrameWriter

from pyspark.sql import SparkSession

from pyspark.sql.functions import length, explode, col, when, regexp\_replace, row\_number, lpad, expr, split

from pyspark.sql.functions import current\_timestamp, year, month, dayofmonth, to\_date, format\_number, lit, lpad, concat

from pyspark.sql.window import Window

from pyspark.sql import functions as F

# I - CREATE dataframe for these SSMS tables: DataStock, DataTime, Fact

# Step 1: Initialize SparkSession - # Link: https://www.machinelearningplus.com/pyspark/pyspark-connect-to-postgresql/

jdbc\_driver\_path = "/opt/spark-3.5.1-bin-hadoop3/jars/postgresql-42.7.3.jar"

odbc\_driver\_path = "/app/mssql-jdbc-12.8.0.jre11.jar"

spark = SparkSession.builder \

    .appName("PostgreSQL Connection with PySpark") \

    .config("spark.jars", jdbc\_driver\_path) \

    .config("spark.jars", odbc\_driver\_path) \

    .getOrCreate()

# Step 2: Define your PostgreSQL database connection details

url = "jdbc:postgresql://postgres:5432/dbstock"

properties = {

    "user": "admin",

    "password": "admin",

    "driver": "org.postgresql.Driver"

}

table\_name = "tbl10stocks"

# Step 3: Load the PostgreSQL table into a PySpark DataFrame

df = spark.read.jdbc(url, table\_name, properties=properties)

df.show()

print("number of rows: {}".format(df.count())) # 5,510 rows

df.dtypes

print(len(df.columns))

print(df.count())

# Step 4: TRANSFORMING DATA - REMOVE DUPLICATES:

df\_non\_STT = df.select("Date", "Stock", "Basic", "Open", "Close", 'High', 'Low', 'Average', 'OrdMatVol', 'OrdMatVal', 'PutThrVol', 'PutThrVal', 'TotalTranVol', 'TotalTranVal', 'MarCap', 'DateChange' )

df\_no\_duplicates = df\_non\_STT.dropDuplicates(df\_non\_STT.columns)

df\_no\_duplicates.dtypes

# Filter out rows where "OrdMatVol" is null or 0

df\_fact\_filtered = df\_no\_duplicates.filter(

    (F.col("OrdMatVol").isNotNull()) & (F.col("OrdMatVol") != 0)

)

df\_fact\_filtered.show()

df\_fact\_filtered.count() # 2,500 rows

print(len(df\_no\_duplicates.columns))

print(df\_no\_duplicates.count())  # 1,525 rows

df\_sorted = df\_fact\_filtered.orderBy(["Stock", "DateChange"])

df\_sorted.show()

# Transform "DateChange" data type into "Date" type:

df\_transformed = df\_sorted.withColumn("DateChange", to\_date("DateChange", 'MM/dd/yyyy'))

# Show the result

df\_transformed.dtypes

print("rows of df\_transformed: {}".format(df\_transformed.count()))

print("rows of df\_sorted: {}".format(df\_sorted.count()))

print("rows of df\_fact\_filtered: {}".format(df\_fact\_filtered.count()))

# Step 5: Create tables in SQL Server database as in the schema

# Table 1 - Data\_time

# Extract unique values from "DateChange"

unique\_dates = df\_transformed.select("DateChange").distinct()

unique\_dates.count() # 253 rows

data\_time = unique\_dates.withColumn("Year", year(col("DateChange"))) \

                        .withColumn("Month", month(col("DateChange"))) \

                        .withColumn("Day", dayofmonth(col("DateChange"))) \

# Sort the DataFrame by TimeID in ascending order

data\_time1 = data\_time.orderBy("DateChange")

data\_time1.show()

print(data\_time1.count()) # 253 rows

# Table 2 - Data\_stock:

# Extract unique Stock Codes

unique\_stocks = df\_transformed.select("Stock").distinct().orderBy("Stock")

data\_stock = unique\_stocks.orderBy("Stock")

data\_stock.show()

# Table 3 - Fact table:

df\_fact = df\_transformed.select("Stock", "DateChange", "Open", "Close", "Low", "High", "PutThrVol", "OrdMatVol", "MarCap")

df\_fact.show()

df\_fact.dtypes

print(df\_fact.count())

df\_fact.show()

df\_fact.dtypes

print("number of columns: "+str(len(df\_fact.columns)))

print("number of rows: "+str(df\_fact.count()))

# II - CREATE dataframe for these SSMS tables: LinkDim, LinkFact:

# Step 1: Initialize SparkSession - # Link: https://www.machinelearningplus.com/pyspark/pyspark-connect-to-postgresql/

jdbc\_driver\_path = "/opt/spark-3.5.1-bin-hadoop3/jars/postgresql-42.7.3.jar"

#odbc\_driver\_path = "/app/mssql-jdbc-12.8.0.jre11.jar"

# Step 2: Define your PostgreSQL database connection details

table\_name1 = "tbcontent"

# Step 3: Load the PostgreSQL table into a PySpark DataFrame

df\_read = spark.read.jdbc(url, table\_name1, properties=properties)

df\_read.show()

print("number of rows: {}".format(df\_read.count())) #

df\_read.dtypes

print(len(df\_read.columns))

print(df\_read.count())

# remove duplicates:

dfundup = df\_read.dropDuplicates()

# Step 4: create dataframe to be inserted into SSMS table named "LinkDim" with these columns "Link", "Link\_ID", "Title", "Contents" ("Body")

linkdimdup = dfundup.select(col("Link"),col("Title"),col("Contents").alias("Body"))

linkdim = linkdimdup.dropDuplicates(["Link"])

## Add "Link\_ID" column contains distinct values generated by timestamp:

linkdim1 = linkdim.withColumn("Link\_ID", expr("uuid()")) # maximum length of uuid() : 36 characters

linkdim2 = linkdim1.select(col("Link\_ID"),col("Link"),col("Title"),col("Body"))

# Add a new column that contains the number of character in url column

df\_with\_url\_length = linkdim.withColumn("url\_length", length(col("Link")))

# Show the resulting DataFrame

df\_with\_url\_length.show()

# Step 5a: Create dataframe to be inserted into SSMS table named "LinkFact" with these columns "Stock", "Datechange", "Link\_ID", "Link", "Contents" ("Body"), "Title", "Count of phat", "Count of tra"

## Select "Stock", "Datechange", "Link", "Contents" ("Body"), "Title" from dfundup:

df11 = dfundup.select(col("Stock"),col("DateChange"),col("Link"),col("Title"),col("Contents").alias("Body")) # 406 rows

# Step 5b: Filter out Dates in the scale of "DateChange" from dbstock.tbl10stocks :

## Find oldest and newest dates from data\_time1:

data\_time1.show()

print(data\_time1.count()) # 251 unique rows

oldest\_date = data\_time1.agg(F.min("DateChange")).collect()[0][0]

newest\_date = data\_time1.agg(F.max("DateChange")).collect()[0][0]

print(f"Oldest Date: {oldest\_date}") # Oldest Date: 2023-08-22

print(f"Newest Date: {newest\_date}") # Newest Date: 2024-08-22

# Filter df11 "DateChange" rows that are in the scale of [oldest, newest]:

filtered\_df11 = df11.filter((F.col("DateChange") >= F.lit(oldest\_date)) & (F.col("DateChange") <= F.lit(newest\_date)))

# Show the result

filtered\_df11.show()

print(filtered\_df11.count()) # 358 rows (48 rows are eliminated from df11, which accounts for 11.8%)

print(filtered\_df11.agg(F.min("DateChange")).collect()[0][0]) # datetime.date(2023, 8, 22), matches Oldest Date

print(filtered\_df11.agg(F.max("DateChange")).collect()[0][0]) # datetime.date(2024, 8, 22),  matches Newest Date

# Find mutual DateChange values between two dataframes:

mutual\_dates = filtered\_df11.join(data\_time1, on="DateChange", how="inner").select("DateChange").distinct()

# Show results

mutual\_dates.show()

print(mutual\_dates.count())  # The number of mutual dates: 71 unique values

# Filter df11 so that its "DateChange" includes only values from mutual\_dates alongwith all of other columns of df11:

df11\_filtered = df11.join(mutual\_dates, on="DateChange", how="inner")

## Step 5c: Create "Count of tra", "Count of phat" columns:

df\_split = df11\_filtered.withColumn("Contents\_array", split(col("Body"), " "))

# Explode the array column `Contents\_array`

df\_exploded = df\_split.withColumn("Content\_exploded", explode(col("Contents\_array")))

# Filter and count occurrences of "phát" and "trả"

linkfact = df\_exploded \

    .filter(F.col("Content\_exploded").like("%phát%") | F.col("Content\_exploded").like("%trả%")) \

    .groupBy("Stock", "DateChange", "Link", "Title", "Body") \

    .agg(

        F.sum(F.when(F.col("Content\_exploded").like("%phát%"), 1).otherwise(0)).alias("Count\_of\_phat"),

        F.sum(F.when(F.col("Content\_exploded").like("%trả%"), 1).otherwise(0)).alias("Count\_of\_tra")

    )

linkfactdup = linkfact.select(col("Stock"),col("DateChange"),col("Link"),col("Title"),col("Body"),col("Count\_of\_phat"),col("Count\_of\_tra"))

linkfact1 = linkfactdup.dropDuplicates(["Stock","DateChange","Link"])

## Step 5d: Add "Link\_ID" from linkdim into linkfact dataframe:

# Join linkfact with data\_time based on matching dates

linkfact1\_with\_link\_id = linkfact1.join(linkdim2, linkfact1["Link"] == linkdim2["Link"], "inner")

linkfact1\_with\_link\_id.count() #  rows

# Select relevant columns:

linkfact1\_with\_link\_id\_1 = linkfact1\_with\_link\_id.select(

    linkfact1\_with\_link\_id["\*"],  # Select all columns

    linkdim2["Link\_ID"]  # Select Link\_ID from linkdim2

)

linkfact2 = linkfact1\_with\_link\_id\_1.select(

    linkfact1["Stock"],linkfact1["DateChange"], \

    linkdim2["Link\_ID"], \

    linkfact1["Link"], \

    linkfact1["Title"], \

    linkfact1["Body"], \

    col("Count\_of\_phat"),col("Count\_of\_tra"))

linkfact2.show()

print(linkfact2.count())

## Find rows that contain PutThrVol or OrdMatVol < 0:

neg = df\_fact.filter((F.col("PutThrVol") < 0) | (F.col("OrdMatVol") < 0))

neg.show()

# III - INSERT INTO 5 SSMS TABLE IN AN ORDER: DataStock > DataTime > LinkDim > Fact > FactLink:

# Define SQL Server database connection details

url\_sql = "jdbc:sqlserver://192.168.10.100:1433;database=dbstock4;encrypt=false;trustServerCertificate=false;hostNameInCertificate=\*.dat"

properties\_sql = {

    "user":"trangkieu4",

     "password":"123456",

    "driver": "com.microsoft.sqlserver.jdbc.SQLServerDriver"

}

# create a loop for inserting each dimension dataframe into corresponding SQL Server table:

tables = ['dbo.DataTime','dbo.LinkDim','dbo.Fact', 'dbo.LinkFact'] # pre-created SQL tables in SSMS

dfs = [data\_time1,linkdim2, df\_fact, linkfact2]

for i in range(len(dfs)):

    try:

        dfs[i].write.mode("append").jdbc(url\_sql, tables[i], properties=properties\_sql)

    except ValueError as error :

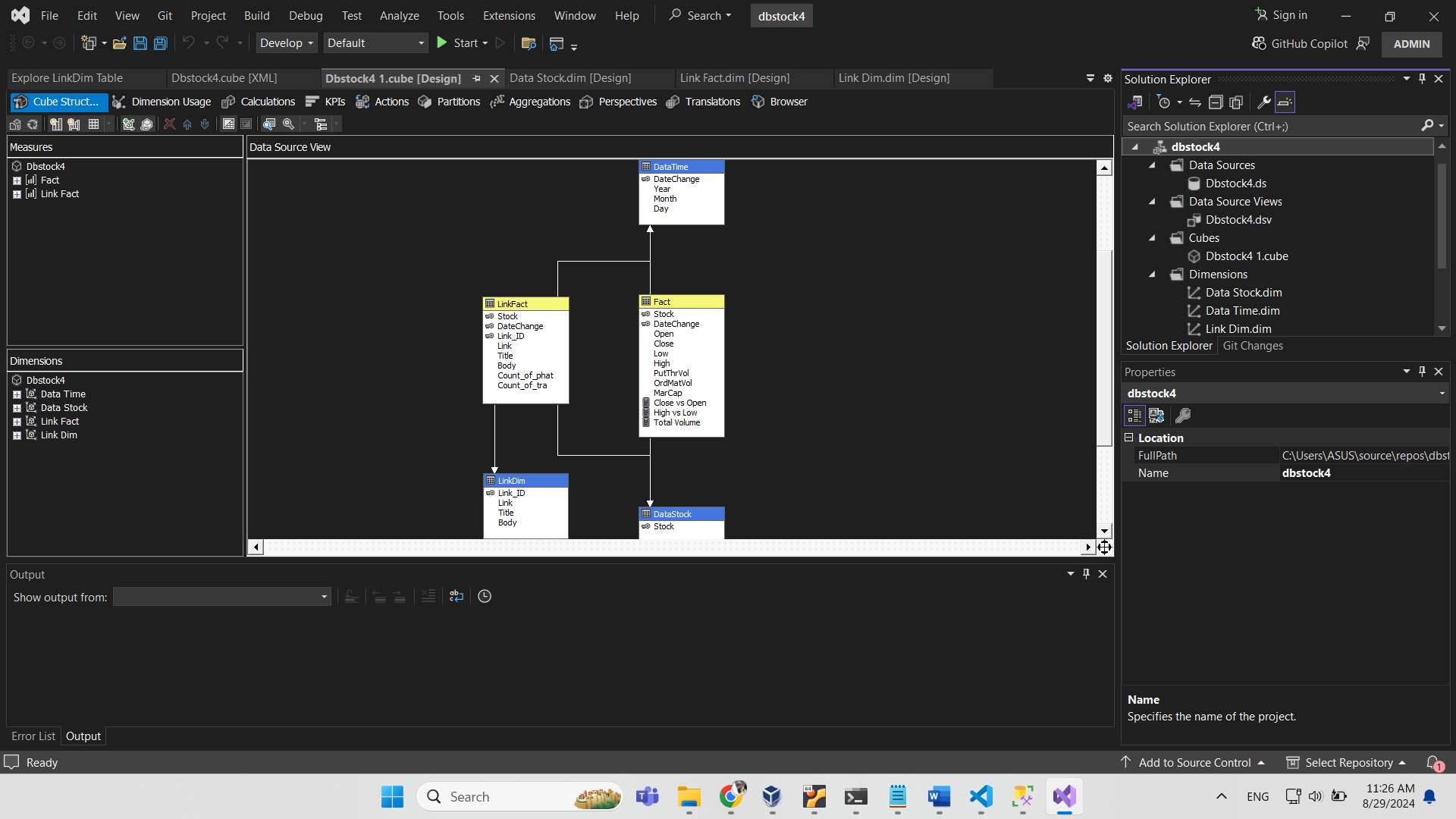
        print("Connector write failed", error)

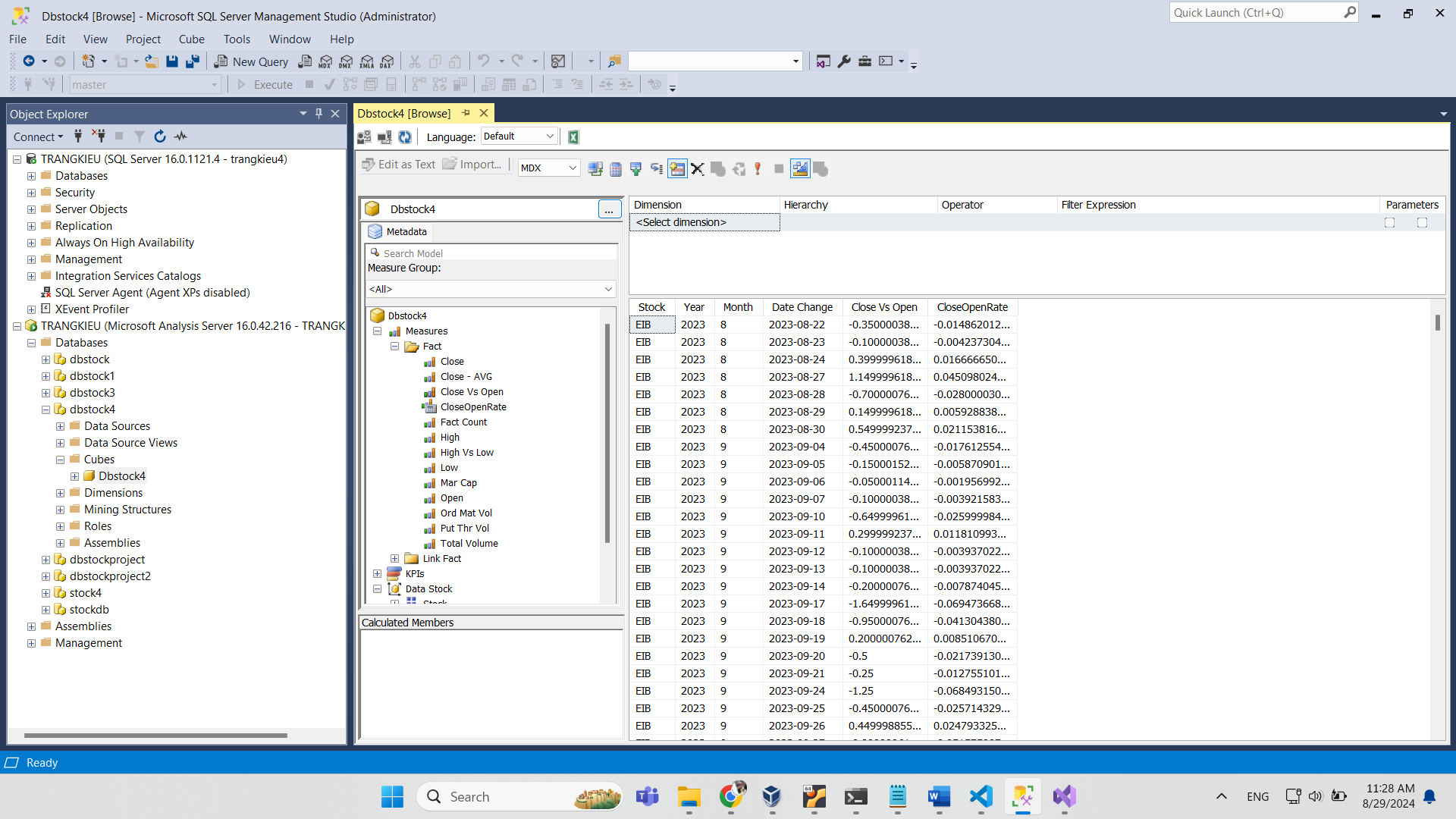
print("Data inserted successfully into SSMS tables")

# Since the second time, inserting data excludes data from DataStock to avoid duplicate conflicts.

spark.stop()

## 6. Creating cube database using MS Visual Studio with SSAS plugin





## 7. Creating Airflow tasks for automatic process

There are some notes when creating Airflow tasks:

* The orders of MongoDB and Kafka services to restart.
* Spark streaming session must be restarted in parallel with scrapy container, but scraping application runs only if PySpark shell is ready for retrieving streamed data in real time.
* Using subprocess method to run docker commands as if running these commands directly in Linux CLI.
* Creating an entrypoint (or .sh file) for the purpose of executing commands inside a bash shell of a Docker container.

This is the source codes of “all\_in.py”:

from airflow import DAG

from airflow.providers.docker.operators.docker import DockerOperator

from airflow.operators.python import PythonOperator

from datetime import datetime, timedelta

import docker

import time

from airflow.exceptions import AirflowException

import subprocess

import os

from airflow.utils.dates import days\_ago

from airflow.operators.dummy\_operator import DummyOperator

from watchdog.observers import Observer

from watchdog.events import FileSystemEventHandler

import glob

from airflow.operators.bash\_operator import BashOperator

# Define default arguments

default\_args = {

    'owner': 'airflow',

    'depends\_on\_past': False,

    'retries': 1,

    'retry\_delay': timedelta(minutes=5),

}

# DAG definition

with DAG(

    'all\_in',

    default\_args=default\_args,

    description='Check if a container is healthy before restarting another container',

    schedule\_interval=None,

    start\_date=datetime(2024, 8, 20) ,

    catchup=False,

    concurrency=10,

) as dag:

    create\_mongo\_db\_collection = BashOperator(

        task\_id='create\_mongo\_db\_and\_collection',

        bash\_command="""

        docker exec mongo1 mongosh --eval 'db = db.getSiblingDB("dbstock"); db.createCollection("tbl10stocks");'

        """,

    )

    create\_mongo\_db\_collection1 = BashOperator(

        task\_id='create\_mongo\_db\_and\_collection1',

        bash\_command="""

        docker exec mongo1 mongosh --eval 'db = db.getSiblingDB("dbstock"); db.createCollection("tbcontent");'

        """,

    )

    # Function to execute postgres creation command inside Docker container

    def run\_postgres\_creation\_func():

        result = subprocess.run(

            "docker exec postgres bash /newdbstock.sh",

            shell=True,

            stdout=subprocess.PIPE,

            stderr=subprocess.PIPE

        )

        print("STDOUT:", result.stdout.decode())

        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:

            raise Exception(f"Command failed with return code {result.returncode}")

    postgres\_creation = PythonOperator(

        task\_id='postgres\_creation',

        python\_callable=run\_postgres\_creation\_func

    )

    # Function to execute postgres rename command inside Docker container

    def postgres\_rename\_func():

        result = subprocess.run(

            "docker exec postgres bash /rename\_db.sh",

            shell=True,

            stdout=subprocess.PIPE,

            stderr=subprocess.PIPE

        )

        print("STDOUT:", result.stdout.decode())

        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:

            raise Exception(f"Command failed with return code {result.returncode}")

    postgres\_rename = PythonOperator(

        task\_id='postgres\_rename',

        python\_callable=postgres\_rename\_func

    )

    # Define the task: Restart the scrapy container

    def restart\_schema\_registry():

        import subprocess

        subprocess.call("docker restart schema-registry", shell=True)

    restart\_schema\_registry = PythonOperator(

        task\_id='restart\_schema\_registry',

        python\_callable=restart\_schema\_registry,

        dag=dag

    )

    # Function to check whether a container is running and restarting it:

    def check\_and\_restart\_container(container\_name, max\_retries=8, wait\_time=30):

        client = docker.from\_env()

        for attempt in range(max\_retries):

            container = client.containers.get(container\_name)

            # Check if the container is running

            if container.status == 'running':

                print(f"Container {container\_name} is running.")

                break  # Exit the loop if the container is running

            print(f"Container {container\_name} is not running. Status: {container.status}. Attempt {attempt + 1}/{max\_retries}. Restarting...")

            container.restart()

            # Wait for the specified time before checking again

            time.sleep(wait\_time)

        else:

            # This else block is executed if the loop finishes without hitting a 'break'

            raise ValueError(f"Container {container\_name} is still not running after {max\_retries} attempts.")

        print(f"Finished checking the status of container {container\_name}.")

    restartadminer = PythonOperator(

        task\_id='restartadminer',

        python\_callable=check\_and\_restart\_container,

        op\_args=['adminer']

    )

    restartpostgres = PythonOperator(

        task\_id='restartpostgres',

        python\_callable=check\_and\_restart\_container,

        op\_args=['postgres']

    )

    restartmongo1 = PythonOperator(

        task\_id='restartmongo1',

        python\_callable=check\_and\_restart\_container,

        op\_args=['mongo1']

    )

    restart\_ubuntu = PythonOperator(

        task\_id='restart\_ubuntu',

        python\_callable=check\_and\_restart\_container,

        op\_args=['ubuntu2204']

    )

    restartzookeeper = PythonOperator(

        task\_id='restartzookeeper',

        python\_callable=check\_and\_restart\_container,

        op\_args=['zookeeper']

    )

    restart\_broker = PythonOperator(

        task\_id='restart\_broker',

        python\_callable=check\_and\_restart\_container,

        op\_args=['broker']

    )

    # Function to retry restarting when health status checking is not healthy:

    def check\_container\_health\_status(container\_name, max\_retries=8, wait\_time=30):

        client = docker.from\_env()

        for attempt in range(max\_retries):

            container = client.containers.get(container\_name)

            # Check if the health status exists

            if 'Health' in container.attrs['State']:

                container\_status = container.attrs['State']['Health']['Status']

            else:

                #RAISE AIRFLOWEXCEPTION(F"CONTAINER '{CONTAINER\_NAME}' DOES NOT HAVE A HEALTH CHECK CONFIGURED.")

                print(f"Container '{container\_name}' does not have a health check configured.")

                break

            if container\_status == 'healthy':

                print(f"Container {container\_name} is healthy.")

                break  # Exit the loop if the container is healthy

            print(f"Container {container\_name} is not healthy. Status: {container\_status}. Attempt {attempt + 1}/{max\_retries}. Restarting...")

            container.restart()

            # Wait for the specified time before checking again

            time.sleep(wait\_time)

        else:

            # This else block is executed if the loop finishes without hitting a 'break'

            raise ValueError(f"Container {container\_name} is still not healthy after {max\_retries} attempts.")

        # Additional code can go here if needed after the loop is complete

        print(f"Finished checking the health status of container {container\_name}.")

    # Task to check the health status of the 'broker' container

    check\_broker\_health = PythonOperator(

        task\_id='check\_broker\_health',

        python\_callable=check\_container\_health\_status,

        op\_args=['broker'],  # Replace 'broker' with your container's name

    )

    # Task to restart the 'rest-proxy' container

    restart\_rest\_proxy = PythonOperator(

            task\_id='restart\_rest\_proxy',

            python\_callable=check\_and\_restart\_container,

            op\_args=['rest-proxy']

    )

    # Task to restart the 'connect' container

    restart\_connect = PythonOperator(

            task\_id='restart\_connect',

            python\_callable=check\_and\_restart\_container,

            op\_args=['connect']

    )

    # Task to check the health status of the 'connect' container

    check\_connect\_health = PythonOperator(

        task\_id='check\_connect\_health',

        python\_callable=check\_container\_health\_status,

        op\_args=['connect'],  # Replace 'broker' with your container's name

    )

    # Function to check if a container is running

    def check\_5container\_status():

        client = docker.from\_env()

        names=['zookeeper','broker','rest-proxy','connect','mongo1']

        for name in names:

            container = client.containers.get(name)

            if container.status.lower() == 'running':

                return True

            else:

                raise ValueError(f"Container {container\_name} is not running. Current status: {container.status}")

                break

    # Define tasks for each container

    check\_5containers = PythonOperator(

        task\_id='check\_5containers',

        python\_callable=check\_5container\_status,

    )

    # Define a function to check Kafka topics

    def check\_kafka\_topics():

        # Execute the command to open bash and run the status command

        result = subprocess.run(

            ["docker", "exec", "mongo1", "bash", "-c", "status"],

            capture\_output=True,

            text=True

        )

        output = result.stdout

        # Check if "dbstock.tbl10stocks" is in the output

        if "topic\": \"dbstock.tbl10stocks\"" in output:

            print("Topic 'dbstock.tbl10stocks' found!")

            return True

        else:

            print("Topic 'dbstock.tbl10stocks' not found.")

            return False

    # Task to check Kafka topics

    check\_kafka\_topics = PythonOperator(

        task\_id='check\_kafka\_topics',

        python\_callable=check\_kafka\_topics,

        dag=dag,

    )

    # Task to check and restart the Spark container

    check\_spark\_running = PythonOperator(

        task\_id='check\_spark\_running',

        python\_callable=check\_and\_restart\_container,

        op\_args=['spark'],

        dag=dag,

        task\_concurrency=3

    )

    # Function to run the entrypoint

    def run\_entrypoint\_func():

        result = subprocess.run(

            "docker exec spark bash entrypoint.sh",

            shell=True,

            stdout=subprocess.PIPE,

            stderr=subprocess.PIPE

        )

        print("STDOUT:", result.stdout.decode())

        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:

            raise Exception(f"Command failed with return code {result.returncode}")

    # Task to start pyspark shell and execute python script

    run\_entrypoint\_task = PythonOperator(

        task\_id='run\_entrypoint\_task',

        python\_callable=run\_entrypoint\_func,

        dag=dag,

    )

    # Define function to check log file to know if the pyspark is ready:

    def check\_log\_file(\*\*kwargs):

        dag\_id = kwargs['dag'].dag\_id

        task\_id = 'run\_entrypoint\_task'

        execution\_date = kwargs['execution\_date']

        log\_dir = f'/home/osboxes/airflow/airflowvenv/logs/dag\_id={dag\_id}'

        log\_pattern = os.path.join(log\_dir, '\*\*', 'attempt=\*.log')

        # Define the string to search for in the log files

        search\_string = f"INFO - Running <TaskInstance: {dag\_id}.{task\_id}"

        max\_wait\_time = 600

        check\_interval = 10  # Interval between checks in seconds

        start\_time = time.time()

        while time.time() - start\_time < max\_wait\_time:

            # Use glob to search for log files matching the pattern

            log\_files = glob.glob(log\_pattern, recursive=True)

            for log\_file in log\_files:

                try:

                    with open(log\_file, 'r') as file:

                        log\_contents = file.read()

                        if search\_string in log\_contents:

                            print(f"Found the string in {log\_file}!")

                            return

                except IOError as e:

                    print(f"Error reading {log\_file}: {e}")

            print(f"Log file matching pattern {log\_pattern} not found or string not found. Waiting...")

            time.sleep(check\_interval)

        raise Exception(f"Log file matching pattern {log\_pattern} not found or string not found after waiting for {max\_wait\_time} seconds.")

    def delete\_previous\_log\_files(task\_id\_to\_delete, \*\*kwargs):

        dag\_id = kwargs['dag'].dag\_id

        execution\_date = kwargs['execution\_date']

        log\_dir = f'/home/osboxes/airflow/airflowvenv/logs/dag\_id={dag\_id}'

        # Define a pattern to match logs for the task to delete

        log\_pattern = os.path.join(log\_dir, f'{task\_id\_to\_delete}', '\*\*', 'attempt=\*.log')

        # Use glob to find all log files matching the pattern for this task

        log\_files = glob.glob(log\_pattern, recursive=True)

        for log\_file in log\_files:

            try:

                # Check if the log file belongs to a previous execution

                log\_timestamp = os.path.basename(os.path.dirname(log\_file))  # Extract timestamp from log file path

                if log\_timestamp < execution\_date.strftime('%Y%m%d%H%M%S'):  # Compare with current execution time

                    os.remove(log\_file)

                    print(f"Deleted previous log file: {log\_file}")

            except OSError as e:

                print(f"Error deleting log file {log\_file}: {e}")

    # Define the delete\_previous\_log\_files task

    delete\_logs\_task = PythonOperator(

        task\_id='delete\_logs\_task',

        python\_callable=delete\_previous\_log\_files,

        op\_kwargs={'task\_id\_to\_delete': 'run\_entrypoint\_task'},  # Pass the task ID of check\_log\_file\_task

        provide\_context=True,

        dag=dag,

    )

    # Task to check log and start Scrapy container

    check\_log\_and\_start\_scrapy\_task = PythonOperator(

        task\_id='check\_log\_and\_start\_scrapy\_task',

        python\_callable=check\_log\_file,

        provide\_context=True,  # Ensures kwargs like execution\_date are passed automatically

        dag=dag,

    )

    # Task to check and restart the Scrapy container

    scrapy\_running = PythonOperator(

        task\_id='scrapy\_running',

        python\_callable=check\_and\_restart\_container,

        op\_args=['scrapy'],

        dag=dag,

    )

    # Function to start crawling

    def start\_crawl\_func():

        subprocess.run("sudo pkill Xvfb || true",

        shell=True,

        stdout=subprocess.PIPE,

        stderr=subprocess.PIPE

    )

        result = subprocess.run(

            "docker exec scrapy ./start\_service.sh",

            shell=True,

            stdout=subprocess.PIPE,

            stderr=subprocess.PIPE

        )

        print("STDOUT:", result.stdout.decode())

        print("STDERR:", result.stderr.decode())

    # Create the crawl task

    start\_crawl = PythonOperator(

        task\_id='start\_crawl',

        python\_callable=start\_crawl\_func,

        dag=dag,

    )

    # Dummy start and end tasks

    start = DummyOperator(

        task\_id='start',

        dag=dag,

    )

    end = DummyOperator(

        task\_id='end',

        dag=dag,

    )

    stop\_xvfb\_processes = BashOperator(

        task\_id='stop\_xvfb\_processes',

        bash\_command='sudo pkill Xvfb || true',

        dag=dag,

    )

    stop\_spark = BashOperator(

        task\_id='stop\_spark',

        bash\_command='docker stop spark',

        dag=dag,

    )

    stop\_scrapy = BashOperator(

        task\_id='stop\_scrapy',

        bash\_command='docker stop scrapy',

        dag=dag,

    )

    restart\_scrapy = PythonOperator(

        task\_id='restartscrapy',

        python\_callable=check\_and\_restart\_container,

        op\_args=['scrapy']

    )

    restart\_spark = PythonOperator(

        task\_id='restart\_spark',

        python\_callable=check\_and\_restart\_container,

        op\_args=['spark'],

        dag=dag,

        task\_concurrency=3

    )

    check\_spark\_running1 = PythonOperator(

        task\_id='check\_spark\_running1',

        python\_callable=check\_and\_restart\_container,

        op\_args=['spark'],

        dag=dag,

        task\_concurrency=3

    )

    def run\_entrypoint\_func1():

        result = subprocess.run(

            "docker exec spark bash entrypoint\_content.sh",

            shell=True,

            stdout=subprocess.PIPE,

            stderr=subprocess.PIPE

        )

        print("STDOUT:", result.stdout.decode())

        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:

            raise Exception(f"Command failed with return code {result.returncode}")

    # Task to start pyspark shell and execute python script again:

    run\_entrypoint\_task1 = PythonOperator(

        task\_id='run\_entrypoint\_task1',

        python\_callable=run\_entrypoint\_func1,

        dag=dag,

    )

    # Task to check log and start Scrapy container again:

    check\_log\_and\_start\_scrapy\_task1 = PythonOperator(

        task\_id='check\_log\_and\_start\_scrapy\_task1',

        python\_callable=check\_log\_file,

        provide\_context=True,  # Ensures kwargs like execution\_date are passed automatically

        dag=dag,

    )

        # Task to check and restart the Scrapy container

    scrapy\_running1 = PythonOperator(

        task\_id='scrapy\_running1',

        python\_callable=check\_and\_restart\_container,

        op\_args=['scrapy'],

        dag=dag,

    )

    def start\_crawl\_func1():

        subprocess.run("sudo pkill Xvfb || true",

        shell=True,

        stdout=subprocess.PIPE,

        stderr=subprocess.PIPE

    )

        result = subprocess.run(

            "docker exec scrapy ./startpubver3.sh",

            shell=True,

            stdout=subprocess.PIPE,

            stderr=subprocess.PIPE

        )

        print("STDOUT:", result.stdout.decode())

        print("STDERR:", result.stderr.decode())

    start\_crawl1 =  PythonOperator(

        task\_id='start\_crawl1',

        python\_callable=start\_crawl\_func1,

        dag=dag,

    )

    stop\_xvfb\_processes1 = BashOperator(

        task\_id='stop\_xvfb\_processes1',

        bash\_command='sudo pkill Xvfb || true',

        dag=dag,

    )

    stop\_spark1 = BashOperator(

        task\_id='stop\_spark1',

        bash\_command='docker stop spark',

        dag=dag,

    )

    def check\_kafka\_topics1():

        # Execute the command to open bash and run the status command

        result = subprocess.run(

            ["docker", "exec", "mongo1", "bash", "-c", "status"],

            capture\_output=True,

            text=True

        )

        output = result.stdout

        # Check if "dbstock.tbl10stocks" is in the output

        if "topic\": \"dbstock.tbcontent\"" in output:

            print("Topic 'dbstock.tbcontent' found!")

            return True

        else:

            print("Topic 'dbstock.tbcontent' not found.")

            return False

    # Task to check Kafka topics

    check\_kafka\_topics1 = PythonOperator(

        task\_id='check\_kafka\_topics1',

        python\_callable=check\_kafka\_topics1,

        dag=dag,

    )

    restart\_spark1 = PythonOperator(

        task\_id='restart\_spark1',

        python\_callable=check\_and\_restart\_container,

        op\_args=['spark'],

        dag=dag,

        task\_concurrency=3

    )

    def run\_entrypoint\_func2():

        result = subprocess.run(

            "docker exec spark bash post2ssms.sh",

            shell=True,

            stdout=subprocess.PIPE,

            stderr=subprocess.PIPE

        )

        print("STDOUT:", result.stdout.decode())

        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:

            raise Exception(f"Command failed with return code {result.returncode}")

    # Task to start pyspark shell and execute python script again:

    run\_entrypoint\_task2 = PythonOperator(

        task\_id='run\_entrypoint\_task2',

        python\_callable=run\_entrypoint\_func2,

        dag=dag,

    )

    def sql\_creation\_func():

        result = subprocess.run(

            "docker exec ubuntu2204 bash /entrypoint.sh",

            shell=True,

            stdout=subprocess.PIPE,

            stderr=subprocess.PIPE

        )

        print("STDOUT:", result.stdout.decode())

        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:

            raise Exception(f"Command failed with return code {result.returncode}")

    # Run rename SQL script inside the running container

    sql\_creation = PythonOperator(

        task\_id='sql\_creation',

        python\_callable=sql\_creation\_func

    )

    def run\_rename\_mongo\_func():

        result = subprocess.run(

            "docker exec mongo1 bash /scratch\_space/rename\_mongo\_db.sh",

            shell=True,

            stdout=subprocess.PIPE,

            stderr=subprocess.PIPE

        )

        print("STDOUT:", result.stdout.decode())

        print("STDERR:", result.stderr.decode())

        if result.returncode != 0:

            raise Exception(f"Command failed with return code {result.returncode}")

    # Task to run the script inside the MongoDB container

    rename\_mongo = PythonOperator(

        task\_id='rename\_mongo',

        python\_callable=run\_rename\_mongo\_func,

        dag=dag,

    )

    # Define the task to stop airflows

    # Task to stop the Airflow Scheduler

    stop\_scheduler = BashOperator(

        task\_id='stop\_airflow\_scheduler',

        bash\_command='pkill -f "airflow scheduler"',

        dag=dag

    )

    # Task to stop the Airflow Web Server

    stop\_webserver = BashOperator(

        task\_id='stop\_airflow\_webserver',

        bash\_command='pkill -f "airflow webserver"',

        dag=dag

    )

    # Define the task to stop Docker containers

    stop\_docker\_containers = BashOperator(

        task\_id='stop\_all\_containers',

        bash\_command='docker stop $(docker ps -q)',

        dag=dag,

    )

    # Set up task dependencies

    restart\_ubuntu >>  postgres\_creation >> restartmongo1 >> restartpostgres >> \

    restartadminer >> restartzookeeper >> restart\_broker >> check\_broker\_health >> \

restart\_schema\_registry >> restart\_rest\_proxy >> restart\_connect >> \

    check\_connect\_health >> check\_5containers >> check\_kafka\_topics >> start

    start >> \

check\_spark\_running >> [run\_entrypoint\_task, check\_log\_and\_start\_scrapy\_task]

    check\_log\_and\_start\_scrapy\_task >> scrapy\_running >> start\_crawl >> \

stop\_xvfb\_processes >> stop\_spark >> stop\_scrapy >> restart\_scrapy >> \

restart\_spark >> check\_kafka\_topics1 >> delete\_logs\_task >> check\_spark\_running1 \

>> [run\_entrypoint\_task1, check\_log\_and\_start\_scrapy\_task1]

check\_log\_and\_start\_scrapy\_task1 >> scrapy\_running1 >> start\_crawl1 >> \

stop\_xvfb\_processes1 >> stop\_spark1 >> sql\_creation >> restart\_spark1 \

>> run\_entrypoint\_task2 >> \

rename\_mongo >> create\_mongo\_db\_collection >> create\_mongo\_db\_collection1 >> \

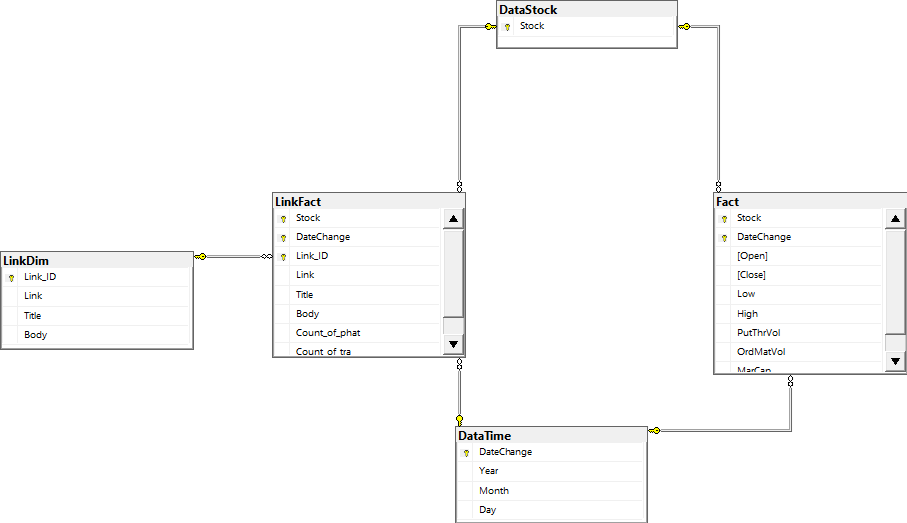
postgres\_rename >> stop\_docker\_containers >> stop\_scheduler >> stop\_webserver >> \

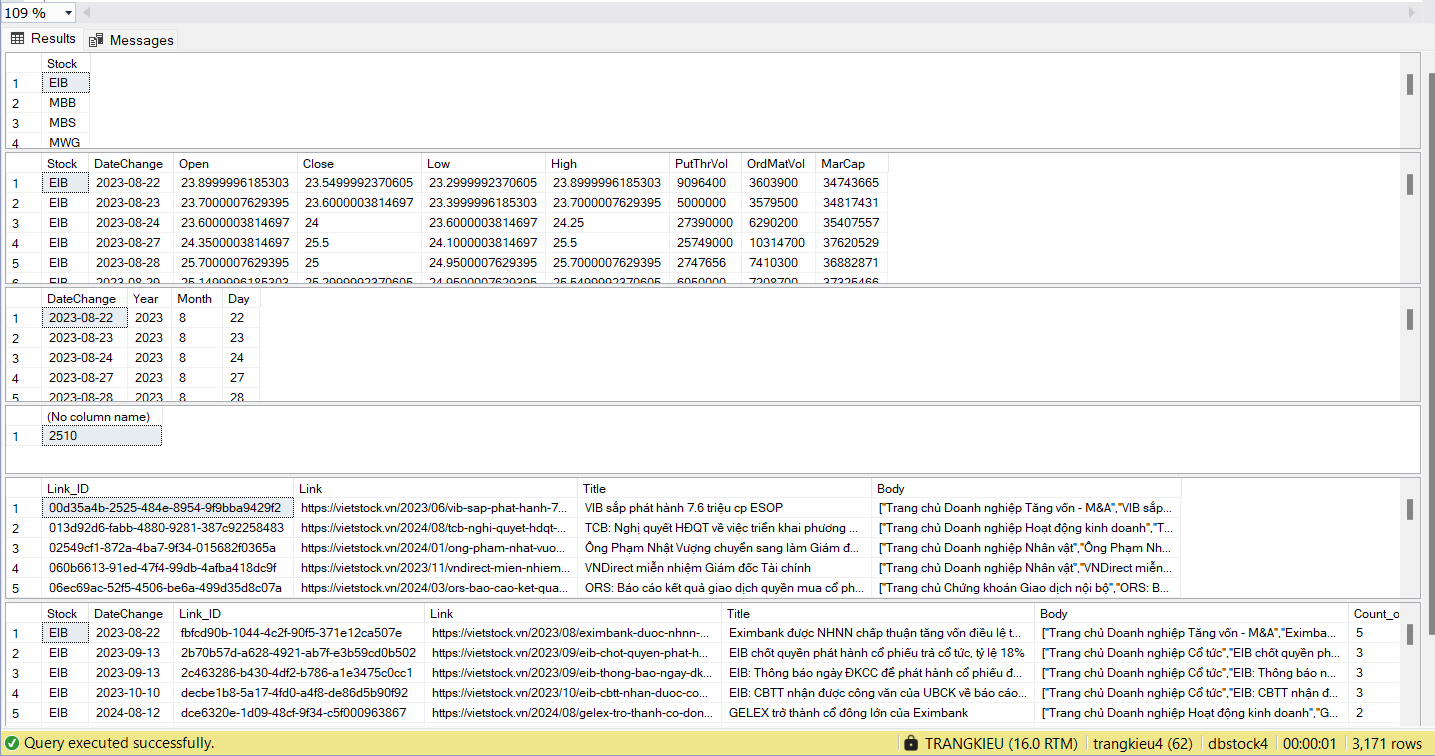
end

# IV. FINAL PRODUCTS

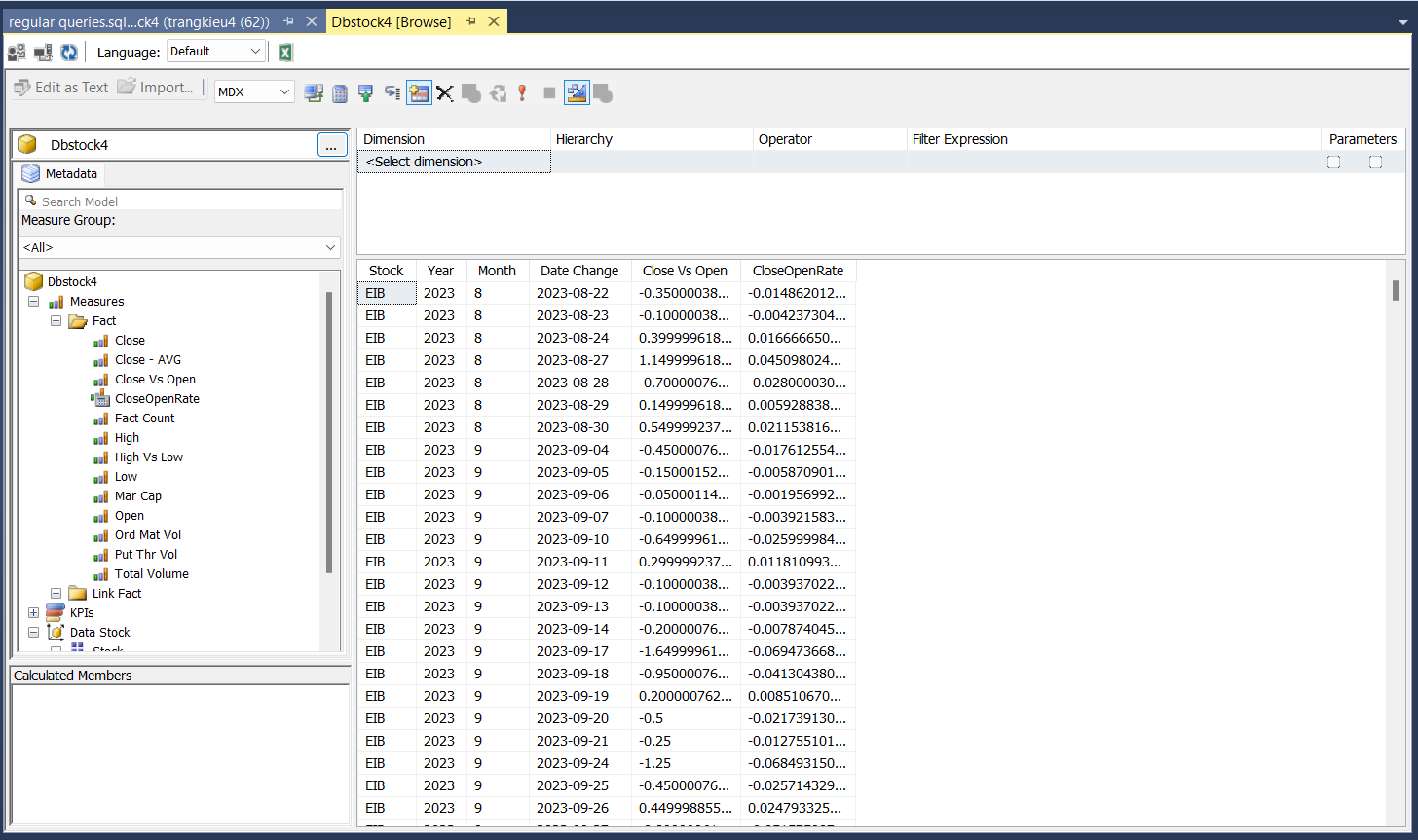
## 1. A galaxy schema data warehouse

Numerous data can tell the relationship between stock price and volume while content data can prove the close relation between the price and news published in the transferred day as well as an aggregation of total numbers of positive and negative words from the news contents. As a result, we can know which stock that giving its stakeholders good benefits. Thus, I create a galaxy schema with two fact tables named “Fact” and “LinkFact”.



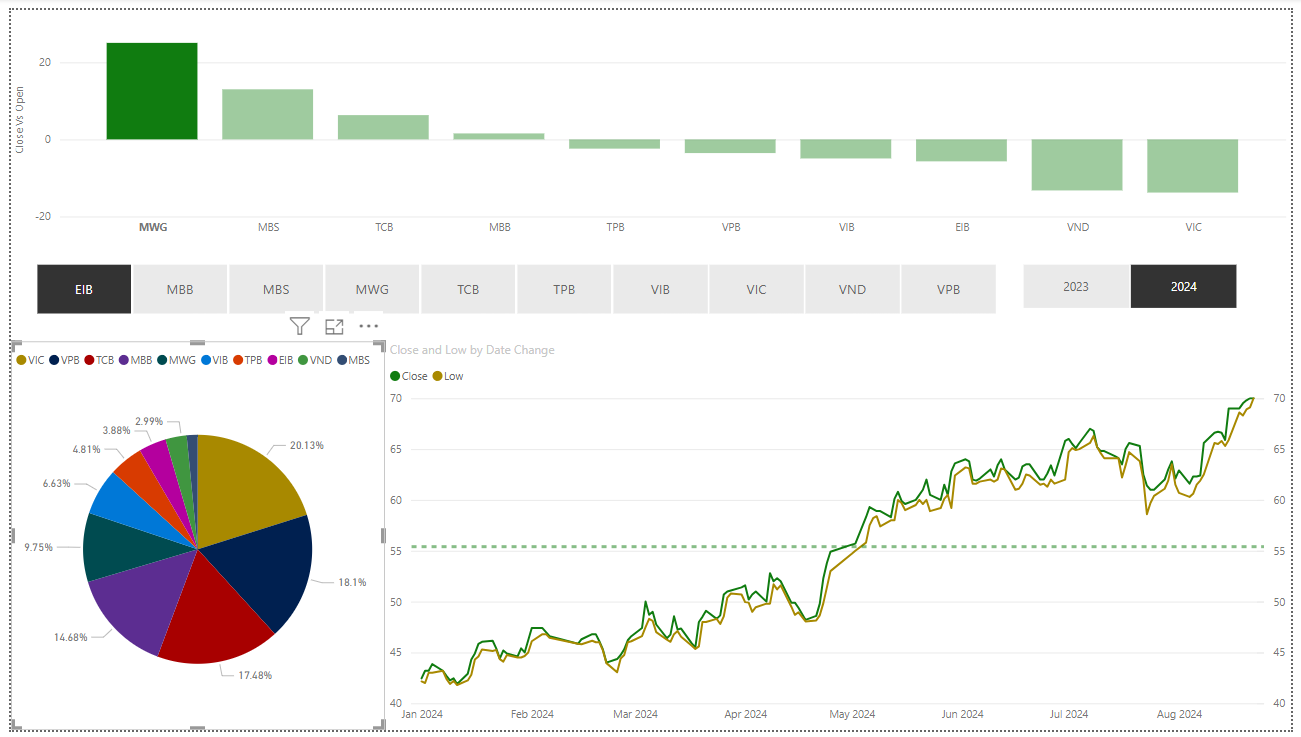
Using “Select … from…” queries to check if data is inserted successfully into data warehouse

Then, we can turn to the next step to create a cube database:



## 2. Dashboards and analysis

### 2.1. Overview of 10 stocks



From the pie chart, we can see the relative size of the market capitalization of the stocks. VIC has the largest proportion among these stocks while VPB, TCB stands at the second and third places respectively. Although MWG holds the fifth position with the moderate rate of under 10%, its price is in an upward trend. Meanwhile, VIC price shows a downward trend.

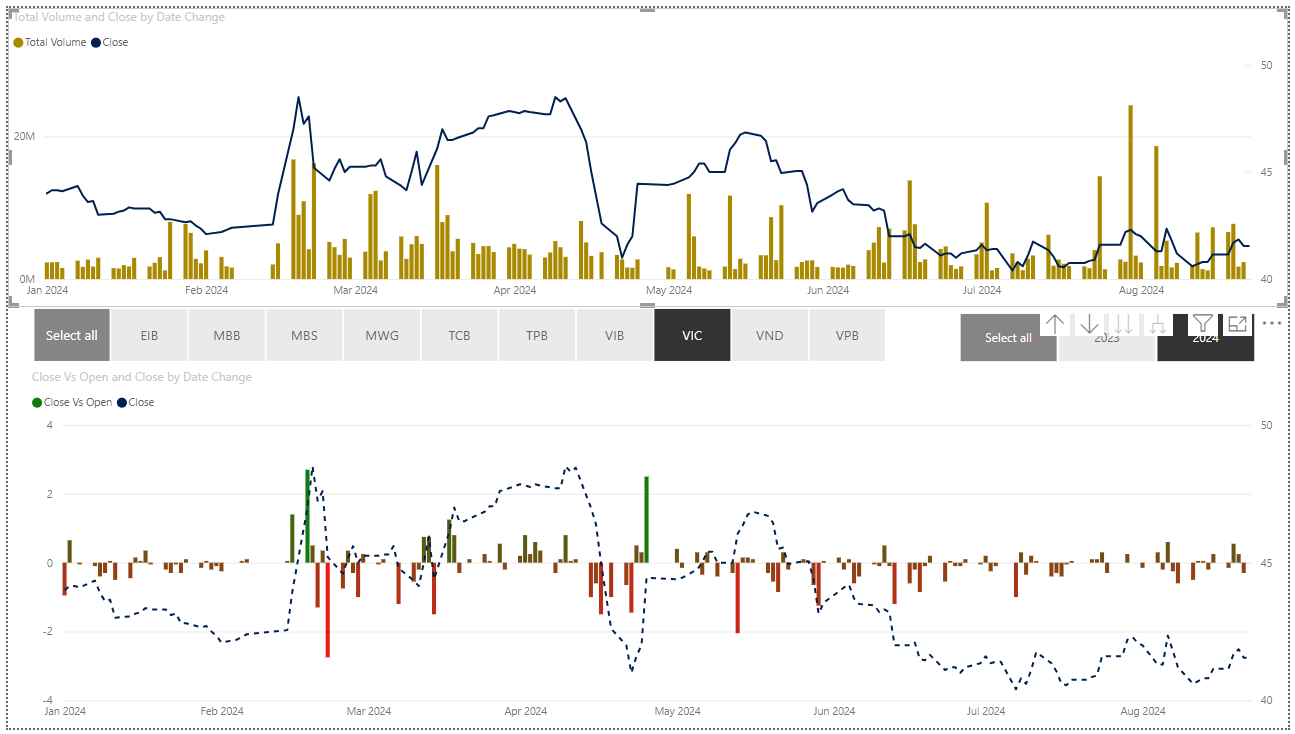
### 2.2. Relationship between news and stock price

May 20th 2024

March 27th 2024

As shown in the column-line combo chart on the top left corner of the dashboard, VND announced the implementation of the plan to issue additional shares on March 27th 2024 which is a negative news to stakeholders as displayed in the Title column of the table under the combo chart. Therefore, the price had started to decrease gradually since then. The same trend repeated on May 20th 2024. During the observation period, VND issued more shares than other stocks in the portfolio.

### 2.3. Relationship between stock price and volume

When price of a stock is increasing but its total volume does not follow the same trend, it demonstrates a signs of price reversal in the near future. Look at the line-column combo chart on the top left corner of the dashboard which shows a reversal in price trend of VIC stock from the mid-March to late April in 2024. Although the price increased by 5% from 46.1 to 48.45 thousand VND during the period from March 17th to April 11th, the total volume decreased strongly from nearly 15.9 million stocks to 1/5 of the beginning of the period at only 3.1 million stocks. It can be said that there is no solid foundation for the incline in the price. As a result, the price started to drop sharply by 18% from the peak to hit the bottom at only 41 thousand VND in a very short time of 11 days since it hit the peak.

🙡***THE END***🙣