**ĐẠI HỌC BÁCH KHOA HÀ NỘI**
**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**


# Báo cáo Project III

## *Đề tài: Financial Data Platform*

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

**Danh sách sinh viên thực hiện:**

| STT | Họ tên | MSSV |
|---|---|---|
| 1 | Phạm Cát Vũ | 20194465 |
| 2 | Phạm Đình Gia Dũng | 20194428 |
| 3 | Nguyễn Văn Hùng | 20194067 |
| 4 | Nguyễn Minh Mạnh | 20194113 |
| 5 | Ngô Hùng Mạnh | 20194112 |

*Hà Nội, Tháng 3 – 2023*

# *Lời cảm ơn*

Trong suốt quá trình thực hiện bài tập lớn, nhờ có sự giúp đỡ và chỉ bảo tận tình của thầy Trần Việt Trung và bạn bè mà nhóm chúng em có thể hoàn thành bài tập lớn này. Chúng em xin chân thành cảm ơn các thầy cô và bạn bè đã giúp đỡ trong thời gian qua.

Mặc dù đã cố gắng hết sức, nhưng do còn hạn chế về kiến thức, kỹ năng cũng như kinh nghiệm nên sản phẩm của nhóm còn nhiều thiếu sót. Nhóm rất mong nhận được những ý kiến đóng góp, chỉ bảo tận tình của thầy cô để hoàn thiện được đề tài này.

Xin chân thành cảm ơn!

# MỤC LỤC


**1. Giới thiệu chung** **4**
1.1. Vấn đề thực tế 4
1.2. Ý tưởng 4
1.3. Ưu điểm 4
1.4. Nhược điểm 4
**2. Công nghệ** **4**
2.1. Python 4
2.2. Postgres 5
TimescaleDB 6
2.3. Airflow 8
2.3.1. Một số thành phần chính của Airflow 8
2.3.2. Kiến trúc của Airflow 9
2.3.3. SparkSubmitOperator 10
2.3.4. Cron 10
2.4. Spark 10
2.4.1. Các thành phần của Spark 11
2.4.2. Kiến trúc của Spark 12
2.4.3 Các loại trừu tượng hóa dữ liệu của Spark 12
2.4.4. PySpark 13
2.5. Docker 13
2.6. Kubernetes 15
**Persistent Storage** **18**
Persistent Volumes (PV) 19
PersistentVolumeClaim (PVC) 19
**3. Data** **21**
3.1. TCBS Website 21
3.2. SSI Website 21
3.3. API 21
**4. Cài đặt** **21**
4.1. Database 21
4.2. Docker 22
4.2.1. Trước khi tiến hành cài đặt 22
4.2.2. Khởi tạo các thành phần phụ trên Docker 22
4.2.3. Tích hợp các thành phần phụ cùng với Airflow và triển khai trên Docker 22
4.2.4. Một số dịch vụ sẽ cần mount dữ liệu 22
4.3. Kubernetes 23
4.3.1 KinD 23
4.3.2 Helm 23
4.3.3 Airflow 23
4.3.4 Spark 23
**5. Kết quả** **24**
5.1. Tổng quan schema 24
5.1.1. listing_companies 24

5.1.2. listing_companies 25
5.1.3. stock_history 25
5.1.4. cash_flow 26
5.1.5. general_rating 26
5.1.6. business_model_rating 27
5.1.7. business_operation_rating 28
5.1.8. financial_health_rating 29
5.1.9. valuation_rating 30
5.1.10. industry_financial_health 30
5.1.11. financial_ratio 31
5.1.12. balance_sheet 34
5.1.13. stock_intraday_transaction 36
5.1.14. income_statement 36
5.1.15. derived_stock_history 38
5.2 Mối quan hệ giữa các bảng dữ liệu 39
5.3. Mô tả cách tính (Với các dữ liệu tính toán) 39
5.3.1. Simple Moving Average 39
5.3.2. Bollinger Band 39
5.3.3. Chaikin Money Flow 40
5.4 Thu thập và lưu trữ dữ liệu về thông tin các mã cổ phiếu từ Web vào Database 41
5.4.1 Cài đặt 41
5.4.2 Thực hiện 42
5.4.2.1 Lấy các mã cổ phiếu để lấy dữ liệu từ API 42
5.4.2.2 Lấy lược đồ cơ sở dữ liệu 42
5.4.2.3. Hàm ghi dữ liệu vào CSDL 42
5.4.2.4. Hàm UDF request API 42
5.4.2.5. Các hàm lấy dữ liệu 42
5.4.3 Các DAG của Airflow 45
5.5 Screenshot từng thành phần trong sản phẩm 47
6. Cải thiện 50
7. Phân chia công việc 51
**Tài liệu tham khảo** **52**

# 1. Giới thiệu chung

## 1.1. Vấn đề thực tế

- Từ trước đến nay việc theo dõi các chỉ số tài chính của các doanh nghiệp vẫn luôn là một nhu cầu thiết thực, có thể xuất phát từ nhiều lý do (Đầu tư sinh lời, nắm bắt tình trạng của doanh nghiệp nói riêng và thị trường nói chung). Khi mà công nghệ thông tin phát triển và được các ứng dụng vào thị trường chứng khoán ngày các tăng cao, theo luồng phát triển đó, nhu cầu cho các phương pháp tiếp cận chỉ số tài chính một cách thuận tiện, cụ thể ở đây là nền dữ liệu tài chính ngày càng trở nên hiện hữu.

## 1.2. Ý tưởng

- Xây dựng 1 nền tảng thu thập, lưu trữ và truy vấn dữ liệu tài chính nhằm hỗ trợ tối đa cho các cá nhân đầu tư cũng như người có nhu cầu tra cứu.
- Dữ liệu tài chính sẽ được thu thập từ các nguồn tin cậy và an toàn (Các trang web đầu tư), lưu trữ trên kho dữ liệu, và người dùng có thể truy vấn dữ liệu từ sử dụng giao diện thân thiện.
- Toàn bộ quá trình thu thập dữ liệu, vận chuyển lên kho dữ liệu và tính toán các dữ liệu sẽ được triển khai trên cụm máy nhằm tăng hiệu quả xử lý khi có nhiều người dùng truy cập/nhiều yêu cầu truy vấn dữ liệu cùng lúc

## 1.3. Ưu điểm

- Đơn giản, dễ sử dụng đối với người dùng, chỉ yêu cầu hiểu biết về lệnh truy vấn cơ bản
- Hệ thống tự động cập nhật dữ liệu không yêu cầu người vận hành

## 1.4. Nhược điểm

- Yêu cầu phần cứng chạy liên tục làm nơi chứa cơ sở dữ liệu và thực thi các tác vụ xử lý.
- Các lệnh truy vấn vẫn yêu cầu hiểu biết về ngôn ngữ SQL để thao tác

# 2. Công nghệ

## 2.1. Python

Python là một ngôn ngữ lập trình thông dịch (interpreted), hướng đối tượng và là ngôn ngữ lập trình bậc cao, ngữ nghĩa động. Python hỗ trợ các module và gói (packages), khuyến khích chương trình module hóa và tái sử dụng mã. Trình thông dịch Python và thư viện chuẩn mở rộng có sẵn dưới dạng mã nguồn hoặc dạng nhị phân miễn phí cho tất cả các nền tảng chính và có thể được phân phối tự do.
Đặc điểm của python:

- Ngữ pháp đơn giản, dễ đọc
- Python là ngôn ngữ mã nguồn mở: Python được phát triển theo giấy phép mã nguồn mở được OSI phê duyệt, khiến nó có thể sử dụng và phân phối miễn phí, ngay cả cho

mục đích thương mại. Giấy phép của Python được quản lý bởi Python Software Foundation.

- Vừa hướng thủ tục, vừa hướng đối tượng
- Hỗ trợ module và hỗ trợ package
- Kiểu dữ liệu động ở mức cao
- Có các bộ thư viện chuẩn và các module ngoài, đáp ứng tất cả các nhu cầu lập trình
- Có khả năng tương tác với các module khác viết trên C/C++
- Có thể nhúng vào ứng dụng như một giao tiếp kịch bản.

Flask là một micro web framework được viết bằng Python. Nó được phân loại là một microframework vì nó không yêu cầu các công cụ hoặc thư viện cụ thể. Nó không có lớp trừu tượng hóa cơ sở dữ liệu, xác thực biểu mẫu hoặc bất kỳ thành phần nào khác nơi các thư viện bên thứ ba có sẵn cung cấp các chức năng chung. Tuy nhiên Flask hỗ trợ tiện ích mở rộng có thể thêm các tính năng của ứng dụng như thể chúng được triển khai trong chính Flask. Tiện ích mở rộng tồn tại cho trình ánh xạ quan hệ tương đối, xác thực biểu mẫu, xử lý upload, nhiều công cụ xác thực mở, template, email, RESTful và một số công cụ liên quan đến các framework phổ biến.

Flask phù hợp cho việc xây dựng các web application có quy mô vừa và nhỏ, các API và web services:

- Xây dựng web application rất giống với việc viết các module Python chuẩn, cấu trúc gọn gàng và rõ ràng.
- Thay vì cung cấp hết tất cả mọi thứ, Flask cung cấp cho người dùng các thành phần cốt lõi thường được sử dụng nhất của khung ứng dụng web như URL routing, request & response object, template, …
- Với Flask, việc chọn component nào cho ứng dụng là việc của chúng ta. Điều này thật tuyệt, vì mỗi web application có những đặc điểm và tính năng riêng, nó không cần phải chứa các component mà nó không dùng.

## 2.2. Postgres

PostgreSQL là một hệ thống cơ sở dữ liệu quan hệ đối tượng mã nguồn mở, mạnh mẽ, sử dụng và mở rộng ngôn ngữ SQL kết hợp với nhiều tính năng giúp lưu trữ và thay đổi quy mô khối lượng công việc dữ liệu phức tạp nhất một cách an toàn.

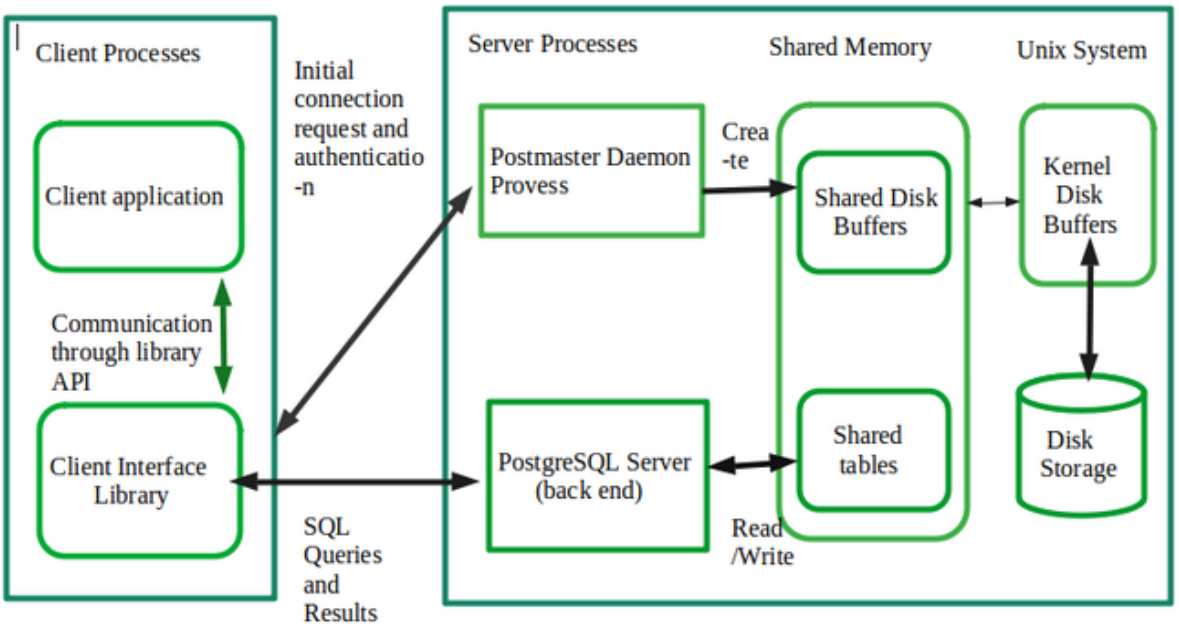
PostgreSQL đi kèm với nhiều tính năng nhằm giúp các nhà phát triển xây dựng ứng dụng, quản trị viên để bảo vệ tính toàn vẹn của dữ liệu và xây dựng môi trường chịu lỗi, đồng thời giúp bạn quản lý dữ liệu của mình bất kể tập dữ liệu lớn hay nhỏ. Ngoài việc là mã nguồn mở và miễn phí , PostgreSQL có khả năng mở rộng cao. PostgreSQL tuân thủ tiêu chuẩn SQL.

PostgreSQL có mô hình kiến trúc client-server:

- Quy trình phía server: Quản lý các tệp cơ sở dữ liệu, chấp nhận các kết nối đến cơ sở dữ liệu từ các ứng dụng máy khách và thực hiện các tác vụ cơ sở dữ liệu thay mặt cho máy khách.
- Quy trình phía client: Người dùng sử dụng để tương tác với cơ sở dữ liệu. Nó thường có giao diện người dùng đơn giản và được sử dụng để giao tiếp giữa người dùng và cơ sở dữ liệu thông thường thông qua các API.

**Quy trình phía client:**

Khi người dùng chạy các truy vấn trong PostgreSQL, client application có thể kết nối với máy chủ PostgreSQL (Quy trình Daemon Postmaster) và gửi truy vấn thông qua một trong nhiều giao diện chương trình Database Client Application được PostgreSQL hỗ trợ như JDBC, Perl DBD, ODBC, … cung cấp các thư viện phía máy khách. Trong quy trình client, giao tiếp giữa client application và thư viện xảy ra với sự giúp đỡ của API như trong hình dưới.




PostgreSQL JDBC Driver cho phép các chương trình Java kết nối với cơ sở dữ liệu PostgreSQL bằng mã Java tiêu chuẩn, độc lập với cơ sở dữ liệu. pgJDBC là JDBC driver mã nguồn mở được viết bằng Pure Java và giao tiếp trong giao thức mạng gốc PostgreSQL. Do đó driver độc lập với nền tảng, sau khi được biên dịch, driver có thể được sử dụng trên bất kỳ hệ thống nào.

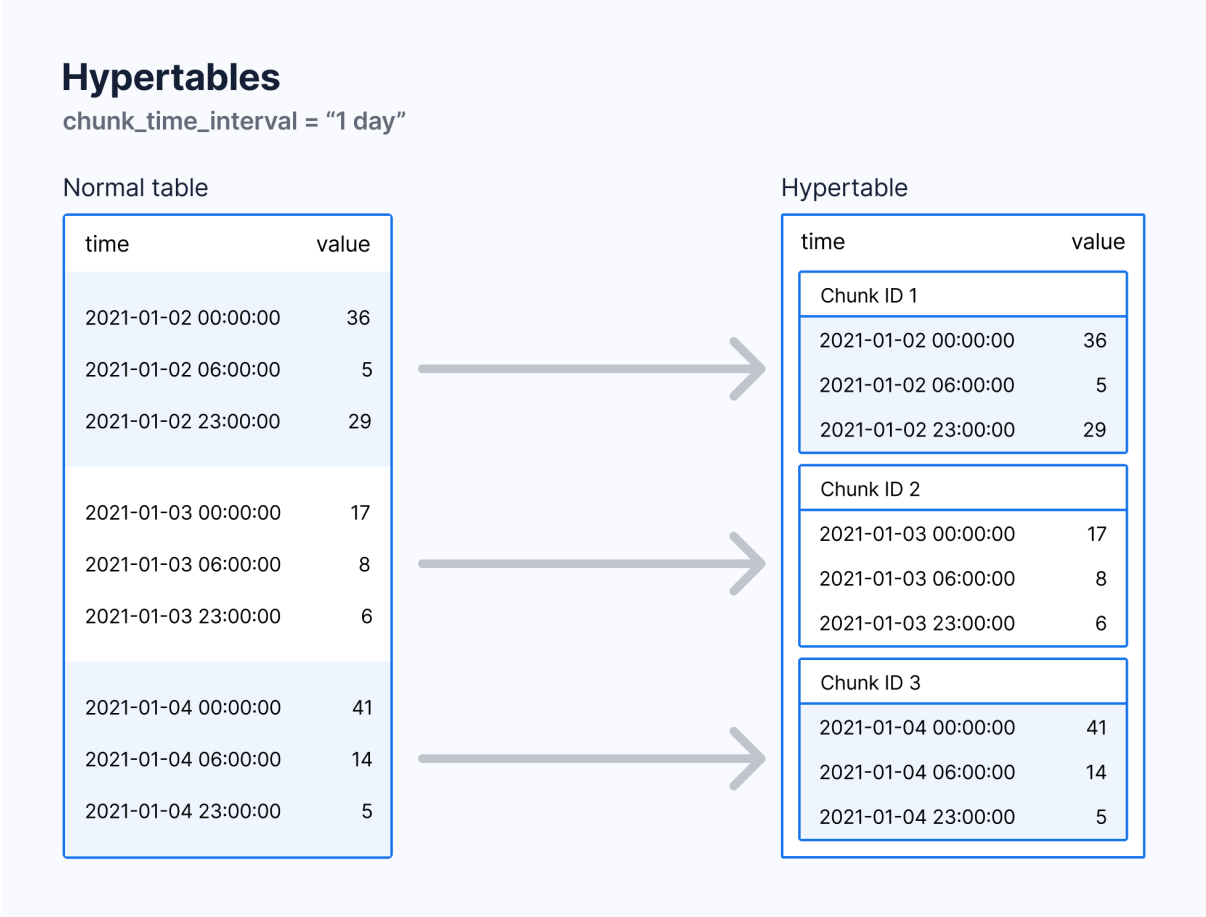
## TimescaleDB

TimescaleDB là một cơ sở dữ liệu chuỗi thời gian, được xây dựng trên PostgreSQL. Tuy nhiên, hơn thế nữa, nó là một cơ sở dữ liệu quan hệ cho chuỗi thời gian. Các nhà phát triển sử dụng TimescaleDB nhận được lợi ích của cơ sở dữ liệu chuỗi thời gian được xây dựng có mục đích, cộng với cơ sở dữ liệu quan hệ cổ điển (PostgreSQL), tất cả trong một, với sự hỗ trợ SQL đầy đủ.

Timescale database chạy bên trong PostgreSQL instance, TimescaleDB nâng cấp các cấu trúc lưu trong PostgreSQL bằng các hook tới query planner, data model, and execution engine.

Hypertables là cốt lõi của TimescaleDB. Hypertables cho phép TimescaleDB hoạt động hiệu quả với dữ liệu chuỗi thời gian. Vì TimescaleDB là PostgreSQL nên tất cả các bảng PostgreSQL thông thường, chỉ mục, thủ tục được lưu trữ và các đối tượng khác có thể được tạo cùng với các siêu bảng TimescaleDB của bạn. Điều này làm cho việc tạo và làm việc với các bảng TimescaleDB tương tự như PostgreSQL thông thường.

Việc tương tác với dữ liệu sẽ thông qua hypertable, đây là một tập hợp các bảng được chia tách một cách liên tục theo space và time intervals, nhưng việc query chỉ cần thực hiện theo các câu SQL tiêu chuẩn.
Bên trong TimescaleDB tự động chia mỗi hypertable thành các chunk nhỏ hơn, với mỗi chunk sẽ đối ứng với một time interval và một phân vùng key space xác định (hashing). Những phần vùng này sẽ không thể trùng nhau, điều đó giúp query tối ưu tập hợp các chunk cho việc giải quyết câu truy vấn. Mỗi chunk sẽ là một table tiêu chuẩn của sql, hay có thể gọi là "child table" của "parent" hypertable.




TimescaleDB đề xuất 3 lợi ích chính hơn PostgreSQL thông thường:

- Tỉ lệ insert cao hơn, đặc biệt cho các database có kích thước size lớn.
- Query performance giữ vững khi database scale.
- Hướng tối ưu cho query theo time.

Đối với PostgreSQL thông thường, performance bắt đầu giảm đáng kể ngay khi indexed tables trở nên quá lớn, không thể lưu được trong memory và buộc phải lưu xuống đĩa. TimescaleDB giải quyết việc này bằng việc phần vùng dữ liệu theo time-space, ngay cả khi trên một machine. Tất cả việc write sẽ chỉ viết trên child table có space time gần nhất, một table vẫn tồn tại trong memory, kích thước size nhỏ hơn dẫn tới việc tạo index rất nhanh. TimescaleDB sử dụng hash partitioning, làm cho việc partitioning dễ dàng phân phát qua một range value giữa các partitions.

## 2.3. Airflow

Airflow là một platform mã nguồn mở, hỗ trợ lên lịch và giám sát các luồng thực thi công việc theo batch. Python framework của Airflow cho phép người dùng có thể xây dựng luồng công việc kết nối với hầu hết mọi công nghệ. Airflow có hỗ trợ giao diện web giúp quản lý quy trình một cách trực quan hóa. Nó hỗ trợ deployment theo nhiều cách, từ chế độ độc lập trên máy tính xách tay đến chạy phân tán.

### 2.3.1. Một số thành phần chính của Airflow

**DAG**

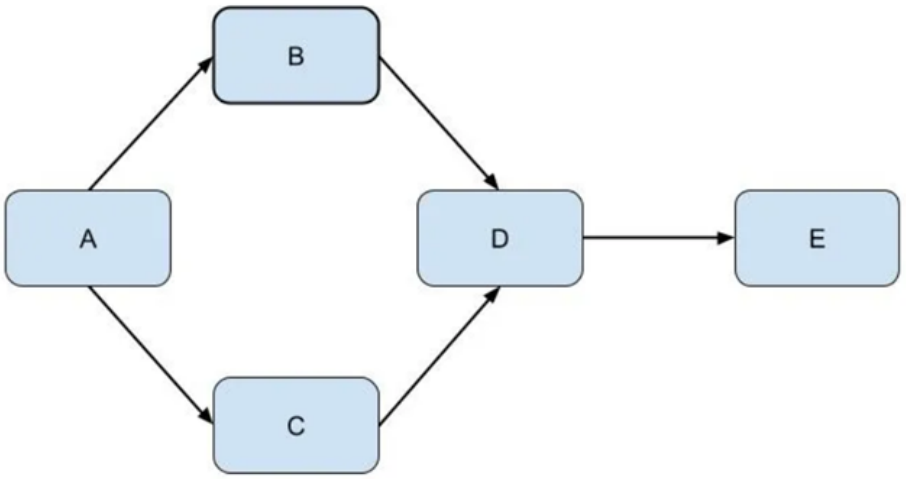
DAG (Directed Acyclic Graph) là một trong những khái niệm cốt lõi của Airflow. DAG là một đồ thi có hướng, là tập hợp nhiều node, được nối với nhau để phản ánh mối quan hệ giữa các node. DAG thể hiện mối quan hệ giữa các node, thứ tự thực hiện và việc thực hiện lại công việc trên node. Mỗi quy trình công việc (workflow) được biểu diễn dưới dạng DAG.

DAG mô tả thứ tự cách thực hiện workflow (tác vụ nào diễn ra khi nào, diễn ra trước tác vụ nào), nhưng không thể hiện nội dung chi tiết trong từng tác vụ. DAG chỉ quan tâm đến việc đảm bảo sao cho các tác vụ diễn ra đúng thời điểm, đúng trình tự, thứ tự, hoặc cách xử lý phù hợp đối với trường hợp lỗi, chứ DAG không quan tâm rằng từng tác vụ làm cụ thể những gì, công việc của từng tác vụ là gì.

Khi một DAG được thực thi  ở một lần chạy nhất định, nó được gọi là DAG run. DAG run là một instance của DAG. DAG run luôn được gắn liền với thời gian thực thi DAG, và phiên bản (version) của DAG.

**Task**

Task xác định một công việc sẽ được thực thi trong workflow, được biểu diễn bởi một node trong DAG, và được viết bằng Python. Những tác vụ này có thể sẽ được thực thi song song hoặc tuần tự - tùy theo cách thiết lập workflow, và có thể truyền thông tin giữa các tác vụ với nhau.




Hình ảnh thể hiện một DAG gồm 5 node A, B, C, D, E, mỗi node thực thi một tác vụ.

Khi task được thực thi trong một lần chạy nhất định, nó được gọi là task instance. Task instance luôn được gắn liền với trạng thái (state) của nó. State thể hiện tình trạng của task: scheduled, queued, running, success, failed, skipped, up_for_retry,…

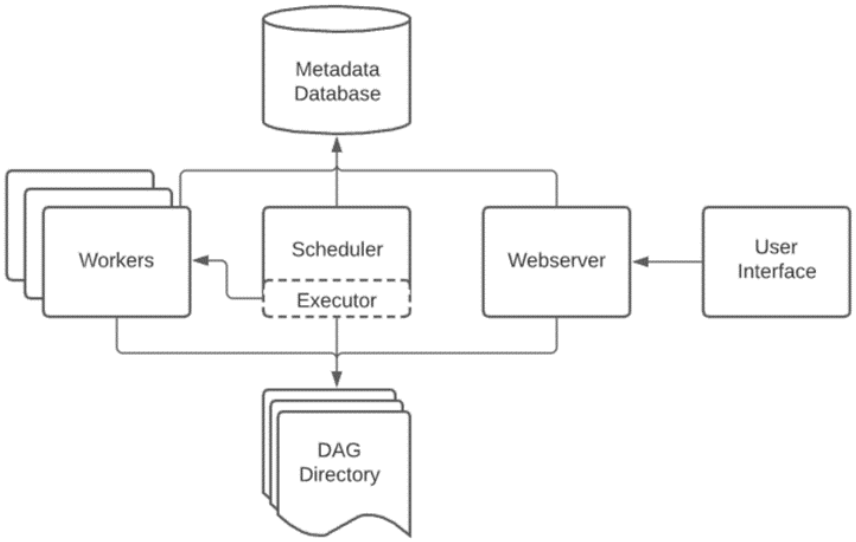
**Operator**

Operator định nghĩa các loại task khác nhau, thể hiện cách dữ liệu được xử lý như thế nào bên trong một task. Mỗi task được định nghĩa là một instance của operator. Operator tương tự như là một class định nghĩa các công việc có thể thực hiện, còn task như một instance của class. Operator cung cấp sẵn các hàm khác nhau cho task.

Một số operator phổ biến như PythonOperator (thực thi các mã Python), BashOperator (thực thi bash script), EmailOperator (gửi email), PostgresOperator (tương tác với cơ sở dữ liệu Postgres),…

Ngoài ra, còn một số các thành phần khác của Airflow như connections (thông tin cần thiết để kết nối Airflow với hệ thống bên ngoài, ví dụ như database,…), XComs (cho phép các task chia sẻ dữ liệu với nhau),…

### 2.3.2. Kiến trúc của Airflow




Người dùng sau khi tạo các DAG, nó sẽ được lưu trữ vào DAG Directory. DAG Directory được chia sẻ chung giữa scheduler, executor, worker, web server.

Scheduler kích hoạt quy trình công việc đã lên lịch và gửi các task cho executor.

Executor quản lý các worker, xử lý các tác vụ đang chạy. Executor sẽ gửi tác vụ đến cho các worker để các worker thực thi.

Worker thực thi các công việc và trả kết quả về, biểu diễn trên web server.

Web server biểu diễn giao diện trực quan để người dùng có thể kiểm tra các luồng công việc, các task thực thi. Người dùng cũng có thể thông qua web server, trực tiếp yêu cầu worker chạy các DAG.

Metadata database là nơi  để scheduler, executor và web server lưu trữ trạng thái công việc mình vào.

### 2.3.3. SparkSubmitOperator

Khi Airflow lên lịch thực thi các spark task, cần phải thiết lập các đường dẫn tới các spark task để đọc task, cũng như các dependencies cần thiết thực thi Spark task, thiết lập các trình quản lý cụm (kubernetes, yarn, standalone,…) hay deploy mode mà Spark hỗ trợ (client hay server), tài nguyên cung cấp,….

SparkSubmitOperator là một operator hỗ trợ cho việc submit các công việc thực thi bởi Spark.

Một số tham số hay sử dụng như spark_conn_id (connection_id kết nối tới Spark), jars (các dependencies packages được dùng để thực thi Spark job), driver_class_path (đường dẫn đến trình quản lý cụm Spark), executor_cores (số lượng lõi CPU thực thi spark job trên mỗi executor của spark), executor_memory (memory tối đa được sử dụng trên mỗi executor của spark),…

### 2.3.4. Cron

Cron là một tiện ích lập lịch công việc, thể hiện công việc được thực hiện vào thời gian nào. Cron có thể thiết lập công việc được chạy định kì theo thời gian nào, vào phút, ngày, giờ, tháng, năm nào.

Cron gồm 5 tham số thể hiện thời gian chạy công việc:

```
# ┌──────────────── phút (0 - 59)
# │ ┌──────────────── giờ (0 - 23)
# │ │ ┌──────────────── ngày trong tháng (1 - 31)
# │ │ │ ┌──────────────── tháng (1 - 12)
# │ │ │ │ │ ──────────────── ngày trong tuần (Thứ hai đến chủ nhật)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * <thời gian thực hiện>
```

Ví dụ: 7 * * * *: tác vụ được chạy 1 lần mỗi 1 giờ, tại thời điểm phút  thứ 7 của giờ đó

* 0 * 3 *: tác vụ được chạy mỗi phút 1 lần lúc 0 giờ, vào các ngày trong Tháng 3.

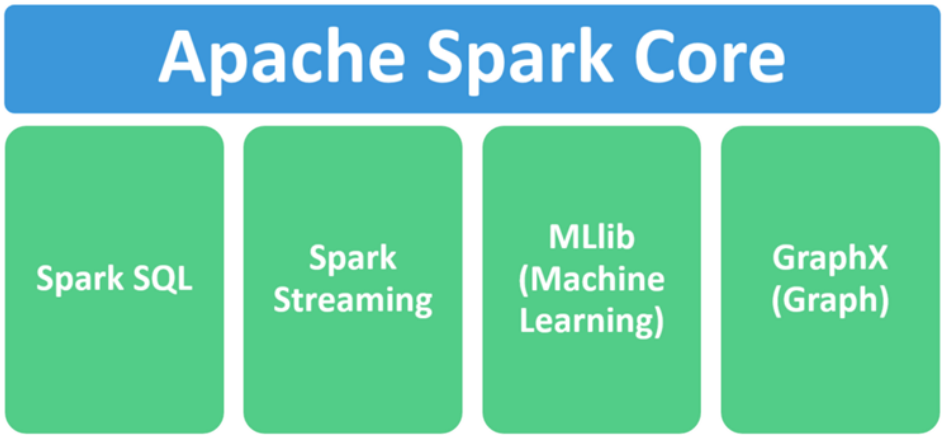
0 0 1 * *: tác vụ được chạy vào lúc 0 giờ 0 phút, ngày 1 hàng tháng

## 2.4. Spark

Apache Spark là một framework xử lý dữ liệu mã nguồn mở trên quy mô lớn. Spark hỗ trợ để lập trình các cụm tính toán song song với khả năng chịu lỗi. Spark hỗ trợ nhiều ngôn ngữ phổ biến (Python, Java, Scala và R), cùng các thư viện cho nhiều tác vụ khác nhau, từ SQL đến streaming và

machine learning. Khả năng tính toán phân tán của Apache Spark khiến nó rất phù hợp với big data, vốn đòi sức mạnh tính toán khổng lồ để làm việc trên các kho dữ liệu lớn. Spark cũng giúp loại bỏ một số gánh nặng lập trình cho các nhà phát triển với một API dễ sử dụng đảm nhiệm phần lớn công việc khó khăn của tính toán phân tán và xử lý dữ liệu lớn.

### 2.4.1. Các thành phần của Spark




Apache Spark gồm có 5 thành phần chính: Spark Core, Spark Streaming, Spark SQL, MLlib và GraphX.

Spark Core là thành phần cốt lõi của Apache Spark, các thành phần khác muốn hoạt động đều cần thông qua Spark Core. Spark Core có vai trò thực hiện công việc tính toán và xử lý trong bộ nhớ (In-memory computing), đồng thời nó cũng tham chiếu đến các dữ liệu được lưu trữ tại các hệ thống lưu trữ bên ngoài.

Spark SQL tập trung vào việc xử lý dữ liệu có cấu trúc. Spark SQL cho phép sử dụng cú pháp SQL để truy vấn dữ liệu, mang sức mạnh của Apache Spark đến các nhà phân tích dữ liệu cũng như các nhà phát triển. Bên cạnh khả năng hỗ trợ SQL, Spark SQL cung cấp một giao diện tiêu chuẩn để đọc và ghi vào các kho dữ liệu khác bao gồm JSON, HDFS, Apache Hive, JDBC, Apache ORC và Apache Parquet, tất cả đều được hỗ trợ trực tiếp. Các cơ sở dữ liệu phổ biến khác như Apache Cassandra, MongoDB, Apache Hbase,… cũng được hỗ trợ thông qua các trình kết nối riêng biệt từ hệ sinh thái Spark Packages.

Spark Streaming giúp Spark giúp nó đáp ứng các yêu cầu xử lý thời gian thực (realtime) hoặc gần như thời gian thực. Spark Streaming chia nhỏ luồng xử lý thành một chuỗi liên tục gồm các microbatch mà sau đó có thể được thao tác bằng API Apache Spark. Bằng cách này, mã trong các xử lý hàng loạt và trực tuyến có thể được tái sử dụng, chạy trên cùng một framework, do đó giảm chi phí cho cả nhà phát triển và nhà điều hành.

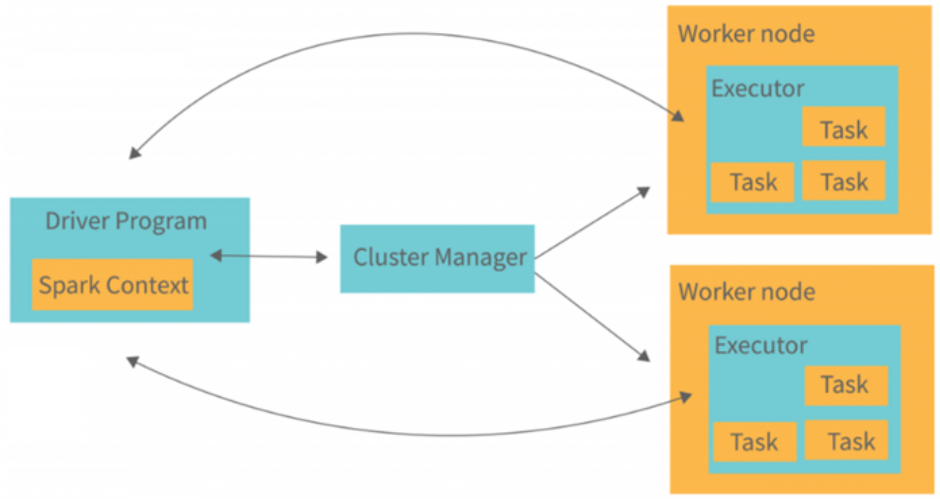
MLlib là một nền tảng học máy phân tán bên trên Spark với kiến trúc phân tán dựa trên bộ nhớ.

GraphX trong Spark cung cấp các thuật toán phân tán để xử lý cấu trúc đồ thị. Các thuật toán này sử dụng phương pháp tiếp cận RDD của Spark Core để lập mô hình dữ liệu; gói GraphFrames cho phép bạn thực hiện các xử lý biểu đồ trên khung dữ liệu, bao gồm cả việc tận dụng trình tối ưu hóa Catalyst cho các truy vấn đồ thị.

### 2.4.2. Kiến trúc của Spark

Apache Spark bao gồm hai thành phần chính: trình điều khiển (driver) và trình thực thi (executors). Trình điều khiển dùng để chuyển đổi mã của người dùng thành nhiều tác vụ (tasks) và phân phối nó đến trình thực thi executors trên các worker node.

Trách nhiệm cốt lõi của executors trên worker node là nhận các nhiệm vụ được giao, chạy chúng và báo cáo lại trạng thái và kết quả thành công hay thất bại của chúng về cho driver. Mỗi ứng dụng Spark có các quy trình thực thi riêng.




Trong quá trình thực hiện, driver tương tác với một thành phần bên ngoài Spark gọi là cluster manager, có thể là kubernetes, yarn,… Trình quản lý cụm (cluster manager) có tác dụng quản lý tài nguyên trong cụm. Khi thực thi, drive gửi yêu cầu đến trình quản lý cụm về số lượng tài nguyên cần cấp phát cho việc thực thi ứng dụng. Trình quản lý cụm sẽ tìm trên các node trong cụm xem node nào phù hợp để cấp phát cho việc khởi chạy ứng dụng mà driver yêu cầu.

### 2.4.3 Các loại trừu tượng hóa dữ liệu của Spark

Spark gồm các loại data abstraction: RDD, dataframe, dataset.

RDD

RDD (Resilient Distributed Dataset) là tập dữ liệu phân tán có khả năng phục hồi. Đây là khái niệm cốt lõi của Spark. Dữ liệu Spark cần xử lý lưu trữ phân tán. được trừu tượng hóa dưới dạng một tập hợp các bản ghi được lưu trữ trải dài, phân vùng trên nhiều máy trong cụm, có khả năng chịu lỗi, khôi phục lại khi gặp lỗi RDD không xác định được cấu trúc dữ liệu. RDD là API cấp thấp của Spark.

Dataframe.

DataFrame là một tập hợp dữ liệu phân tán không thay đổi. Không giống như RDD, dữ liệu được sắp xếp thành các cột được đặt tên, giống như một bảng trong cơ sở dữ liệu quan hệ. Được thiết kế để làm cho việc xử lý các tập dữ liệu lớn trở nên dễ dàng hơn, DataFrame cho phép các nhà phát triển áp đặt cấu trúc lên một tập hợp dữ liệu phân tán, cho phép trừu tượng hóa ở cấp độ cao hơn; nó cung cấp API ngôn ngữ dành riêng cho trường để thao tác dữ liệu phân tán; và làm cho Spark có thể tiếp cận

được với nhiều đối tượng hơn, ngoài các data engineer. Dataframe là API cấp cao của Spark, được xây dựng trên RDD.

Dataset**.**

Dataset là tập dữ liệu gồm nhiều hàng, với mỗi hàng trong dataset là một bảng. Mỗi bảng trong dataset được xác định kiểu dữ liệu một cách chặt chẽ thông qua JVM. Có thể nói dataset là nhiều hàng, mỗi hàng gồm một dataframe, nhưng được định kiểu một cách tự động và chặt chẽ thông qua JVM. Dataset là API cấp cao của Spark, được xây dựng trên RDD.

### 2.4.4. PySpark

PySpark là API Python dành cho Apache Spark, là một framework mã nguồn mở, hỗ trợ tính toán phân tán, có bộ thư viện để xử lý dữ liệu quy mô lớn, thời gian thực. Spark được viết bằng Scala, PySpark được phát hành để hỗ trợ sự chuyển đổi giữa Spark và Python.

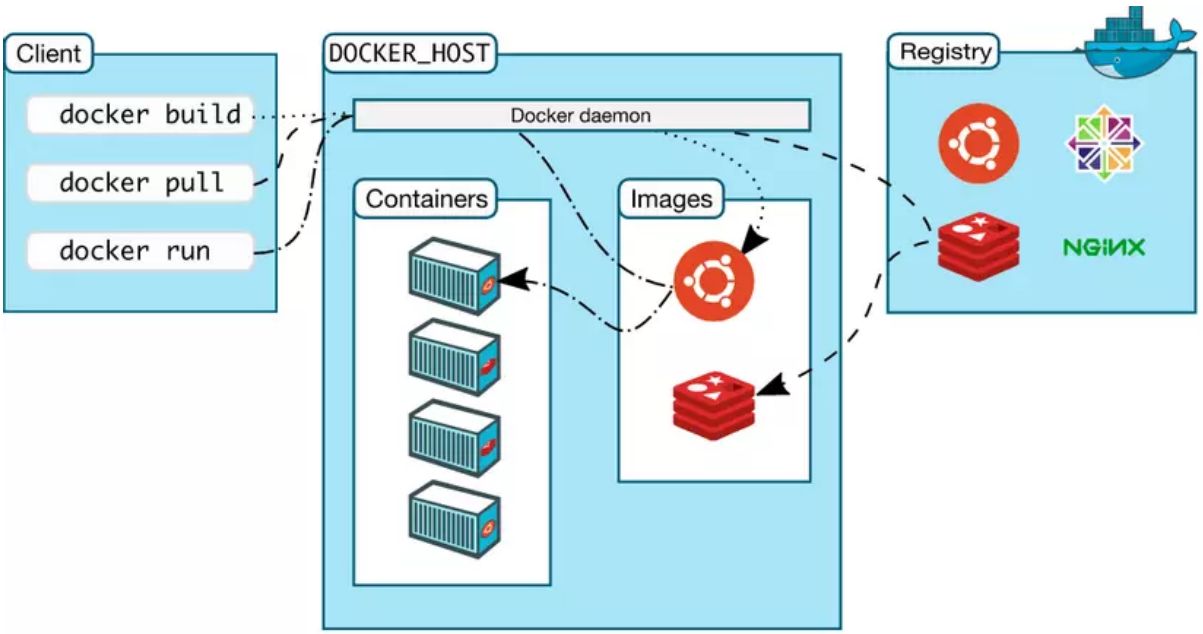
Loại dữ liệu chính được sử dụng trong PySpark là khung dữ liệu Spark. (Spark dataframe) Đối tượng này có thể được coi là một bảng được phân phối trên cụm và có chức năng tương tự như các dataframe trong R và Pandas. Nếu muốn thực hiện tính toán phân tán bằng PySpark, cần thực hiện các thao tác trên khung dữ liệu Spark chứ không phải các loại dữ liệu Python khác. Ngoài ta, PySpark hỗ trợ RDD và Spark SQL. PySpark không hỗ trợ dataset.

## 2.5. Docker

Docker là một nền tảng cho phép đóng gói, triển khai và chạy các ứng dụng một cách nhanh chóng. Ứng dụng Docker chạy trong vùng chứa (container) có thể được sử dụng trên bất kỳ hệ thống nào: máy tính xách tay của nhà phát triển, hệ thống trên cơ sở hoặc trong hệ thống đám mây. Và là một công cụ tạo môi trường được "đóng gói" (còn gọi là Container) trên máy tính mà không làm tác động tới môi trường hiện tại của máy, môi trường trong Docker sẽ chạy độc lập.

Docker được tạo ra để làm việc trên nền tảng Linux , nhưng đã mở rộng để cung cấp hỗ trợ lớn hơn cho các hệ điều hành không phải Linux, bao gồm Microsoft Windows và Apple OS X.

Kiến trúc của Docker:




**- Docker Client:** Docker CLI có vai trò là client, cung cấp giao diện tương tác với người dùng (command line) và gửi các RESTful requests tương ứng đến Docker Daemon.
**- Docker Daemon:** Docker Daemon (Docker Engine) là một lightweight runtime giúp build, run và quản lý các containers và các thành phần liên quan.
**- Docker Engine** quản lý 4 đối tượng chính. Tất cả các đối tượng đều có tên và ID.

**- Image**: là read-only template dùng để tạo containers. Được cấu tạo theo dạng layers, và tất cả các layers là read-only. Có thể tạo Docker Image bằng cách buid từ Dockerfile hoặc commit từ Docker Container.

**- Container**: là một runnable instance của một Docker Image. Khi chạy một container từ một Image, sẽ có một writable layer được tạo ra và tất cả các thay đổi về dữ liệu sẽ diễn ra ở layer này. Lúc này, có thể sử dụng lệnh commit một container để tạo ra một image mới.

**- Network**: cung cấp private network (VLAN) để các container trên một host có thể liên lạc được với nhau, hoặc các container trên nhiều hosts có thể liên lạc được với nhau (multi-host networking).

**- Volume**: dùng để lưu trữ các dữ liệu độc lập với vòng đời của container và giúp cho các containers có thể chia sẻ dữ liệu với nhau và với host.

**- Docker Images:**
Docker Images là một file read-only, không thể thay đổi được, nó chứa các thư viện, công cụ, dịch vụ hay packages, những cấu hình để chạy và cần thiết để tạo nên ứng dụng. Ta cũng không thể start hoặc run images giống như container và có thể tạo được image riêng hoặc lấy images đã public trên registry để về và customize thành một image gồm những công cụ cần thiết cho nhu cầu.
Để tạo ra một image, ta cần tạo một Docker File. Docker dựa vào file này để sử dụng và build ra images. Dockerfile bao gồm tất cả các instruction (hướng dẫn) để docker build ra một image.
**- Docker Containers:**
Docker Container có đặc điểm là chóng tàn (**ephemeral**), nghĩa có thể khởi tạo nhanh và cũng có thể xóa nhanh. Không nên lưu dữ liệu trong container. Nếu cần lưu thì nên sử dụng Volume.

Docker Container sử dụng cơ chế **process-based**. Người dùng có thể tạo Docker Container bằng:

- docker run <image_name> <command>
- docker run <image_name>

Khi thực hiện một trong hai cách trên, một Docker Container sẽ được tạo và thực thi **câu lệnh truyền vào** hoặc **câu lệnh mặc định** đã được định nghĩa trong Docker Image. Câu lệnh này sẽ sinh process có pid = 1.

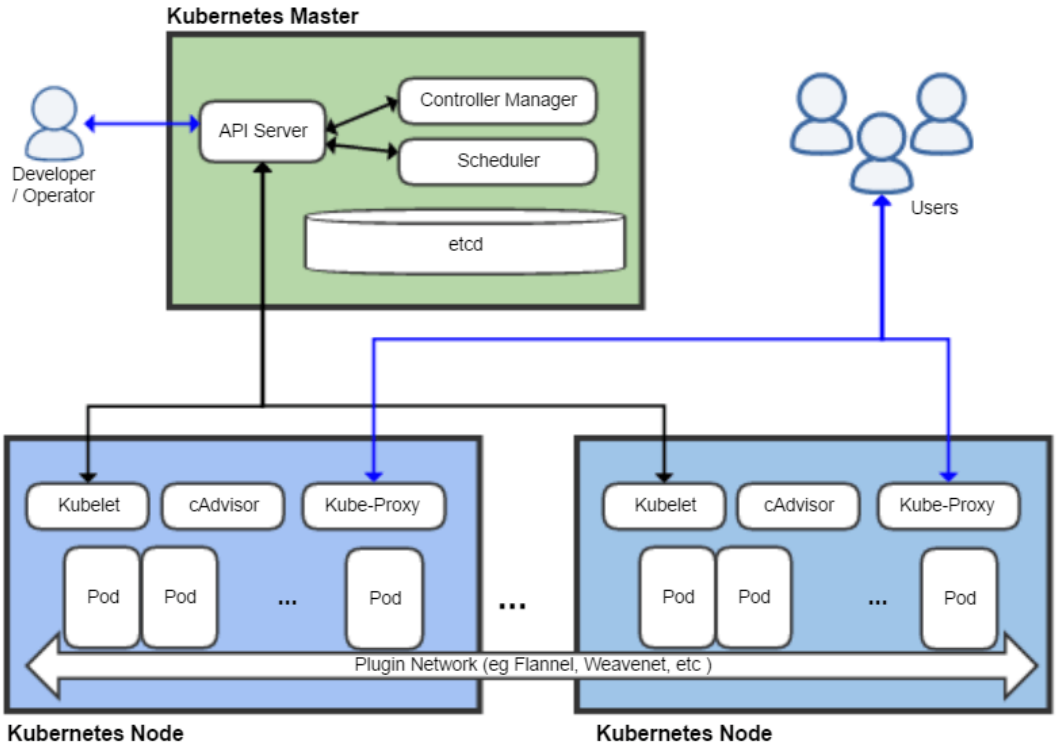
Nếu process này kết thúc thì Docker Container kết thúc. Khi đó, tất cả các process con của pid = 1 sẽ bị kill.

**- Docker Compose**
Với Dockerfile, Ta sẽ có thể tạo được 1 container chứa tất cả mọi thứ vào trong đó. Tuy nhiên, việc nhét quá nhiều thứ vào Dockerfile sẽ gây ra nhiều vấn đề trong việc build image nếu ta cần chỉnh sửa (tăng thời gian build chẳng hạn). Ngoài ra, một Dockerfile đảm nhận nhiều nhiệm vụ thì hoàn toàn không tốt chút nào với các principle KISS, SRP. Nếu bạn tách nó ra thành các Dockerfile riêng biệt để tránh các vấn đề trên thì việc chạy từng Dockerfile một cũng không thích hợp nếu như đó là hàng chục hoặc hàng trăm image. Ngoài ra, nếu ta có một container cần dùng chung (chẳng hạn như DB) hay đơn giản là có thể hoạt động với mọi môi trường như *dev, test, prod*... thì Dockerfile không hề dễ để thực hiện. Chính vì thế mà **Docker Compose** ra đời để giải quyết những vấn đề mà Dockerfile không thể làm tốt.
**Những tính năng chính của Compose bao gồm:**

- Thiết lập và cấu hình đa môi trường container hoàn toàn độc lập nhau trên cùng một máy chủ
- Bảo lưu các phân vùng bộ nhớ khi container được tạo ra
- Chỉ tạo lại container nào có config thay đổi trong khi vẫn bảo lưu dữ liệu của container
- Cho phép định nghĩa các biến variables trong file YAML để tùy chỉnh cho các môi trường dev và product.

## 2.6. Kubernetes

**Kubernetes** (còn gọi là k8s) là một hệ thống để chạy, quản lý, điều phối các ứng dụng được container hóa trên một cụm máy (1 hay nhiều) gọi là cluster. Với Kubernetes ta có thể cấu hình để chạy các ứng dụng, dịch vụ sao cho phù hợp nhất khi chúng tương tác với nhau cũng như với bên ngoài. Ta có thể điều chỉnh tăng giảm tài nguyên, bản chạy phục vụ cho dịch vụ (scale), có thể cập nhật (update), thu hồi update khi có vấn đề …
Kiến trúc của Kubernetes:

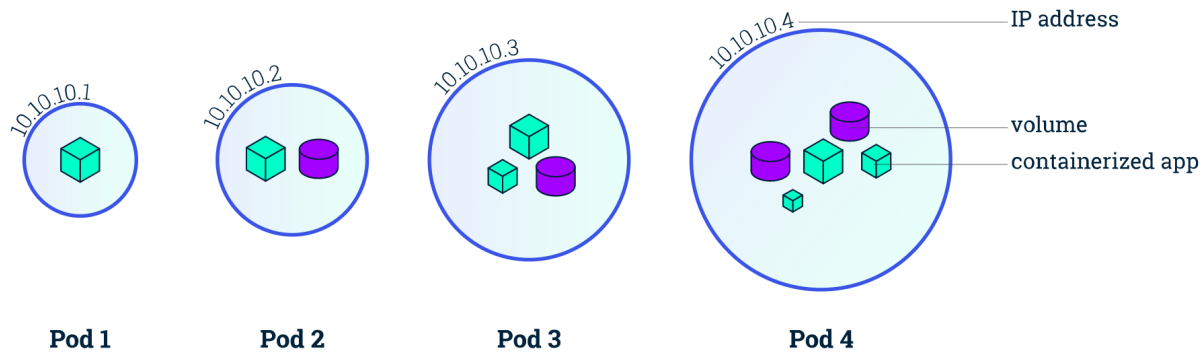



**- Cluster**

- **Control Plane**
  - **kube-apiserver** chạy tại máy master, cung cấp các API Restful để các client (như kubectl) tương tác với Kubernetes
  - **etcd** là thành phần cơ bản cần thiết cho Kubernetes, nó lưu trữ các cấu hình chung cho cả cụm máy, etcd chạy tại máy master. etcd là một dự án mã nguồn mở, nó cung cấp dịch vụ lưu dữ liệu theo cặp key/value
  - **kube-scheduler** chạy tại master, thành phần này giúp lựa chọn Node nào để chạy các ứng dụng căn cứ vào tài nguyên và các thành phần khác sao cho hệ thống ổn định.
  - **kube-controller** chạy tại master, nó điều khiển trạng thái cluster, tương tác để thực hiện các tác vụ tạo, xóa, cập nhật ... các tài nguyên
- **Node**
  - **Kubelet** dịch vụ vụ chạy trên tất cả các máy (Node), nó đảm đương giám sát chạy, dừng, duy trì các ứng dụng chạy trên node của nó.
  - **Kube-proxy**: cung cấp mạng proxy để các ứng dụng nhận được traffic từ ngoài mạng vào cluster.

**- Pods**

Kubernetes không chạy các container một cách trực tiếp, thay vào đó nó bọc một hoặc vài container vào với nhau trong một cấu trúc gọi là POD. Các container cùng một pod thì chia sẻ với nhau tài nguyên và mạng cục bộ của pod.



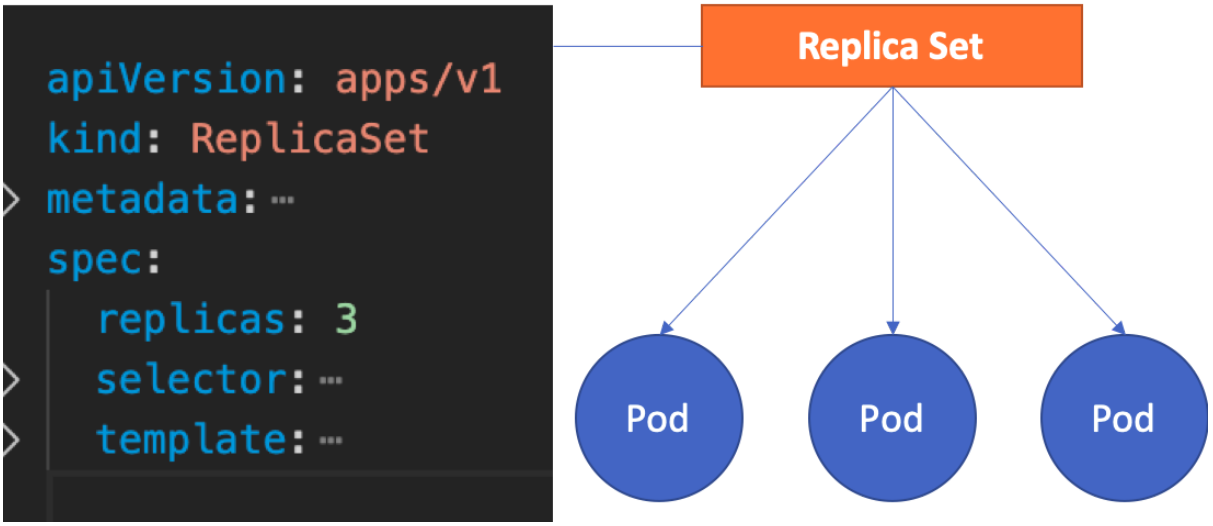

Pod là thành phần đơn vị (nhỏ nhất) để Kubernetes thực hiện việc nhân bản (replication), có nghĩa là khi cần thiết thì Kubernetes có thể cấu hình để triển khai nhân bản ra nhiều pod có chức năng giống nhau để tránh quá tải, thậm chí nó vẫn tạo ra nhiều bản copy của pod khi không quá tải nhằm phòng lỗi (ví dụ node bị die).

Pod có thể có nhiều container mà pod là đơn vị để scale (có nghĩa là tất cả các container trong pod cũng scale theo) nên nếu có thể thì cấu hình ứng dụng sao cho một Pod có ít container nhất càng tốt.

- Cách sử dụng hiệu quả và thông dụng là dùng loại Pod trong nó chỉ chạy một container.
- Pod loại chạy nhiều container trong đó thường là đóng gói một ứng dụng xây dựng với sự phối hợp chặt chẽ từ nhiều container trong một khu vực cách ly, chúng chia sẻ tài nguyên ổ đĩa, mạng cho nhau.

Deployment và ReplicaSet

**- ReplicaSet** là một điều khiển Controller - nó đảm bảo ổn định các nhân bản (số lượng và tình trạng của POD, replica) khi đang chạy.

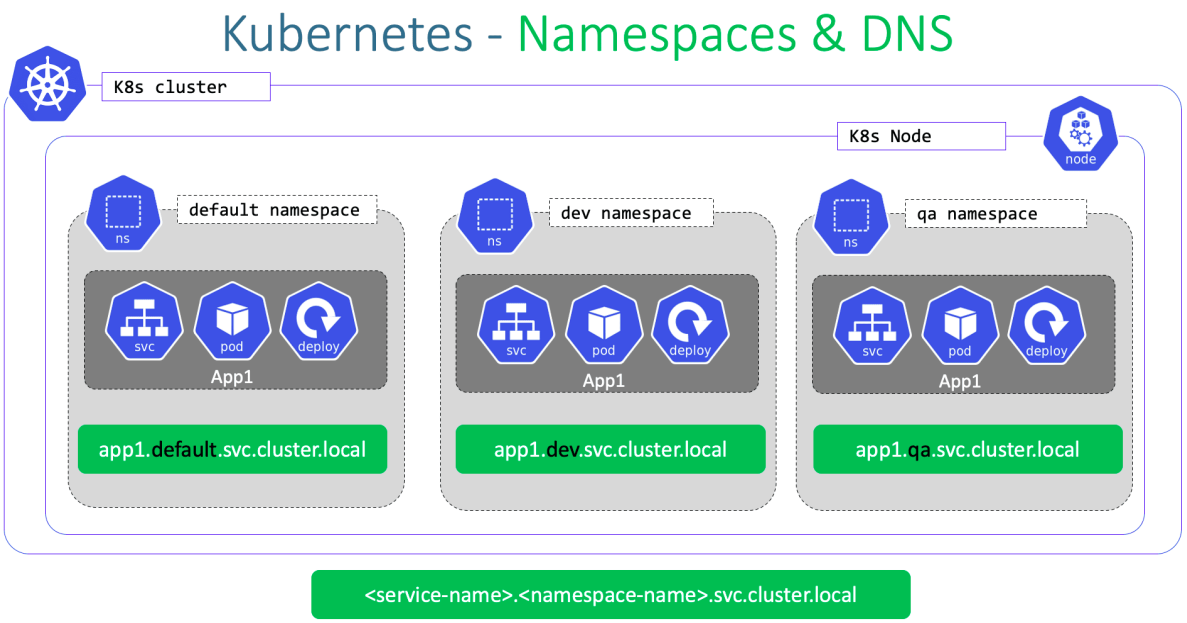



Khi định nghĩa một ReplicaSet (định nghĩa trong file .yaml) gồm các trường thông tin, gồm có trường selector để chọn ra các các Pod theo label, từ đó nó biết được các Pod nó cần quản lý(số lượng POD có đủ, tình trạng các POD). Trong nó nó cũng định nghĩa dữ liệu về Pod trong spec template, để nếu cần tạo Pod mới nó sẽ tạo từ template đó. Khi ReplicaSet tạo, chạy, cập nhật nó sẽ thực hiện tạo / xóa POD với số lượng cần thiết trong khai báo (replicas).

**- Deployment** quản lý một nhóm các Pod - các Pod được nhân bản, nó tự động thay thế các Pod bị lỗi, không phản hồi bằng pod mới nó tạo ra. Như vậy, deployment đảm bảo ứng dụng có một (hay nhiều) Pod để phục vụ các yêu cầu.

Deployment sử dụng mẫu Pod (Pod template - chứa định nghĩa / thiết lập về Pod) để tạo các Pod (các nhân bản replica), khi template này thay đổi, các Pod mới sẽ được tạo để thay thế Pod cũ ngay lập tức.

**Namespace**




Đây là một công cụ dùng để nhóm hoặc tách các nhóm đối tượng. Namespaces được sử dụng để kiểm soát truy cập, kiểm soát truy cập network, quản lý resource và quoting.

**- Storage**

Khi làm việc với docker ta đã quen với khái niệm docker volume. Hiểu một cách đơn giản thì Pod hay Server đều cần có resource để chạy gồm Memory/CPU, ngoài ra cần thêm ổ cứng nữa. Vì với server sẽ là disk còn với container sẽ là Volume. Volume là một thư mục để các container của Pod có thể truy cập và sử dụng để lưu trữ dữ liệu được.

Kubernetes hỗ trợ khá nhiều loại Volume, cho cả Public cloud và On Premise, có thể kể đến như sau:

- persistentvolumeclaim
- configMap
- secret
- hostPath
- emptyDir
- local
- nfs
- cephfs
- ...

**Persistent Storage**

Các loại Ephemeral Storage (lưu trữ tạm thời) có thời gian tồn tại song song với vòng đời của Pod. Nghĩa là khi Pod bị xóa đi hoặc bị tắt thì các dữ liệu tạm thời này cũng bị mất đi.

Ngược lại với nó, thì **Persistent Storage** vẫn còn tồn tại độc lập với Pod mà không bị xóa đi khi Pod bị xóa. Persistent Storage được sử dụng rất phổ biến trong thực tế với các ứng dụng cần lưu trữ trên K8S (stateful app).

Để tạo Persistent Storage cho các Pod/Deployment.. trên kubernetes thì ta sẽ làm việc với 2 loại tài nguyên là **PersistentVolume** (PV) and **PersistentVolumeClaim** (PVC).

**Persistent Volumes (PV)**

PersistentVolume (PV) là một phần phân vùng lưu trữ dữ liệu được cấp phát trên một hệ thống lưu trữ nhất định được thực hiện thủ công với admin hoặc tự động thông qua **Storage Class**. PV là tài nguyên của K8S Cluster giống như node cũng là tài nguyên của cluster vậy. PV được sử dụng như Volume Plugins nhưng vòng đời của nó độc lập với Pod sử dụng PV, nghĩa là Pod có thể đã bị xóa nhưng PV thì vẫn tiếp tục tồn tại.

**PersistentVolumeClaim (PVC)**

PVC là một yêu cầu về việc cấp phát một phân vùng lưu trữ bởi một user. PVC sẽ sử dụng tài nguyên PV giống như Pod sử dụng tài nguyên của Node vậy. Pod có thể yêu cầu một tài nguyên nhất định (về CPU/Memory) mà Node cần cấp phát cho nó. Thì tương tự PVC sẽ yêu cầu một phân vùng lưu trữ có dung lượng và phương thức truy cập (access mode) nhất định (ví dụ ReadWriteOnce hay ReadWriteMany..)

**- Configmap**

Configmap là một tài nguyên mức namespace trên k8s dùng để lưu trữ các dữ liệu dạng key-value. Pod có thể sử dụng configmaps dưới dạng các biến môi trường, dùng như tham số trong câu lệnh commandline trong pod hoặc sử dụng như một file cấu hình trong một volume. Lưu ý configmap không mã hóa dữ liệu, dữ liệu lưu trong configmap dưới dạng plain text.

Sử dụng configmap giúp cho ứng dụng trở nên linh động hơn. Các tham số của ứng dụng đều được cấu hình từ các biến môi trường hoặc từ file config. Các biến môi trường/file config này khi triển khai ở các môi trường cụ thể sẽ được gán giá trị từ configmap tương ứng.

Lưu ý khi sử dụng configmap cho Pod thì configmap và pod phải ở cùng namespace.

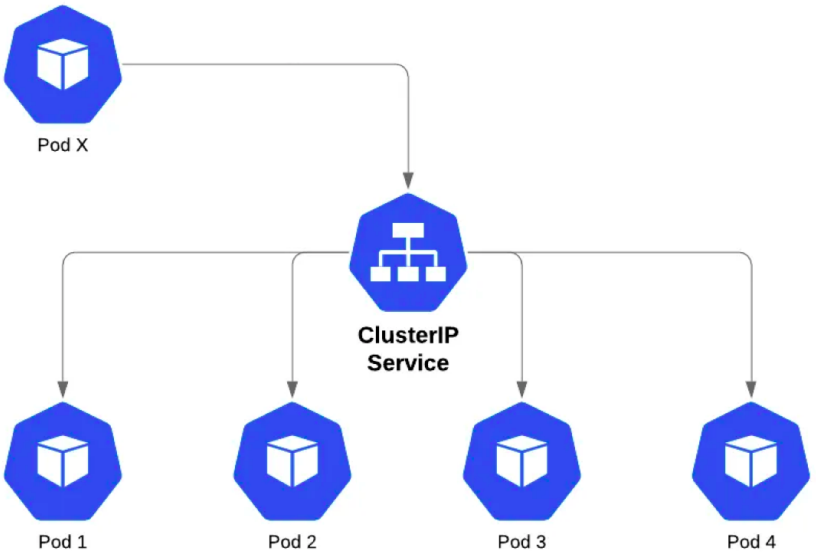
**- Secrets**

Secret tương tự với configmap, là một tài nguyên mức namespace trên k8s dùng để lưu trữ các dữ liệu dạng key-value. Điểm khác biệt lớn nhất giữa secret và configmap đó là secret sinh ra để lưu những thông tin nhạy cảm (sensitive data) như username, password, token... và các thông tin này sẽ được mã hóa bằng base64 khi lưu vào hệ thống.

Về cách sử dụng thì gần như không có khác biệt gì so với configmap, chúng ta sẽ vẫn có một số cách sử dụng cơ bản như:

- Tạo biến môi trường từ key của secret
- Sử dụng secret như một volume chứa các file (ví dụ các file certificate, key..)

**- Services**




Các **POD** được quản lý trong Kubernetes, trong vòng đời của nó chỉ diễn ra theo hướng - được tạo ra, chạy và khi nó kết thúc thì bị xóa và khởi tạo POD mới thay thế. ! Có nghĩa ta không thể có tạm dừng POD, chạy lại POD đang dừng ...

Mặc dù mỗi POD khi tạo ra nó có một IP để liên lạc, tuy nhiên vấn đề là mỗi khi POD thay thế thì là một IP khác, nên các dịch vụ truy cập không biết IP mới nếu ta cấu hình nó truy cập đến POD nào đó cố định. Để giải quyết vấn đề này sẽ cần đến Service.

Service (microservice) là một đối tượng trừu tượng nó xác định ra một nhóm các POD và chính sách để truy cập đến POD đó. Nhóm cá POD mà Service xác định thường dùng kỹ thuật Selector (chọn các POD thuộc về Service theo label của POD).

Cũng có thể hiểu Service là một dịch vụ mạng, tạo cơ chế cân bằng tải (load balancing) truy cập đến các điểm cuối (thường là các Pod) mà Service đó phục vụ.

**- Helm**

Helm là một trình quản lý gói và công cụ quản lý ứng dụng cho Kubernetes, gói nhiều tài nguyên Kubernetes vào một đơn vị triển khai logic duy nhất được gọi là **Chart.**

Lợi ích của Helm:

- Deployment đơn giản hơn mang tính chất lặp lại với chỉ vài câu lệnh ngắn
- Quản lý sự phụ thuộc của ứng dụng với các version cụ thể
- Thực hiện nhiều deployment với các môi trường khác nhau như: test, staging, production ...
- Thực thi các jobs liên quan đến chạy ứng dụng trước khi deployment
- Dễ dàng update rollback và test deployment khi có vấn đề xảy ra hay muốn cập nhật phiên bản mới (zero downtime server)

**- Kind**

Kind là một công cụ để chạy các cụm Kubernetes cục bộ bằng cách sử dụng các “Node” của Container của Docker. kind được thiết kế chủ yếu để thử nghiệm chính Kubernetes, nhưng có thể được sử dụng để phát triển cục bộ hoặc CI.

# 3. Data

## 3.1. TCBS Website

TCInvest là ứng dụng đầu tư chứng khoán của công ty Cổ phần chứng khoán Kỹ thương Techcom Securities (TCBS). Đây là công ty nằm trong hệ sinh thái tài chính của Ngân hàng Techcombank. Bảng giá chứng khoán TCBS là nơi thể hiện tất cả thông tin liên quan đến giá và các giao dịch cổ phiếu của thị trường chứng khoán.

Ngoài ra TCBS cũng sử dụng hình thức cung cấp dữ liệu lên web thông qua public api, do đó việc tận dụng các api này cho việc lấy dữ liệu trở nên đơn giản hơn rất nhiều so với cách truyền thống là scrape html table hay selenium để tải file.

## 3.2. SSI Website

Với SSI, có khá nhiều dữ liệu phân tích sử dụng nguồn cấp từ Fiin Trade rất có giá trị và sử dụng thuận tiện, đôi khi đặc biệt và hữu ích hơn những thông tin do TCBS cung cấp. Dữ liệu từ SSI cung cấp qua thư viện vnstock cũng đến từ public api (tương tự truy cập website thông qua trình duyệt). Có thể dễ dàng truy cứu các public api này thông qua công cụ của Chrome: Inspect >> Network >> Fetch/XHR.

## 3.3. API

Tại giai đoạn ban đầu của dự án, dữ liệu được lấy bằng cách sử dụng thư viện vnstock, trong khi đó dữ liệu được truyền qua api và được đọc bằng pandas rất nhanh chóng, hầu hết các chỉ cần đâu đó 1-2 giây để có được toàn bộ lịch sử giá của 1 cổ phiếu bất kỳ trong vòng 1 năm.
Tuy nhiên để phù hợp với mục đích của dự án (Các tính toán được thực hiện trên pyspark), chúng tôi đã viết lại các thư viện này để đọc dữ liệu trực tiếp trên pyspark, sau đó nhập vào bảng để đạt hiệu quả tối đa

# 4. Cài đặt

## 4.1. Database

Nhóm thực hiện cài đặt PostgreSQL với extension là TimescaleDB. Các thành viên tự chọn tên đăng nhập, mật khẩu và tên Database sẽ sử dụng. Database sẽ được cài đặt trên máy Host của Docker và Kubernetes.
Ngoài ra nhóm cũng sử dụng DBeaver, phần mềm quản lý cơ sở dữ liệu hỗ trợ JDBC driver, để trực quan hóa Database, tuy nhiên điều này hoàn toàn không có ảnh hưởng đến kết quả cuối cùng.

## 4.2. Docker

### 4.2.1. Trước khi tiến hành cài đặt

- Docker: Tiến hành cài đặt Docker trên máy, do thiết bị sử dụng hệ điều hành Ubuntu cho phép docker sử dụng toàn bộ tài nguyên trên máy, nhóm không thay đổi cài đặt gì và sử dụng thiết lập mặc định.

- Docker Compose: Tiến hành cài đặt docker compose trên máy, sử dụng thiết lập mặc định

- File docker-compose.yaml: Toàn bộ Airflow có thể được triển khai trên docker sử dụng file docker-compose.yaml, có thể tải xuống từ đường dẫn ‘https://airflow.apache.org/docs/apache-airflow/2.5.1/docker-compose.yaml’

### 4.2.2. Khởi tạo các thành phần phụ trên Docker

- Với mục đích triển khai giao diện người dùng ngay trên Docker, bước đầu nhóm tạo file tên Dockerfile tại thư mục chứa chương trình python.
- Trong file Docker file này sẽ điền các lệnh và chỉ dẫn phù hợp cũng như các thư viện đính kèm để docker, các thư viện ở đây gồm có psycopg2 (Dùng để kết nối và truy vấn từ bảng PostgreSQL), flask (Thư viện dùng để xây dựng giao diện người dùng, vnstock (Thu thập dữ liệu từ các nguồn TCBS và SSI) và pandas (Phục vụ 1 số tác vụ xử lý dữ liệu trong giao diện)
- Ngoài ra vì 1 trong những tính năng của giao diện cho phép người dùng nhập các biểu thức và tạo nên các tác vụ tính toán trong luồng công việc trên airflow, điều này yêu cầu giao diện phải có khả năng tạo các file python mới và lưu trong thư mục chứa DAG của airflow. Do đó chúng ta sẽ yêu cầu docker tạo 1 thư mục mới chứa các file python tạo mới bởi giao diện để sau này mount vào chính thư mục chứa DAG kể trên.
- Cuối cùng, nhóm sẽ xác định cổng port được sử dụng để biểu diễn giao diện người dùng, trong dự án, giao diện được xác định sẽ chạy trên port 5000

Sau khi đã có file Dockerfile, ta chạy lệnh để Docker xây dựng image mới từ chương trình python và file Dockerfile theo chỉ dẫn.

### 4.2.3. Tích hợp các thành phần phụ cùng với Airflow và triển khai trên Docker

Trong file docker-compose.yaml sẽ có các dịch vụ, với từng dịch vụ đóng 1 vai trò nhất định:

- web-ui: Giao diện người dùng, được triển khai từ image đã tạo ở mục trên
- airflow-scheduler: Bộ lập lịch giám sát tất cả các tác vụ và DAG, sau đó kích hoạt các phiên bản tác vụ sau khi các phần phụ thuộc của chúng hoàn tất.
- airflow-webserver: Giao diện web cho Airflow có sẵn, trong dự án nhóm xác định địa chỉ của giao diện tại http://localhost:8080
- airflow-worker: Công nhân thực hiện các tác vụ do bộ lập lịch trình đưa ra.
- airflow-init: Đóng vai trò khởi tạo Airflow
- postgres: Cơ sở dữ liệu chứa metadata của Airflow
- redis: broker chuyển tiếp tin nhắn từ bộ lập lịch tới worker.
- flower: Chương trình giám sát môi trường Airflow, có sẵn giao diện tại địa chỉ http://localhost:5555

### 4.2.4. Một số dịch vụ sẽ cần mount dữ liệu

Airflow: Mount dữ liệu vào 5 thư mục lần lượt như sau: dags (Thư mục lưu các file python DAG), logs (Thư mục chứa log Airflow), plugins, data (Thư mục chứa các dữ liệu tạo ra trong quá trình thực hiện luồng công việc) và script (Thư mục chứa các script python sẽ được gọi trong DAG)

## 4.3. Kubernetes

Trước khi tiến hành cài đặt, nhóm phải cài đặt một số công cụ như sau:

- Docker: Cho phép chạy container KinD
- Docker Compose
- KinD: Xây dựng 1 cluster Kubernetes với mỗi node là 1 Container trên Docker
- Helm
- Kubectl

Với K8S, nhóm sẽ thực hiện deploy 1 deployment Airflow và 1 deployment Spark bằng KinD và Helm

### 4.3.1 KinD

Với kind, tạo 1 cluster Kubernetes Local gồm 1 control-plane và 2 workers, mỗi node sẽ được chạy như là 1 Container trong Docker của máy host.
Ở 2 node worker sẽ mount trực tiếp Folder Airflow để thực hiện khởi tạo PersonalVolume và PersonalVolumeClaim cho DAG và log của Airflow.

### 4.3.2 Helm

Ta sẽ sử dụng Helm để cài đặt các Chart được đề cập đến ở bên dưới vào cluster KinD được tạo ở trên.

### 4.3.3 Airflow

Với Airflow, sử dụng Helm chart User-Community Airflow Helm Chart để thực hiện tạo 1 deployment với phiên bản Airflow 2.2.5 và CeleryExecutor.

- Ở trong Docker Image của image được sử dụng để khởi tạo, thực hiện cài đặt Java 11, PySpark, thư viện provider cho Spark của Airflow, Pandas và Requests để lấy dữ liệu từ API. JDBC Driver của Postgresql và các chương trình Spark sẽ được mount thẳng vào folder DAG của các pod Airflow thông qua PV và PVC đã đề cập ở trên.
- Deployment sẽ bao gồm các pod:
    - airflow-cluster-db-migrations: để gửi thông tin hoạt động của airflow về meta-database
    - airflow-cluster-flower: giao diện của Celery, dùng để giám sát worker
    - airflow-cluster-pgbouncer: pgbouncer for meta-database
    - airflow-cluster-postgresql: meta-database của airflow trong deployment
    - airflow-cluster-redis-master: database backend của Celery
    - airflow-cluster-scheduler: scheduler của Airflow
    - airflow-cluster-sync-users: synchronize thông tin của các người dùng
    - airflow-cluster-triggerer: triggerer của Airflow
    - airflow-cluster-web: giao diện web của Airflow
    - airflow-cluster-worker: worker của Airflow, sẽ có 2 worker trong deployment này

### 4.3.4 Spark

Với Spark, nhóm sử dụng Helm chart bitnami/spark để thực hiện tạo 1 deployment với phiên bản 3.3.2.

- Ở trong Docker Image của image được sử dụng để khởi tạo chart, tải JDBC Driver cho Postgres và cài thư viện Pandas và Requests của Python để lấy dữ liệu từ API.
- Tuy nhiên vì một vài lý do được giải thích ở phần sau, Deployment của Spark này sẽ không được sử dụng trong phiên bản cuối.

# 5. Kết quả

## 5.1. Tổng quan schema

### 5.1.1. listing_companies

| **STT** | **Tên trường** | **Loại Dữ liệu** | **Ràng buộc** | **Mô tả** |
|---|---|---|---|---|
| 1 | ticker | varchar(8) | PK | Mã cổ phiếu |
| 2 | exchange | varchar(5) | | Sàn giao dịch |
| 3 | short_name | varchar(256) | | Tên ngắn gọn |
| 4 | industry_id | double | | Mã ngành |
| 5 | industry_idv2 | integer | | Mã ngành phiên bản 2 |
| 6 | industry | varchar(256) | | Lĩnh vực cổ phiếu |
| 7 | industry_en | varchar(256) | | Lĩnh vực cổ phiếu (tiếng anh) |
| 8 | established_year | integer | | năm lên sàn |
| 9 | company_type | varchar(2) | | Loại công ty |

5.1.2. listing_companies

| STT | Tên trường | Loại Dữ liệu | Ràng buộc | Mô tả |
|---|---|---|---|---|
| 1 | ticker | varchar(8) | PK | Mã cổ phiếu (chỉ gồm những mã còn được niêm yết) |
| 2 | no_employees | integer | | số lượng nhân viên |
| 3 | no_shareholders | integer | | số lượng cổ đông |
| 4 | foreign_percent | double | | Tỉ lệ sở hữu của nước ngoài |
| 5 | website | varchar(256) | | website công ty |
| 6 | stock_rating | double | | mức độ tăng trưởng cổ phiếu |
| 7 | delta_in_week | double | | Giá trị Delta tuần |
| 8 | delta_in_month | double | | Giá trị Delta tháng |
| 9 | delta_in_year | double | | Giá trị Delta năm |
| 10 | outstanding_share | double | | cổ phiếu đang lưu hành |
| 11 | issue_share | double | | cổ phiếu đã phát hành |

5.1.3. stock_history

| STT | Tên trường | Loại Dữ liệu | Ràng buộc | Mô tả |
|---|---|---|---|---|
| 1 | ticker | varchar(3) | PK, FK | Mã cổ phiếu |
| 2 | time_stamp | timestamp | PK | Ngày giao dịch |
| 3 | open | double | | Giá cổ phiếu đầu phiên |
| 4 | high | double | | Giá cao nhất trong phiên |
| 5 | low | double | | Giá thấp nhất trong phiên |
| 7 | close | double | | GIá cuối phiên |

| | | | | |
|---|---|---|---|---|
| 8 | volume | integer | | Số lượng cổ phiếu giao dịch |

5.1.4. cash_flow

| STT | Tên trường | Loại Dữ liệu | Ràng buộc | Mô tả |
|---|---|---|---|---|
| 1 | ticker | varchar(3) | PK, FK | Mã cổ phiếu |
| 2 | year | integer | PK | Năm |
| 3 | quarter | integer | PK | Quý |
| 4 | invest_cost | integer | | Chi phí đầu tư |
| 5 | from_invest | integer | | Dòng tiền đầu tư |
| 6 | from_financial | integer | | Dòng tiền từ hoạt động tài chính |
| 7 | from_sale | integer | | Dòng tiền từ hoạt động kinh doanh |
| 8 | free_cash_flow | double | | Dòng tiền tự do |

5.1.5. general_rating

| STT | Tên trường | Loại Dữ liệu | Ràng buộc | Mô tả |
|---|---|---|---|---|
| 1 | ticker | varchar(3) | PK, FK | Mã cổ phiếu |
| 2 | year | integer | PK | Năm |
| 3 | quarter | integer | PK | Quý |
| 4 | stock_rating | double | | Điểm đánh giá cổ phiếu |
| 5 | valuation | double | | Điểm đánh giá định giá |
| 6 | financial_health | double | | Điểm đánh giá sức khỏe tài chính |
| 7 | business_model | double | | Điểm đánh giá mô hình kinh doanh |
| 8 | business_operation | double | | Điểm đánh giá hiệu quả hoạt động |

| | | | | |
|---|---|---|---|---|
| 9 | rs_rating | double | | Sức mạnh giá (Khả năng tăng giá của cổ phiếu |
| 10 | ta_score | double | | Điểm số phân tích kỹ thuật dựa trên các tín hiệu kỹ thuật |
| 11 | highest_price | double | | Giá cổ phiếu cao nhất trong 1 năm qua |
| 12 | lowest_price | double | | Giá cổ phiếu thấp nhất trong 1 năm qua |
| 13 | price_change3m | double | | Thay đối giá cổ phiếu trong 3 tháng qua |
| 14 | price_change1y | double | | Thay đối giá cổ phiếu trong 1 năm qua |
| 15 | beta | double | | Hệ số đo lường mức độ biến động của giá cổ phiếu so với giá trị danh mục toàn thị trường |
| 16 | alpha | double | | Tỉ suất sinh lời cổ phiếu không phụ thuộc vào rủi ro hệ thống |
| 17 | | | | |

### 5.1.6. business_model_rating

| **STT** | **Tên trường** | **Loại Dữ liệu** | **Ràng buộc** | **Mô tả** |
|---|---|---|---|---|
| 1 | ticker | varchar(3) | PK, FK | Mã cổ phiếu |
| 2 | year | integer | PK | Năm |
| 3 | quarter | integer | PK | Quý |
| 4 | business_model | integer | | Điểm đánh giá mô hình kinh doanh |
| 5 | business_efficiency | integer | | Điểm đánh giá hiệu quả hoạt động kinh doanh |
| 6 | asset_quality | double | | Điểm đánh giá chất lượng tài sản |
| 7 | cash_flow_quality | double | | Điểm đánh giá chất lượng dòng tiền |
| 8 | bom | double | | Điểm đánh giá chủ sở hữu, ban lãnh đạo và hội đồng quản trị |

| | | | | |
|---|---|---|---|---|
| 9 | business_administration | double | | Điểm đánh giá quản trị doanh nghiệp |
| 10 | product_service | double | | Điểm đánh giá sản phẩm và dịch vụ |
| 11 | business_advantage | double | | Điểm đánh mức độ thuận lợi trong SXKD |
| 12 | company_position | double | | Điểm đánh giá vị thế công ty trong ngành |
| 13 | industry | double | | Ngành mà công ty đang hoạt động |
| 14 | operation_risk | double | | Điểm đánh giá rủi ro chung về hoạt động |

### 5.1.7. business_operation_rating

| **STT** | **Tên trường** | **Loại Dữ liệu** | **Ràng buộc** | **Mô tả** |
|---|---|---|---|---|
| 1 | ticker | varchar(3) | PK, FK | Mã cổ phiếu |
| 2 | industry_en | double | | Ngành mà công ty đang hoạt động |
| 3 | year | integer | PK | Năm |
| 4 | quarter | integer | PK | Quý |
| 5 | loan_growth | double | | Mức tăng trong tổng các khoản vay của Công ty trừ đi dự phòng rủi ro cho vay trong 1 năm |
| 6 | deposit_growth | double | | Mức tăng tiền gửi vào công ty trong 1 năm |
| 7 | net_interest_income_growth | double | | Tăng trưởng chênh lệch giữa thu nhập lãi mà ngân hàng kiếm được từ hoạt động cho vay và tiền lãi mà ngân hàng trả cho người gửi tiền |
| 8 | net_interest_margin | double | | Chênh lệch giữa thu nhập từ lãi và số tiền lãi trả cho người cho vay, so với số tài sản công ty |
| 9 | cost_to_income | double | | Chi phí vận hành / Doanh thu vận hành |
| 10 | net_income_to_i | double | | Thu nhập ròng / I |

| | | | | |
|---|---|---|---|---|
| 11 | business_operation | double | | Điểm đánh giá hiệu quả hoạt động công ty |
| 12 | avg_ro_e | double | | Trung bình ROE |
| 13 | avg_ro_a | double | | Trung bình ROA |
| 14 | last5years_net_profit_growth | double | | Tăng trưởng LN ròng 5 năm |
| 15 | last5years_revenue_growth | double | | Tăng trưởng Doanh thu 5 năm |
| 16 | last5years_operating_profit_growth | double | | Tăng trưởng LN hoạt động 5 năm |
| 17 | last5years_eb_it_da_growth | double | | Tăng trưởng EBITDA 5 năm |
| 18 | last5years_fc_ff_growth | double | | Tăng trưởng FCFF 5 năm |
| 19 | last_year_gross_profit_margin | double | | Biên LN gộp năm trước |
| 20 | last_year_operating_profit_margin | double | | Biên LN hoạt động năm trước |
| 21 | last_year_net_profit_margin | double | | Biên LN ròng năm trước |
| 22 | t_oi_growth | double | | Tăng trưởng T/OI |

### 5.1.8. financial_health_rating

| **STT** | **Tên trường** | **Loại Dữ liệu** | **Ràng buộc** | **Mô tả** |
|---|---|---|---|---|
| 1 | ticker | varchar(3) | PK, FK | Mã cổ phiếu |
| 2 | industry_en | double | | Ngành mà công ty đang hoạt động |
| 3 | year | integer | PK | Năm |
| 4 | quarter | integer | PK | Quý |
| 5 | loan_deposit | double | | Dư nợ/Huy động |
| 6 | bad_loan_gross_loan | double | | Nợ xấu/Tổng dư nợ |

| | | | | |
|---|---|---|---|---|
| 7 | bad_loan_asset | double | | Nợ xấu/Tổng tài sản |
| 8 | provision_bad_loan | double | | Dự phòng/Nợ xấu |
| 9 | financial_health | double | | Điểm đánh giá sức khỏe tài chính |
| 10 | net_debt_equity | double | | Nợ ròng/VCSH |
| 11 | current_ratio | double | | Hệ số thanh toán lãi vay |
| 12 | interest_coverage | double | | Khả năng thanh toán hiện hành |
| 13 | net_debt_eb_it_da | double | | Nợ ròng / EBITDA |

5.1.9. valuation_rating

| **STT** | **Tên trường** | **Loại Dữ liệu** | **Ràng buộc** | **Mô tả** |
|---|---|---|---|---|
| 1 | ticker | varchar(3) | PK, FK | Mã cổ phiếu |
| 2 | industry_en | double | | Ngành mà công ty đang hoạt động |
| 3 | year | integer | PK | Năm |
| 4 | quarter | integer | PK | Quý |
| 5 | valuation | double | | Điểm đánh giá định giá cổ phiếu |
| 6 | pe | double | | Giá thị trường của cổ phiếu / Giá trị sổ sách của cổ phiếu |
| 7 | pb | double | | Giá thị trường của cổ phiếu / Lợi nhuận ròng doanh nghiệp |
| 8 | ps | double | | Giá thị trường của cổ phiếu / Doanh thu thuần trên 1 cổ phiếu |
| 9 | evebitda | double | | EV / EBITDA |
| 10 | dividend_rate | double | | Tỷ suất cổ tức = Giá trị cổ tức trên một cổ phiếu / Giá trị thị trường của cổ phiếu |

5.1.10. industry_financial_health

| STT | Tên trường | Loại Dữ liệu | Ràng buộc | Mô tả |
|---|---|---|---|---|
| 1 | ticker | varchar(3) | PK, FK | Mã cổ phiếu |
| 2 | industry_en | double | | Ngành mà công ty đang hoạt động |
| 3 | year | integer | PK | Năm |
| 4 | quarter | integer | PK | Quý |
| 5 | loan_deposit | double | | Dư nợ/Huy động |
| 6 | bad_loan_gross_loan | double | | Nợ xấu/Tổng dư nợ |
| 7 | bad_loan_asset | double | | Nợ xấu/Tổng tài sản |
| 8 | provision_bad_loan | double | | Dự phòng/Nợ xấu |
| 9 | financial_health | double | | Điểm đánh giá sức khỏe tài chính |
| 10 | net_debt_equity | double | | Nợ ròng/VCSH |
| 11 | current_ratio | double | | Hệ số thanh toán lãi vay |
| 12 | interest_coverage | double | | Khả năng thanh toán hiện hành |
| 13 | net_debt_eb_it_da | double | | Nợ ròng / EBITDA |

### 5.1.11. financial_ratio

| STT | Tên trường | Loại Dữ liệu | Ràng buộc | Mô tả |
|---|---|---|---|---|
| 1 | ticker | varchar(3) | PK, FK | Mã cổ phiếu |
| 2 | year | integer | PK | Năm |
| 3 | quarter | integer | PK | Quý |
| 4 | price_to_earning | double | | Giá trị thường một cổ phiếu (P) /EPS |
| 5 | price_to_book | double | | Giá trị thưởng một cổ phiếu (P) /Giá trị sổ sách một cổ phiếu (B) |

| | | | | |
|---|---|---|---|---|
| 6 | value_before_ebitda | double | | Lợi nhuận trước lãi vay và thuế |
| 7 | dividend | double | | Tỷ lệ lợi tức thuần được công ty chi trả cho các cổ đông. |
| 8 | roe | double | | Tỷ suất lợi nhuận trên vốn |
| 9 | roa | double | | Tỷ suất lợi nhuận trên tài sản |
| 10 | days_receivable | double | | Số ngày phải thu |
| 11 | days_inventory | double | | Số ngày tồn kho |
| 12 | days_payable | double | | Số ngày phải trả |
| 13 | ebit_on_interest | double | | Lợi nhật hoạt động (EBIT) /Lãi vay |
| 14 | earning_per_share | integer | | Lợi nhuận trên một cổ phiếu |
| 15 | book_value_per_share | double | | Vốn chủ sở hữu /Tổng số lượng cổ phiếu đang lưu hành |
| 16 | interest_margin | double | | Biên lãi thuần |
| 17 | non_interest_on_toi | double | | Thu nhập ngoài lãi /Thu nhập hoạt động (TOI) |
| 18 | bad_debt_percentage | double | | Tỷ lệ nợ xấu |
| 19 | provision_on_bad_debt | double | | Dự phòng /Nợ xấu |
| 20 | cost_of_financing | double | | Chi phí tài chính |
| 21 | equity_on_total_asset | double | | Vốn chủ sở hữu /Tổng tài sản |
| 22 | equity_on_loan | double | | Vốn chủ sở hữu /Cho vay |
| 23 | cost_to_income | double | | Tổng chi phí hoạt động /Tổng doanh thu hoạt động |
| 24 | equity_on_liability | double | | Vốn chủ sở hữu /Nợ phải trả |
| 25 | current_payment | double | | Thanh toán hiện hành |
| 26 | quick_payment | double | | Thanh toán nhanh |
| 27 | eps_change | double | | Mức thay đổi EPS so với kỳ trước |

| | | | | |
|---|---|---|---|---|
| 28 | ebitda_on_stock | double | | Lợi nhuận trước lãi vay, thuế mỗi cổ phiếu |
| 29 | gross_profit_margin | double | | Biên lợi nhuận gộp |
| 30 | operating_profit_margin | double | | Biên lợi nhuận hoạt động |
| 31 | post_tax_margin | double | | Biên lợi nhuận sau thuế |
| 32 | debt_on_equity | double | | Nợ vay /Vốn chủ sở hữu |
| 33 | debt_on_asset | double | | Nợ vay /Tài sản |
| 34 | debt_on_ebitda | double | | Nợ vay /EBITDA |
| 35 | short_on_long_debt | double | | Vay ngắn hạn /Vay dài hạn |
| 36 | asset_on_equity | double | | Tổng tài sản /Vốn chủ sở hữu |
| 37 | capital_balance | double | | Cân đối vốn |
| 38 | cash_on_equity | double | | Tiền mặt /Vốn chủ sở hữu |
| 39 | cash_on_capitalize | double | | Tiền mặt /Vốn hóa |
| 40 | cash_circulation | double | | Chu kỳ tiền mặt |
| 41 | revenue_on_work_capital | double | | Doanh thu thuần /Vốn lưu động bình quân |
| 42 | capex_on_fixed_asset | double | | Chi phí đầu tư /Tài sản cố định |
| 43 | revenue_on_asset | double | | Doanh thu thuần /Tài sản |
| 44 | post_tax_on_pre_tax | double | | Lợi nhuận sau thuế /Lợi nhuận trước thuế |
| 45 | ebit_on_revenue | double | | Lợi nhuận hoạt động /Doanh thu thuần |
| 46 | pre_tax_on_ebit | double | | Lợi nhuận trước thuế /Lợi nhuận hoạt động |
| 47 | pre_provision_on_toi | double | | Lợi nhuận trước dự phòng /TOI |
| 48 | post_tax_on_toi | double | | Thu nhập ngoài lãi /TOI |
| 49 | loan_on_earn_asset | double | | Cho vay /Tài sản sinh lãi |

| | | | | |
|---|---|---|---|---|
| 50 | loan_on_asset | double | | Cho vay /Tổng tài sản |
| 51 | loan_on_deposit | double | | Cho vay /Tiền gửi khách hàng |
| 52 | deposit_on_earn_asset | double | | Tiền gửi khách hàng /Tài sản sinh lãi |
| 53 | bad_debt_on_asset | double | | Nợ xấu /Tổng tài sản |
| 54 | liquidity_on_liability | double | | Tài sản thanh khoản /Nợ phải trả |
| 55 | payable_on_equity | double | | Nợ phải trả /Vốn chủ sở hữu |
| 56 | cancel_debt | double | | Tỷ lệ xóa nợ |
| 57 | ebitda_on_stock_change | double | | Mức thay đổi EBITDA/cp so với kỳ trước |
| 58 | book_value_per_share_change | double | | Mức thay đổi BVPS so với kỳ trước |
| 59 | credit_growth | double | | Tỷ lệ tăng trưởng tín dụng |

### 5.1.12. balance_sheet

| **STT** | **Tên trường** | **Loại Dữ liệu** | **Ràng buộc** | **Mô tả** |
|---|---|---|---|---|
| 1 | ticker | varchar(3) | PK, FK | Mã cổ phiếu |
| 2 | year | integer | PK | Năm |
| 3 | quarter | integer | PK | Quý |
| 4 | shortAsset | double | | Tài sản ngắn hạn |
| 5 | cash | integer | | Tiền các loại |
| 6 | shortInvest | double | | Tài sản đầu tư ngắn hạn |
| 7 | shortReceivable | double | | Các khoản thu ngắn hạn |
| 8 | inventory | double | | Hàng tồn kho |
| 9 | longAsset | double | | Tài sản dài hạn |
| 10 | fixedAsset | integer | | Tài sản cố định |

| | | | | |
|---|---|---|---|---|
| 11 | asset | integer | | Tài sản |
| 12 | debt | integer | | Tổng nợ |
| 13 | shortDebt | double | | Nợ ngắn hạn |
| 14 | longDebt | double | | Nợ dài hạn |
| 15 | equity | integer | | Vốn chủ sở hữu |
| 16 | capital | integer | | Vốn điều lệ |
| 17 | centralBankDeposit | double | | Tiền gửi tại NHNN |
| 18 | otherBankDeposit | double | | Tiền gửi TCTD khác |
| 19 | otherBankLoan | double | | Cho vay TCTD khác |
| 20 | stockInvest | double | | Chứng khoán đầu tư |
| 21 | customerLoan | double | | Cho vay khách hàng |
| 22 | badLoan | double | | Nợ xấu |
| 23 | provision | double | | Dự phòng cho vay KH |
| 24 | netCustomerLoan | double | | Cho vay KH ròng |
| 25 | otherAsset | double | | Tài sản khác |
| 26 | otherBankCredit | double | | Tiền gửi của TCTD |
| 27 | oweOtherBank | double | | Vay các TCTD |
| 28 | oweCentralBank | double | | Nợ CP & NHNN |
| 29 | valuablePaper | double | | PH giấy tờ có giá |
| 30 | payableInterest | double | | Lãi & phí phải trả |
| 31 | receivableInterest | double | | Lãi phải thu |
| 32 | deposit | double | | Tiền gửi của TCTD |
| 33 | otherDebt | integer | | Nợ phải trả khác |
| 34 | fund | double | | Quỹ của TCTD |

| | | | | |
|---|---|---|---|---|
| 35 | unDistributedIncome | double | | Lợi nhuận chưa PP |
| 36 | minorShareHolderProfit | double | | Lợi ích của cổ đông thiểu số |
| 37 | payable | integer | | Các khoản phải trả |

### 5.1.13. stock_intraday_transaction

| **STT** | **Tên trường** | **Loại Dữ liệu** | **Ràng buộc** | **Mô tả** |
|---|---|---|---|---|
| 1 | id | serial | PK | id của giao dịch |
| 2 | ticker | varchar(3) | FK | Mã cổ phiếu |
| 3 | price | double | | Giá cổ phiếu được đặt trong lệnh |
| 4 | volume | integer | | Số lượng cổ phiếu được đặt trong lệnh |
| 5 | cp | double | | |
| 6 | rcp | double | | |
| 7 | a | varchar(2) | | Hành động của giao dịch<br>(“BU”: Buy Up, “SD”: Sell Down) |
| 8 | ba | double | | Tổng số cổ phiếu được mua vào từ đầu phiên tới thời điểm của giao dịch |
| 9 | sa | double | | Tổng số cổ phiếu được bán ra từ đầu phiên trong lịch sử tới thời điểm của giao dịch |
| 10 | hl | boolean | | |
| 11 | pcp | double | | Số lượng cổ phiếu PCP<br>(Participating Convertible Preferred) giao dịch |
| 12 | timeStamp | timestamp | | Ngày giao dịch |

### 5.1.14. income_statement

| **STT** | **Tên trường** | **Loại Dữ liệu** | **Ràng buộc** | **Mô tả** |
|---|---|---|---|---|

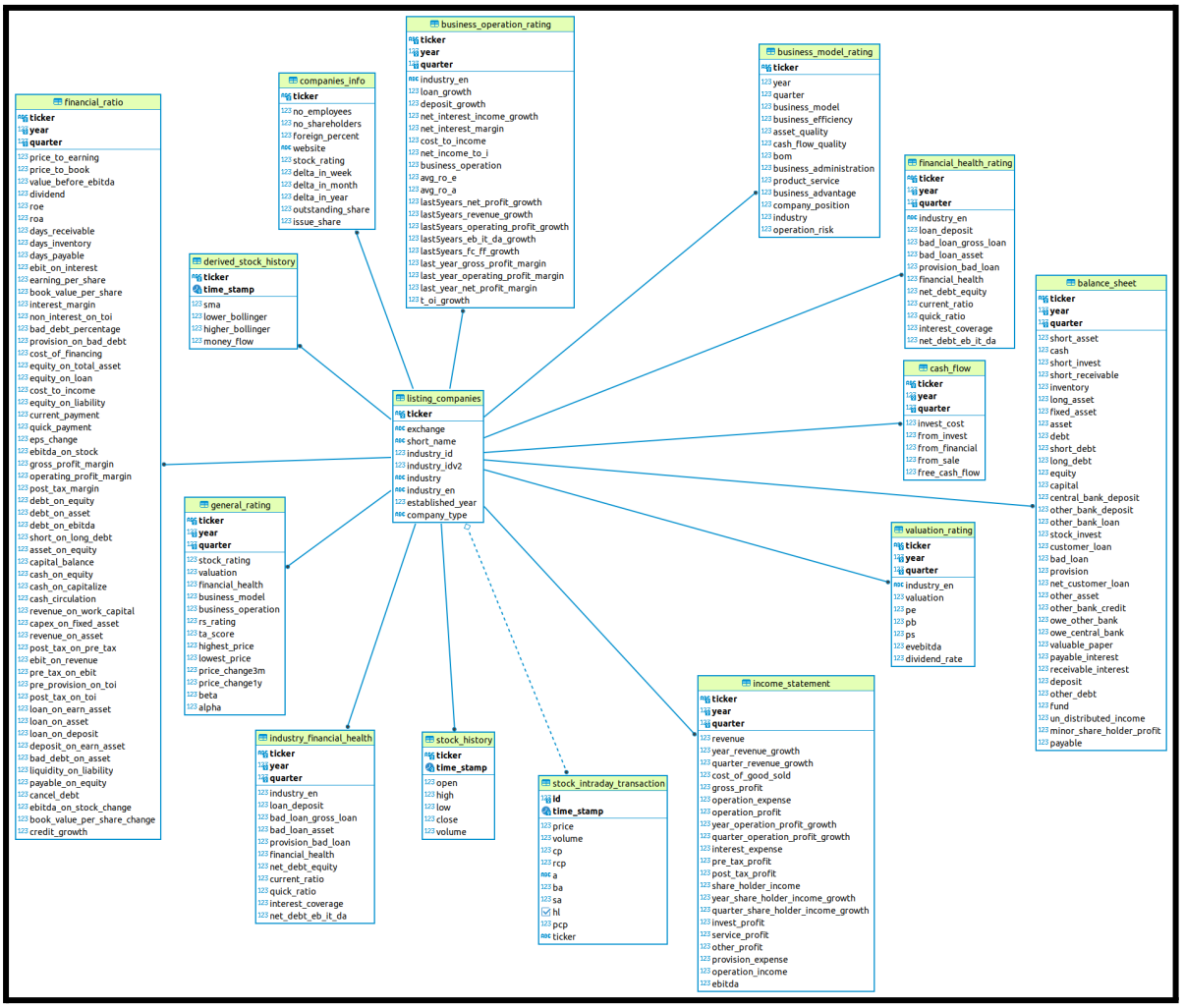
| | | | | |
|---|---|---|---|---|
| 1 | ticker | varchar(3) | PK, FK | Mã cổ phiếu |
| 2 | year | integer | PK | Năm |
| 3 | quarter | integer | PK | Quý |
| 4 | revenue | double | | Doanh thu |
| 5 | yearRevenueGrowth | double | | Tăng trưởng doanh thu theo năm |
| 6 | quarterRevenueGrowth | double | | Tăng trưởng doanh thu theo quý |
| 7 | costOfGoodSold | double | | Giá vốn hàng bán |
| 8 | grossProfit | double | | Lợi nhuận gộp |
| 9 | operationExpense | integer | | Chi phí hoạt động |
| 10 | operationProfit | integer | | Lợi nhuận hoạt động |
| 11 | yearOperationProfitGrowth | double | | Tăng trưởng lợi nhuận hoạt động theo năm |
| 12 | quarterOperationProfitGrowth | double | | Tăng trưởng lợi nhuận hoạt động theo quý |
| 13 | interestExpense | double | | Chi phí lãi vay |
| 14 | preTaxProfit | integer | | Lợi nhuận trước thuế |
| 15 | postTaxProfit | integer | | Lợi nhuận sau thuế |
| 16 | shareHolderIncome | integer | | Thu nhập của chủ sở hữu |
| 17 | yearShareHolderIncomeGrowth | double | | Thu nhập của chủ sở hữu theo năm |
| 18 | quarterShareHolderIncomeGrowth | double | | Thu nhập của chủ sở hữu theo quý |
| 19 | investProfit | double | | Lợi nhuận đầu tư |
| 20 | serviceProfit | double | | Lợi nhuận dịch vụ |
| 21 | otherProfit | double | | Các lợi nhuận khác |
| 22 | provisionExpense | double | | Chi phí dự phòng |
| 23 | operationIncome | double | | Thu nhập hoạt động |

| | | | | |
|---|---|---|---|---|
| 24 | ebitda | double | | Thu nhập trước thuế |

5.1.15. derived_stock_history

| **STT** | **Tên trường** | **Loại Dữ liệu** | **Ràng buộc** | **Mô tả** |
|---|---|---|---|---|
| 1 | ticker | varchar(3) | PK, FK | Mã cổ phiếu |
| 2 | time_stamp | double | PK | Ngày giao dịch |
| 3 | sma | double | | Đường trung bình động đơn giản, tính bằng trung bình cộng các mức giá đóng cửa trong một khoảng thời gian giao dịch nhất định |
| 4 | lower_bollinger | double | | Dải dưới Boillinger |
| 5 | higher_bollinger | double | | Dải trên Boillinger |
| 6 | money_flow | double | | Dòng tiền |

## 5.2 Mối quan hệ giữa các bảng dữ liệu




## 5.3. Mô tả cách tính (Với các dữ liệu tính toán)

### 5.3.1. Simple Moving Average

Simple Moving Average (SMA) được tính là giá trị trung bình của $n$ giá trị gần nhất của 1 giá trị:

$$SMA_i = \frac{\sum_{k=i-n}^{i} x_k}{n}$$

### 5.3.2. Bollinger Band

Dải bollinger (Bollinger Band) là một công cụ phân tích kỹ thuật được phát minh bởi John Bollinger vào những năm 1980. Phát triển từ khái niệm dải giao dịch (trading

band), Dải bollinger có thể được sử dụng để đo lường mức "cao" hoặc "thấp" của giá so với các giao dịch trước đó.
Để tính ta thực hiện:

1. Tính SMA trong $n$-ngày ($n = 30$) của giá trị
2. Tính độ lệch chuẩn của giá trị với SMA đã tính được:

$$\sigma_i = \sqrt[2]{\frac{\sum_{k=i-n}^{i} \left(x_k - SMA_i\right)^2}{n}}$$

3. Cuối cùng, dải Bollinger được tính bằng

$$BBands_i = x_i \pm \sigma_i * d$$

Với: $d$: hằng số ($d = 1$)

### 5.3.3. Chaikin Money Flow

Chỉ số dòng tiền (MFI) là một chỉ số được tính toán trong khoảng thời gian *n*-ngày, nằm trong khoảng từ 0 đến 100, cho biết dòng tiền vào những ngày tăng giá theo tỷ lệ phần trăm của tổng số ngày tăng và giảm.
Để tính ta thực hiện:

4. Tính Giá thông thường (typical price):

$$typical\ price = \frac{high + low + close}{3}$$

5. Tính Dòng tiền (Money flow):

$$money\ flow = typical\ price \times volume$$

6. Sau đó, tổng dòng tiền trong *n*-ngày nhất định sẽ được tính. Dòng tiền dương (Positive money flow) là tổng của những ngày mà Giá thông thường cao hơn giá thông thường của ngày hôm trước và Dòng tiền âm (Negative money flow) khi thấp hơn. Nếu giá thông thường không đổi thì ngày đó sẽ bỏ qua.
7. Tính tỉ lệ dòng tiền (money ratio):

$$money\ ratio = \frac{positive money\ flow}{negative money\ flow}$$

8. Cuối cùng ta sẽ tính chỉ số dòng tiền (MFI):

$$MFI = 100 - \frac{100}{1 + money\ ratio}$$

## 5.4 Thu thập và lưu trữ dữ liệu về thông tin các mã cổ phiếu từ Web vào Database

### 5.4.1 Cài đặt

Để kết nối vào Database, nhóm sử dụng Hostname default của Docker để chỉ máy Host: host.internal.docker

Nhóm ban đầu có ý tưởng là sẽ sử dụng deployment Airflow để gửi job sang cho deployment Spark thực hiện, tuy nhiên khi sử dụng SparkSubmitOperator để submit job dưới dạng Client mode, Worker của Airflow sẽ đóng vai trò là Driver, chạy các lệnh trong file mã nguồn của Job, tại bước đọc và ghi vào Database dữ liệu, các Worker của deployment Spark sẽ không thể phân giải DNS của tên miền mà Driver gửi sang. Điều này khiến nhóm chuyển hướng sang thực hiện các cách tiếp theo.

Vì vậy nhóm có 3 giải pháp để có thể thực hiện nhiệm vụ của hệ thống bằng Spark:

1. Submit job dưới dạng Local, điều này sẽ khiến Worker thực hiện job local, số lượng tài nguyên của mỗi job có thể được assign trong file source code của từng DAG.
2. Sử dụng Ingress để phân giải DNS ở các pod Worker của Spark
3. Thiết lập cách để worker Airflow thực hiện lệnh submit Job trong pod Master của Spark

Tuy nhiên nhóm chỉ mới hoàn thành/thực hiện thành công giải pháp thứ nhất trong project này. Với việc phải chạy trên pod của Worker Airflow, người dùng sẽ không thể truy cập vào UI của Spark.

### 5.4.2 Thực hiện

Một số các hàm lấy thông tin từ API sẽ có tham số init, tham số này dùng để xác định là đây có phải là lần chạy đầu tiên/ lần chạy cập nhật không, nếu True thì API sẽ được gọi để lấy tất cả các thông tin, nếu False thì API được gọi chỉ lấy thông tin mới nhất.

#### 5.4.2.1 Lấy các mã cổ phiếu để lấy dữ liệu từ API

Hàm sẽ trả về 1 danh sách mã các cổ phiếu sẽ được sử dungj trong quá trình pull dữ liệu từ API.
Nếu init = True, ta sẽ thu thập thông tin từ danh sách các mã cổ phiếu hiện có trên trang TCBS. Nhóm có tham khảo hàm listing_companies() của thư viện vnstock để lấy ra danh sách các mã cổ phiếu có trên trang TCBS. Nếu False, ta sẽ chỉ lấy các mã cổ phiếu ở trong bảng **companies_info**, điều này giúp chúng ta loại bỏ các mã cổ phiếu không còn được niêm yết.

#### 5.4.2.2 Lấy lược đồ cơ sở dữ liệu

Đọc Schema từ cơ sở dữ liệu, dùng để chuẩn đổi kiểu khi thêm dữ liệu vào cơ sở dữ liệu theo đúng schema của bảng.
Ta thiết lập các cấu hình của cơ sở dữ liệu thông qua hàm option bao gồm:

- path đến JDBCdriver của hệ CSDL tương ứng
- tên của bảng
- địa chỉ và tên của CSDL thông qua giá trị url theo cú pháp quy định
- tên người dùng
- mật khẩu của CSDL

#### 5.4.2.3. Hàm ghi dữ liệu vào CSDL

Đẩy dữ liệu từ định dạng data frame vào trong CSDL với những thông số đã khai báo thông qua các giá trị giống như hàm đọc dữ liệu.

#### 5.4.2.4. Hàm UDF request API

Sẽ có hai hàm, loại thứ nhất dành cho những API call trả về kết quả phức tạp, các trường thông tin phải được định nghĩa trước, loại thứ hai sẽ được tự động định nghĩa theo cấu trúc của các bảng trong cơ sở dữ liệu.
Ta sẽ dùng lần lượt 2 hàm này để định nghĩa 1 Pyspark UDF (User-defined Function), cho phép Spark xử lý các API request một cách song song trên từng partition của DataFrame gồm URL của các request.

#### 5.4.2.5. Các hàm lấy dữ liệu

Các hàm đều sử dụng API User-defined Functions của Spark để phân phối các yêu cầu API về các Worker.

a. Hàm get_ratio()
Hàm lấy các báo cáo tài chính của một mã cổ phiếu.

Bao gồm các loại báo cáo sau:

- Báo cáo dòng tiền: Báo cáo lưu chuyển tiền tệ là báo cáo tài chính tổng hợp, phản ánh việc hình thành và sử dụng lượng tiền phát sinh trong kỳ báo cáo của doanh nghiệp. Thông tin về

lưu chuyển tiền tệ của doanh nghiệp cung cấp cho người sử dụng thông tin có cơ sở để đánh giá khả năng tạo ra các khoản tiền và việc sử dụng những khoản tiền đã tạo ra đó trong hoạt động sản xuất kinh doanh của doanh nghiệp.

- Báo cáo thu nhập: là báo cáo lời - lỗ ghi chi tiết những hoạt động tài chính của công ty trong một thời kỳ cụ thể, bao gồm lợi nhuận hoặc thua lỗ thuần trong một thời kỳ xem xét.
- Báo cáo bảng cân đối kế toán: Bảng cân đối kế toán là báo cáo tài chính tổng hợp, phản ánh tổng quát toàn bộ giá trị tài sản, nợ phải trả và nguồn vốn của doanh nghiệp tại một thời điểm nhất định. Căn cứ vào bảng cân đối kế toán có thể nhận xét, đánh giá khái quát tình hình tài chính của doanh nghiệp.
- Báo cáo hệ số tài chính: Cung cấp nhiều hệ số tài chính khác nhau liên quan đến mã cổ phiếu.

Để xác định loại báo cáo cần tải xuống, ta dùng tham số report_type. Ta thực hiện xóa bỏ dữ liệu trùng lặp, sau đấy sắp xếp dữ liệu theo thứ tự thời gian và đẩy vào cơ sở dữ liệu thông qua hàm ghi dữ liệu định nghĩ ở trên.

b. Hàm get_rating()

Ta sử dụng hàm này để lấy ra các chỉ số đánh giá tài chính đối với một mã cổ phiếu. Bao gồm các loại đánh giá sau:

- Đánh giá tổng quát
- Đánh giá mô hinh kinh doanh
- Đánh giá hoạt động kinh doanh
- Đánh giá sức khỏe tài chính
- Đánh giá giá trị

Sau khi thu được dữ liệu, ta tiến hành lược bỏ một số trường thông tin không có giá trị đến mã cổ phiếu trước khi lưu trữ vào trong cơ sở dữ liệu.

Hàm này sẽ ghi đè lên bảng tương ứng (xóa dữ liệu rồi ghi) vì thông tin có thể không được cập nhật ở thời gian chạy.

c. Hàm get_stock_history()

Ta dùng hàm này để lấy thông tin của các ngày giao dịch đối với từng mã cổ phiếu, từ đó cho ta thấy xu hướng của thị trong trong một khoảng thời gian. Thông tin bao gồm:

- Giá mở cửa
- Giá cao nhất
- Giá thấp nhất
- Giá đóng cửa
- …

Ta thay đổi định dạng của trường dữ liệu thời gian trước khi lưu trữ vào cơ sở dữ liệu.

d. Hàm get_intraday_transaction()

Ta sử dụng hàm này để lấy ra những giao dịch trong ngày của một mã cổ phiếu. Ở đây, ta cần xử lý hai trường hợp:

Một là ngày giao dịch trong tuần

Hai là ngày giao dịch cuối tuần

Với mỗi trường hợp, ta có api khác nhau để lấy thông tin liên quan đến trường hợp đấy.

Với dữ liệu thu thập được, ta tiến hành đổi tên trường dữ liệu. Và thay đổi dạng của trường dữ liệu thời gian.

e. Hàm get_listing_companies()

Ta sử dụng hàm này để lấy thêm các thông tin liên quan đến mỗi mã cổ phiếu mà ta đã thu được ở hàm get_tickers(). Những thông tin này bao gồm:

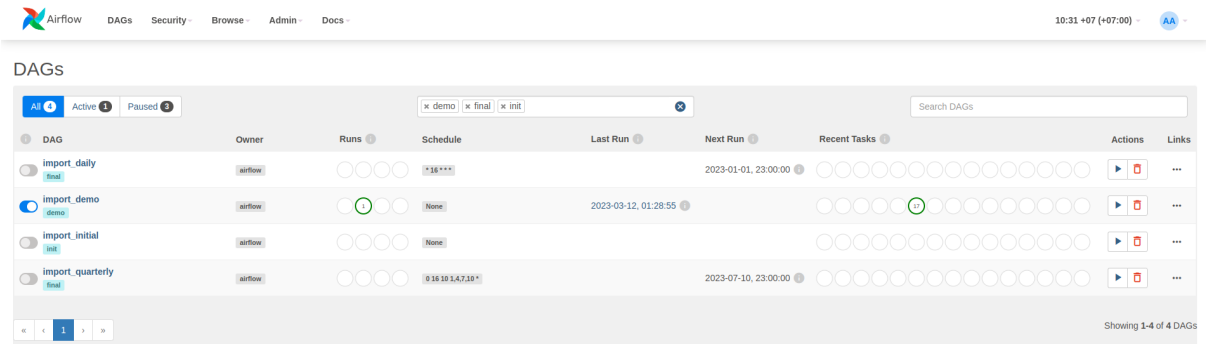
- Sàn giao dịch niêm yết
- Tên mã cổ phiếu
- Tên công ty
- Số lượng nhân viên
- …

Ta sẽ tách các thông tin không đổi để ghi vào bảng **listing_companies**, sau đó ghi đè những thông tin có thể thay đổi vào bảng **companies_info**.Bởi vì các bảng khác đều dựa vào PK ticker của **listing_companies**, việc sử dụng bảng **companies_info** sẽ giúp chung ta chỉ gửi yêu cầu về các mã cổ phiếu còn niêm yết, đồng thời vẫn có thể lưu trữ những thông tin ở các bảng khác về thông tin của các mã không còn niêm yết.

f. Hàm calculate_indicators()

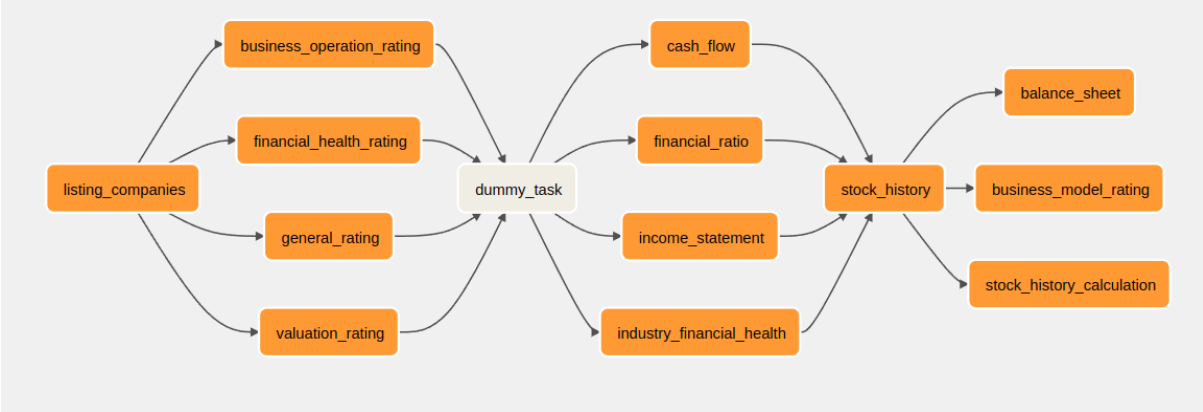
Ta sử dụng hàm này để tính toán 3 dữ liệu phái sinh được đề cập ở trên. Spark sẽ query Database để lấy dữ liệu 30 ngày gần nhất của giá cổ phiếu rồi thực hiện tính toán. Với các giá trị liên quan tới giá không được đề cập trong cách tính, Nhóm sử dụng giá đóng cửa để tính toán.

### 5.4.3 Các DAG của Airflow

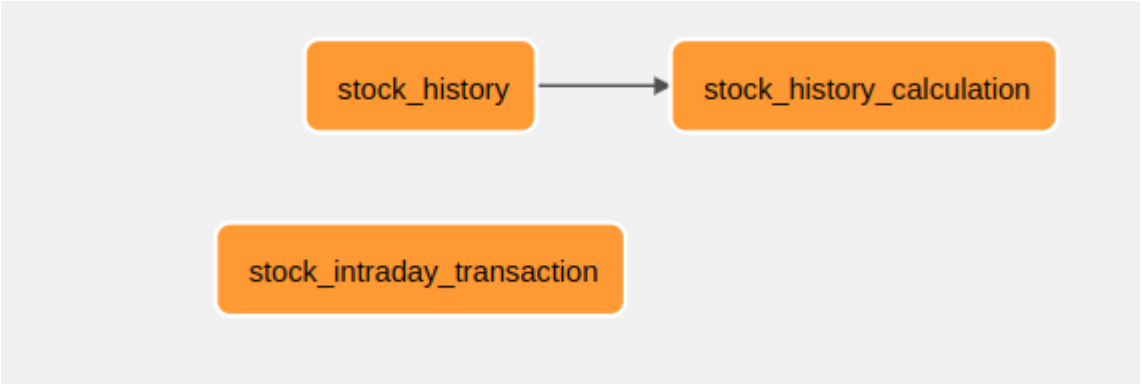



Trong Airflow sẽ có 4 DAG như sau:

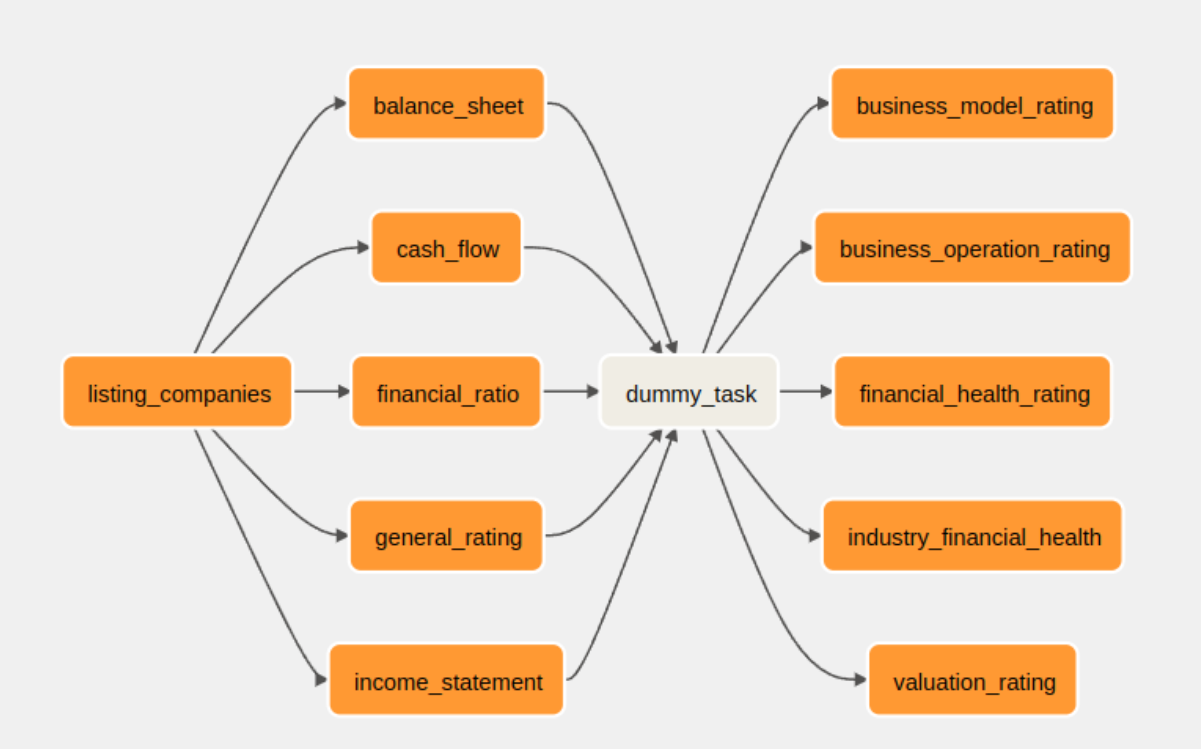
- import_initial
    - Pull tất cả các thông tin từ API từ trước tới thời điểm chạy (không gồm dữ liệu về giá cổ phiếu trong ngày chạy và các lệnh trong ngày)
    - Chạy 1 lần do người dùng khởi tạo



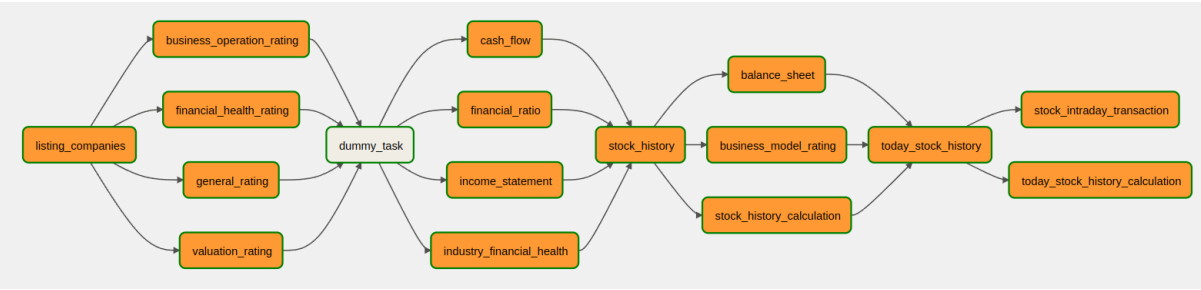

- import_daily
    - Pull thông tin về lịch sử các lệnh và thông tin về giá của các cổ phiếu trong ngày
    - Chạy vào 4H chiều mỗi ngày, sau khi các sàn đã kết thúc giao dịch




- import_quarterly
    - Pull tất cả các thông tin (trừ các báo cáo cuối năm)
        - Cập nhật các mã cổ phiếu mới niêm yết
    - Chạy vào 4H chiều ngày 10 hàng tháng 1, 4, 7, 10




- import_demo
    - Giống như import_init và import_daily
    - Pull tất cả các thông tin từ API từ trước tới thời điểm chạy (kể cả dữ liệu về giá cổ phiếu trong ngày chạy và các lệnh trong ngày) cho 3 mã cổ phiếu đầu tiên
    - Dùng để demo

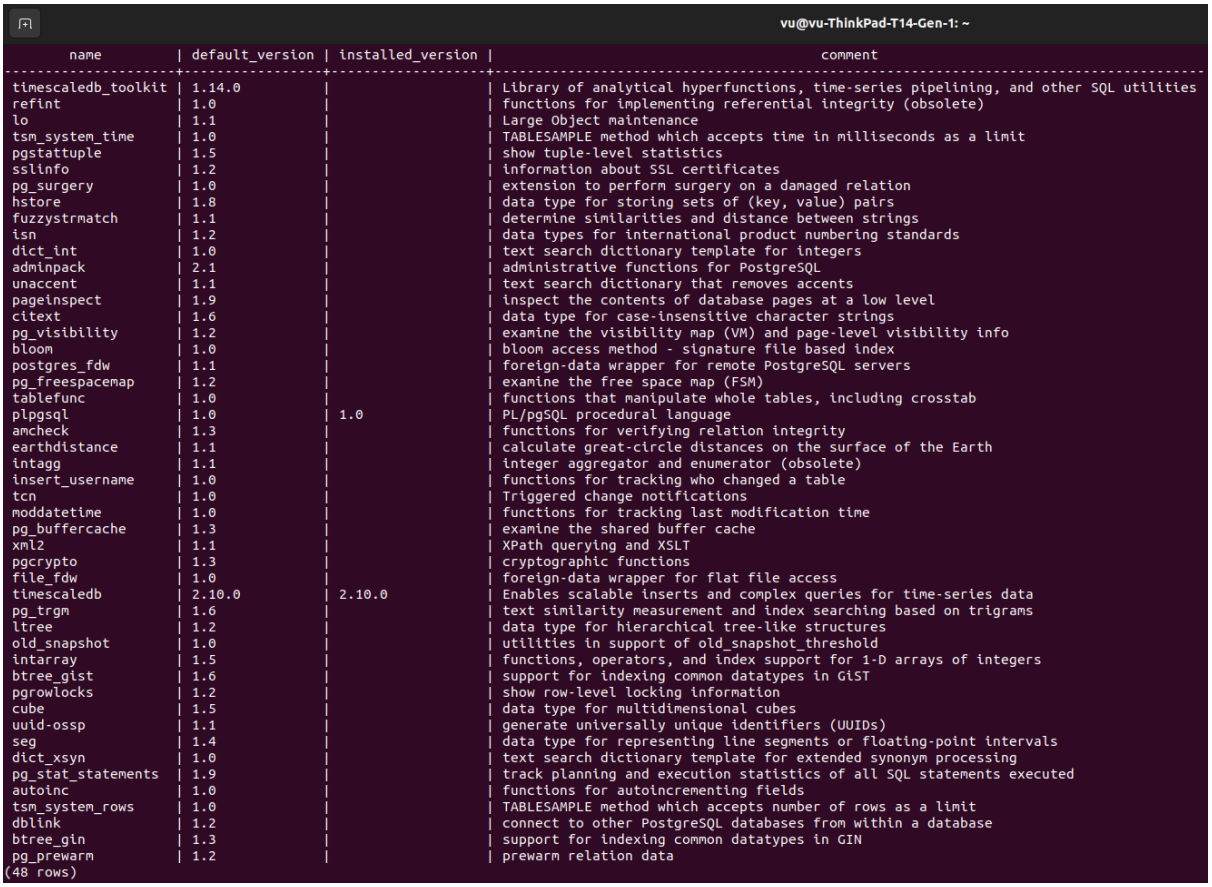



Lưu ý:

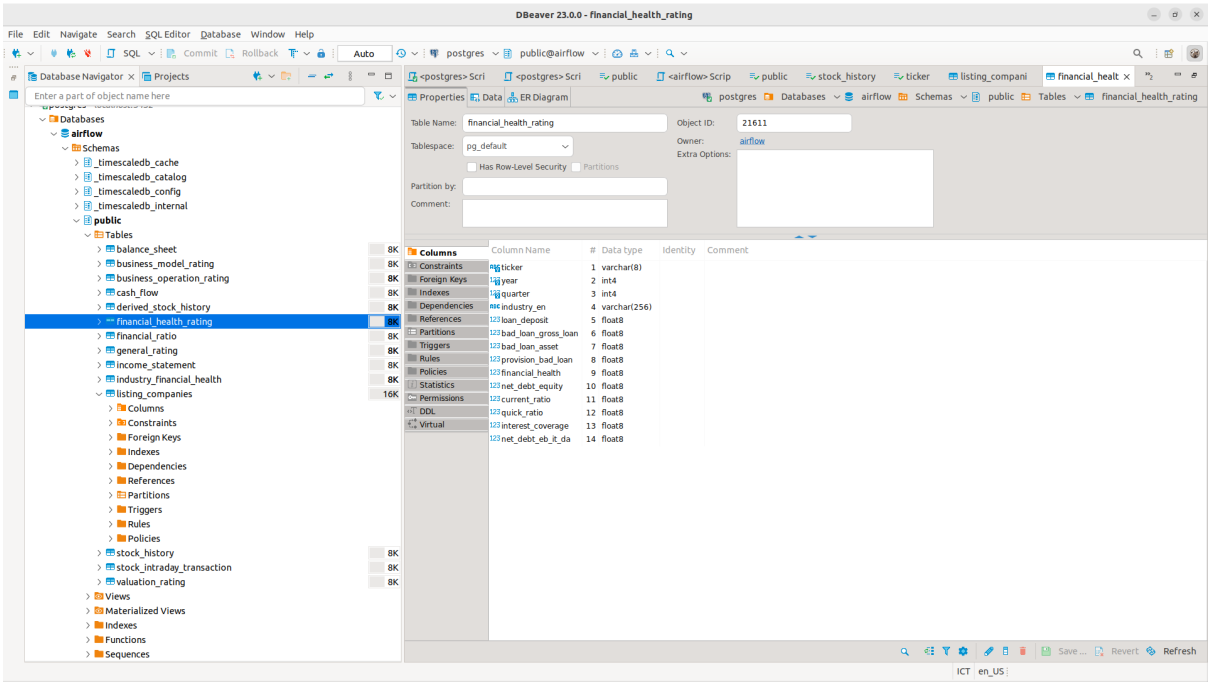
- Task dummy_task chỉ là 1 task để synchronize các task.
- Các Task được sắp xếp như trên chỉ để hạn chế việc Airflow sử dụng quá nhiều tài nguyên (trừ listing_companies phải chạy trước trong import_init và import_quarterly để có được những công ty đang niêm yết).
- Chỉ có task stock_history_calculation dùng để tính các dữ liệu phái sinh từ lịch sử giá cổ phiếu mới phụ thuộc vào task stock_history.
- Người dùng có thể sắp xếp lại các Task theo các ràng buộc trên
- Tất cả các task (trừ dummy_task đều sử dụng SparkSubmitOperator từ thư viện Provider của Airflow dành cho Spark.
    - Operator này gửi 1 lệnh spark-submit
    - Cho phép định nghĩa các params của lệnh spark-submit
- Ngoài ra người dùng cũng có thể định nghĩa số lượng Spark-Executors và số core mà mỗi task sẽ sử dụng ở trong file .py của các dag tương ứng

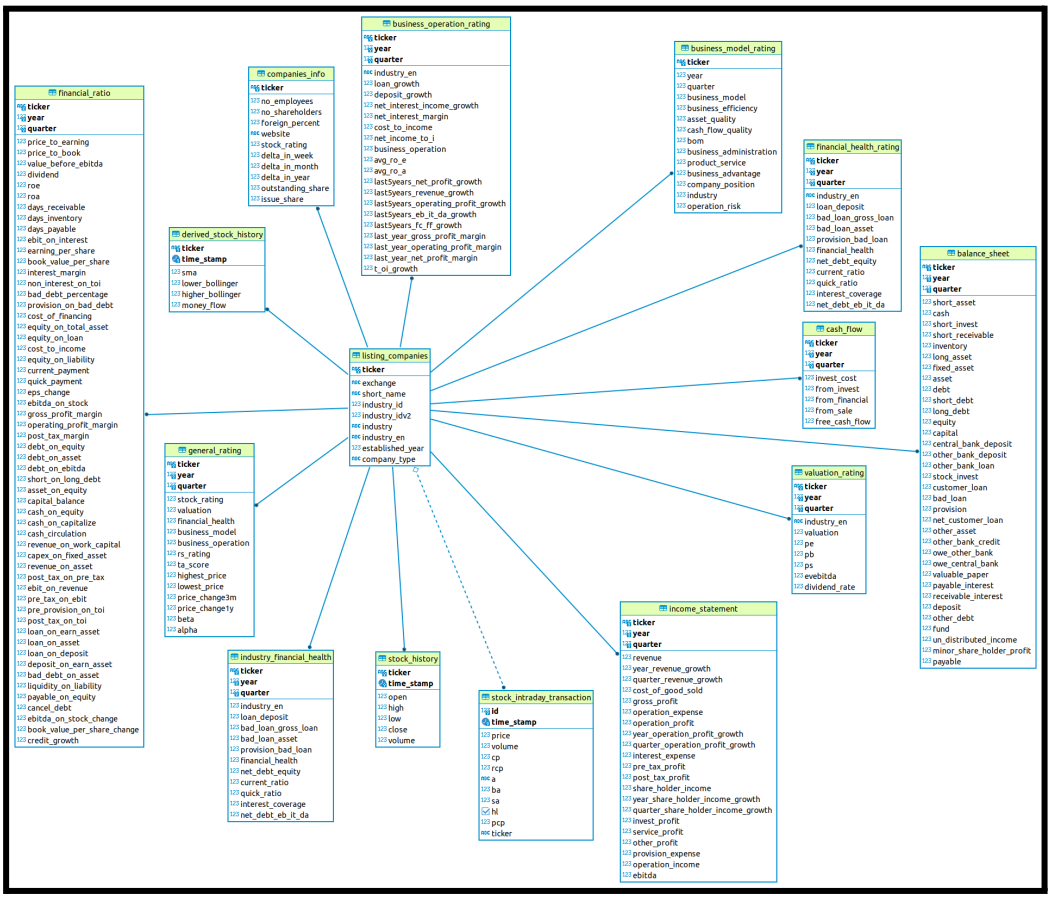
## 5.5 Screenshot từng thành phần trong sản phẩm

Database

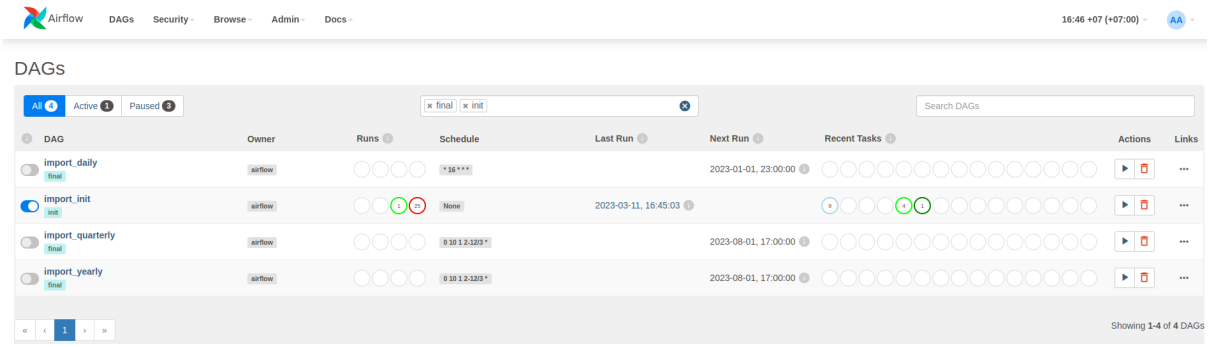


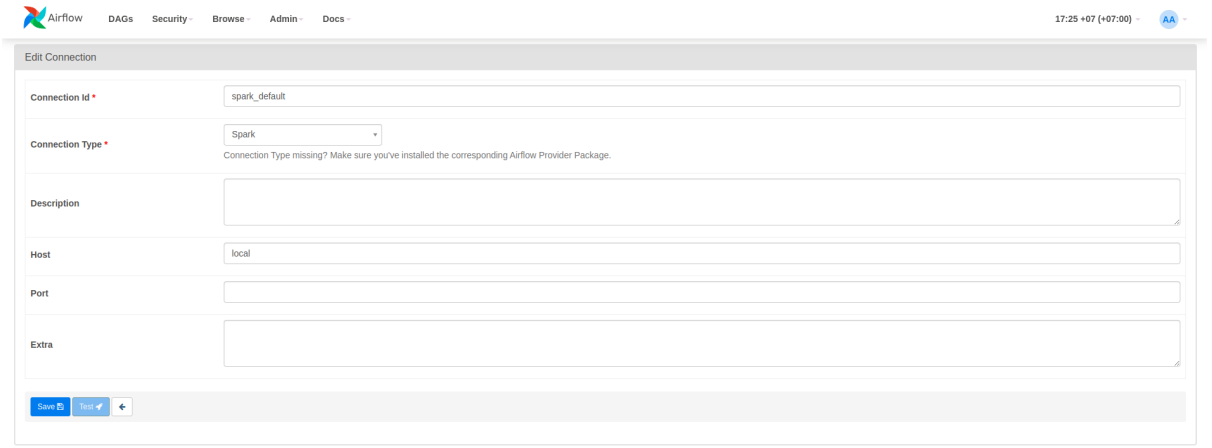

```
       name         | default_version | installed_version |                                                          comment
--------------------+-----------------+-------------------+-----------------------------------------------------------------------------------------------
 timescaledb_toolkit | 1.14.0          |                   | Library of analytical hyperfunctions, time-series pipelining, and other SQL utilities
 refint              | 1.0             |                   | functions for implementing referential integrity (obsolete)
 lo                  | 1.1             |                   | Large Object maintenance
 tsm_system_time     | 1.0             |                   | TABLESAMPLE method which accepts time in milliseconds as a limit
 pgstattuple         | 1.5             |                   | show tuple-level statistics
 sslinfo             | 1.2             |                   | information about SSL certificates
 pg_surgery          | 1.0             |                   | extension to perform surgery on a damaged relation
 hstore              | 1.8             |                   | data type for storing sets of (key, value) pairs
 fuzzystrmatch       | 1.1             |                   | determine similarities and distance between strings
 isn                 | 1.2             |                   | data types for international product numbering standards
 dict_int            | 1.0             |                   | text search dictionary template for integers
 adminpack           | 2.1             |                   | administrative functions for PostgreSQL
 unaccent            | 1.1             |                   | text search dictionary that removes accents
 pageinspect         | 1.9             |                   | inspect the contents of database pages at a low level
 citext              | 1.6             |                   | data type for case-insensitive character strings
 pg_visibility       | 1.2             |                   | examine the visibility map (VM) and page-level visibility info
 bloom               | 1.0             |                   | bloom access method - signature file based index
 postgres_fdw        | 1.1             |                   | foreign-data wrapper for remote PostgreSQL servers
 pg_freespacemap     | 1.2             |                   | examine the free space map (FSM)
 tablefunc           | 1.0             |                   | functions that manipulate whole tables, including crosstab
 plpgsql             | 1.0             | 1.0               | PL/pgSQL procedural language
 amcheck             | 1.3             |                   | functions for verifying relation integrity
 earthdistance       | 1.1             |                   | calculate great-circle distances on the surface of the Earth
 intagg              | 1.1             |                   | integer aggregator and enumerator (obsolete)
 insert_username     | 1.0             |                   | functions for tracking who changed a table
 tcn                 | 1.0             |                   | Triggered change notifications
 moddatetime         | 1.0             |                   | functions for tracking last modification time
 pg_buffercache      | 1.3             |                   | examine the shared buffer cache
 xml2                | 1.1             |                   | XPath querying and XSLT
 pgcrypto            | 1.3             |                   | cryptographic functions
 file_fdw            | 1.0             |                   | foreign-data wrapper for flat file access
 timescaledb         | 2.10.0          | 2.10.0            | Enables scalable inserts and complex queries for time-series data
 pg_trgm             | 1.6             |                   | text similarity measurement and index searching based on trigrams
 ltree               | 1.2             |                   | data type for hierarchical tree-like structures
 old_snapshot        | 1.0             |                   | utilities in support of old_snapshot_threshold
 intarray            | 1.5             |                   | functions, operators, and index support for 1-D arrays of integers
 btree_gist          | 1.6             |                   | support for indexing common datatypes in GiST
 pgrowlocks          | 1.2             |                   | show row-level locking information
 cube                | 1.5             |                   | data type for multidimensional cubes
 uuid-ossp           | 1.1             |                   | generate universally unique identifiers (UUIDs)
 seg                 | 1.4             |                   | data type for representing line segments or floating-point intervals
 dict_xsyn           | 1.0             |                   | text search dictionary template for extended synonym processing
 pg_stat_statements  | 1.9             |                   | track planning and execution statistics of all SQL statements executed
 autoinc             | 1.0             |                   | functions for autoincrementing fields
 tsm_system_rows     | 1.0             |                   | TABLESAMPLE method which accepts number of rows as a limit
 dblink              | 1.2             |                   | connect to other PostgreSQL databases from within a database
 btree_gin           | 1.3             |                   | support for indexing common datatypes in GIN
 pg_prewarm          | 1.2             |                   | prewarm relation data
(48 rows)
```


DBeaver 23.0.0 - financial_health_rating
File Edit Navigate Search SQL Editor Database Window Help
Database Navigator
Projects
Databases
airflow
Schemas
_timescaledb_cache
_timescaledb_catalog
_timescaledb_config
_timescaledb_internal
public
Tables
balance_sheet
business_model_rating
business_operation_rating
cash_flow
derived_stock_history
financial_health_rating
financial_ratio
general_rating
income_statement
industry_financial_health
listing_companies
Columns
Constraints
Foreign Keys
Indexes
Dependencies
References
Partitions
Triggers
Rules
Policies
stock_history
stock_intraday_transaction
valuation_rating
Views
Materialized Views
Indexes
Functions
Sequences
Properties
Data
ER Diagram
Table Name: financial_health_rating
Object ID: 21611
Tablespace: pg_default
Owner: airflow
Extra Options:
Has Row-Level Security
Partitions
Partition by:
Comment:
Column Name
Data type
Identity
Comment
ticker varchar(8)
year int4
quarter int4
industry_en varchar(256)
loan_deposit float8
bad_loan_gross_loan float8
bad_loan_asset float8
provision_bad_loan float8
financial_health float8
net_debt_equity float8
current_ratio float8
quick_ratio float8
interest_coverage float8
net_debt_eb_it_da float8
Save...
Revert
Refresh
ICT
en_US

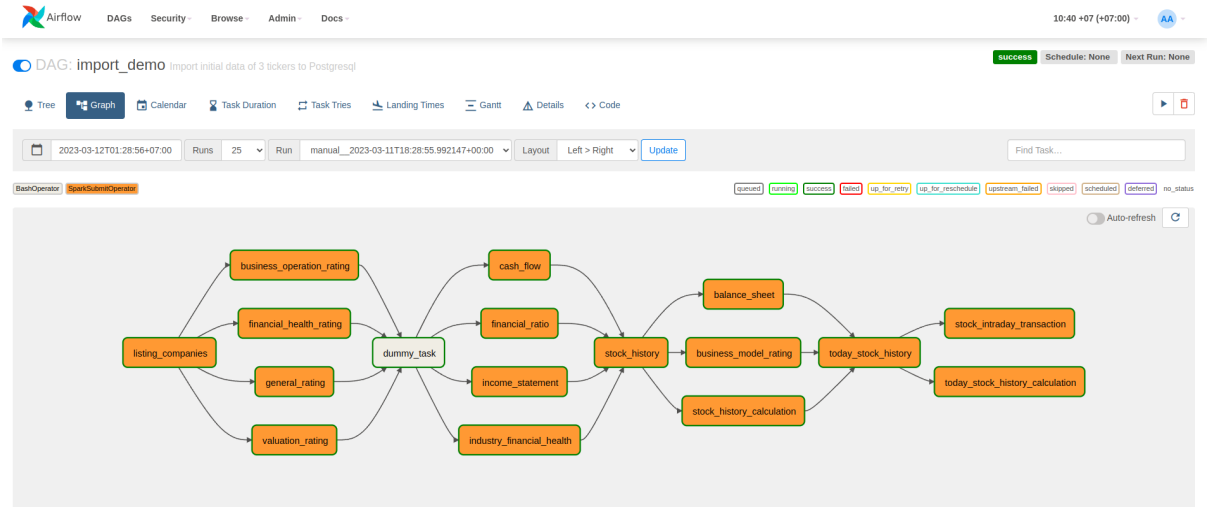

financial_ratio
companies_info
business_operation_rating
business_model_rating
financial_health_rating
derived_stock_history
listing_companies
cash_flow
balance_sheet
general_rating
valuation_rating
industry_financial_health
stock_history
stock_intraday_transaction
income_statement

Airflow UI



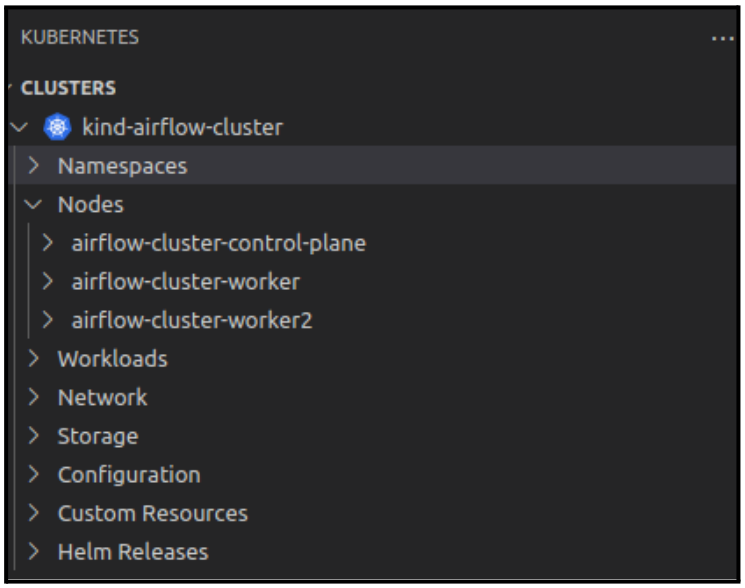




DAG graph




Spark UI


Kubernetes




# 6. Cải thiện

- Tiếp tục thực hiện các cách được đề cập ở phần 5.4.1
- Xây dựng DAG định kỳ để khảo sát dữ liệu thiếu hay sai và pull lại
- Tích hợp Web App vào k8s
- Thiết lập thêm các cách song song hóa việc pull dữ liệu bằng Spark

## 7. Phân chia công việc

- Thiết kế và cài đặt cơ sở dữ liệu
    - Phạm Đinh Gia Dũng: 80%
    - Phạm Cát Vũ: 20%
- Lập trình các hàm lấy dữ liệu
    - Phạm Đinh Gia Dũng: 50%
    - Phạm Cát Vũ: 50%
- Lập trình các hàm tính toán dữ liệu
    - Phạm Cát Vũ: 70%
    - Nguyễn Văn Hùng: 30%
- Thiết kế luồng công việc Airflow
    - Nguyễn Văn Hùng: 30%
    - Nguyễn Minh Mạnh: 30%
    - Ngô Hùng Mạnh: 40%
- Cài đặt triển khai trên Docker
    - Phạm Đinh Gia Dũng: 60%
    - Nguyễn Văn Hùng: 20%
    - Nguyễn Minh Mạnh: 20%
- Cài đặt triển khai trên Kubernetes
    - Phạm Cát Vũ: 70%
    - Ngô Hùng Mạnh: 30%
- Viết báo cáo
    - Cả nhóm thực hiện

# Tài liệu tham khảo


1. https://www.python.org/about/
2. https://www.geeksforgeeks.org/postgresql-system-architecture/
3. https://docs.timescale.com/getting-started/latest/
4. https://redbag.vn/blog/tcinvest-la-gi-chi-tiet-cach-doc-bang-gia-chung-khoan-tcbs?fbclid=IwAR0n6IfBIp4ozJOENUyXGpGHZzHXWfSdaCIxIqtwK2N-kkcBousnNKYT6WI#1644974426-h2-0
5. https://jdbc.postgresql.org/
6. https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html
7. https://spark.apache.org/docs/latest/cluster-overview.html
8. https://kubernetes.io/docs/home/