

ĐẠI HỌC BÁCH KHOA HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP

**Nền tảng dữ liệu chứng khoán Việt Nam
Phân hệ quản lý và lập lịch trình tính toán chỉ báo**

MAI TRƯỜNG SƠN

son.mt183620@sis.hust.edu.vn

**Trường Công nghệ thông tin và truyền thông
Ngành Khoa học Máy tính**

Giảng viên hướng dẫn: TS. Trần Việt Trung

Chữ kí GVHD

Trường:

Công nghệ thông tin và Truyền thông

HÀ NỘI, 03/2023

LỜI CẢM ƠN

Đầu tiên, tôi xin gửi lời cảm ơn đến bố mẹ, em gái và những người thân trong gia đình vì đã luôn ủng hộ và động viên tôi trong suốt gần 05 năm theo học tại Đại học Bách khoa Hà Nội.

Tôi xin gửi lời cảm ơn đến TS. Trần Việt Trung vì đã tận tình hướng dẫn, chỉ bảo để tôi có thể hoàn thiện đề án này; cảm ơn Tuấn, Dương, Tú vì đã đồng hành cùng tôi với đề tài này trong nhiều tháng vừa qua. Tôi cũng xin gửi lời cảm ơn đến các thầy cô giáo ở Bách khoa Hà Nội, đến tập thể IT1-04 K63 vì đã giúp quãng đời sinh viên của tôi trở nên vô cùng đáng nhớ.

Tiếp theo, tôi xin gửi lời cảm ơn đến những đồng nghiệp thuộc phòng Công nghệ số, Ban Công nghệ Thông tin, Tập đoàn Công nghiệp - Viễn thông Quân đội (Viettel) - nơi tôi đang thực tập - vì đã tạo điều kiện công việc tối đa cho tôi trong quá trình thực hiện đề án này.

Cuối cùng, cảm ơn Trang vì đã đến và luôn là chỗ dựa tinh thần, luôn động viên tôi trong những thời điểm khó khăn nhất.

Xin trân trọng cảm ơn!

TÓM TẮT NỘI DUNG ĐỒ ÁN

Thị trường chứng khoán Việt Nam trong những năm trở lại đây chứng kiến một giai đoạn phát triển nhanh chóng với xu hướng đầu tư cá nhân tăng mạnh; từ đó đặt ra yêu cầu từ về những nền tảng ứng dụng theo dõi dữ liệu biến động của thị trường chứng khoán. Hiện nay ở Việt Nam có thể dễ dàng tiếp cận với các ứng dụng như vậy với máy tính và điện thoại thông minh. Những ứng dụng này cung cấp những chức năng cần thiết để người dùng theo dõi thị trường với giao diện dễ sử dụng, bảng biểu, sơ đồ dữ liệu được cập nhật theo thời gian thực.

Bên cạnh những ưu điểm nêu trên, các nền tảng này vẫn còn những hạn chế về dữ liệu của các chỉ báo chứng khoán. Chúng chủ yếu cung cấp những chỉ báo cơ bản, người dùng không thể thêm mới hay tùy biến những chỉ báo theo công thức của riêng mình. Trước thực tế này, nhóm chúng tôi đã quyết định thực hiện xây dựng "Nền tảng dữ liệu thị trường chứng khoán Việt Nam" với chức năng cập nhật dữ liệu giao dịch thị trường, tính toán chỉ báo chứng khoán, cung cấp giao diện website để người dùng có thể tạo mới và chỉnh sửa các chỉ báo theo nhu cầu cá nhân của người sử dụng.

Trong khuôn khổ ĐATN này, mục tiêu của tôi là xây dựng "Phân hệ quản lý và lập lịch trình tính toán chỉ báo", đóng vai trò xử lý và tính toán dữ liệu chỉ báo của thị trường chứng khoán. Sau quá trình nghiên cứu, tìm hiểu và triển khai, tôi đã xây dựng được phân hệ gồm các trình tính toán được quản lý và lập lịch tự động với ngôn ngữ lập trình Python [1] và công nghệ lập lịch Apache Airflow [2], từ đó đóng góp vào hoàn thiện sản phẩm nền tảng dữ liệu của cả nhóm.

MỤC LỤC

| | |
|---|-----------|
| CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI..... | 1 |
| 1.1 Đặt vấn đề..... | 1 |
| 1.2 Mục tiêu và phạm vi đề tài..... | 1 |
| 1.3 Định hướng giải pháp..... | 2 |
| 1.4 Bố cục đồ án | 2 |
| CHƯƠNG 2. XÂY DỰNG NỀN TẢNG DỮ LIỆU THỊ TRƯỜNG CHỨNG KHOÁN | 4 |
| 2.1 Thông tin chung về TTCK Việt Nam..... | 4 |
| 2.1.1 Các khái niệm cơ bản | 4 |
| 2.1.2 Các chỉ báo chứng khoán phổ biến..... | 5 |
| 2.1.3 Khảo sát các ứng dụng nền tảng dữ liệu TTCK tại Việt Nam..... | 9 |
| 2.2 Tổng quan thiết kế nền tảng dữ liệu chứng khoán..... | 10 |
| 2.2.1 Tổng quan yêu cầu chức năng..... | 10 |
| 2.2.2 Yêu cầu phi chức năng..... | 10 |
| 2.2.3 Thiết kế hệ thống | 11 |
| CHƯƠNG 3. CÔNG NGHỆ SỬ DỤNG..... | 13 |
| 3.1 Công nghệ lưu trữ dữ liệu | 13 |
| 3.1.1 Apache Druid[3] | 13 |
| 3.1.2 Apache Kafka[4]..... | 15 |
| 3.2 Công nghệ xử lý dữ liệu - Python[1] và pandas [9]..... | 17 |
| 3.3 Công nghệ lập lịch cho các trình tính toán - Apache Airflow [2] | 18 |
| 3.4 Công nghệ tự động sinh tệp tin theo mẫu - jinja[11] | 20 |
| 3.5 Công nghệ đóng gói và triển khai phần mềm - Docker[12] | 21 |

| | |
|---|-----------|
| CHƯƠNG 4. XÂY DỰNG PHÂN HỆ QUẢN LÝ VÀ LẬP LỊCH TRÌNH TÍNH TOÁN CHỈ BÁO | 22 |
| 4.1 Tính toán chỉ báo theo công thức..... | 22 |
| 4.1.1 Thiết kế cơ sở dữ liệu | 22 |
| 4.1.2 Thiết kế luồng tính toán..... | 23 |
| 4.1.3 Đặc tả lớp Miner | 25 |
| 4.1.4 Sơ đồ lớp các trình tính toán..... | 29 |
| 4.2 Lập lịch các trình tính toán | 30 |
| 4.2.1 Nhu cầu lập lịch các trình tính toán chỉ báo | 30 |
| 4.2.2 Lập lịch sử dụng Apache Airflow | 31 |
| 4.3 Chức năng tự động tạo mới chỉ báo cá nhân | 32 |
| 4.3.1 Biểu đồ hoạt động của chức năng tạo mới chỉ báo | 33 |
| 4.3.2 Dữ liệu đầu vào của người dùng | 34 |
| 4.3.3 Vai trò của phân hệ quản lý và lập lịch các trình tính toán chỉ báo... | 35 |
| 4.4 Yêu cầu phi chức năng | 39 |
| CHƯƠNG 5. KẾT QUẢ THỰC NGHIỆM VÀ ĐÁNH GIÁ..... | 41 |
| 5.1 Xây dựng ứng dụng..... | 41 |
| 5.1.1 Thư viện và công cụ sử dụng..... | 41 |
| 5.1.2 Cấu hình thiết bị | 42 |
| 5.2 Triển khai | 42 |
| 5.2.1 Triển khai trình tính toán chỉ báo..... | 42 |
| 5.2.2 Triển khai lập lịch trình tính toán chỉ báo | 46 |
| 5.2.3 Kết quả đạt được | 47 |
| 5.2.4 Minh họa các chức năng chính | 47 |
| 5.3 Kiểm thử..... | 50 |

| | |
|---|-----------|
| CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN | 53 |
| 6.1 Kết luận..... | 53 |
| 6.2 Hướng phát triển..... | 53 |
| TÀI LIỆU THAM KHẢO..... | 54 |

DANH MỤC HÌNH VẼ

| | | |
|-----------|--|----|
| Hình 2.1 | Mô hình hệ thống cho nền tảng dữ liệu | 12 |
| Hình 3.1 | Kiến trúc và các thành phần của Apache Druid [7] | 13 |
| Hình 3.2 | Kiến trúc và các thành phần của Apache Kafka | 16 |
| Hình 3.3 | Mối tương quan giữa <i>Series</i> và <i>DataFrame</i> trong pandas | 18 |
| Hình 3.4 | Kiến trúc và các thành phần của Apache Airflow [10] | 19 |
| Hình 3.5 | Các bước tạo file theo template của jinja | 20 |
| Hình 4.1 | Apache Druid data model | 23 |
| Hình 4.2 | Luồng thực thi của một trình tính toán chỉ báo | 24 |
| Hình 4.3 | Biểu đồ các lớp Miner, DruidConnection và KafkaConnection | 24 |
| Hình 4.4 | Biểu đồ lớp Miner | 25 |
| Hình 4.5 | Lớp DruidConnection | 26 |
| Hình 4.6 | Lớp KafkaConnection | 26 |
| Hình 4.7 | Sơ đồ lớp gồm các trình tính toán riêng biệt | 30 |
| Hình 4.8 | Khai báo một Airflow DAG trong file Python | 31 |
| Hình 4.9 | Khai báo một Task trong file Python | 31 |
| Hình 4.10 | Khai báo thứ tự các Tasks trong một DAG | 32 |
| Hình 4.11 | DAG trình tự các trình tính toán chỉ báo | 32 |
| Hình 4.12 | Sơ đồ hoạt động tính năng tạo mới chỉ báo | 33 |
| Hình 4.13 | Template tạo Druid spec file | 36 |
| Hình 4.14 | Template tạo class cho chỉ báo mới | 37 |
| Hình 4.15 | Các bước cập nhật DAG trên Airflow | 38 |
| Hình 4.16 | Template cho script khai báo Task cho chỉ báo mới | 39 |
| Hình 5.1 | Mã nguồn lớp DruidConnection | 43 |
| Hình 5.2 | Mã nguồn lớp KafkaConnection | 44 |
| Hình 5.3 | Mã nguồn lớp Miner | 45 |
| Hình 5.4 | Mã nguồn lớp OBVMiner | 46 |
| Hình 5.5 | Các container thành phần của Airflow | 47 |
| Hình 5.6 | Các datasources trên Apache Druid | 48 |
| Hình 5.7 | Kết quả chỉ báo OBV mã HPG trong tháng 1/2023 | 48 |
| Hình 5.8 | Giao diện web của Airflow | 49 |
| Hình 5.9 | Giao diện thông tin 1 DAG trên Airflow | 49 |

DANH MỤC BẢNG BIỂU

| | | |
|----------|--|----|
| Bảng 5.1 | Danh sách thư viện và công cụ sử dụng | 41 |
| Bảng 5.2 | Cấu hình máy tính cá nhân | 42 |
| Bảng 5.3 | Các thống kê khác của phân hệ khi triển khai | 47 |

DANH MỤC THUẬT NGỮ

| Tên thuật ngữ | Ý nghĩa |
|-------------------------|--|
| Ticker | Mã cổ phiếu, xác định duy nhất một cổ phiếu cụ thể trên một sàn giao dịch cụ thể |
| Apache Druid | Hệ thống lưu trữ dữ liệu time-series |
| Druid datasource | Thành phần lưu trữ dữ liệu trong Apache Druid |
| Apache Kafka | Nền tảng lưu trữ dữ liệu streaming phân tán |
| Kafka topic | Một đơn vị tổ chức lưu trữ dữ liệu trong Apache Kafka |
| File | Tệp tin |
| Class | Lớp (trong Lập trình hướng đối tượng) |
| Miner | Trình tính toán chỉ báo |
| Template | Bản mẫu |
| Docker | Công nghệ đóng gói và triển khai ứng dụng |
| MySQL Server | Hệ thống quản trị cơ sở dữ liệu quan hệ mã nguồn mở phát triển bởi Oracle |
| PostgreSQL | Một thống quản trị cơ sở dữ liệu mã nguồn mở |
| MsSQL | Hệ thống quản trị cơ sở dữ liệu phát triển bởi Microsoft |

DANH MỤC TỪ VIẾT TẮT

| Viết tắt | Tên tiếng Anh | Tên tiếng Việt |
|----------------------------|---------------------------------------|---|
| CNTT | | Công nghệ thông tin |
| DATN | | Đồ án tốt nghiệp |
| TTCK | | Thị trường chứng khoán |
| HoSE | Ho Chi Minh City Stock Exchange | Sở Giao dịch Chứng khoán Thành phố Hồ Chí Minh |
| VN-Index | | Chỉ số thể hiện biến động giá cổ phiếu giao dịch trên HoSE |
| HNX | Hanoi Stock Exchange | Sở Giao dịch Chứng khoán Hà Nội |
| HNX-Index | | Chỉ số thể hiện biến động giá cổ phiếu giao dịch trên HNX |
| OBV | On-Balance Value | Chỉ số Cân bằng Khối lượng |
| A/D Line | Accumulation/ Distribution Line | Đường Tích lũy/ Phân phối |
| EMA | Exponential Moving Average | Trung bình động lũy thừa |
| MACD | Moving Average Convergence Divergence | Trung bình động hội tụ phân kỳ |
| RS | Comparative Relative Strength | Chỉ số mạnh tương quan |
| RSI | Relative Strength Index | Chỉ số sức mạnh tương đối |
| API | Application Programming Interface | Giao diện lập trình ứng dụng |
| RESTAPI/ RESTfulAPI | REpresentational State Transfer API | Tiêu chuẩn thiết kế API cho ứng dụng web |
| ReactJS | | Framework lập trình giao diện web sử dụng ngôn ngữ JavaScript |
| FastAPI | | Framework lập trình máy chủ web sử dụng ngôn ngữ Python |
| JSON | JavaScript Object Notation | Định dạng dữ liệu chuyển đổi JavaScript |
| CSV | Comma-separated values | |
| IDE | Integrated Development Environment | Môi trường lập trình tích hợp |
| SQL | Structured Query Language | Ngôn ngữ truy vấn có cấu trúc |

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Đặt vấn đề

Thị trường chứng khoán Việt Nam ngày càng thu hút nhiều nhà đầu tư tham gia, đặc biệt là làn sóng đầu tư cá nhân trong và sau đại dịch COVID-19. Nhiều người dân tìm đến thị trường chứng khoán như một hình thức đầu tư sinh lời, đem lại nguồn thu cho cá nhân và gia đình bên cạnh các nguồn thu sẵn có khác.

Tuy nhiên, nhà đầu tư cá nhân không phải ai cũng có thể có mặt thường xuyên, liên tục tại các Sở giao dịch chứng khoán hay các công ty chứng khoán để cập nhật liên tục các số liệu về giao dịch của thị trường. Do vậy nhu cầu về một nền tảng dữ liệu chứng khoán được đưa ra, nhằm giúp tất cả mọi người có thể theo dõi diễn biến thị trường ở bất kì đâu với các thiết bị thông minh.

Nhưng ngay cả khi có được các số liệu đầy đủ và nhanh chóng, các quyết định mua-bán của các nhà đầu tư mới vẫn chủ yếu dựa vào cảm tính hay lời chỉ dẫn của các môi giới (“broker”), làm giảm hiệu quả đầu tư, gây nhiễu thị trường. Để có thể đưa ra các quyết định sáng suốt hơn trong giao dịch chứng khoán, nhiều nhà đầu tư đã tham khảo đến các chỉ báo, là những đại lượng được tính toán dựa trên số liệu giá cả thị trường theo thời gian để thể hiện quá trình biến động của các mã cổ phiếu, dự đoán xu hướng tương lai, qua đó tăng hiệu quả đầu tư. Các chỉ báo thường được nhà đầu tư quan tâm có thể kể đến như: EMA, MACD hay OBV (được giới thiệu chi tiết ở phần 2.1.2). Ngoài ra, các nhà đầu tư còn có mong muốn tự định nghĩa ra các chỉ báo của riêng mình, để phân tích thị trường với góc nhìn cá nhân dựa vào kinh nghiệm theo dõi và giao dịch trên thị trường.

1.2 Mục tiêu và phạm vi đề tài

Dựa vào nhu cầu sẵn có của thị trường, nhóm chúng tôi đã có những tìm hiểu về các sản phẩm công nghệ phục vụ cho việc theo dõi biến động thị trường chứng khoán. Trên thị trường hiện có các sản phẩm như: SSI iBoard (phát triển bởi Công ty Cổ phần Chứng khoán SSI), VnDirect (phát triển bởi Công ty Cổ phần Chứng khoán VNDIRECT) hay Entrade X (phát triển bởi Công ty Cổ phần Chứng khoán DNSE). Ưu điểm của những sản phẩm này là dữ liệu được cập nhật nhanh chóng, hiển thị số liệu qua các biểu đồ thân thiện với người dùng; một điểm nữa phải kể đến là có nhiều sản phẩm liên kết với tài khoản của người dùng tại các công ty chứng khoán, tích hợp thêm tính năng đặt lệnh mua-bán ngay trên ứng dụng. Tuy nhiên bên cạnh các ưu điểm nêu trên, các hệ thống nền tảng thông tin TTCK đều có những hạn chế trong cung cấp thông tin các chỉ báo, người dùng chỉ được chọn các chỉ báo có sẵn, không thể tự thêm mới các chỉ báo theo nhu cầu cá nhân.

Do đó, nhóm chúng tôi đã quyết định lựa chọn đề tài xây dựng một nền tảng dữ liệu thị trường chứng khoán. Nền tảng này sẽ có chức năng thu thập và lưu trữ các dữ liệu giao dịch của thị trường, tính toán các thông tin chỉ báo, cung cấp giao diện web thuận tiện để người dùng dễ dàng truy cập. Ngoài ra, nền tảng còn cho phép người dùng tự định nghĩa các chỉ báo mới, qua đó đưa ra những quyết định đầu tư hiệu quả cao hơn.

Trong quyển đồ án này, tôi sẽ trình bày về quá trình thiết kế và xây dựng phân hệ quản lý và lập lịch các trình tính toán chỉ báo, là một phần quan trọng của nền tảng dữ liệu TTCK nêu trên. Phân hệ này đóng vai trò định kỳ tính toán các chỉ báo mỗi khi dữ liệu giao dịch được cập nhật, ngoài ra hỗ trợ người dùng tạo mới các chỉ báo theo ý muốn.

1.3 Định hướng giải pháp

Để xây dựng nên phân hệ quản lý và lập lịch các trình tính toán chỉ báo với phạm vi và mục tiêu đã nêu ở phần 1.2, tôi đề xuất giải pháp triển khai như sau:

Dữ liệu giao dịch của các mã cổ phiếu sau khi được định kỳ thu thập sẽ được lưu lại trong cơ sở dữ liệu. Từ đây, các chỉ báo sẽ được tính toán dựa theo công thức tương ứng. Sau khi quá trình tính toán hoàn tất, giá trị của các chỉ báo sẽ được lưu vào cơ sở dữ liệu để dễ dàng hiển thị cho người dùng. Ngoài ra phân hệ cũng sẽ đáp ứng yêu cầu tạo mới chỉ báo của người dùng.

Với phương án trên, tôi đã sử dụng ngôn ngữ lập trình Python 3.8 [1] để đọc, ghi và xử lý dữ liệu chỉ báo. Các cơ sở dữ liệu được sử dụng hỗ trợ cho phân hệ bao gồm Apache Druid [3] và Apache Kafka [4]. Những công nghệ này sẽ được đề cập cụ thể hơn ở Chương 3.

Dựa vào những định hướng nêu trên, phân hệ quản lý và lập lịch các trình tính toán chỉ báo đã được xây dựng với các chức năng tính toán chỉ báo, lập lịch tính toán chỉ báo và đặc biệt nhất là thêm mới các trình tính toán ngay khi hệ thống đang hoạt động. Phân hệ này đã góp phần xây dựng nền tảng dữ liệu TTCK với các chức năng như đã đề ra.

1.4 Bố cục đồ án

Phần còn lại của báo cáo đồ án tốt nghiệp này được tổ chức như sau.

Chương 2 trình bày về quá trình xây dựng và thiết kế tổng thể nền tảng, các phân hệ trong nền tảng. Nội dung bao gồm từ các kiến thức lý thuyết cơ bản về thị trường chứng khoán cho đến các sơ đồ thiết kế tổng quan cho toàn bộ nền tảng dữ liệu này.

Đến Chương 3, tôi giới thiệu về các công nghệ được sử dụng để xây dựng nên phân hệ quản lý và lập lịch các trình tính toán chỉ báo. Các công nghệ được đề cập

sẽ phân thành các nhóm liên quan đến chức năng: chức năng lưu trữ, chức năng xử lý, chức năng lập lịch, chức năng tạo mới file. Các chức năng này đều là một phần trong quá trình xây dựng nên phân hệ.

Ở chương 4, tôi sẽ trình bày về thiết kế, triển khai các thành phần của phân hệ quản lý và lập lịch các trình tính toán chỉ báo. Nội dung bao gồm đặc tả các chức năng của phân hệ, các đoạn mã giả minh họa, thiết kế lớp và cơ sở dữ liệu. Việc áp dụng các công nghệ nêu ở chương 3 cũng sẽ được nói đến chi tiết.

Chương 5 sẽ trình bày về quá trình triển khai thực tế phân hệ. Những nội dung được nêu bao gồm: công cụ sử dụng, thông tin đóng gói phân hệ, minh họa triển khai những chức năng chính của phân hệ, thông tin cấu hình và kiểm thử phân hệ.

Cuối cùng, phần nội dung về kết luận và hướng phát triển được trình bày ở chương 6.

CHƯƠNG 2. XÂY DỰNG NỀN TẢNG DỮ LIỆU THỊ TRƯỜNG CHỨNG KHOÁN

Ở chương 2, đồ án sẽ trình bày những nội dung kiến thức chung và cơ bản về thị trường chứng khoán, những ứng dụng hiện có để theo dõi chứng khoán tại Việt Nam. Tiếp sau đó sẽ là nội dung về tổng quan yêu cầu và thiết kế hệ thống của nền tảng dữ liệu chứng khoán mà nhóm lựa chọn xây dựng.

2.1 Thông tin chung về TTCK Việt Nam

2.1.1 Các khái niệm cơ bản

Chứng khoán là bằng chứng xác nhận quyền và lợi ích hợp pháp của người sở hữu đối với tài sản hoặc phần vốn của tổ chức phát hành. Chứng khoán được thể hiện bằng hình thức chứng chỉ, bút toán ghi sổ hoặc dữ liệu điện tử. Chứng khoán bao gồm các loại: cổ phiếu, trái phiếu, chứng chỉ quỹ đầu tư, chứng khoán phái sinh.

Thị trường chứng khoán hay sàn chứng khoán là nơi phát hành giao dịch mua bán, trao đổi các loại cổ phiếu chứng khoán và được thực hiện chủ yếu tại sở giao dịch chứng khoán hoặc thông qua các công ty môi giới chứng khoán. Thị trường chứng khoán được chia thành 2 loại đó là thị trường sơ cấp và thị trường thứ cấp. Thị trường sơ cấp là nơi cổ phiếu lần đầu phát hành từ công ty để hút một nguồn vốn đầu tư, điều này giúp họ có thể huy động một số vốn trên thị trường chứng khoán. Phần lớn những người mua trên thị trường sơ cấp là các tổ chức lớn hay quỹ đầu tư. Với thị trường chứng khoán thứ cấp, cổ phiếu được mua bán lại sau khi phát hành sơ cấp. Người mua tại thị trường sơ cấp sẽ tiến hành mua bán đối với các nhà đầu tư chứng khoán khác trên thị trường. Chính vì thế sẽ không có tiền mới được sinh ra mà chỉ là thay đổi quyền sở hữu cổ phiếu giữa người mua và bán. Đây cũng là nơi các nhà đầu tư cá nhân có thể tham gia giao dịch chứng khoán.

Các thành phần tham gia TTCK:

- Nhà phát hành: là các tổ chức thực hiện huy động vốn thông qua TTCK dưới hình thức phát hành các chứng khoán
- Nhà đầu tư: là những người thực sự mua và bán chứng khoán trên TTCK. Nhà đầu tư có thể đượ chia thành 2 loại nhà đầu tư cá nhân và nhà đầu tư có tổ chức.
- Các công ty chứng khoán: là những công ty hoạt động tron lĩnh vực chứng khoán, có thể đảm nhận một hoặc nhiều trong số các nghiệp vụ chính là môi giới, bảo lãnh phát hành, quản lý quỹ đầu tư, tư vấn đầu tư chứng khoán và tự

doanh.

Một số khái niệm khác:

- **Cổ phiếu:** là loại chứng khoán thường gặp nhất trên thị trường chứng khoán. Cổ phiếu đại diện cho một phần vốn của một công ty. Việc sở hữu cổ phiếu giúp các nhà đầu tư trở thành cổ đông của công ty và có quyền được tham gia vào việc quyết định quan trọng của công ty.
- **Chỉ số chứng khoán:** là thước đo đại diện cho sự biến động của giá chứng khoán trên sàn giao dịch. Ở Việt Nam, có hai chỉ số chính là VN-Index của HOSE và HNX-Index của HNX.
- **Khối lượng giao dịch:** Là số lượng chứng khoán được giao dịch trong một khoảng thời gian nhất định. Khối lượng giao dịch thường được xem là chỉ số thể hiện độ sôi động của thị trường chứng khoán.
- **Giá trị vốn hóa thị trường:** Là giá trị của tất cả các cổ phiếu đang được giao dịch trên sàn chứng khoán. Giá trị vốn hóa thị trường thường được sử dụng để đánh giá giá trị của một công ty.
- **Biến động giá chứng khoán:** Là sự thay đổi giá của chứng khoán trong một khoảng thời gian nhất định. Việc theo dõi biến động giá chứng khoán giúp nhà đầu tư đưa ra quyết định đầu tư hợp lý.
- **Lệnh mua và lệnh bán:** Là các yêu cầu mua hoặc bán chứng khoán được đưa ra trên sàn giao dịch chứng khoán.

2.1.2 Các chỉ báo chứng khoán phổ biến

Mỗi chỉ báo của TTCK được xác định qua các công thức tính toán riêng. Hệ thống hiện tại gồm 07 chỉ báo có sẵn, với các công thức được định nghĩa dựa vào các thông tin giao dịch của các mã chứng khoán.

Để thống nhất cho việc trình bày các công thức của các chỉ báo khác nhau, các kí hiệu được quy ước chung như sau:

O_i (open): giá khởi điểm (giá tham chiếu) trong phiên ngày i

C_i (close): giá chốt phiên trong phiên ngày i

H_i (high): giá cao nhất trong phiên ngày i

L_i (low): giá thấp nhất trong phiên ngày i

V_i (volume): khối lượng giao dịch trong phiên ngày i

1. Chỉ báo OBV (On-Balance Volume) - Chỉ số Cân bằng Khối lượng

(a) Ý nghĩa

OBV là giá trị tích lũy khối lượng giao dịch thành công trừ các phiên cộng thêm khối lượng giao dịch nếu tăng giá và trừ đi khối lượng giao dịch nếu giảm giá. Ý nghĩa của OBV đánh giá sức tăng hoặc giảm của giá dựa trên khối lượng được giao dịch thành công.

Nếu giá tăng nhưng khối lượng giao dịch nhỏ, đồ thị OBV tăng chậm, giá giảm nhưng khối lượng giao dịch nhỏ, đồ thị OBV giảm chậm. Nếu giá tăng với khối lượng giao dịch lớn OBV tăng mạnh, nếu giá giảm với khối lượng giao dịch lớn thì OBV giảm mạnh.

Như vậy căn cứ vào sức tăng của OBV và việc hình thành phân kỳ âm hoặc dương của OBV để kết luận và khẳng định tính chắc chắn của xu thế tăng hoặc giảm giá hiện tại.

(b) Công thức

$$OBV_i = OBV_{i-1} + \text{sgn}(C_i - C_{i-1}) * V_i$$

2. Chỉ báo A/D (Accumulation/ Distribution) Line - Đường Tích lũy/ Phân phối

(a) Ý nghĩa

Chỉ báo A/D được hiểu là chỉ báo kỹ thuật sử dụng hai yếu tố là khối lượng giao dịch và giá cả để xem xét sự thay đổi của một cổ phiếu. Lúc này, khối lượng càng lớn thì ảnh hưởng của biến động càng cao và ngược lại

Accumulation (tích lũy): Khối lượng giao dịch được xem là tích lũy trong trường hợp giá đóng cửa hiện tại cao hơn giá đóng cửa phiên giao dịch của ngày hôm trước. Điều này có nghĩa là thị trường đang lái giá lên cao để bán ra. Khối lượng càng lớn, tức là hệ số A/D càng lớn sẽ chứng tỏ biến động của giá càng cao.

Distribution (phân phối): Khối lượng giao dịch được xem là phân phối trong trường hợp giá đóng cửa hiện tại thấp hơn giá đóng cửa phiên giao dịch của ngày hôm trước. Điều này có nghĩa là thị trường đang kéo giá xuống thấp để mua vào, các nhà đầu tư có thể canh điểm để vào hàng.

(b) Công thức

$$MFM_i = \frac{(C_i - L_i) - (H_i - C_i)}{H_i - L_i}$$

$$MFV_i = MFM_i * V_i$$

$$A/D_i = A/D_{i-1} + MFV_i$$

CHƯƠNG 2. XÂY DỰNG NỀN TẢNG DỮ LIỆU THỊ TRƯỜNG CHỨNG KHOÁN

3. Chỉ báo Aroon - Chỉ báo Xu hướng giá

(a) Ý nghĩa

Chỉ báo Aroon là một công cụ giao dịch cho biết giá đang có xu hướng hay trong một phạm vi. Nếu đường Aroon Up cắt qua đường Aroon Down, tâm lý thị trường là tăng, và khi ngược lại, sẽ giảm.

Sử dụng chỉ báo Aroon sẽ giúp bạn có cái nhìn rõ ràng hơn về tâm lý thị trường, đặc biệt là khi xu hướng mới sẽ bắt đầu. Chỉ báo Aroon được hiển thị trong một cửa sổ riêng biệt với hai đường động – đường Aroon Up và Aroon Down. Hai đường này dao động và hiển thị hướng tiềm năng dựa trên các điều kiện sau:

- Khi Aroon Up cắt qua Aroon Down, thị trường bắt đầu một xu hướng tăng mới.
- Khi Aroon Down cắt qua Aroon Up, thị trường bắt đầu xu hướng giảm mới.
- Xu hướng tăng mạnh khi Aroon Up ở mức 100.
- Xu hướng giảm mạnh khi Aroon Down ở mức 100.

(b) Công thức

i. Aroon up

$$Aroonup = \frac{25 - \text{Số chu kỳ từ giá đỉnh trong 25 ngày}}{25} * 100$$

ii. Aroon down

$$Aroondown = \frac{25 - \text{Số chu kỳ từ giá đáy trong 25 ngày}}{25} * 100$$

4. Chỉ báo EMA (Exponential Moving Average) - Đường Trung bình động lũy thừa

(a) Ý nghĩa

Đường EMA (Exponential Moving Average) được gọi là đường trung bình động lũy thừa. EMA là công cụ chỉ báo phản ánh sự biến động của giá được tính theo cấp số nhân dùng để tạo tín hiệu mua, bán dựa trên giao thoa và phân kỳ so với mức giá trung bình ở quá khứ.

Dựa vào công thức tính của EMA là đặt trọng số theo cấp số nhân nên EMA phản ánh được xu hướng giá trong khoảng thời gian biến động giá

gần nhất, khoảng thời gian này thường là 10 ngày, 20 phút, 30 tuần hay bất kỳ khoảng thời gian nào mà nhà đầu tư lựa chọn. Do vậy mà đường EMA thường nhạy cảm với các tín hiệu bất thường trong ngắn hạn hơn và kết quả nhận được chuẩn xác hơn so với đường SMA. Từ đó nhà đầu tư sẽ có được các dự báo để phản ứng nhanh và kịp thời hơn với các biến động này.

Ngoài ra EMA còn giúp nhận biết được xu hướng của thị trường thời điểm hiện tại và xác định được các ngưỡng hỗ trợ và kháng cự của giá.

(b) Công thức

$$EMA_i = C_i * \frac{2}{N+1} + EMA_{i-1} * (1 - \frac{2}{N+1})$$

N thường có giá trị bằng 12, 26, 50 và 100, tương ứng là các chỉ số EMA_{12} , EMA_{26} , EMA_{50} và EMA_{100} .

5. Chỉ báo MACD (Moving Average Convergence/ Divergence) - Đường Trung bình hội tụ phân kì

(a) Ý nghĩa MACD là từ viết tắt của Moving Average Convergence Divergence – Trung bình động hội tụ phân kỳ. Đây là một đường chỉ báo kỹ thuật được tạo ra bởi nhà phát minh Gerald Appel vào năm 1979. Đường MACD được tính bằng độ chênh lệch giữa 2 trung bình trượt số mũ. Thông thường đó là 2 trung bình trượt số mũ của 2 chu kỳ 12 ngày và 26 ngày.

(b) Công thức

$$MACD_i = EMA_{12_i} - EMA_{26_i}$$

6. Chỉ báo RS (Comparative Relative Strength) - Chỉ số sức mạnh tương quan

(a) Ý nghĩa

Chỉ số Sức mạnh tương quan RS (Comparative Relative Strength) là một chỉ báo cơ bản để so sánh sức mạnh và khả năng sinh lời của một cổ phiếu so với các cổ phiếu khác trên thị trường.

Nếu sử dụng chỉ báo cũng có thể được sử dụng để so sánh với ngành, từ đây có thể xác định xem cổ phiếu đang xem xét mạnh hay yếu hơn các cổ phiếu trong ngành. Chỉ báo RS cũng có thể được sử dụng để tìm các cổ phiếu tăng tốt hơn trong thị trường Uptrend và giảm ít hơn trong thị trường Downtrend.

(b) Công thức

$$RS_i = \frac{AG_i}{AL_i}$$

AG: Average Gain (trung bình giá tăng trong chu kì tính toán)

AL: Average Loss (trung bình giá giảm trong chu kì tính toán)

Chu kì thường lấy giá trị 14 (ví dụ: 14 ngày, 14 giờ, ...)

7. Chỉ báo RSI (Relative Strength Index) - Chỉ số Sức mạnh tương đối

(a) Ý nghĩa RSI là viết tắt của Relative Strength Index - chỉ số sức mạnh tương đối. Chỉ báo RSI tính toán tỷ lệ giữa mức tăng giá và giảm giá trung bình trong một khoảng thời gian nhất định, thể hiện tình trạng quá mua và quá bán của thị trường.

(b) Công thức

$$RSI_i = 100 - \frac{100}{RS_i + 1}$$

Chu kì thường lấy giá trị 14 (ví dụ: 14 ngày, 14 giờ, ...)

2.1.3 Khảo sát các ứng dụng nền tảng dữ liệu TTCK tại Việt Nam

Như ở chương 1 đã đề cập, xu hướng đầu tư chứng khoán ở Việt Nam đang ngày càng tăng cao, kéo theo đó là nhu cầu cập nhật dữ liệu chứng khoán thường xuyên, liên tục. Hiện nay trên thị trường đã có nhiều ứng dụng dưới cả dạng web và mobile app cho phép người dùng theo dõi biến động chứng khoán, và thường được phát triển bởi các công ty chứng khoán. Những ví dụ có thể liệt kê ra như: ứng dụng SSI iBoard (phát triển bởi Công ty Cổ phần Chứng khoán SSI), ứng dụng VnDirect (phát triển bởi Công ty Cổ phần Chứng khoán VNDIRECT) hay ứng dụng Entrade X (phát triển bởi Công ty Cổ phần Chứng khoán DNSE).

Những ứng dụng này đã có thời gian dài hoạt động và cho thấy những ưu điểm nổi trội. Dữ liệu trên những ứng dụng này được hiển thị theo dạng bảng và được cập nhật theo thời gian thực. Các mã cổ phiếu cũng được chia ra thành các nhóm ngành khác nhau, cùng với đó là số liệu thể hiện xu hướng tăng/ giảm theo từng nhóm ngành. Hơn nữa, số liệu được hiển thị cùng với màu sắc theo xu hướng tăng, giảm của những mã cổ phiếu mà người dùng theo dõi, tăng sự trực quan khi người dùng nhìn vào các bảng số liệu.

Một phần thông tin dữ liệu nữa cũng được những ứng dụng này tính toán và hiển thị, đó là các chỉ báo chứng khoán. Nhiều nhà đầu tư không chỉ theo dõi những thông tin về giá cả hay khối lượng giao dịch mà còn rất quan tâm đến xu hướng biến động của các chỉ báo chứng khoán. Những chỉ báo này tùy theo công

thức tính toán mà đem lại những cái nhìn khác nhau về biến động thị trường, qua đó giúp nhà đầu tư có thêm những căn cứ để đưa ra quyết định mua-bán của mình.

Ngoài ra, hầu hết những ứng dụng trên thị trường hiện nay được phát triển bởi những công ty chứng khoán tại Việt Nam. Do đó người sử dụng hoàn toàn có thể đặt lệnh mua-bán các mã cổ phiếu ngay trên ứng dụng di động sau khi đăng nhập vào tài khoản chứng khoán tương ứng. Với ưu điểm này, việc phát triển các ứng dụng hoạt động ổn định, có thông tin cập nhật chính xác và giao diện bắt mắt sẽ dễ dàng giúp công ty chứng khoán thu hút thêm người dùng mở tài khoản mới. Bên cạnh những thông tin số liệu của thị trường, nhiều ứng dụng còn tích hợp thêm các trang tin tức về chủ đề đầu tư - kinh doanh, mang lại bức tranh đa chiều hơn về thị trường tài chính cho những nhà đầu tư chứng khoán.

Tuy nhiên, bên cạnh các ưu điểm nêu trên, các ứng dụng theo dõi chứng khoán hiện nay còn có những hạn chế liên quan đến các chỉ báo chứng khoán. Danh sách chỉ báo trên các ứng dụng còn sơ sài, đôi khi không đáp ứng được nhu cầu theo dõi của người dùng. Người dùng cũng không thể chỉnh sửa hay thậm chí là thêm mới các chỉ báo theo công thức mà mình mong muốn, gây bất tiện cho quá trình sử dụng.

2.2 Tổng quan thiết kế nền tảng dữ liệu chứng khoán

2.2.1 Tổng quan yêu cầu chức năng

Về mặt tính năng, nền tảng cần cung cấp chức năng đăng kí và đăng nhập cho người dùng. Tiếp đó, nền tảng phải đáp ứng khả năng thu thập và lưu trữ dữ liệu giao dịch của tất cả các mã đang niêm yết trên sàn chứng khoán, đồng thời hiển thị dữ liệu này dưới dạng biểu đồ. Bên cạnh dữ liệu giao dịch, dữ liệu của các chỉ báo phổ biến trên thị trường cũng được tính toán và hiển thị trên giao diện web. Một tính năng nữa mà nền tảng cần đáp ứng là cho phép người dùng tạo mới chỉ báo chứng khoán cá nhân thông qua việc nhập liệu trên giao diện.

Về mặt dữ liệu, bên cạnh việc thu thập, lưu trữ và trực quan hóa, nền tảng cần đảm bảo tính chính xác và cập nhật của dữ liệu. Ngoài ra còn có các yêu cầu về bảo mật và khả năng khôi phục dữ liệu trong trường hợp sự cố cũng cần được đáp ứng.

Về giao diện web, cần được bố cục rõ ràng, mạch lạc, hiển thị đầy đủ các biểu đồ dữ liệu chứng khoán và các chức năng quản lý người dùng.

2.2.2 Yêu cầu phi chức năng

Về các yêu cầu phi chức năng, nền tảng cần hoạt động ổn định, dữ liệu cập nhật định kì. Giao diện web màu sắc hiện đại, bắt mắt. Nền tảng cần phản hồi nhanh với yêu cầu của người dùng, giảm thiểu tình trạng treo, đơ gây bất tiện trong quá trình

người dùng tương tác với hệ thống.

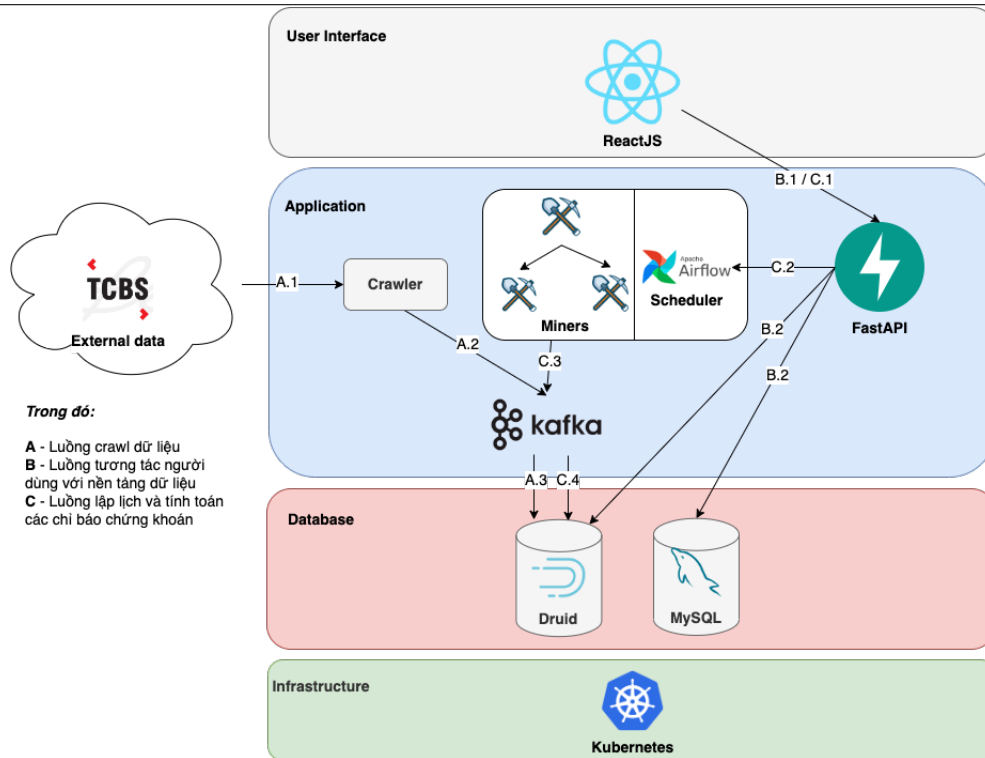
Hệ thống lưu trữ dữ liệu đảm bảo tính phân tán và tính chịu lỗi. Ngoài ra nền tảng cần đáp ứng khả năng mở rộng tài nguyên trong trường hợp số lượng người dùng tăng lên vượt quá khả năng sẵn có.

2.2.3 Thiết kế hệ thống

Với các yêu cầu chức năng và phi chức năng trên, nhóm chúng tôi có thống nhất và đưa ra mô hình thiết kế hệ thống gồm các phân hệ như sau:

- Phân hệ ứng dụng và quản lý nền tảng: bao gồm 3 thành phần chính là một webserver được xây dựng bằng FastAPI[5] cho backend và ReactJS[6] cho frontend, một cơ sở dữ liệu người dùng được triển khai bằng MySQL Server .
- Phân hệ quản lý và lập lịch trình tính toán chỉ báo: bao gồm các trình tính toán chỉ báo (Miner). Các miner này được lập lịch và quản lý bởi Apache Airflow.
- Phân hệ thu thập, tổ chức lưu trữ và cảnh báo chứng khoán: bao gồm 2 thành phần chính là Kafka và Druid. Các dữ liệu của hệ thống được lưu toàn bộ trong Druid. Kafka được sử dụng như một vùng đệm cho dữ liệu, trước khi đẩy vào Druid.
- Phân hệ hạ tầng triển khai Kubernetes: đóng vai trò kết nối các phân hệ còn lại trong hệ thống, đóng gói chúng và triển khai trên cùng một nền tảng.

Dưới đây là sơ đồ thiết kế tổng quan các phân hệ của toàn bộ nền tảng dữ liệu chứng khoán mà nhóm xây dựng:



Hình 2.1: Mô hình hệ thống cho nền tảng dữ liệu

Mô hình trên là minh họa cho luồng hoạt động cũng như các thành phần được sử dụng trong hệ thống.

Luồng thu thập (crawl) dữ liệu chứng khoán hoạt động bằng cách lấy dữ liệu từ trang TCBS thông qua thư viện "vnstock" của Python. Dữ liệu sau khi được thu thập sẽ được đưa vào hệ thống hàng đợi phân tán Apache Kafka rồi đi tới lưu trữ cố định tại Apache Druid - nền tảng lưu trữ dữ liệu chuỗi thời gian.

Với phần tính năng tính toán các chỉ báo chứng khoán, các Miner (trình tính toán) được xây dựng và lập lịch bởi Apache Airflow. Các trình tính toán này lấy dữ liệu từ Druid, tính toán theo công thức và trình tự được định sẵn, sau đó ghi lại dữ liệu ra các Kafka topic. Dữ liệu sau đó tiếp tục được lưu trữ lâu dài trên Apache Druid.

Người dùng thực hiện tương tác với nền tảng thông qua giao diện web phát triển bằng framework ReactJS. Giao diện gửi và nhận thông tin với máy chủ web được triển khai với framework FastAPI của Python. Để có được dữ liệu giao dịch thị trường và chỉ báo chứng khoán hiển thị cho người dùng, máy chủ web gửi các truy vấn lấy dữ liệu đến Apache Druid và MySQL.

Mỗi phân hệ đều đóng các vai trò quan trọng trong hệ thống, nhằm đảm bảo mọi yêu cầu về chức năng hay phi chức năng đã nêu trên.

CHƯƠNG 3. CÔNG NGHỆ SỬ DỤNG

Ở chương trước, đồ án đã cung cấp những thông tin tổng quan về TTCK cũng như thiết kế tổng thể các phân hệ chính cùng với chức năng của nền tảng dữ liệu TTCK mà nhóm xây dựng. Đến chương này, đồ án sẽ đi vào giới thiệu những công nghệ được sử dụng trong quá trình xây dựng phân hệ quản lý và lập lịch các trình tính toán chỉ báo.

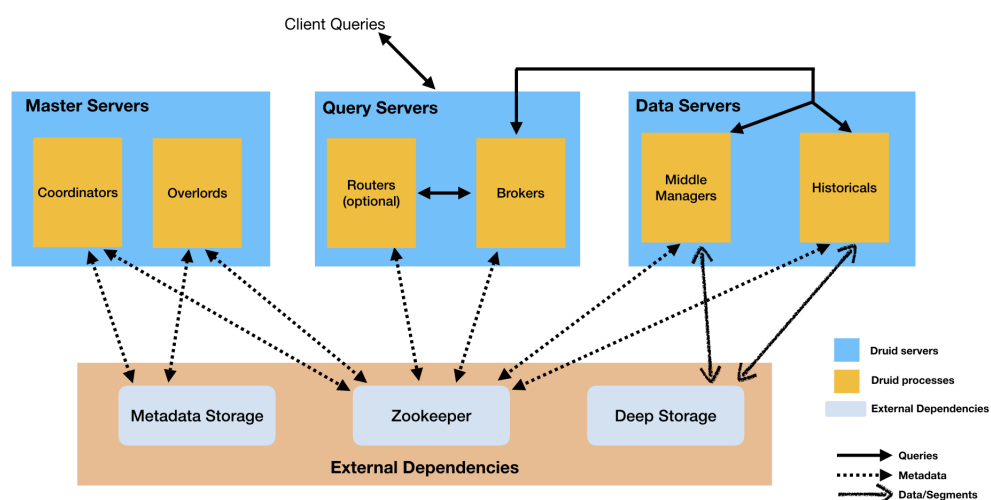
3.1 Công nghệ lưu trữ dữ liệu

Dữ liệu thu thập từ thị trường chứng khoán là dữ liệu theo chuỗi thời gian (time-series). Để lưu trữ và truy vấn hiệu quả dữ liệu này, Apache Druid và Apache Kafka đã được sử dụng trong sản phẩm này.

3.1.1 Apache Druid[3]

Apache Druid là cơ sở dữ liệu chuỗi thời gian (time-series) với những ưu điểm là thời gian truy vấn nhanh chóng, khả năng mở rộng và chịu lỗi. Druid thường được sử dụng vào những yêu cầu về dữ liệu thời gian thực (real-time), truy vấn dữ liệu tốc độ nhanh và đảm bảo hệ thống luôn có thể đáp ứng được nhu cầu tính toán.[3]

Dữ liệu khi được đọc và lưu trữ bởi Druid được chia thành các phân mảnh (segment) dựa theo khoảng thời gian. Trong một khoảng thời gian có thể có nhiều segment nếu Druid thực hiện đọc dữ liệu từ nhiều nguồn. Segment là một đơn vị xử lý dữ liệu trong Apache Druid. Để lưu trữ và quản lý các segment này, các thành phần kiến trúc tổng quan của Druid được mô tả như hình vẽ sau.



Hình 3.1: Kiến trúc và các thành phần của Apache Druid [7]

Trong đó, vai trò của mỗi thành phần của Apache Druid được định nghĩa như sau:

- *Deep Storage* (Kho lưu trữ sâu) là thành phần lưu trữ tất cả dữ liệu đã được hệ thống Druid nhận vào. Kho lưu trữ này được chia sẻ quyền truy cập đến tất cả các Druid server trong hệ thống. Khi được triển khai trên các cụm, Deep Storage có thể là các hệ thống S3 hay HDFS. Còn khi triển khai trên một máy, nó sẽ sử dụng bộ nhớ trên ổ đĩa máy tính.

Deep Storage là một thành phần quan trọng đảm bảo tính chất chịu lỗi của Druid.

- *Coordinator* (Người điều phối) là tiến trình chịu trách nhiệm chính trong việc quản lý và phân phối các phân mảnh dữ liệu (segment). Cụ thể hơn, Coordinator gửi và nhận tín hiệu từ Historical để thực hiện nhận một segment mới hay xóa bỏ các segment hết hạn, đảm bảo các segment này được đọc bởi nhiều node Historical.
- *Overlord* (Lãnh chúa) là tiến trình làm nhiệm vụ chấp thuận các tác vụ nhận dữ liệu vào Druid. Overlord có thể được chạy ở hai chế độ local (tại chỗ) và remote (từ xa). Khi chạy ở chế độ local, Overlord chịu trách nhiệm tạo các Peons để thực thi các tác vụ nhận dữ liệu. Overlord khi hoạt động local cần sự trợ giúp của Middle Manager.
- *Broker* (Người môi giới) là tiến trình nhận các truy vấn từ máy khách (client). Họ xác định những node Historical và MiddleManager nào đang phục vụ các segment đó và gửi một truy vấn con cho mỗi tiến trình này. Sau đó, nó thu thập và hợp nhất kết quả và trả lời client. Bên trong, các tiến trình Historical phản hồi với các truy vấn con tương ứng với các segment dữ liệu đã được lưu trữ xuống deep storage, trong khi tiến trình Middle Manager phản hồi các truy vấn con tương ứng với dữ liệu mới được nhập, chưa được lưu tại **Deep Storage**).
- *Router* (Bộ định tuyến) đóng vai trò định tuyến các yêu cầu tới các Broker, Coordinator, Overlord bằng cách cung cấp một cổng API thống nhất trước các Broker, Coordinator, Overlord.
- *Historical* (Lịch sử) là tiến trình xử lý lưu trữ dữ liệu có thể truy vấn được. Các node đang chạy các tiến trình Historical tìm nạp các segment từ Deep Storage đến đĩa cục bộ của chúng và trả lời các truy vấn về các phân đoạn này.
- *Middle Manager* (Quản lý trung gian) quản lý các tác vụ nhập dữ liệu vào cụm, chịu trách nhiệm kết nối đến các hệ thống lưu trữ dữ liệu. Middle Manager sẽ

chuyển tiếp các tác vụ đến thực thi tại các Peon (một tiến trình thực thi việc nhập dữ liệu). Mỗi Peon chỉ chịu trách nhiệm cho một tác vụ, mỗi Middle Manager có thể quản lý nhiều Peons cùng lúc.

Một khái niệm nữa của Druid được nhắc đến rất nhiều trong đề án này, đó là Druid datasources [8]. Druid datasources là đối tượng để thực hiện truy vấn dữ liệu. Kiểu datasource phổ biến nhất là datasource dạng bảng. Đôi khi nhắc đến datasource sẽ được ngầm hiểu là datasource dạng bảng. Mỗi trường dữ liệu (cột) Druid datasource dạng bảng thuộc một trong ba loại sau:

- **Primary timestamp:**

Do Apache Druid là cơ sở dữ liệu time-series, nên trong mỗi bảng bắt buộc có một cột timestamp. Cột timestamp này được Druid sử dụng để phân mảnh và sắp xếp dữ liệu trong bảng.

- **Dimension:**

Cột thuộc kiểu Dimension lưu trữ thông tin trạng thái của dữ liệu. Những cột này có thể áp dụng những thao tác groupby, filter.

- **Metrics:**

Cột Metrics lưu dữ liệu dạng số. Những cột này có thể được áp dụng các hàm tính toán như max(), min(), avg(), sum(), ...

3.1.2 Apache Kafka[4]

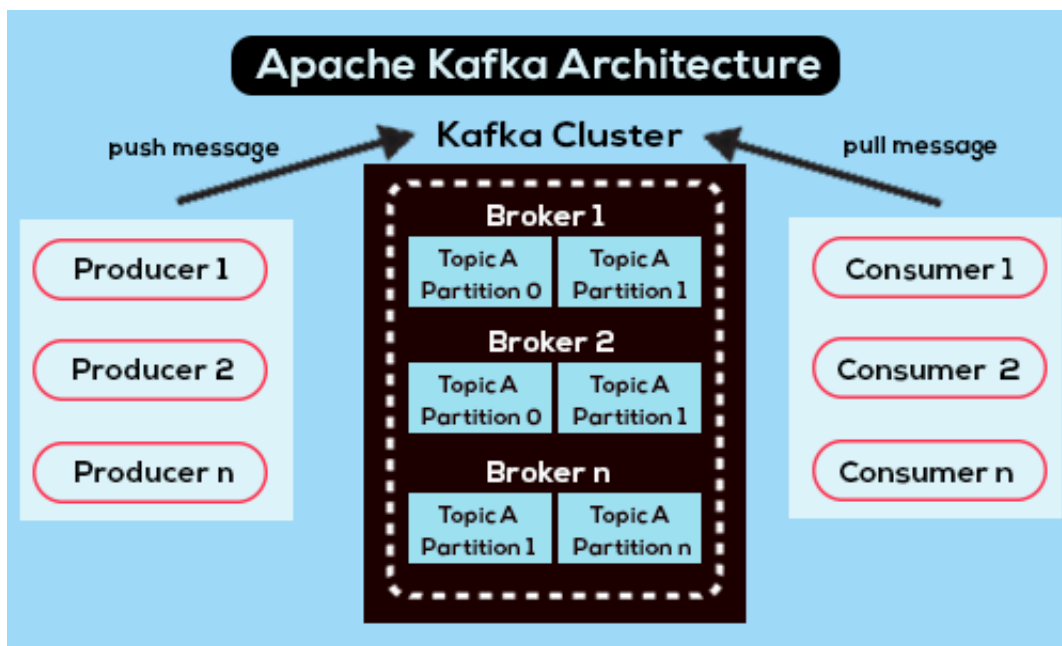
Apache Kafka là nền tảng streaming (trực tiếp) dữ liệu sự kiện. "Dữ liệu sự kiện" ở đây có nghĩa rằng đối tượng mà Kafka quan tâm là những gì đang xảy ra xung quanh chúng ta. Mỗi sự kiện, mỗi điều xảy ra đều có thể được chuyển thành các bản ghi và được lưu trữ trong Kafka.

Là một nền tảng streaming, dữ liệu trong Kafka được xử lý ngay lập tức. Kafka hỗ trợ kết nối đến các nguồn sinh dữ liệu (một máy chủ web, một ứng dụng di động hay một tiến trình nào đó) và liên tục nhận về dữ liệu dạng sự kiện từ các nguồn này theo dạng hàng đợi tin (message queue). Đồng thời, Kafka cũng có thể được kết nối đến các đích lưu trữ hay sử dụng dữ liệu, ví dụ như dữ liệu từ các nguồn đi qua Kafka và được tiếp tục đưa vào các hệ thống lưu trữ lâu dài như các cơ sở dữ liệu hoặc các ứng dụng phân tích.

Về mặt lưu trữ, một đơn vị dữ liệu trong Kafka được gọi là message (dòng tin). Nếu tiếp cận Kafka từ góc nhìn của nền tảng cơ sở dữ liệu quan hệ thì có thể coi message tương tự như một hàng (row) trong một bảng dữ liệu. Một message chỉ đơn giản là một mảng byte, vì vậy dữ liệu chứa trong đó không có định dạng cụ

thể.

Dưới đây là hình vẽ mô tả kiến trúc và các thành phần tổng quan của hệ thống Apache Kafka.



Hình 3.2: Kiến trúc và các thành phần của Apache Kafka

Source: <https://www.projectpro.io/article/apache-kafka-architecture-/442>

- *Topic* (Chủ đề) là cách mà Kafka tổ chức lưu trữ các message của mình. Vĩn theo cách nhìn từ cơ sở dữ liệu quan hệ, nếu mỗi message là một hàng trong bảng, thì topic sẽ chính là một bảng dữ liệu. Một topic được định danh bởi tên của nó và một cụm Kafka có thể có nhiều topic. Thông thường, một topic được chia thành nhiều phân vùng nhỏ (partition). Các message được viết vào các topic sẽ luôn luôn được thêm vào cuối hàng đợi. Các message với cùng key sẽ được ghi vào cùng một phân vùng.
- *Producer* (Nhà sản xuất) là thành phần đưa dữ liệu từ nguồn (máy chủ web, ứng dụng, ...) vào Kafka dưới dạng các message. Producer khi ghi dữ liệu cần chỉ rõ topic mong muốn, tuy nhiên theo mặc định, các producer sẽ không quan tâm việc các message được cho vào phân vùng nào của topic. Trong một số trường hợp, producer sẽ gửi message đến các phân vùng cụ thể bằng cách sử dụng khóa. Trình phân vùng sẽ tạo ra một hàm băm của khóa và ánh xạ nó tới một phân vùng cụ thể. Điều này đảm bảo rằng tất cả các message được tạo bằng một khóa sẽ được ghi vào cùng một phân vùng duy nhất.
- *Consumer* (Người tiêu dùng) là thành phần đọc các message từ các topic. Consumer đăng ký một hoặc nhiều topic và đọc các message theo thứ tự mà chúng được tạo ra. Consumer giữ việc theo dõi những message mà nó đã đăng

kí bằng cách theo dõi các offset (là giá trị số nguyên tăng lên mỗi khi message được tạo ra). Mỗi message trong một phân vùng đều giá trị offset là duy nhất. Việc sử dụng giá trị offset này giúp consumers có thể dừng hay khi khởi động lại có thể đọc tiếp các message mà không bị mất vị trí đã đọc được trước đó.

- *Broker* là một máy chủ Kafka đơn và một thành phần trong một cụm Kafka. Broker nhận các message từ các producer, gán offsets cho chúng và lưu trữ chúng trên ổ đĩa. Nó cũng phục vụ các consumer, đáp ứng yêu cầu tìm phân vùng và trả lại các message trong ổ đĩa. Tùy vào phần cứng cụ thể và các cấu hình mà một broker có thể xử lý hàng ngàn phân vùng và hàng triệu message mỗi giây. Trong một cụm các broker sẽ có một broker hoạt động như cluster controller (quản lý cụm) - được bầu tự động từ các broker cụm. Controller chịu trách nhiệm cho các hoạt động quản trị, bao gồm việc gán các phân vùng cho các broker và giám sát các lỗi của chúng. Một phân vùng có thể được gán cho nhiều broker, điều này sẽ dẫn đến phân vùng có thể được nhân bản. Điều này cho phép một broker khác trở thành leader nếu broker leader hiện tại dừng hoạt động.

Khi cài đặt Kafka trên cụm với nhiều máy tính, mỗi máy tính sẽ đóng vai trò là Broker (khái niệm này sẽ được giải thích ở phần tiếp theo). Dữ liệu đưa vào Kafka sẽ được nhân bản và lưu trên các Broker khác nhau, số lượng nhân bản tùy thuộc vào cấu hình hệ thống, cho thấy *tính phân tán* trong lưu trữ dữ liệu của Kafka. Do vậy nếu có một Broker dừng hoạt động, dữ liệu sẽ không bị mất mát bởi đã được sao lưu trên các Broker khác, đảm bảo *tính chịu lỗi* của hệ thống. Đồng thời việc mở rộng phạm vi cụm Kafka bằng việc bổ sung các Broker cũng không hề phức tạp, không ảnh hưởng đến các Broker sẵn có đang hoạt động, thể hiện *tính khả mở* của công nghệ Apache Kafka.

Tất cả những đặc tính đó khiến cho Kafka trở thành một hệ thống truyền thông điệp có hiệu năng vượt trội dưới khối lượng dữ liệu tải lớn. Do vậy công nghệ này được lựa chọn để sử dụng triển khai trong nền tảng dữ liệu TTCK của nhóm.

3.2 Công nghệ xử lý dữ liệu - Python[1] và pandas [9]

Với việc xử lý dữ liệu, ngôn ngữ Python được sử dụng cho các thao tác xử lý dữ liệu trong phân hệ quản lý và lập lịch trình tính toán chỉ báo.

Python là ngôn ngữ lập trình hướng đối tượng bậc cao được giới thiệu từ những năm cuối của thế kỉ XX và ngày càng được sử dụng phổ biến trong nhiều lĩnh vực lập trình. Những tác vụ thường được các lập trình viên thực hiện bằng Python như đọc, ghi và xử lý dữ liệu, triển khai các thuật toán Machine Learning hay xây dựng, huấn luyện các mô hình Deep Learning. Với số lượng thư viện hỗ trợ rất lớn,

Python đã thể hiện rõ lợi thế của mình trong các tác vụ xử lý dữ liệu.

Do vậy các bước xử lý và tính toán các chỉ báo trong phân hệ này được cụ thể hóa bằng ngôn ngữ Python, cụ thể là sử dụng thư viện *pandas*. *pandas* là thư viện được phát triển phục vụ cho thao tác xử lý và phân tích dữ liệu dạng bảng và chuỗi thời gian, rất phù hợp với yêu cầu đặt ra của hệ thống.

Hai thành phần quan trọng nhất của *pandas* là *Series* và *DataFrame*. Trong khi *Series* là lớp đối tượng dữ liệu dạng cột, thì *DataFrame* là cấu trúc dữ liệu đa chiều dạng bảng gồm các hàng và cột. Hay nói cách khác, một đối tượng *DataFrame* là tập hợp của các đối tượng *Series*.

| Series | | | Series | | | DataFrame | |
|--------|--------|---|--------|---------|---|-----------|---------|
| | apples | | | oranges | | | |
| 0 | 3 | + | 0 | 0 | = | 0 | 3 |
| 1 | 2 | | 1 | 3 | | 1 | 2 |
| 2 | 0 | | 2 | 7 | | 2 | 0 |
| 3 | 1 | | 3 | 2 | | 3 | 1 |
| | | | | | | | oranges |
| | | | | | | | 0 |
| | | | | | | | 3 |
| | | | | | | | 7 |
| | | | | | | | 2 |

Hình 3.3: Mối tương quan giữa *Series* và *DataFrame* trong *pandas*

Source: <https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/>

pandas cung cấp nhiều API để xử lý dữ liệu được thể hiện ở dạng *Series* và *DataFrame*. Ngoài ra, *pandas* cũng hỗ trợ thao tác đọc, ghi dữ liệu từ các định dạng file phổ biến như CSV, JSON hay từ SQL database.

Phân hệ quản lý và lập lịch trình tính toán chỉ báo không có yêu cầu quá cao về lượng dữ liệu cần xử lý trong một chu kỳ hoạt động, ngoài ra các dữ liệu cũng đều thuộc dạng có cấu trúc, chủ yếu là dữ liệu dạng số và chuỗi thời gian. Vậy nên việc lựa chọn Python và thư viện *pandas* trong quá trình phát triển phân hệ là phù hợp.

3.3 Công nghệ lập lịch cho các trình tính toán - Apache Airflow [2]

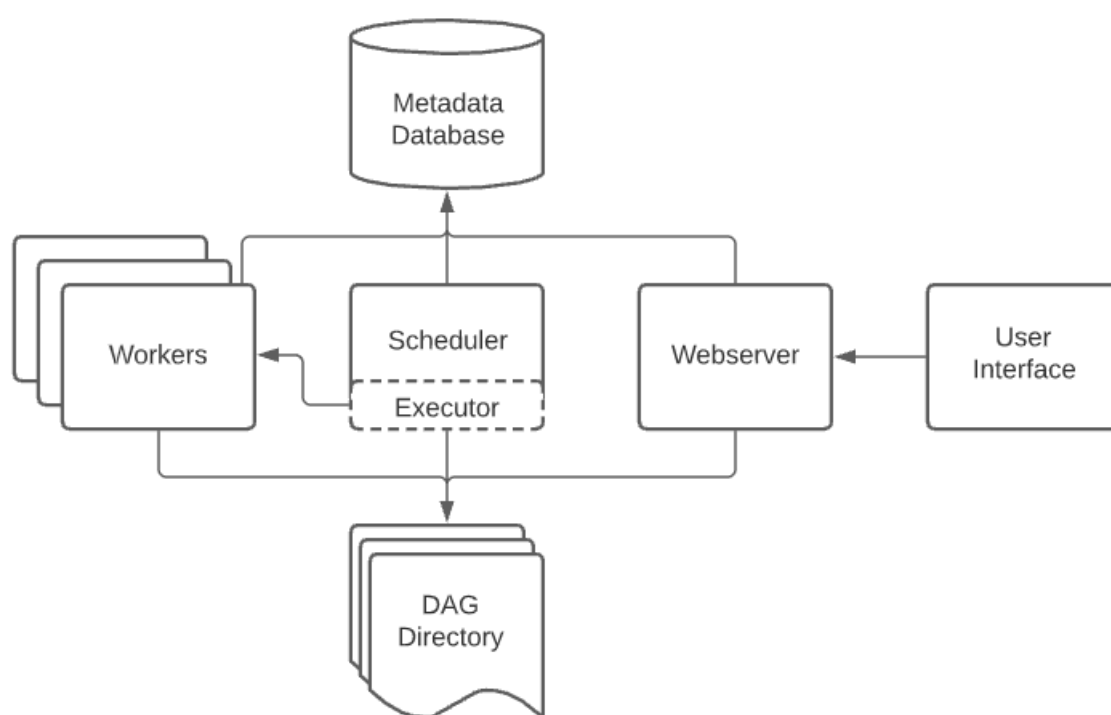
Ngôn ngữ Python và thư viện *pandas* đã được lựa chọn cho các bước xử lý dữ liệu và tính toán giá trị của các chỉ báo. Ngoài ra, để lập lịch cho các trình tính toán chỉ báo thực hiện theo đúng tần suất và thứ tự, phân hệ sử dụng công nghệ Apache Airflow.

Apache Airflow là một công cụ lập trình mã nguồn mở (open-source) phục vụ cho mục đích lập lịch và quản trị các luồng xử lý dữ liệu. Airflow khởi nguồn là một dự án mã nguồn mở tại Airbnb, trong bối cảnh các trình xử lý dữ liệu của công ty ngày càng gia tăng về mặt số lượng, các kỹ sư tại đây đặt ra yêu cầu về một công

cụ giúp lập lịch và quản lý những luồng tính toán đó. Aiflow chính thức gia nhập Apache Foundation Incubator vào năm 2016 và tiếp tục được phát triển cho đến ngày hôm nay.

Khi sử dụng Airflow, hai đối tượng mà người dùng thường xuyên làm việc và tương tác là *Task* và *DAG*. *DAG* (Directed Acyclic Graph - Đồ thị có hướng không chu trình) là một khái niệm căn bản của Airflow. Đồ thị này đại diện cho một luồng công việc (workflow) gồm đỉnh là các Task, các cạnh có hướng mũi tên thể hiện sự phụ thuộc và quan hệ trước sau giữa các Task. Mỗi *Task* là một thành phần trong một DAG, mô tả một bước trong workflow. Mỗi lượt mà một Task và một DAG được khởi chạy, sẽ tạo ra các đối tượng *Task Instance* và *DAG Run*, bao gồm các thông tin về thời gian, trạng thái của mỗi lần khởi chạy của Task và DAG.

Về mặt kiến trúc hệ thống, Apache Airflow gồm các thành phần chính với các vai trò khác nhau tạo nên một hệ thống hoạt động hoàn thiện.



Hình 3.4: Kiến trúc và các thành phần của Apache Airflow [10]

Các thành phần được mô tả trong 3.4 có vai trò cụ thể như sau:

- *Webserver* là một server được phát triển bằng Fast và Gunicorn để hiển thị thông tin giao diện web cho người sử dụng.
- *Scheduler* là tiến trình Python đa luồng chịu trách nhiệm lập lịch cho các Task và DAG trong hệ thống. Cụ thể, Scheduler quyết định xem Task nào được thực

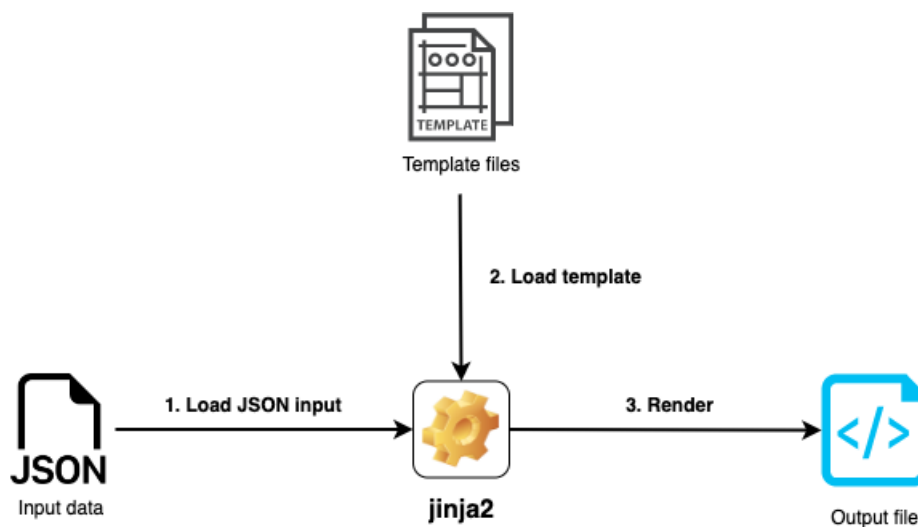
thi, thực thi vào thời điểm nào cũng như thực thi trên tài nguyên nào.

- *Executor* là cơ chế xác định nguồn tài nguyên tính toán được sử dụng để thực thi các Task. Executor chạy như một thành phần trong Scheduler mỗi khi hệ thống Airflow hoạt động.
- *Workers* là các tiến trình chịu trách nhiệm thực thi các Task, theo sự điều phối của Executor.
- *Metadata Database* cơ sở dữ liệu lưu trữ thông tin của Task và DAG. Thông thường cơ sở dữ liệu của Airflow sẽ được triển khai với PostgreSQL, ngoài ra MySQL, MsSQL hay SQLite cũng được hỗ trợ sử dụng.

3.4 Công nghệ tự động sinh tệp tin theo mẫu - jinja[11]

Một chức năng quan trọng khác của phân hệ quản lý và lập lịch trình tính toán cũng như toàn bộ nền tảng là cho phép người dùng tạo mới trình tính toán chỉ báo theo công thức cá nhân. Để tạo ra một chỉ báo mới và thêm vào luồng các trình tính toán khác đang hoạt động, phân hệ cần tạo mới các tệp tin định nghĩa lớp chỉ báo mới, tệp tin chứa thông tin DAG mới. Với ngôn ngữ Python, thư viện *jinja* hỗ trợ chức năng tạo mới tệp tin này và được sử dụng trong quá trình phát triển phân hệ.

jinja là một công cụ sinh tệp tin (file) dựa theo bản mẫu (template). Mỗi template thể hiện nội dung file đầu ra với các phần nội dung cố định và những phần nội dung tùy biến cho mỗi lần sinh file. Mỗi khi có dữ liệu đầu vào và yêu cầu tạo file, jinja sẽ tạo ra file mới bằng cách điền những thông tin đầu vào vào những vị trí thích hợp được định nghĩa trong template. Quá trình thư viện jinja sinh file được mô tả trong sơ đồ sau:



Hình 3.5: Các bước tạo file theo template của jinja

3.5 Công nghệ đóng gói và triển khai phần mềm - Docker[12]

Trong quá trình triển khai các thành phần của hệ thống Apache Airflow, thay vì triển khai bằng việc cài đặt thuần túy trên máy, tôi sử dụng Docker để giúp cho việc khởi chạy các thành phần của Airflow dễ dàng hơn.

Cụ thể, Docker là một nền tảng mở cho phép lập trình viên phát triển, khởi chạy và vận chuyển ứng dụng. Docker cho phép người dùng tách biệt ứng dụng với nền tảng phần cứng của thiết bị, tạo ra môi trường ảo hóa với tài nguyên riêng để các ứng dụng thực thi trong đó.

Khi làm việc với Docker, hai khái niệm cần quan tâm nhất là *Docker Image* và *Docker Container*. Như đã nói, Docker là nền tảng cho phép khu biệt tài nguyên và tạo ra môi trường ảo hóa với đầy đủ tài nguyên và môi trường để ứng dụng hoạt động. Một môi trường như vậy là một *Docker Container*.

Vậy còn **Docker Image**, đây là một template chứa thông tin hướng dẫn cách tạo ra Container. Nói cách khác, trong Docker Image sẽ gồm các bước cấu hình cho các thành phần của Container và Container chính là thành quả khi chạy Image. Docker Image có thể được lưu trữ, gửi và nhận một cách dễ dàng dưới các dạng tệp nén. Điều này giúp việc triển khai ứng dụng trên nhiều môi trường khác nhau rất thuận lợi, bởi thực chất ứng dụng chỉ hoạt động trong không gian Container của nó mà thôi.

Trong phân hệ quản lý và lập lịch trình tính toán, các thành phần của Airflow như web-server, scheduler, worker đều được triển khai trong các Docker Container riêng biệt.

CHƯƠNG 4. XÂY DỰNG PHÂN HỆ QUẢN LÝ VÀ LẬP LỊCH TRÌNH TÍNH TOÁN CHỈ BÁO

Nội dung của chương 3 đã đề cập đến các công nghệ được sử dụng trong phân hệ quản lý và lập lịch trình chỉ báo. Đến chương này, đồ án sẽ trình bày nội dung xây dựng các yêu cầu chức năng và phi chức năng của phân hệ.

Các chức năng của phân hệ quản lý và lập lịch các trình tính toán chỉ báo bao gồm: tính toán các chỉ báo theo công thức, lập lịch các trình tính toán và tự động tạo mới trình tính toán theo yêu cầu của người dùng. Các chức năng này sẽ được mô tả chi tiết ở phần sau.

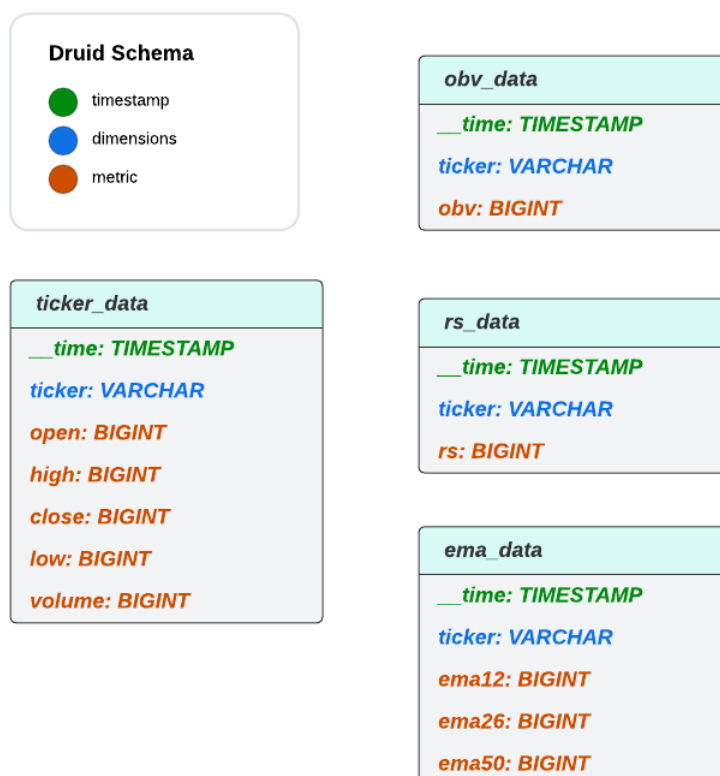
4.1 Tính toán chỉ báo theo công thức

Ở mục 2.1.2, đồ án đã đề cập đến các chỉ báo phổ biến trên TTCK. Đặc điểm chung của các chỉ báo là thường được tính theo các công thức dưới dạng hồi quy. Giá trị của chỉ báo trong chu kỳ này được tính toán dựa theo các giá trị trong các chu kỳ trước.

4.1.1 Thiết kế cơ sở dữ liệu

Dữ liệu phục vụ cho chức năng tính toán chỉ báo bao gồm: dữ liệu giao dịch và dữ liệu các chỉ báo đã được tính toán. Chúng được lưu trên Apache Druid trong các datasource dạng bảng riêng biệt. Như đã đề cập và giới thiệu ở mục 3.1.1, mỗi trường dữ liệu (cột) trong Druid datasource dạng bảng thuộc một trong ba loại: *Primary timestamp*, *Dimension*, *Metrics*.

Dưới đây là thiết kế mô hình dữ liệu lưu trữ trên Apache Druid gồm các bảng dữ liệu giao dịch ("ticker_data") và dữ liệu chỉ báo ("obv_data", "rs_data", "ema_data").

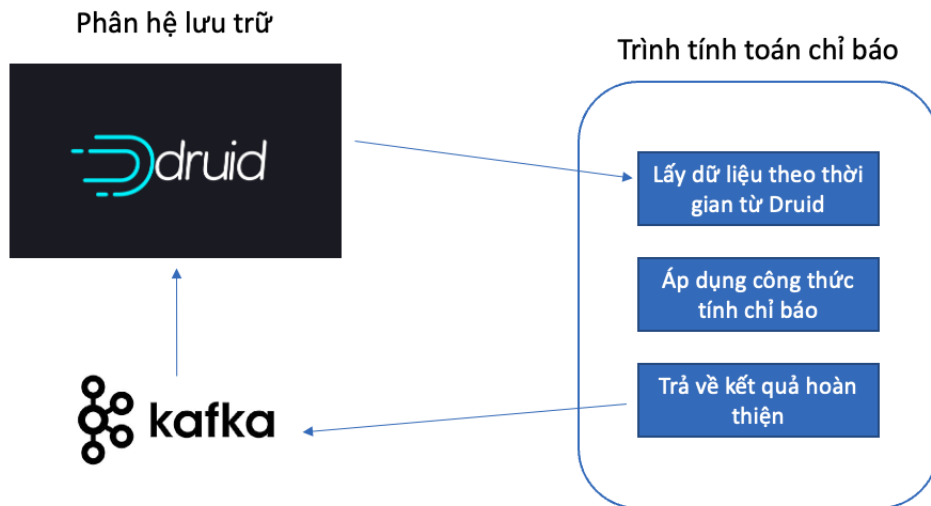


Hình 4.1: Apache Druid data model

Apache Druid không hỗ trợ ghi trực tiếp dữ liệu vào datasource dạng bảng. Do vậy khi tạo datasource trên Druid, cần cấu hình để datasource đọc dữ liệu streaming từ một topic Kafka cụ thể. Topic này đóng vai trò vùng đệm, khi dữ liệu được đẩy vào topic sẽ lập tức được đẩy vào datasource đã cấu hình.

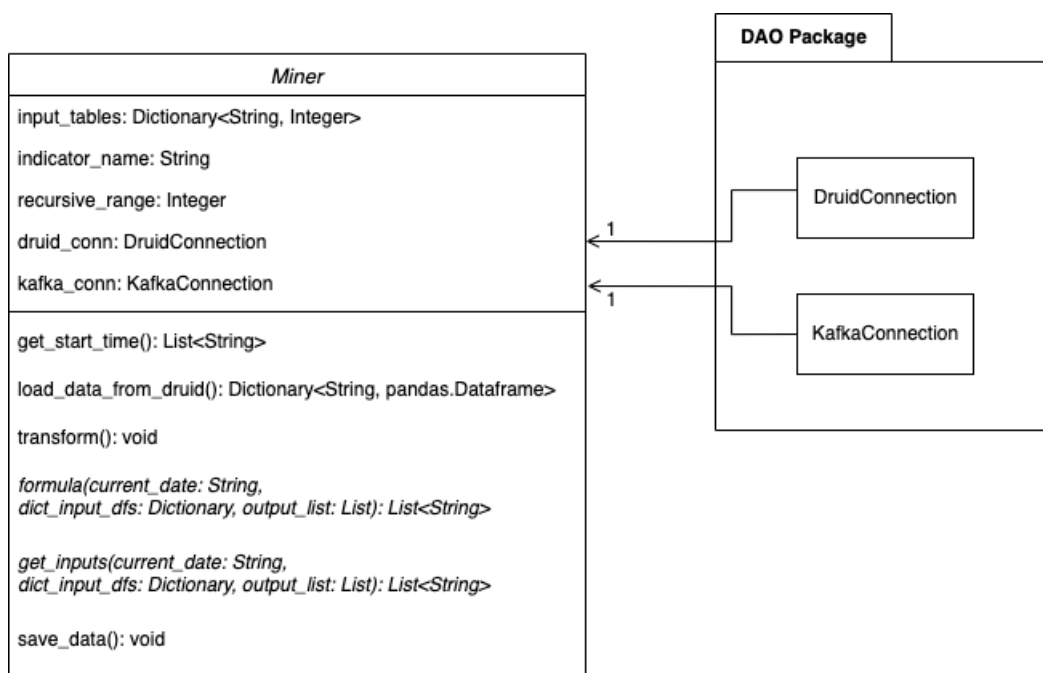
4.1.2 Thiết kế luồng tính toán

Với yêu cầu tính toán như vậy, luồng thực thi của một trình tính toán được thể hiện qua sơ đồ sau:



Hình 4.2: Luồng thực thi của một trình tính toán chỉ báo

Dựa vào luồng thực thi này, các class chính được xây dựng theo biểu đồ lớp dưới đây:



Hình 4.3: Biểu đồ các lớp Miner, DruidConnection và KafkaConnection

Ở biểu đồ 4.3, class Miner thể hiện đối tượng trình tính toán chỉ báo, các class DruidConnection và KafkaConnection thuộc package DAO phục vụ thao tác kết nối đến các thành phần thuộc phân hệ lưu trữ (Apache Druid và Apache Kafka).

4.1.3 Đặc tả lớp Miner

Lớp trừu tượng (abstract class) Miner thể hiện cho đối tượng trình tính toán chỉ báo. Các thuộc tính và phương thức được mô tả trong hình sau:

| <i>Miner</i> |
|---|
| <code>input_tables: Dictionary<String, Integer></code> <code>indicator_name: String</code> <code>recursive_range: Integer</code> <code>druid_conn: DruidConnection</code> <code>kafka_conn: KafkaConnection</code> |
| <code>execute(): void</code> <code>get_start_time(): List<String></code> <code>load_data_from_druid(): Dictionary<String, pandas.DataFrame></code> <code>transform(): void</code> <code>formula(current_date: String, dict_input_dfs: Dictionary, output_list: List): List<String></code> <code>get_inputs(current_date: String, dict_input_dfs: Dictionary, output_list: List): List<String></code> <code>save_data(): void</code> |

Hình 4.4: Biểu đồ lớp Miner

- Các thuộc tính

- **input_tables:**

Mỗi công thức để tính toán chỉ báo gồm các giá trị đầu vào, những giá trị này được lưu trữ trong một hoặc nhiều bảng dữ liệu.

Thuộc tính này liệt kê các bảng dữ liệu chứa giá trị đầu vào phục vụ cho việc tính toán các công thức.

Ví dụ với công thức tính chỉ báo OBV:

$$OBV_i = OBV_{i-1} + \text{sgn}(C_i - C_{i_1}) * V_i$$

Giá trị OBV được tính dựa trên giá đóng cửa và khối lượng giao dịch. Vậy nên thuộc tính input_tables sẽ là danh sách gồm bảng lưu dữ liệu giao dịch (ticker_data).

- **indicator_name:**

Thuộc tính kiểu String thể hiện tên của chỉ báo, không gồm kí tự space. Ví dụ: "obv", "ad", "rs", "ema", ...

– **recursive_range:**

Công thức tính chỉ báo có đặc thù thường được viết dưới dạng hồi quy. Thuộc tính này thể hiện số chu kỳ tối đa mà chỉ báo được tính toán hồi quy.

Vẫn xét ví dụ chỉ báo OBV phía trên, giá trị recursive_range = 1. Với các chỉ báo được tính từ những công thức không hồi quy, giá trị của thuộc tính bằng 0.

– **druid_conn và kafka_conn:**

Để phục vụ cho thao tác đọc ghi dữ liệu từ các hệ thống lưu trữ, class DruidConnection và KafkaConnection được xây dựng gồm các phương thức kết nối đến hai hệ thống Apache Druid và Apache Kafka.

| DruidConnection |
|--|
| druid_host: String |
| druid_coordinator: String |
| query: PyDruid |
| load_ticker_data(start_date: String, end_date: String, ticker_table: String): pandas.DataFrame |
| load_indicator_data(indicator_table: String, start_date: String, end_date: String, metric: String): pandas.DataFrame |
| create_streaming_datasource(): void |

Hình 4.5: Lớp DruidConnection

| KafkaConnection |
|--|
| bootstrap_servers: String |
| admin_client: KafkaAdminClient |
| producer: KafkaProducer |
| consumer: KafkaConsumer |
| create_new_topic(topic_names: List<String>): void |
| get_existed_list_topics(): List<String> |
| produce_df_to_kafka(topic: String, df: pandas.DataFrame): void |

Hình 4.6: Lớp KafkaConnection

Hai thuộc tính "druid_conn" và "kafka_conn" là instance của các lớp tương ứng.

- Các phương thức

Các phương thức trong class Miner mô tả các bước trong luồng tính toán chỉ báo.

– **get_start_time(indicator_name):**

Phương thức trả về phiên giao dịch mà bắt đầu từ đó chỉ báo được tính toán.

Nếu chỉ báo chưa từng được lưu trữ, trình tính toán sẽ bắt đầu từ phiên giao dịch đầu tiên của năm 2023. Ngược lại, trình tính toán sẽ tính từ phiên giao dịch gần nhất mà chỉ báo đã được lưu trữ trong hệ thống.

Algorithm 1: Mã giả hàm `get_start_time()`

```
indicator_name start_date
list_datasources ← All existing Druid datasources
if indicator_name not in list_datasources then
    | start_date ← "2023 - 01 - 01"
else
    | indicator_data ← query_druid_data(indicator_name)
    | last_date ← get_last_date_from_data(indicator_data)
    | start_date ← get_next_date(last_date)
```

– **load_data_from_druid()**

Dựa vào thời gian lấy được từ phương thức **get_start_time()**, dữ liệu tương ứng được lấy ra từ Apache Druid.

Algorithm 2: Mã giả hàm `load_data_from_druid()`

```
start_date, input_tables data_map data_list = []
if "ticker_data" is in input_tables then
    | ticker_df ← query_data_from_ticker_table()
    | AddKeyValue(data_map, "ticker_data", ticker_df)
    | RemoveItem(input_tables, "ticker_data")
for indicator in input_tables do
    | indicator_df ← query_data_from_indicator_table(indicator)
    | AddKeyValue(data_map, indicator, indicator_df)
```

– **get_inputs():**

Các giá trị đầu vào cần cho việc tính toán theo công thức được trích xuất từ các bảng dữ liệu đầu vào.

Hàm `get_inputs()` được định nghĩa theo dạng abstract method. Các trình tính toán khi được khai báo sẽ tự định nghĩa lại phương thức này.

Algorithm 3: Mã giả hàm `get_inputs()`

```
current_date, data_map input_value_list
if current_date is valid then
    | for data in data_map.values() do
    | | input_value ← get_input_value(data)
    | | AddItem(input_value_list, input_value)
else
    | input_value_list ← NULL
```

– **formula()**:

Phương thức này triển khai công thức tính toán chỉ báo, dựa theo từng chỉ báo và từng phiên giao dịch.

Việc tính toán dựa theo các giá trị đầu vào được trả về từ hàm `get_inputs()`. Phương thức `formula()` cũng được định nghĩa là `abstract`, để có thể tùy chỉnh với các chỉ báo khác nhau.

Algorithm 4: Mã giả hàm `formula()`

```

current_date, data_map indicator_value
input_value_list ← get_inputs(current_date, data_map)
if input_value_list is not NULL then
    | indicator_value ← compute by formula
else
    | indicator_value ← NULL

```

– **transform()**:

Sau khi các bảng dữ liệu đầu vào được đọc từ Druid, việc tính toán dữ liệu được thực hiện thông qua các vòng lặp trên từng dòng dữ liệu.

Với mỗi dòng dữ liệu cần tính toán, hai phương thức `get_inputs()` và `formula()` được gọi đến để áp dụng công thức chỉ báo.

Algorithm 5: Mã giả hàm `transform()`

```

data_map, start_date output_indicator_data
ticker_list ← get_all_tickers(data_map)
output_indicator_data ←
for ticker in ticker_list do
    | indicator_output_list ← []
    | each_ticker_data_map ← filter_by_ticker(data_map)
    | for date in [start_date...today] do
        | indicator_output_value = formula(date, each_ticker_data_map)
        | AddItem(indicator_output_list, indicator_output_value)
    | ticker_indicator_data ← create_new_df(indicator_output_list)
    | AddItem(output_indicator_data, ticker, ticker_indicator_data)

```

– **save_data()**:

Khi phương thức `transform()` hoàn tất, dữ liệu được trả về là các dataframe chứa dữ liệu chỉ báo sau khi được tính toán.

Ở bước cuối cùng trong luồng thực thi, trình tính toán tạo mới Kafka topic,

Druid datasource và đẩy dữ liệu vào Kafka topic tương ứng.

Algorithm 6: Mã giả hàm `save_data()`

output_topic, output_indicator_data

```
for ticker in output_indicator_data.keys() do  
    ticker_indicator_data  $\leftarrow$  output_indicator_data[ticker]  
    SendToKafka(output_topic, ticker_indicator_data)
```

– **execute()**:

Phương thức `execute()` tập hợp lệnh gọi các phương thức thực hiện từng bước trong luồng tính toán.

Algorithm 7: Mã giả hàm `execute()`

indicator_name, output_topic, input_tables, output_indicator_data

start_date \leftarrow *get_start_time*(*indicator_name*)

data_map \leftarrow *load_data_from_druid*(*start_date*, *input_tables*)

output_indicator_data \leftarrow *transform*(*data_map*, *start_date*)

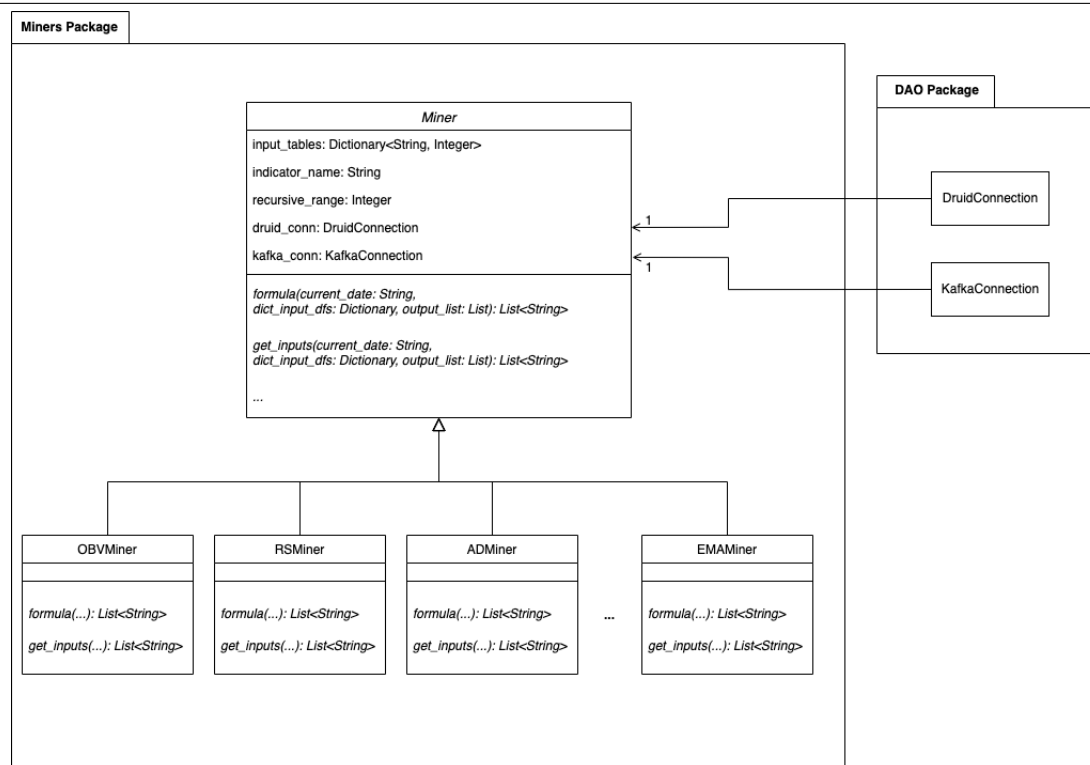
save_data(*output_topic*, *output_indicator_data*)

4.1.4 Sơ đồ lớp các trình tính toán

Với lớp trừu tượng `Miner` được mô tả ở phần ??, với mỗi chỉ báo cần tính toán, hệ thống sẽ tạo mới một lớp riêng kế thừa từ lớp `Miner`.

Lớp kế thừa này sẽ định nghĩa lại hai phương thức trừu tượng của `Miner` là `get_inputs()` và `formula()`.

Sơ đồ các lớp thể hiện các trình tính toán chỉ báo riêng biệt được mô tả như sau:



Hình 4.7: Sơ đồ lớp gồm các trình tính toán riêng biệt

4.2 Lập lịch các trình tính toán

4.2.1 Nhu cầu lập lịch các trình tính toán chỉ báo

Như vậy, chức năng tính toán các chỉ báo đã được trình bày thông qua việc mô tả các bước và luồng thực hiện cũng như kèm theo các ví dụ. Nhưng không chỉ đảm bảo tính toán, phân hệ còn cần tổ chức lập lịch các trình tính toán bởi những lý do sau đây:

- Các chỉ báo chạy định kỳ

Các trình tính toán chỉ báo sẽ được chạy sau khi dữ liệu giao dịch được cập nhật. Cụ thể với phân hệ này, các trình tính toán được chạy ngay sau khi trình thu thập hoàn thiện phần cập nhật dữ liệu của phiên giao dịch vào datasource "ticker_data"

Do vậy các chỉ báo cần được đặt lịch chạy định kỳ 01 lần/ ngày sau khi phiên giao dịch của thị trường kết thúc.

- Các chỉ báo phụ thuộc lẫn nhau

Các chỉ báo đều được thực hiện sau khi có hệ thống cập nhật dữ liệu giao dịch mới từ thị trường chứng khoán. Tuy nhiên không phải tất cả các chỉ báo đều được chạy đồng thời, bởi có những chỉ báo phụ thuộc vào kết quả của những chỉ báo khác. Ví dụ như để tính MACD phải có số liệu của EMA12 và

EMA26.

Các yêu cầu về thứ tự tính toán có thể được biểu diễn thành đồ thị tính toán, với đỉnh là các trình tính toán, mỗi cạnh có hướng đi từ chỉ báo cần tính toán trước cho đến chỉ báo được tính toán sau. Khởi đầu của đồ thị là phần thu thập dữ liệu giao dịch, và không tồn tại chu trình trong đồ thị tính toán trên.

4.2.2 Lập lịch sử dụng Apache Airflow

Với những yêu cầu cụ thể về thời gian và DAG trình tự thực hiện như trên, Apache Airflow đã được sử dụng để triển khai tính năng này. Ở nội dung phần 3.3, các khái niệm chính của Airflow đã được đề cập, trong đó có Task và DAG.

Trong phân hệ này, mỗi Task tương đương với một trình tính toán, mỗi lần tính toán sẽ là một Task Instance. Mỗi quan hệ về mặt thứ tự giữa các Task được thể hiện trong DAG. Để định nghĩa DAG gồm các Task, cần tạo file script Python để khai báo các Task và xác định mối quan hệ phụ thuộc giữa các Task.

Trước tiên cần định nghĩa ra một đối tượng DAG trong file, ví dụ như sau:

```
full_dag = DAG(
    dag_id='full_dag',
    default_args={
        'retries': 1
    },
    description="DAG for stock indicators computation",
    schedule=timedelta(days=1),
    start_date=datetime.today(),
    tags=['indicator'])

crawler_task >> [ad_task, ema_task, obv_task, aroon_task, rs_task]
rs_task >> rsi_task
ema_task >> macd_task
```

Hình 4.8: Khai báo một Airflow DAG trong file Python

Sau khi DAG đã được định nghĩa, lần lượt các Task thuộc DAG đó cũng sẽ được khai báo. Ở đây mỗi Task là một trình tính toán chỉ báo, được khai báo như ví dụ sau:

```
def compute_obv():
    print("OBV Miner executing...")
    obv_miner = OBVMiner(input_tables={"ticker_data": 0}, indicator_name="obv", recursive_range=1)
    obv_miner.execute()
    return "OBV_MINER"
```

Hình 4.9: Khai báo một Task trong file Python

Task được khai báo trong ví dụ này là một trình tính toán chỉ báo OBV. Hàm `compute_obv()` khởi tạo đối tượng thuộc lớp `OBVMiner` bằng cách truyền vào các

CHƯƠNG 4. XÂY DỰNG PHÂN HỆ QUẢN LÝ VÀ LẬP LỊCH TRÌNH TÍNH TOÁN CHỈ BÁO

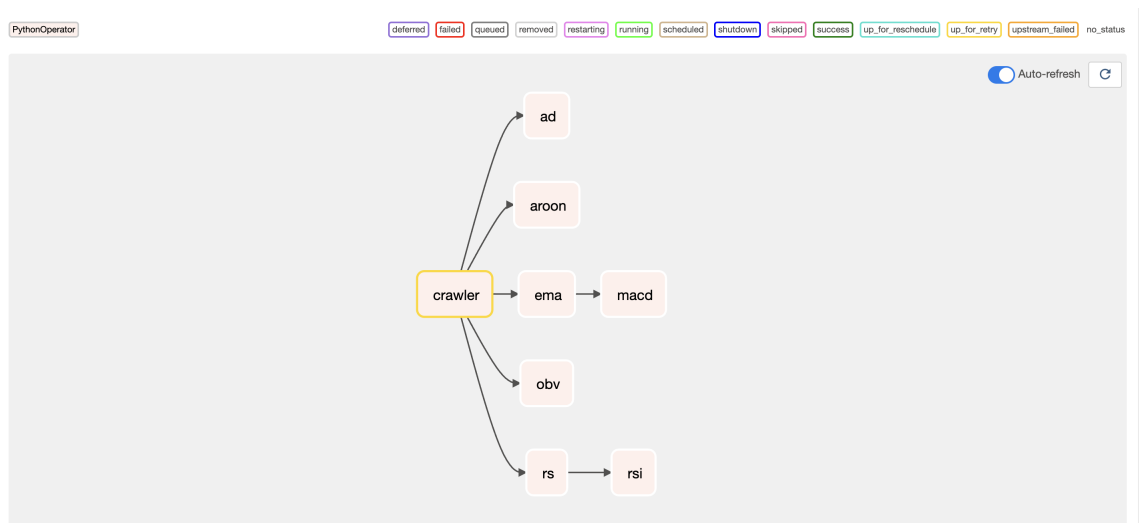
tham số cần thiết Sau đó gọi phương thức `execute()` để thực hiện các bước tính toán đã được mô tả ở phần **??**. Với DAG và các Tasks đã được khai báo, bước tiếp theo trong quá trình tạo file Airflow DAG là định nghĩa thứ tự thực hiện, sử dụng kí hiệu ">".

```
crawler_task >> [ad_task, ema_task, obv_task, aroon_task, rs_task]
rs_task >> rsi_task
ema_task >> macd_task
```

Hình 4.10: Khai báo thứ tự các Tasks trong một DAG

DAG này ngoài các trình tính toán chỉ báo còn có thành phần Crawler (thu thập dữ liệu giao dịch sau khi kết thúc phiên). Các chỉ báo sẽ được thực thi sau khi crawler hoàn tất việc cập nhật dữ liệu giao dịch.

Sau khi file DAG được hoàn thiện, hệ thống Airflow sẽ đọc và hiển thị kết quả trên giao diện của thành phần Webserver như hình dưới đây.



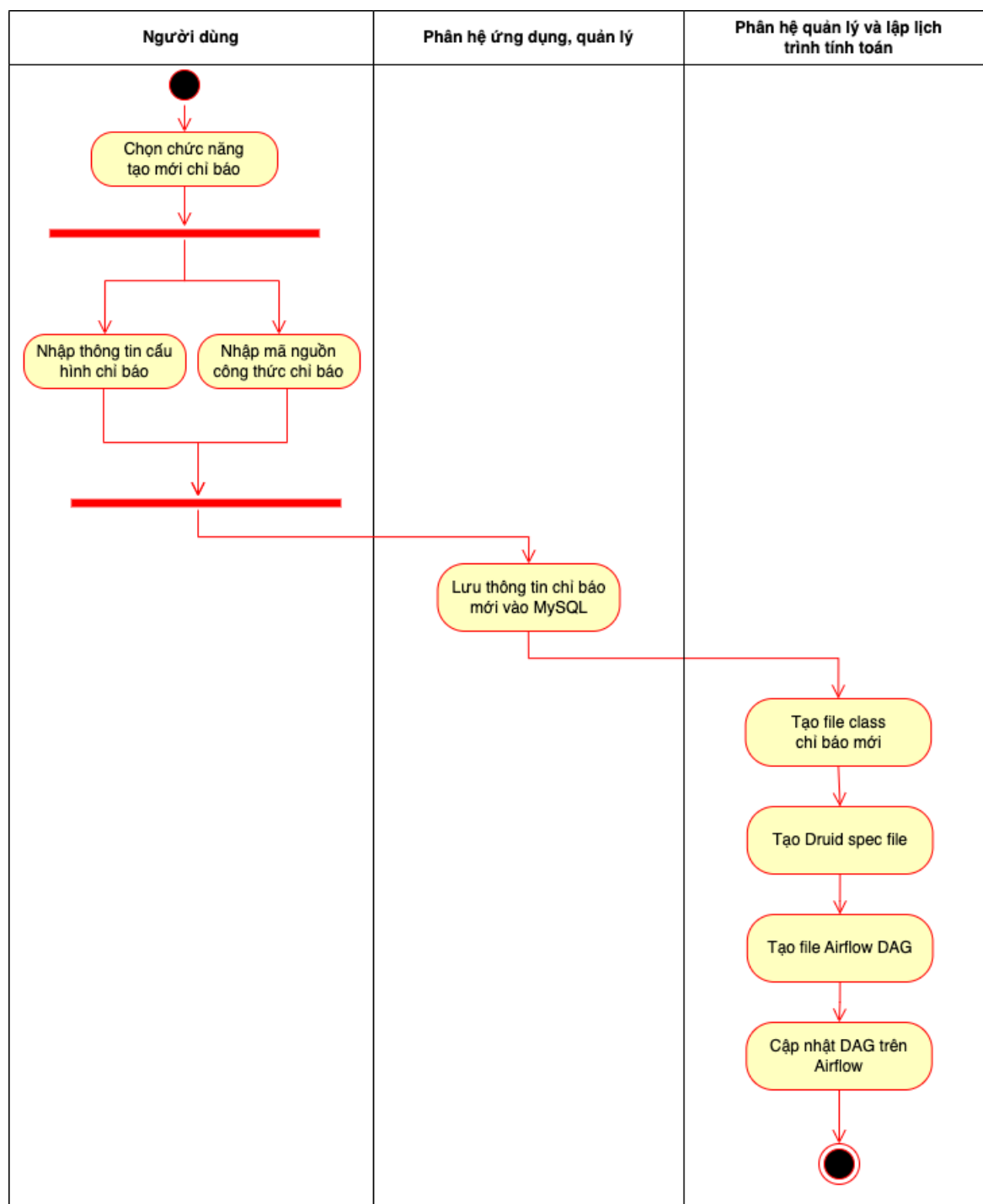
Hình 4.11: DAG trình tự các trình tính toán chỉ báo

4.3 Chức năng tự động tạo mới chỉ báo cá nhân

Với hai chức năng kể trên, phân hệ đã hoàn thiện luồng lập lịch tính toán cho các chỉ báo phổ biến trên thị trường. Tuy nhiên, như đã đề cập ở các phần trước, hiện nhiều nhà đầu tư muốn theo dõi biến động thị trường dựa trên những khía cạnh khác nhau, làm nảy sinh nhu cầu muốn tạo mới những chỉ báo theo công thức của riêng mình. Vậy nên phân hệ này được phát triển thêm một tính năng nổi bật là tự động tạo mới chỉ báo cá nhân.

4.3.1 Biểu đồ hoạt động của chức năng tạo mới chỉ báo

Để phục vụ cho tính năng tạo mới chỉ báo cá nhân, cần sự liên kết giữa các phân hệ của toàn bộ nền tảng. Biểu đồ dưới đây thể hiện quá trình các phân hệ hoạt động để thực hiện chức năng này:



Hình 4.12: Sơ đồ hoạt động tính năng tạo mới chỉ báo

4.3.2 Dữ liệu đầu vào của người dùng

Người dùng sử dụng chức năng tạo mới chỉ báo thông qua tương tác với giao diện web của phân hệ ứng dụng, quản lý của nền tảng dữ liệu chứng khoán này. Sau khi được xử lý, thông tin được gửi đến phân hệ quản lý và lập lịch trình tính toán chỉ báo ở dạng JSON gồm các trường như sau.

`input_tables`: liệt kê danh sách các bảng dữ liệu đầu vào để tính toán chỉ báo mới. Trường thông tin này có dạng key-value, với key là tên các bảng dữ liệu như `ticker_data`, `obv_data` còn value sẽ là số chu kỳ thời gian hồi quy mà dữ liệu trong bảng đó được sử dụng. Ví dụ để tính toán chỉ báo RS, ta cần dữ liệu về **giá đóng cửa** của 14 phiên giao dịch gần nhất, trường thông tin sẽ có dạng:

```
{"ticker_data": 14}
```

`indicator_name`: trường thông tin khai báo tên chỉ báo, dưới dạng xâu không gồm ký tự dấu cách. Tên của Kafka topic và Druid datasource tương ứng cũng sẽ được đặt có dạng `<indicator_name>_data`.

`recursive_range`: trường thông tin dạng số nguyên dương thể hiện số chu kỳ hồi quy của công thức chỉ báo. Với chỉ báo OBV có công thức

$$OBV_i = OBV_{i-1} + \text{sgn}(C_i - C_{i_1}) * V_i,$$

thì giá trị `recursive_range` = 1.

`get_inputs_method` và `formula_method`: hai trường thông tin chứa mã nguồn của phương thức `get_inputs()` và `formula()` để khởi tạo chỉ báo mới. Chức năng này đòi hỏi người dùng cần có kiến thức cơ bản về lập trình với ngôn ngữ Python và thư viện pandas.

Một đầu vào dữ liệu JSON hoàn chỉnh cho việc khai báo chỉ báo cá nhân (ví dụ với RSI) như sau:

```
{
  "indicator_name": "rsi",
  "input_tables": {"rs_data": 0},
  "recursive_range": 1,
  "get_inputs_method": "...",
  "formula_method": "...",
}
```

4.3.3 Vai trò của phân hệ quản lý và lập lịch các trình tính toán chỉ báo

Theo như nội dung của biểu đồ hoạt động phía trên, phân hệ quản lý và lập lịch các trình tính toán chỉ báo đóng vai trò tạo mới các tệp tin, các thành phần theo giá trị đầu vào mà người dùng khai báo.

Để tạo mới file với yêu cầu như vậy, phân hệ sử dụng thư viện jinja2 trên Python. Khi sử dụng jinja2, ứng với mỗi file cần sinh mới, hệ thống cần khởi tạo trước các template. Những template này định hình sẵn những nội dung cố định của file, cùng với đó là các vị trí sẽ được điền dựa theo dữ liệu đầu vào từ người dùng. Thông tin giới thiệu về thư viện jinja2 cũng như mô tả về cơ chế sinh file đã được đề cập ở mục 3.4. Dựa trên cơ chế này, phân hệ thực hiện chức năng tạo mới chỉ báo theo các bước sau đây:

- **Tạo mới Kafka topic và Druid datasource**

Khi nhận thông tin về một chỉ báo mới, phân hệ tạo yêu cầu đến hai hệ thống Kafka và Apache Druid để tạo mới những thành phần lưu trữ dữ liệu.

Trước tiên, phân hệ thông qua API của Apache Kafka để tạo mới một topic tương ứng với luồng dữ liệu từ chỉ báo mới. Khi Kafka topic đã được tạo, phân hệ tiếp tục thực hiện bước tạo mới Druid datasource.

Để tạo mới Druid datasource, phân hệ sử dụng thư viện jinja2 để tạo file spec chứa cấu hình của datasource nhận luồng dữ liệu đầu vào từ Kafka topic. Trong file spec template, trường thông tin về tên của chỉ báo (indicator_name) sẽ được đánh dấu để jinja điền vào mỗi khi người dùng tạo chỉ báo mới. Nội dung template spec file như sau:

```
{
  "type": "kafka",
  "spec": {
    "ioConfig": {
      "type": "kafka",
      "consumerProperties": {
        "bootstrap.servers": "localhost:9092"
      },
      "topic": "{{indicator_name}}_data",
      "inputFormat": {
        "type": "json"
      },
      "useEarliestOffset": true
    },
    "tuningConfig": {
      "type": "kafka"
    },
    "dataSchema": {
      "dataSource": "{{indicator_name}}_data",
      "timestampSpec": {
        "column": "time",
        "format": "auto"
      },
      "dimensionsSpec": {
        "dimensions": [
          "ticker",
          {
            "type": "double",
            "name": "{{indicator_name}}"
          }
        ]
      },
      "granularitySpec": {
        "queryGranularity": "none",
        "rollup": false,
        "segmentGranularity": "day"
      }
    }
  }
}
```

Hình 4.13: Template tạo Druid spec file

Với nội dung file spec theo template như trên, dữ liệu đẩy từ Kafka topic vào Druid datasource buộc phải theo schema gồm 3 trường: ticker, time và indicator_name. Đây là schema mặc định cho mỗi bảng chứa chỉ báo cá nhân thêm mới, tính năng này hiện chưa áp dụng nếu người dùng muốn bảng dữ liệu có schema khác với schema mặc định này.

- **Tạo file class chỉ báo mới**

Theo thiết kế biểu đồ lớp 4.7, các lớp chỉ báo cụ thể sẽ đều kế thừa lớp cha Miner. Lớp cha Miner đã được triển khai sẵn các phương thức đọc, ghi và xử lý chung cho luồng dữ liệu. Các lớp con cần khai báo thêm hai phương thức get_inputs() và formula() tương ứng với công thức của chỉ báo cần tính toán.

Như vậy, file template khai báo class cho chỉ báo mới có nội dung như sau:

CHƯƠNG 4. XÂY DỰNG PHÂN HỆ QUẢN LÝ VÀ LẬP LỊCH TRÌNH TÍNH TOÁN CHỈ BÁO

```
class {{indicator_name_upper}}Miner(Miner):
    def __init__(self, input_tables, indicator_name, recursive_range):
        super().__init__(input_tables, indicator_name, recursive_range)

    def get_inputs(self, current_date: str, dict_input_dfs: Dict[str, pd.DataFrame], output_list: list):
        {{get_inputs_method}}

    def formula(self, current_date: str, dict_input_dfs: Dict[str, pd.DataFrame], output_list: list):
        {{formula_method}}
```

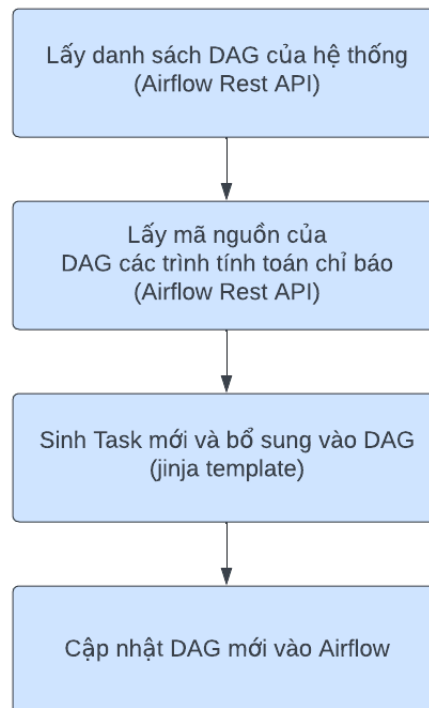
Hình 4.14: Template tạo class cho chỉ báo mới

Đầu tiên, trường `indicator_name_upper` (tên chỉ báo viết in hoa) được sử dụng để đặt tên cho class mới. File template cũng khai báo lớp mới tạo sẽ kế thừa từ lớp cha `Miner`. Các đoạn mã nguồn của hai phương thức `get_inputs()` và `formula()` cũng được điền vào thông qua hai trường thông tin `get_inputs_method()` và `formula_method()`.

- **Tạo mới và cập nhật file Airflow DAG**

Bước cuối cùng trong quá trình tạo mới chỉ báo, đó là tạo mới và cập nhật file Airflow DAG. Trước khi nhận yêu cầu tạo mới các chỉ báo, phân hệ đã có một DAG gồm các chỉ báo có sẵn của hệ thống như thể hiện ở sơ đồ 4.11.

Airflow hiện chưa cung cấp API cho phép thay đổi nội dung mã nguồn qua HTTP request. Vì vậy, các bước thực hiện cập nhật file DAG như sau:



Hình 4.15: Các bước cập nhật DAG trên Airflow

– Lấy danh sách DAG của hệ thống Airflow

Trước tiên, cần kiểm tra các DAG của Airflow đang hoạt động. Ở quy mô hiện tại của nền tảng chỉ có duy nhất 1 DAG với id = "indicator_dag". URL hoàn chỉnh để gửi yêu cầu sẽ là:

<airflow_server_host>:<airflow_port>/api/v1/dags

– Lấy mã nguồn của file DAG các trình tính toán chỉ báo

Bước này bao gồm hai lần truy vấn thông qua Airflow Rest API. Đầu tiên, gửi request đến endpoint **/api/v1/dags/dag_id** để lấy thông tin chi tiết về một DAG, được định danh qua dag_id. Với DAG có id là "indicator_dag" thì URL hoàn chỉnh sẽ là:

<airflow_server_host>:<airflow_port>/api/v1/dags/indicator_dag

Từ kết quả trả về cần trích xuất ra trường thông tin **file_token**. Trường thông tin này tiếp tục được gửi theo yêu cầu đến URL sau:

<airflow_server_host>:<airflow_port>/api/v1/dagSources/{file_token}

Kết quả trả về là mã nguồn của file DAG có id là "indicator_dag".

– Sinh Task mới và bổ sung vào DAG

Ở bước này phân hệ sẽ sinh ra đoạn script khai báo Task cho chỉ báo mới

trong file DAG, sử dụng jinja với template sau:

```
def compute_{{indicator_name}}():
    {{indicator_name}}_miner = {{indicator_name_upper}}Miner(input_tables={{input_tables}}, indicator_name="{{indicator_name}}",
                                                            recursive_range={{recursive_range}})
    {{indicator_name}}_miner.execute()
    print("{{indicator_name_upper}} Miner executing...")
    return "{{indicator_name_upper}}_MINER"

{{indicator_name}}_task = PythonOperator(task_id="{{indicator_name}}", python_callable=compute_{{indicator_name}}, dag=indicator_dag)
```

Hình 4.16: Template cho script khai báo Task cho chỉ báo mới

Sau khi được khai báo trong file DAG, mỗi quan hệ phụ thuộc của Task mới tạo sẽ được định nghĩa với các task tương ứng với tên các bảng trong trường thông tin `input_tables` mà người dùng nhập vào.

– Cập nhật DAG mới vào Airflow

Với nội dung file DAG hoàn tất, việc còn lại là cập nhật nội dung mới cho DAG hiện đang chạy trên Airflow. Như đã đề cập ở phần trên, Rest API của Airflow hiện chưa hỗ trợ thao tác cập nhật mã nguồn của DAG. Để thực hiện cập nhật nội dung DAG, tôi thực hiện xóa DAG đang có và tạo một DAG mới thông qua việc ghi file mới vào folder "dags" của Airflow.

Trên đây là các bước phục vụ cho chức năng tạo mới chỉ báo cá nhân được phân hệ quản lý và lập lịch trình tính toán chỉ báo thực hiện. Đây cũng là chức năng cuối trong 03 chức năng chính được mô tả của phân hệ này.

4.4 Yêu cầu phi chức năng

Các yêu cầu phi chức năng với phân hệ này cũng tương đồng với yêu cầu phi chức năng chung cho toàn bộ nền tảng dữ liệu.

Thời gian tính toán và phản hồi nhanh là yêu cầu được đặt ra cho phân hệ với vai trò tính toán, bên cạnh việc dữ liệu phải chính xác, trung thực. Với lượng dữ liệu không quá nhiều cho mỗi ngày giao dịch, thời gian tính toán và phản hồi của hệ thống là nhanh chóng, không chậm trễ.

Tính chịu lỗi là yêu cầu trong quá trình hệ thống hoạt động có thể chấp nhận rủi ro về phần cứng hoặc phần mềm trong một phạm vi nhất định. Hệ thống cần đảm bảo hoạt động thông suốt, không mất mát dữ liệu nếu có sự cố xảy ra. Để thực hiện được yêu cầu này, các thành phần của hệ thống cần được cài đặt dưới dạng cụm, dữ liệu được phân chia ra nhiều bản sao để đảm bảo chịu lỗi.

Tính khả mở là yêu cầu về khả năng mở rộng của nền tảng nói chung và phân hệ nói riêng. Với nền tảng dữ liệu, để có thể mở rộng dễ dàng và không ảnh hưởng đến những thành phần đang có, giải pháp đưa ra bên cạnh việc cài đặt các thành

CHƯƠNG 4. XÂY DỰNG PHÂN HỆ QUẢN LÝ VÀ LẬP LỊCH TRÌNH TÍNH TOÁN CHỈ BÁO

phần trên các cụm máy tính thì còn có thể sử dụng các công nghệ đóng gói ứng dụng như Docker.

Ở phân hệ quản lý và lập lịch các trình tính toán chỉ báo, yêu cầu phi chức năng được đáp ứng gồm thời gian tính toán nhanh. Ngoài ra các yêu cầu phi chức năng khác do điều kiện không thể triển khai được trên cụm nên không thể đáp ứng được. Trong tương lai, các yêu cầu phi chức năng này sẽ được đáp ứng đầy đủ.

CHƯƠNG 5. KẾT QUẢ THỰC NGHIỆM VÀ ĐÁNH GIÁ

Như vậy là ở chương 4, tôi đã trình bày về các yêu cầu chức năng và phi chức năng khi xây dựng phân hệ, nội dung từ thiết kế cho các bước triển khai thuật toán, lập trình. Ở chương này, tôi sẽ trình bày về quá trình triển khai và kiểm thử của phân hệ quản lý và lập lịch trình tính toán chỉ báo.

5.1 Xây dựng ứng dụng

5.1.1 Thư viện và công cụ sử dụng

Để xây dựng nên phân hệ quản lý và lập lịch trình tính toán chỉ báo, các công cụ được sử dụng như sau:

| Mục đích | Công cụ | Địa chỉ URL |
|---------------------------------------|-------------------------------|---|
| IDE lập trình | PyCharm Professional 2021.3.2 | https://www.jetbrains.com/pycharm/ |
| Text Editor | Sublime Text 3 | https://www.sublimetext.com/ |
| Nền tảng API | Postman | https://www.postman.com/ |
| Hệ thống quản lý phiên bản | Git | https://git-scm.com/ |
| Nền tảng quản lý phiên bản trực tuyến | Github | https://github.com/about |
| Quản lý môi trường ảo | Anaconda | https://www.anaconda.com/ |
| Nền tảng đóng gói môi trường | Docker | https://www.docker.com/ |

Bảng 5.1: Danh sách thư viện và công cụ sử dụng

Do ngôn ngữ lập trình được lựa chọn là Python và các thư viện như pandas hay jinja, mã nguồn của phân hệ chủ yếu được xây dựng trên IDE Pycharm. Với Python, việc tạo lập một môi trường thư viện ảo là cần thiết để tách biệt không gian làm việc giữa các project khác nhau, đây là vai trò được đảm nhiệm bởi Anaconda.

Tiếp đến, để thử nghiệm và kiểm tra các đầu API đến các thành phần lưu trữ như Apache Druid và Apache Kafka, nền tảng API Postman đóng vai trò tạo các yêu cầu và lấy kết quả trả về.

Git và Github là những công cụ quen thuộc để kiểm soát phiên bản của mã nguồn. Còn Docker là một nền tảng rất phổ biến trong việc tạo môi trường ảo hóa để tách biệt tài nguyên phân phối cho các ứng dụng hoạt động. Trong phân hệ này, các thành phần của Airflow được xây dựng trên các Docker container riêng biệt.

5.1.2 Cấu hình thiết bị

Quá trình triển khai và kiểm thử được thực hiện trên máy tính cá nhân với thông tin cấu hình như sau:

| | |
|---------------------|---|
| Tên thiết bị | Apple MacBook Pro M1 2020 |
| RAM | 8GB |
| Bộ nhớ | 512GB |
| Vi xử lý | Apple M1 8 nhân (4 nhân hiệu năng cao và 4 nhân tiết kiệm điện) |

Bảng 5.2: Cấu hình máy tính cá nhân

Với cấu hình máy không quá mạnh, việc triển khai các ứng dụng trên máy gặp khó khăn khi không thể đảm bảo cài đặt trên cụm. Các bước xây dựng và triển khai phân hệ được mô tả rõ hơn ở các phần sau.

5.2 Triển khai

5.2.1 Triển khai trình tính toán chỉ báo

Sau khi đã có được dữ liệu trên Apache Druid và các Kafka Topic tương ứng, các trình tính toán bắt đầu được triển khai theo thiết kế lớp và mã giả thuật toán ở ???. Mã nguồn lớp **DruidConnection** và **KafkaConnection** sau khi hoàn thiện như sau:

```
import yaml
from pydruid.client import *
from pydruid.utils.aggregators import *
from pydruid.utils.filters import Dimension

def load_configs(path="../../config/config.yml"):...

class DruidConnection:
    def __init__(self):...

    def load_ticker_data(self, start_date, end_date, ticker_table="ticker_data"):...

    def load_indicator_data(self, indicator_table, start_date, end_date, metric):
        if start_date is None:
            start_date = "2022-01-01T00:00"
        self.query.topn(
            datasource=indicator_table,
            granularity="day",
            intervals=start_date + "/" + end_date,
            dimension="ticker",
            metric=metric,
            aggregations={metric: longmax(metric)},
            threshold=5000
        )
        df = self.query.export_pandas()
        return df
```

Hình 5.1: Mã nguồn lớp DruidConnection

```
import json
import numpy
import pandas
import yaml
from kafka.admin import KafkaAdminClient, NewTopic
from kafka import KafkaProducer, KafkaConsumer
from typing import List

def load_configs(path="../../config/config.yml"):...

class KafkaConnection:
    def __init__(self, bootstrap_servers="localhost:9092", client_id="sonmt"):...

    def create_new_topic(self, topic_names: str or List[str]):
        topic_list = []
        if type(topic_names) == list:...

        if type(topic_names) == str:
            topic_list = [NewTopic(name=topic_names, num_partitions=1, replication_factor=1)]

        self.admin_client.create_topics(new_topics=topic_list)

    def get_existed_list_topics(self):
        list_topics = self.consumer.topics()
        return list(list_topics)

    def produce_df_to_kafka(self, topic: str, df: pandas.DataFrame):...
```

Hình 5.2: Mã nguồn lớp KafkaConnection

Các API của Druid và Kafka đã được đóng gói lại qua các phương thức nhằm thuận tiện hơn cho việc tái sử dụng mã nguồn. Tiếp đó là mã nguồn của lớp trừu tượng Miner, với các thuộc tính và phương thức được triển khai như sau:

```

class Miner:
    def __init__(self, input_tables: Dict[str, int],
                 indicator_name: str,
                 recursive_range: int):...

    def execute(self):
        print("Start loading data...")
        dict_input_dfs, start_date, end_date = self.load_data_from_druid()
        print(f"Finish loading from {start_date} to {end_date}")
        for k in dict_input_dfs.keys():...
        print("Start transforming data...")
        output = self.transform(dict_input_dfs, start_date, end_date)
        self.save_data(output)

    def get_start_time(self):...

    def load_data_from_druid(self):...

    @abstractmethod
    def get_inputs(self, current_date: str, dict_input_dfs: Dict[str, pd.DataFrame], output_list: list):...

    @abstractmethod
    def formula(self, current_date: str, dict_input_dfs: Dict[str, pd.DataFrame], output_list: list):...

    def transform(self, dict_inputs_df, start_date, end_date):...

    def save_data(self, output):...

```

Hình 5.3: Mã nguồn lớp Miner

Từ mã nguồn trong hình 5.3 có thể thấy, các bước của quá trình tính toán chỉ báo được mô tả qua các phương thức của lớp Miner, phương thức `execute()` sẽ lần lượt gọi đến các phương thức đại diện cho từng bước tính toán. Hai phương thức trừu tượng `get_inputs()` () và `formula()` sẽ được ghi đè và triển khai bởi các lớp con kế thừa. Ví dụ dưới đây là mã nguồn của lớp con `OBVMiner` kế thừa lớp cha `Miner`:


```
class OBVMiner(Miner):
    def __init__(self, input_tables, indicator_name, recursive_range):
        super().__init__(input_tables, indicator_name, recursive_range)

    def get_inputs(self, current_date: str, dict_input_dfs: Dict[str, pd.DataFrame], output_list: list):
        if not current_date.endswith(".000Z"):
            current_date += ".000Z"
        ticker_data = dict_input_dfs["ticker_data"]
        ticker_data = ticker_data.reset_index(drop=True)

        row_index_list = ticker_data.index[ticker_data['timestamp'] == current_date].tolist()
        if len(row_index_list) == 0:
            return None

        current_row_index = row_index_list[0]
        cur_close = float(ticker_data["close"][current_row_index])
        cur_volume = float(ticker_data["volume"][current_row_index])
        prev_close = float(ticker_data["close"][current_row_index - 1]) if current_row_index > 0 else 0

        prev_1_obv = output_list[-1] if len(output_list) > 0 else None

        return cur_close, cur_volume, prev_close, prev_1_obv

    def formula(self, current_date: str, dict_input_dfs: Dict[str, pd.DataFrame], output_list: list):
        inputs = self.get_inputs(current_date, dict_input_dfs, output_list)
        if inputs is None:
            cur_obv = None
        else:
            cur_close, cur_volume, prev_close, prev_1_obv = inputs
            if prev_1_obv is None:
                cur_obv = 0
            else:
                cur_obv = prev_1_obv + np.sign(cur_close - prev_close) * cur_volume
        output_list.append(cur_obv)
        return output_list, cur_obv
```

Hình 5.4: Mã nguồn lớp OBVMiner

Ở hình 5.4, lớp OBVMiner được khởi tạo kế thừa từ lớp cha Miner, và triển khai tiếp hai phương thức trừu tượng tương ứng với công thức tính chỉ báo OBV. Các chỉ báo khác cũng được triển khai tương tự.

5.2.2 Triển khai lập lịch trình tính toán chỉ báo

Sau khi đã có mã nguồn hoàn thiện của các trình tính toán, tôi tiếp tục triển khai việc lập lịch cho những trình tính toán này. Tôi lựa chọn 3 chỉ báo là OBV, RS và RSI để thực hiện lập lịch và tính toán trên Airflow. Các thành phần của Airflow cũng được triển khai trên các Docker container giúp dễ dàng khởi chạy và quản lý.

Các tiến trình của Airflow được định nghĩa trong file docker-compose.yml và tiến hành cài đặt theo hướng dẫn tại [13]. Các container được khởi chạy thành công

như dưới đây:

```
(miners-env) sonmt@Sons-MacBook-Pro Miners % docker ps
```

| CONTAINER ID | IMAGE | COMMAND | CREATED | STATUS | PORTS | NAMES |
|--------------|---------------------|--------------------------|----------------|-------------------------|------------------------|----------------------------|
| 7e57ba25b125 | sonmt/airflow:2.5.1 | "/usr/bin/dumb-init ..." | 19 minutes ago | Up 19 minutes (healthy) | 8080/tcp | miners-airflow-scheduler-1 |
| ddf58816478d | sonmt/airflow:2.5.1 | "/usr/bin/dumb-init ..." | 19 minutes ago | Up 19 minutes (healthy) | 8080/tcp | miners-airflow-worker-1 |
| 110c87563de8 | sonmt/airflow:2.5.1 | "/usr/bin/dumb-init ..." | 19 minutes ago | Up 19 minutes (healthy) | 8080/tcp | miners-airflow-triggerer-1 |
| b746420d1aa4 | sonmt/airflow:2.5.1 | "/usr/bin/dumb-init ..." | 19 minutes ago | Up 19 minutes (healthy) | 0.0.0.0:8080->8080/tcp | miners-airflow-webserver-1 |
| f2c5aabf9bb1 | postgres:13 | "docker-entrypoint.s..." | 19 minutes ago | Up 19 minutes (healthy) | 5432/tcp | miners-postgres-1 |
| f8599b19d7e6 | redis:latest | "docker-entrypoint.s..." | 19 minutes ago | Up 19 minutes (healthy) | 6379/tcp | miners-redis-1 |

Hình 5.5: Các container thành phần của Airflow

Các containers hoạt động ổn định, không gặp lỗi khi triển khai.

5.2.3 Kết quả đạt được

Sau thời gian lập trình và triển khai các hệ thống, phân hệ quản lý và lập lịch chỉ báo đã được xây dựng với hệ thống Airflow lập lịch tính toán chỉ báo, cũng như tích hợp thêm mã nguồn cho chức năng tạo mới chỉ báo cá nhân.

Mã nguồn gồm hai gói chính là Miner Package và DAO Package. Trong đó DAO Package gồm hai lớp DruidConnection và KafkaConnection với mã nguồn như mô tả ở 5.1 và 5.2. Miner Package chứa class Miner và các class chỉ báo kế thừa gồm 5 class: OBVMiner, RSMMiner, ADMMiner, RSIMMiner và AroonMiner.

Một vài thống kê khác liên quan đến phân hệ được liệt kê trong bảng sau:

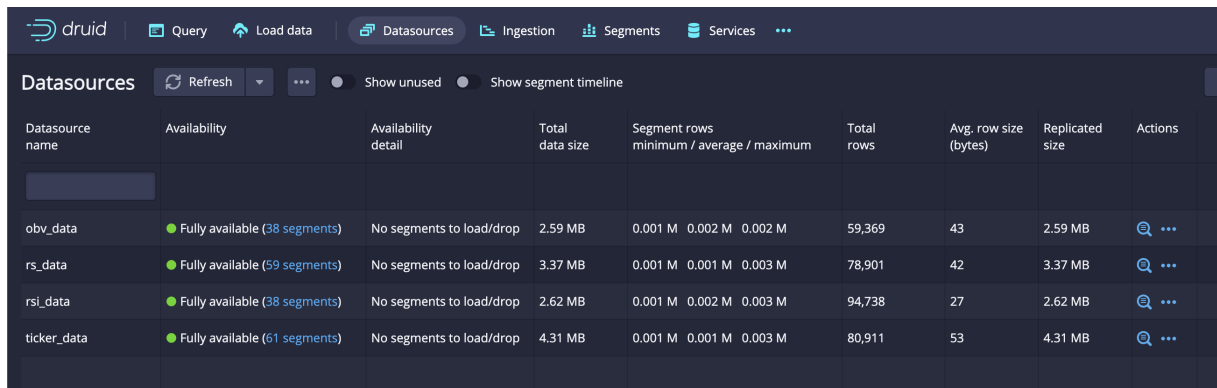
| Tiêu chí | Số liệu |
|--|------------------|
| Tổng số dòng mã nguồn | khoảng 2100 dòng |
| Dung lượng các images | 1.5GB |
| Dung lượng các containers | 2.83GB |
| Số chỉ báo triển khai thực tế | 3 |
| Thời gian tính toán mỗi lần chạy DAG | 10s/lần |
| Số lượng mã chứng khoán được tính toán | 1617 mã |

Bảng 5.3: Các thống kê khác của phân hệ khi triển khai

5.2.4 Minh họa các chức năng chính

a, Minh họa tính toán chỉ báo

Các chỉ báo được tính toán và lưu lại trong các Druid datasources, như hình dưới đây:



| Datasource name | Availability | Availability detail | Total data size | Segment rows minimum / average / maximum | Total rows | Avg. row size (bytes) | Replicated size | Actions |
|-----------------|-------------------------------|--------------------------|-----------------|--|------------|-----------------------|-----------------|---------|
| obv_data | Fully available (38 segments) | No segments to load/drop | 2.59 MB | 0.001 M 0.002 M 0.002 M | 59,369 | 43 | 2.59 MB | |
| rs_data | Fully available (59 segments) | No segments to load/drop | 3.37 MB | 0.001 M 0.001 M 0.003 M | 78,901 | 42 | 3.37 MB | |
| rsi_data | Fully available (38 segments) | No segments to load/drop | 2.62 MB | 0.001 M 0.002 M 0.003 M | 94,738 | 27 | 2.62 MB | |
| ticker_data | Fully available (61 segments) | No segments to load/drop | 4.31 MB | 0.001 M 0.001 M 0.003 M | 80,911 | 53 | 4.31 MB | |

Hình 5.6: Các datasources trên Apache Druid

Các datasources này có tên tương ứng với dữ liệu mà chúng chứa bên trong. Các thông tin mà hình 5.6 hiển thị cho người dùng biết không chỉ tên mà còn các thông tin về dung lượng tổng, dung lượng các segments, dung lượng trung bình của mỗi hàng dữ liệu và tổng số bản ghi trong một datasource.

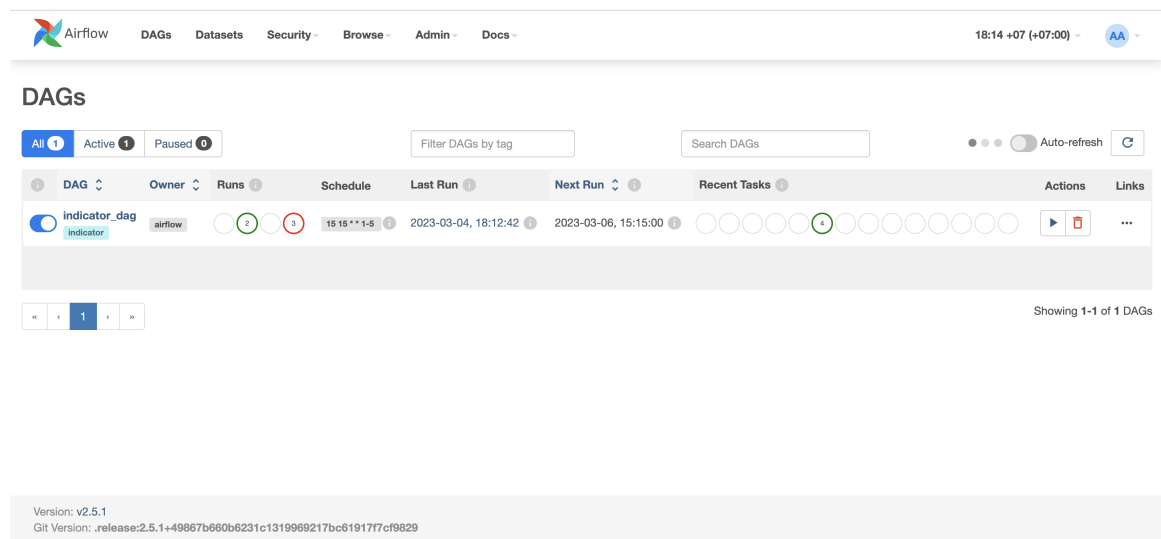
Để minh họa cho chức năng tính toán chỉ báo, dưới đây là kết quả tính chỉ báo OBV cho cổ phiếu HPG (Công ty cổ phần Tập đoàn Hòa Phát) trong các phiên giao dịch của tháng 01/2023.

| _time | A ticker | ¹²³ obv |
|--------------------------|----------|--------------------|
| 2023-01-03T00:00:00.000Z | HPG | -793,715,585 |
| 2023-01-04T00:00:00.000Z | HPG | -773,477,206 |
| 2023-01-05T00:00:00.000Z | HPG | -751,963,032 |
| 2023-01-06T00:00:00.000Z | HPG | -771,602,464 |
| 2023-01-09T00:00:00.000Z | HPG | -755,410,064 |
| 2023-01-10T00:00:00.000Z | HPG | -728,547,097 |
| 2023-01-11T00:00:00.000Z | HPG | -702,824,060 |
| 2023-01-12T00:00:00.000Z | HPG | -723,003,408 |
| 2023-01-13T00:00:00.000Z | HPG | -742,184,970 |
| 2023-01-16T00:00:00.000Z | HPG | -719,852,558 |
| 2023-01-17T00:00:00.000Z | HPG | -678,240,483 |

Hình 5.7: Kết quả chỉ báo OBV mã HPG trong tháng 1/2023

b, Minh họa tính năng lập lịch trình tính toán chỉ báo

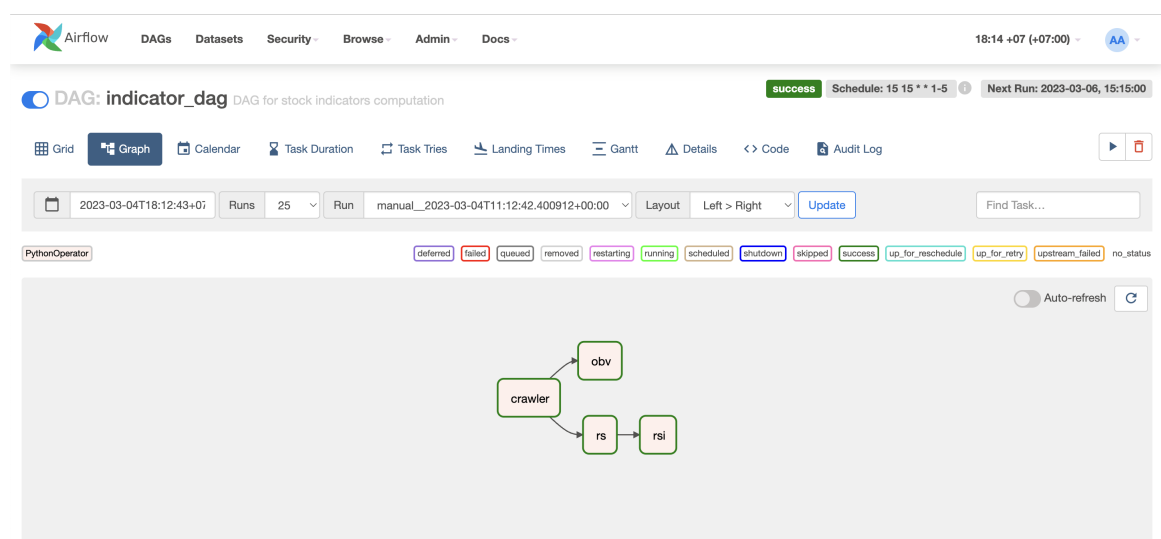
Các trình tính toán được lập lịch bởi Apache Airflow. Giao diện màn hình chính của Airflow web-server sau khi triển khai như sau:



Hình 5.8: Giao diện web của Airflow

Trên giao diện ở hình 5.8, danh sách các DAG của hệ thống được hiển thị, ở đây có duy nhất "indicator_dag" là DAG chứa các trình tính toán chỉ báo chứng khoán. Bên cạnh thông tin về tên, các thông tin khác của DAG cũng được cung cấp trên giao diện như: owner (chủ nhân) của DAG, lịch sử những lần chạy, thông tin lập lịch, thời gian lần chạy gần nhất, thời gian lần chạy tiếp theo và các thông tin khác.

Khi bấm vào tên DAG, các thông tin chi tiết hơn về DAG được hiển thị như giao diện sau:



Hình 5.9: Giao diện thông tin 1 DAG trên Airflow

Trong giao diện ở hình 5.9, thông tin về Graph (đồ thị) được hiển thị, cho thấy thứ tự các tác vụ (trình tính toán) được thực hiện trong mỗi lần khởi chạy DAG.

5.3 Kiểm thử

Sau đây đề án sẽ trình bày các kịch bản kiểm thử cho các chức năng của phân hệ quản lý và lập lịch trình tính toán chỉ báo gồm: chức năng tính toán chỉ báo và chức năng thêm mối chỉ báo cá nhân.

a, Kiểm thử chức năng tính toán chỉ báo

| ID | Case | Quy trình test | Kết quả mong muốn | Kết quả thực hiện | Kết luận |
|-----|--|--|---|---|-------------|
| 1.1 | Tính toán chỉ báo khi chưa tạo Kafka topic | <ol style="list-style-type: none"> 1. Không tạo Kafka topic tương ứng 2. Khởi chạy trình tính toán chỉ báo | Tự động tạo mới Kafka topic nếu chưa có | Trình tính toán tự động tạo mới Kafka topic và ghi dữ liệu | PASS |
| 1.2 | Tính toán chỉ báo khi đã tạo Kafka topic, chưa tạo Druid data-source | <ol style="list-style-type: none"> 1. Tạo Kafka topic tương ứng 2. Không tạo Druid data-source tương ứng 3. Khởi chạy trình tính toán chỉ báo | Trình tính toán tự động tạo mới Druid datasource và ghi dữ liệu vào Druid | Trình tính toán chỉ ghi dữ liệu vào Kafka, không ghi được vào Druid | FAIL |

| | | | | | |
|-----|--|--|---|---|-------------|
| 1.3 | Tính toán chỉ báo khi đã tạo Kafka topic, đã tạo Druid data-source kết nối đến Kafka topic tương ứng | <ol style="list-style-type: none"> 1. Tạo Kafka topic tương ứng 2. Tạo Druid datasource với ingestion task tương ứng 3. Khởi chạy trình tính toán chỉ báo | Trình tính toán ghi thành công dữ liệu vào Druid datasource | Trình tính toán ghi thành công dữ liệu vào Druid datasource | PASS |
|-----|--|--|---|---|-------------|

Trường hợp kiểm thử ID=1.2 lỗi do trong luồng hoạt động của trình tính toán không có bước tạo Druid datasource kết nối đến Kafka topic. Do vậy, luôn cần cài đặt các Druid datasource và Kafka topic cần thiết trước khi khởi chạy các trình tính toán chỉ báo có sẵn của hệ thống.

b, Kiểm thử chức năng tạo mới chỉ báo cá nhân

| ID | Case | Quy trình test | Kết quả mong muốn | Kết quả thực hiện | Kết luận |
|-----|--|---|--------------------------------|--------------------------------|-------------|
| 2.1 | Sai định dạng dữ liệu đầu vào | Dữ liệu đầu vào có dạng: <code>{... "input_tables": 123456 ... }</code> gửi đến phân hệ | Báo lỗi, không tạo chỉ báo mới | Báo lỗi, không tạo chỉ báo mới | PASS |
| 2.2 | Đúng định dạng dữ liệu đầu vào, tên chỉ báo trùng với chỉ báo có sẵn | Gửi đến dữ liệu với tên chỉ báo trùng với chỉ báo có sẵn | Báo lỗi, không tạo chỉ báo mới | Báo lỗi, không tạo chỉ báo mới | PASS |

| | | | | | |
|-----|--|--|------------------------------------|---|-------------|
| 2.3 | Đúng định dạng dữ liệu, không trùng tên với chỉ báo có sẵn | Gửi dữ liệu đầu vào đúng định dạng và không trùng lặp tên đến phân hệ | Tạo mới trình tính toán thành công | Tạo mới trình tính toán thành công | PASS |
| 2.4 | Đúng định dạng dữ liệu, không trùng tên với chỉ báo có sẵn, sai mã nguồn | Gửi dữ liệu đầu vào đúng định dạng và không trùng lặp tên nhưng mã nguồn ở trường <code>get_inputs_method</code> và <code>formula_method</code> không đúng | Báo lỗi không tạo được chỉ báo mới | Tạo chỉ báo mới, thay đổi DAG nhưng gặp lỗi khi khởi chạy | FAIL |

Từ những trường hợp kiểm thử trong bảng trên, cho thấy dữ liệu đầu vào mà người dùng nhập đóng vai trò quan trọng cho chức năng này. Dữ liệu cần được tổ chức đúng định dạng. Trường hợp kiểm thử ID = 2.4 thất bại do chưa có cơ chế kiểm tra tính đúng đắn của mã nguồn do người dùng nhập vào hệ thống.

CHƯƠNG 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1 Kết luận

Như vậy, đề tài xây dựng phân hệ quản lý và lập lịch trình tính toán chỉ báo đã được hoàn thành, góp phần tạo nên toàn bộ nền tảng dữ liệu chứng khoán. Phân hệ đã hoàn thành các chức năng: tính toán chỉ báo, lập lịch tính toán chỉ báo định kì, tự động tạo mới chỉ báo cá nhân. Chức năng tự động tạo mới chỉ báo cá nhân cũng là điểm nổi trội khi đem so sánh với các ứng dụng theo dõi chứng khoán trên thị trường hiện nay, đáp ứng nhu cầu của người tham gia đầu tư chứng khoán. Tuy nhiên, phân hệ vẫn còn nhiều thiếu sót so với các sản phẩm thực tế, có thể kể đến như tần suất tính toán thấp (1 lần/ ngày) dẫn đến dữ liệu không được cập nhật theo thời gian thực, hay việc triển khai phân hệ trên cụm còn chưa tối ưu.

Trong quá trình thực hiện đồ án này, bản thân tôi đã biết thêm các khái niệm được sử dụng trên thị trường chứng khoán, xây dựng luồng tính toán các chỉ báo, hoàn thiện chức năng tạo mới chỉ báo cá nhân với sự kết hợp của nhiều hệ thống (Druid, Kafka, Airflow) cũng như đóng gói các thành phần bằng Docker. Bên cạnh đó, còn những thiếu sót trong quá trình thực hiện như mã nguồn chưa được tổ chức tối ưu, dựng các thành phần trên môi trường cụm chưa được hoàn thiện. ĐATN đã đem lại cho tôi nhiều kinh nghiệm trong việc thiết kế hệ thống, thiết kế cơ sở dữ liệu, lập trình ứng dụng với Python, đóng gói và triển khai các thành phần hệ thống với Docker. Đây là những kinh nghiệm quý giá cho tôi trong quá trình tiếp tục phát triển chuyên môn trong tương lai.

6.2 Hướng phát triển

Với những ưu và nhược điểm của phân hệ được nêu những phần trên của quyển đồ án, phân hệ tính toán và lập lịch trình tính toán chỉ báo vẫn tiếp tục cần được hoàn thiện. Với chức năng tính toán và lập lịch các chỉ báo, sẽ bổ sung thêm nhiều chỉ báo hơn, cải thiện tốc độ tính toán, tăng tần suất tính toán của các chỉ báo lên 1 lần/ giờ và tiến tới cập nhật dữ liệu thời gian thực.

Bên cạnh đó, để phân hệ hoạt động ổn định hơn, có thể xem xét sử dụng một vài công nghệ và framework như Spark để xử lý dữ liệu streaming hay Apache Dolphin để lập lịch cho các trình tính toán với giao diện thân thiện hơn. Với chức năng tạo mới chỉ báo cá nhân, cần có biện pháp xác thực và kiểm tra dữ liệu mã nguồn được người dùng nhập vào hệ thống, tiến tới trong tương lai sử dụng giao diện đồ họa thân thiện hơn để mở rộng đối tượng người dùng cho tính năng này.

TÀI LIỆU THAM KHẢO

- [1] Python Software Foundation, *Python 3.8.16 documentation*. [Online]. Available: <https://docs.python.org/3.8/>.
- [2] Apache Software Foundation, *What is airflow?* [Online]. Available: <https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html>.
- [3] Apache Software Foundation, *Introduction to apache druid*. [Online]. Available: <https://druid.apache.org/docs/latest/design/index.html>.
- [4] Apache Software Foundation, *Introduction*. [Online]. Available: <https://kafka.apache.org/intro>.
- [5] Tiangolo, *Fastapi*. [Online]. Available: <https://fastapi.tiangolo.com/>.
- [6] Facebook Open Source, *Getting started - react*. [Online]. Available: <https://reactjs.org/docs/getting-started.html>.
- [7] Apache Software Foundation, *Druid design*. [Online]. Available: <https://druid.apache.org/docs/latest/design/architecture.html>.
- [8] Apache Software Foundation, *Druid datasources*. [Online]. Available: <https://druid.apache.org/docs/latest/querying/datasource.html>.
- [9] NumFOCUS, Inc., *Pandas documentation*. [Online]. Available: <https://pandas.pydata.org/docs/>.
- [10] Apache Software Foundation, *Architecture overview*. [Online]. Available: <https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/overview.html>.
- [11] Pallets, *Jinja*. [Online]. Available: <https://jinja.palletsprojects.com/en/3.1.x/>.
- [12] Docker Inc., *Docker overview*. [Online]. Available: <https://docs.docker.com/get-started/overview/>.
- [13] Apache Software Foundation, *Running airflow in docker*. [Online]. Available: <https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html>.