

**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO THỰC TẬP CƠ SỞ  
HỆ THỐNG DATA PIPELINE  
GLAMIRA USER FLOW INSIGHT**

<b>GIẢNG VIÊN HƯỚNG DẪN</b>	<b>TS. KIM NGỌC BÁCH</b>
<b>HỌ VÀ TÊN SINH VIÊN</b>	<b>NGUYỄN HỮU PHÚC</b>
<b>MÃ SINH VIÊN</b>	<b>B22DCAT224</b>
<b>LỚP</b>	<b>E22CQCN04-B</b>

*Hà Nội – 2025*

## MỤC LỤC

<b>DANH MỤC HÌNH VẼ.....</b>	<b>4</b>
<b>DANH MỤC CÁC TỪ VIẾT TẮT.....</b>	<b>6</b>
<b>LỜI MỞ ĐẦU.....</b>	<b>7</b>
<b>CHƯƠNG 1: GIỚI THIỆU VỀ HỆ THỐNG VÀ CÁC CÔNG NGHỆ.....</b>	<b>8</b>
<b>SỬ DỤNG.....</b>	<b>8</b>
1.1. Giới thiệu về hệ thống.....	8
1.1.1. Mục tiêu và ý nghĩa của dự án.....	8
1.1.2. Phạm vi của dự án.....	8
1.1.3. Các bài toán cần giải quyết.....	9
1.1.4. Phương hướng giải quyết bài toán.....	9
1.2. Công nghệ sử dụng.....	10
1.2.1. Hệ điều hành Linux.....	10
1.2.2. Docker.....	12
1.2.3. Database.....	16
1.2.4. Kafka.....	17
1.2.5. Apache Spark.....	22
1.3. Quy trình ETL (Extract, Transform, Loading).....	26
1.3.1. Định nghĩa.....	26
1.3.2. Cách thức hoạt động.....	27
1.3.3. Áp dụng vào hệ thống.....	29
1.4. Mô tả hệ thống.....	30
1.4.1. Kiến trúc tổng quan.....	30
1.4.2. Quy trình ETL.....	31
1.4.3. Đặc điểm nổi bật.....	31
1.5. Kết luận.....	31
<b>CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG.....</b>	<b>32</b>
2.1. Phân tích dữ liệu cần xử lí.....	32
2.1.1. Mô tả dữ liệu.....	32
2.1.2. Những dữ liệu cần lấy ra để phân tích.....	34
2.2. Thiết kế hệ thống.....	35
2.2.1. Quản lý và kiểm tra dữ liệu trong MongoDB.....	36
2.2.2. Dùng python để lấy những dữ liệu cần thiết.....	38
2.2.3. Đẩy dữ liệu từ python lên kafka.....	39
2.2.4. Dùng spark để lấy dữ liệu từ kafka và phân tích.....	41
2.2.5. Đẩy dữ liệu đã phân tích từ Spark lên mysql.....	42
2.2.6. Trực quan hóa dữ liệu.....	43

2.3. Kết luận.....	44
<b>CHƯƠNG 3: CÀI ĐẶT CHƯƠNG TRÌNH.....</b>	<b>45</b>
3.1. Cài đặt Docker.....	45
3.1.1. Cài đặt Docker trên Linux.....	45
3.1.2. Cài đặt Docker Compose.....	46
3.1.3. Cấu hình Quyền Truy cập Docker.....	46
3.1.4. Kiểm tra sau khi cài đặt.....	47
3.2. Cài đặt các container trên Docker.....	48
3.2.1. Cài đặt MongoDB.....	48
3.2.2. Cài đặt MySQL.....	50
3.2.3. Cài đặt Kafka.....	51
3.3. Cài đặt PySpark.....	53
3.3.1. Yêu cầu hệ thống.....	53
3.3.2. Quy trình cài đặt.....	54
3.4. Kết luận.....	56
<b>PHẦN KẾT LUẬN.....</b>	<b>57</b>
<b>TÀI LIỆU THAM KHẢO.....</b>	<b>58</b>

## DANH MỤC HÌNH VẼ

Hình 1.1. Kiến trúc Linux.....	13
Hình 1.2. Các thành phần của Docker Engine.....	15
Hình 1.3. Mô tả cách thức các Docker Client hoạt động và tương tác với nhau. 16	
Hình 1.4. Kiến trúc Docker.....	17
Hình 1.5. Hệ sinh thái của Kafka.....	20
Hình 1.6. Kiến trúc Kafka.....	21
Hình 1.7. Cách thức hoạt động của Kafka.....	23
Hình 1.8. Hình ảnh về Apache Spark.....	24
Hình 1.9. Hệ sinh thái của Apache Spark.....	25
Hình 1.10. Các thành phần của Apache Spark.....	26
Hình 1.11. Kiến trúc của Apache Spark.....	27
Hình 1.12. Cách thức hoạt động ETL.....	29
Hình 1.13. Hệ thống data pipeline.....	32
Hình 2.1. Dữ liệu mẫu.....	34
Hình 2.2. Hệ thống data pipeline.....	38
Hình 2.3. Đưa file chứa dữ liệu lên MongoDB.....	38
Hình 2.4. Kết quả trả về sau khi đưa dữ liệu.....	39
Hình 2.5. Kiểm tra số lượng bản ghi.....	39
Hình 2.6. Kiểm tra dữ liệu đã đưa vào.....	40
Hình 2.7. Kết quả trả về sau khi chạy file main.py.....	42
Hình 2.8. Dữ liệu sau khi được python xử lí.....	43
Hình 2.9. Kết quả trả về sau khi chạy file spark_kafka_consumer.py.....	44
Hình 2.10. Giá trị 10 sản phẩm đầu tiên được trả theo Euro.....	45
Hình 2.11. Giá trị 10 sản phẩm đầu tiên được trả theo USD.....	46
Hình 3.1. Phiên bản của docker và docker compose.....	49
Hình 3.2. Trạng thái dịch vụ của Docker.....	49
Hình 3.3. Khởi động Container MongoDB và kiểm tra.....	51
Hình 3.4. Truy cập vào MongoDB shell.....	51
Hình 3.6. Khởi động Container MySQL và kiểm tra.....	53
Hình 3.7. Truy cập vào MySQL shell.....	53
Hình 3.8. File docker-compose.....	54
Hình 3.9. Khởi động các Container và kiểm tra.....	55
Hình 3.10. Kiểm tra topic.....	55

Hình 3.11. Kiểm tra phiên bản java sau khi cài đặt.....	56
Hình 3.12. Kiểm tra phiên bản python sau khi cài đặt.....	57
Hình 3.13. Kiểm tra phiên bản pyspark sau khi cài đặt.....	58

## DANH MỤC CÁC TỪ VIẾT TẮT

Từ viết tắt	Ý nghĩa
RDBMS	Relational Database Management System
API	Application Programing Interface
JSON	Javascript Object Notation
SQL	Structured Query Language
NoSQL	Not Only SQL
RAM	Random Access Mempory
CPU	Central Processing Unit
JDBC	Java Database Connectivity
BI	Business Intelligence
URI	Uniform Resource Identifier

## LỜI MỞ ĐẦU

Trong bối cảnh phát triển nhanh chóng của công nghệ thông tin và hệ thống thương mại điện tử, dữ liệu trở nên phổ biến và ngày qua ngày lại một nhiều hơn. Dữ liệu chính là chìa khóa, để mở ra những cánh cổng mới với thật nhiều cơ hội cho những ai biết tận dụng sức mạnh của chúng. Từ đó, việc phân tích dữ liệu quy mô lớn đã trở thành một yêu cầu thiết yếu để tối ưu hóa hiệu suất và ra quyết định chiến lược. Xuất phát từ thực tế đó, đề tài “Glamira User Flow Insight” được thực hiện như một phần của môn Thực tập cơ sở nhằm trình bày một cách khoa học và có hệ thống để xây dựng một đường ống dữ liệu. Nhằm xử lý dữ liệu log từ website thương mại điện tử Glamira.

Mục tiêu của hệ thống nhằm xử lý dữ liệu, đảm bảo tính chính xác, hiệu suất và khả năng mở rộng. Luận văn này sẽ trình bày quá trình nghiên cứu, thiết kế và triển khai ứng dụng, đồng thời đánh giá những khó khăn và bài học kinh nghiệm thu được. Thông qua rằng đề tài này, không chỉ giúp nâng cao kỹ năng lập trình mà còn là tiền đề để phát triển các dự án thực tế trong tương lai.

Bố cục của luận văn gồm chương chính:

- Chương 1: Giới thiệu về hệ thống và các công nghệ sử dụng  
Ở chương đầu tiên, tập trung vào giới thiệu tổng quan hệ thống cũng như các công nghệ được sử dụng, từ đó có cái nhìn tổng quát về toàn bộ dự án.
- Chương 2: Phân tích và thiết kế hệ thống  
Chương tiếp theo chính là phân tích sâu và thiết kế hệ thống hoàn chỉnh.
- Chương 3: Cài đặt chương trình  
Chương cuối cùng là các bước để cài đặt phần mềm, công nghệ lên hệ thống, giúp hệ thống hoạt động một cách trơn tru.

# CHƯƠNG 1: GIỚI THIỆU VỀ HỆ THỐNG VÀ CÁC CÔNG NGHỆ

## SỬ DỤNG

### 1.1. Giới thiệu về hệ thống

#### 1.1.1. Mục tiêu và ý nghĩa của dự án

Dự án nhằm mục đích thiết kế và xây dựng một data pipeline có khả năng xử lý, làm sạch và chuyển dữ liệu log của hệ thống thương mại điện tử. Với những mục tiêu sau đây:

- **Đảm bảo tính toàn vẹn dữ liệu:** Triển khai và xây dựng một data pipeline có thể đảm bảo dữ liệu lớn được truyền tải và xử lý mà không bị mất mát, sai lệch và trùng lặp dữ liệu.
- **Tối ưu hóa hiệu suất xử lý:** Giúp tăng tốc độ và hiệu suất xử lý dữ liệu lớn, giảm độ trễ trong thời gian thực để hỗ trợ đưa ra các quyết định kinh doanh.
- **Ứng dụng công nghệ tiên tiến:** Data pipeline được tích hợp các công nghệ và framework hiện đại như Apache Spark, Apache Kafka, ... Từ đó xây dựng được một hệ thống linh hoạt với khả năng mở rộng cao.
- **Tạo giá trị kinh doanh:** Từ dữ liệu đã được xử lý, có thể cung cấp các insight giá trị, hỗ trợ doanh nghiệp tối ưu hóa chiến lược kinh doanh và mang đến những trải nghiệm tốt nhất cho khách hàng.

Dự án mang ý nghĩa quan trọng trong việc giúp các doanh nghiệp, đặc biệt là trong lĩnh vực thương mại điện tử, khai thác tối đa giá trị từ dữ liệu log, từ đó đưa ra các quyết định dựa trên dữ liệu.

#### 1.1.2. Phạm vi của dự án

Hệ thống được thiết kế để xử lý khối lượng dữ liệu lớn, cụ thể là **32GB dữ liệu log**, tương đương với khoảng **41 triệu bản ghi** từ website bán đồ trang sức **Glamira**. Phạm vi của dự án bao gồm:

- **Thu thập dữ liệu:** Thu thập dữ liệu log từ các nguồn khác nhau.
- **Xử lý và làm sạch dữ liệu:** Thực hiện các quy trình làm sạch dữ liệu, loại bỏ trùng lặp, chuẩn hóa định dạng và xử lý các giá trị thiếu hoặc bất thường.



- **Phân tích dữ liệu:** Triển khai các kỹ thuật phân tích dữ liệu như phân tích hành vi khách hàng, phát hiện xu hướng và dự đoán nhu cầu.
- **Lưu trữ và truy vấn:** Lưu trữ dữ liệu đã xử lý trong một kho dữ liệu để hỗ trợ truy vấn hiệu quả và phân tích sâu.

### 1.1.3. Các bài toán cần giải quyết

#### 1.1.3.1. Xử lý dữ liệu lớn

Hệ thống cần xử lý một khối lượng dữ liệu lớn trong thời gian ngắn, với các yêu cầu cụ thể:

- **Tối ưu hóa hiệu suất:** Sử dụng các công nghệ xử lý phân tán như Apache Spark để phân chia khối lượng công việc, giảm thời gian xử lý.
- **Khả năng mở rộng:** Đảm bảo hệ thống có thể xử lý khối lượng dữ liệu tăng trưởng trong tương lai mà không cần tái cấu trúc lớn.
- **Độ tin cậy:** Đảm bảo dữ liệu được xử lý không bị mất mát hoặc hỏng hóc.
- **Hiệu quả tài nguyên:** Tối ưu hóa việc sử dụng CPU, RAM và băng thông mạng để giảm chi phí vận hành.

#### 1.1.3.2. Làm sạch và chuẩn hóa dữ liệu

Dữ liệu log từ website Glamira thường chứa nhiều vấn đề như:

- **Dữ liệu không đồng nhất:** Các bản ghi có định dạng khác nhau do nguồn thu thập đa dạng.
- **Dữ liệu nhiễu:** Bao gồm các bản ghi trùng lặp, giá trị thiếu hoặc dữ liệu không hợp lệ.
- **Tích hợp đa nguồn:** Kết hợp dữ liệu từ nhiều nguồn như log server, sự kiện người dùng và dữ liệu giao dịch, yêu cầu chuẩn hóa schema trước khi xử lý.

### 1.1.4. Phương hướng giải quyết bài toán

#### 1.1.4.1. Xử lý dữ liệu lớn

- Sử dụng Apache Spark với tính toán in-memory và phân tán, chia nhỏ workload qua DataFrame API.
- Tích hợp Apache Kafka để xử lý luồng dữ liệu thời gian thực, giảm độ trễ.

#### 1.1.4.2. Làm sạch và chuẩn hóa dữ liệu

- Dùng Python kết hợp Spark SQL để parse và chuẩn hóa JSON.
- Áp dụng schema-on-read trong Spark để xử lý định dạng đa dạng.
- Loại bỏ trùng lặp bằng Spark dropDuplicates() hoặc Python với window functions.
- Phân tích nguồn, thiết kế pipeline ETL bằng Spark và Python.

## **1.2. Công nghệ sử dụng**

### **1.2.1. Hệ điều hành Linux**

#### **1.2.1.1. Giới thiệu về hệ điều hành Linux**

##### **a. Hệ điều hành Linux là gì?**

Linux là một hệ điều hành phát triển dựa vào hệ điều hành Unix và được phát hành miễn phí. Hệ điều hành này được cài đặt từ máy tính cá nhân đến các server chuyên dụng.

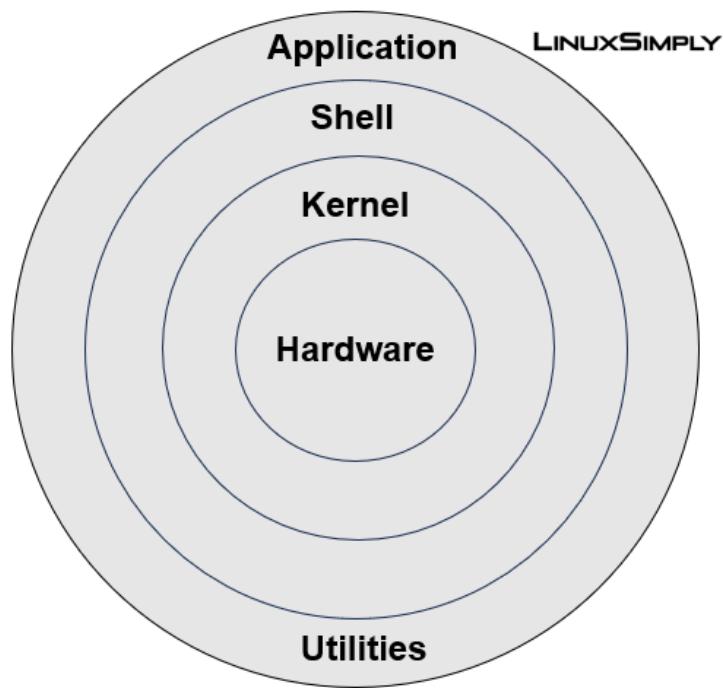
##### **b. Ưu điểm**

- Mã nguồn mở: Miễn phí, cho phép tùy chỉnh và tối ưu theo nhu cầu của mỗi người. Cùng với đó, cộng đồng người sử dụng và phát triển rất lớn.
- Ổn định và đáng tin cậy: Có thể chạy liên tục không cần khởi động lại, phù hợp cho các pipeline ETL và hệ thống Kafka/Spark.
- Hiệu suất cao: Tối ưu hóa tài nguyên như CPU và RAM
- Hỗ trợ multitasking và multi-user

##### **c. Nhược điểm**

- Phức tạp cho người mới sử dụng: yêu cầu phải viết nhiều dòng lệnh hơn ở Windows, cần nhiều thời gian để làm quen với các công cụ
- Khả năng tương thích phần mềm: không hỗ trợ tốt một số phần mềm, giao diện như trên Windows

#### **1.2.1.2. Tổng quan về kiến trúc Linux**



*Hình 1.1. Kiến trúc Linux*

a. Các thành phần

- Kernel: Hay còn được gọi là nhân hệ điều hành, đây là phần cốt lõi và quan trọng nhất, được ví như trái tim của hệ điều hành. Kernel chứa các module, thư viện để quản lý và giao tiếp với phần cứng và các ứng dụng.
- Shell: là một chương trình giao diện dòng lệnh có chức năng thực thi các lệnh từ người dùng hoặc từ các ứng dụng, tiện ích yêu cầu chuyển đến cho Kernel xử lý. Shell là cầu nối quan trọng giữa người dùng và Kernel.
- Applications: Là các phần mềm và tiện ích mà người dùng cài đặt trên hệ điều hành Linux, cung cấp các chức năng để người dùng sử dụng. Các ứng dụng có thể chia thành nhiều loại:
  - Ứng dụng người dùng: bao gồm các phần mềm trình duyệt web, soạn thảo văn bản, các công cụ phát triển code, ...
  - Tiện ích: các công cụ quản trị hệ thống
  - Ứng dụng server: các phần mềm chạy trên máy chủ Linux như MySQL, Apache Spark, Docker, ...

b. Tương tác giữa các thành phần

- Người dùng -> Shell/Application: Người dùng nhập các câu lệnh lên giao diện dòng lệnh hoặc yêu cầu ở các ứng dụng
- Shell/Application -> Kernel: Shell chuyển các yêu cầu từ người dùng hoặc ứng dụng đến kernel để thực thi

- Kernel -> Phần cứng: Kernel giao tiếp trực tiếp với phần cứng, từ đó thực hiện các tác vụ
- Kernel -> Shell/Applications: Kernel sau khi xử lý, trả kết quả xử lý về Shell hoặc các ứng dụng, từ đó hiển thị cho người dùng

## **1.2.2. Docker**

### **1.2.2.1. Giới thiệu về Docker**

#### **a. Docker là gì**

Docker là một nền tảng mã nguồn mở cung cấp các công cụ để các developers, admins/systems có thể phát triển, thực thi, chạy các ứng dụng với containers.

Nói một cách dễ hiểu: Khi chúng ta muốn chạy app thì chúng ta phải thiết lập môi trường chạy cho nó. Thay vì chúng ta sẽ đi cài môi trường chạy cho nó thì chúng ta sẽ chạy docker.

#### **b. Ưu điểm**

- **Tiện lợi và nhanh chóng**

Docker giúp việc cài đặt một dự án mới nhanh gọn và đơn giản hơn rất nhiều. Chỉ cần vài dòng lệnh, có thể nhanh chóng tạo được môi trường ảo hóa chứa đầy đủ những cài đặt cần thiết cho project.

- **Khả năng di động**

Môi trường develop được dựng lên bằng docker có thể chuyển từ người này sang người khác mà không làm thay đổi cấu hình ở trong.

- **Hiệu quả tài nguyên**

Các containers nhẹ hơn máy ảo, giảm sử dụng CPU, RAM. Phù hợp cho các tác vụ xử lý dữ liệu lớn

#### **c. Nhược điểm**

- **Khó khăn cho người mới**

Người dùng mới phải học nhiều khái niệm như container, image, Dockerfile, ... trước khi bắt đầu làm việc với Docker. Vì sự phức tạp đó mà khá khó khăn cho người mới bắt đầu

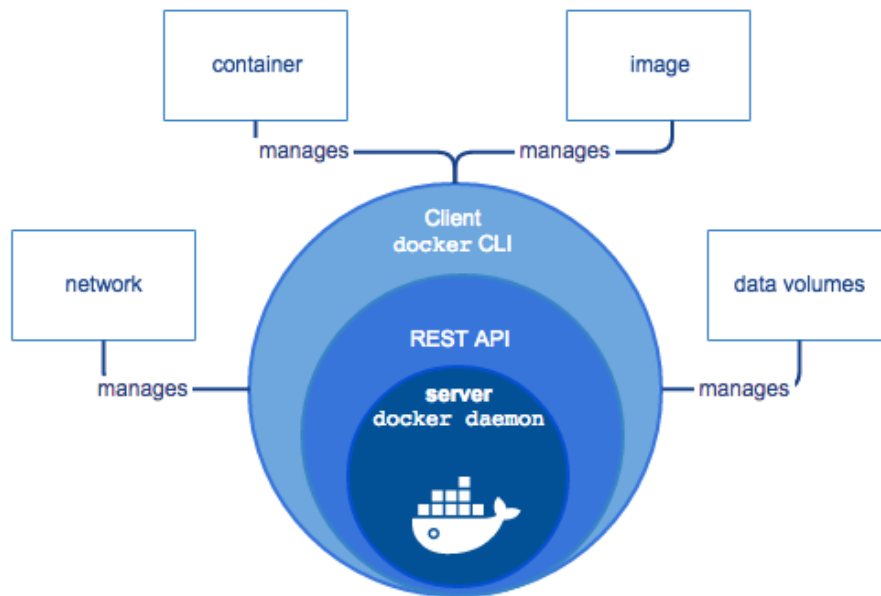
- **Phức tạp trong việc quản lý**

Khi triển khai nhiều container trên nhiều host, việc quản lý trở nên phức tạp. Đòi hỏi người quản lý phải có nhiều kinh nghiệm và học thêm các công cụ hỗ trợ như Kubernetes hay Docker Swarm.

### **1.2.2.2. Tổng quan về kiến trúc Docker**

#### **1.2.2.2.1. Các thành phần cơ bản của Docker**

##### **a. Docker Engine**



*Hình 1.2. Các thành phần của Docker Engine*

Docker engine là một ứng dụng client-server. Nó cung cấp nền tảng để xây dựng, chạy và quản lý các container. Cùng với đó, nó đảm bảo tính nhất quán khi triển khai container trên nhiều môi trường khác nhau.

Docker Engine có 2 phiên bản phổ biến:

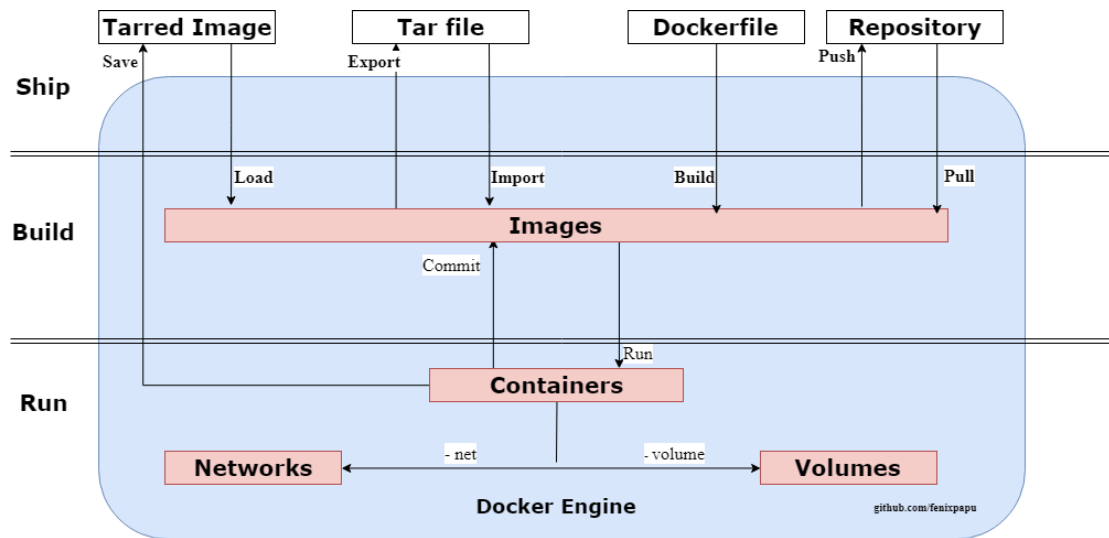
- **Docker Community Edition (CE):** Là phiên bản miễn phí, mã nguồn mở. Nó cung cấp các tính năng cơ bản, nhưng không đi kèm hỗ trợ thương mại.
- **Docker Enterprise(EE):** Là phiên bản trả phí, khi sử dụng phiên bản này bạn sẽ nhận được sự hỗ trợ của nhà phát hành, có thêm các tính năng quản lý và bảo mật hệ thống.

Các thành phần chính của docker engine gồm có:

- **Server:**
  - Hay còn được gọi là docker daemon, sẽ chịu trách nhiệm tạo và quản lý các Docker objects như images, containers, networks, volume.
  - Nó nhận yêu cầu từ Docker Client qua REST API, sau đó thực thi các thao tác.
  - Có thể giao tiếp với các daemon khác trong cluster.
- **REST API:**
  - Là giao diện lập trình ứng dụng, Docker Daemon cung cấp các API cho Docker Client sử dụng để thao tác với Docker.

- Hỗ trợ giao tiếp thông qua UNIX socket, được mặc định trên Linux

- **Docker Client:**



*Hình 1.3. Mô tả cách thức các Docker Client hoạt động và tương tác với nhau*

- Là giao diện dòng lệnh, nơi mà người dùng sử dụng để tương tác với Docker Daemon. Người dùng có thể gửi các câu lệnh ở đây
- Nó gửi các câu đến Docker Daemon thông qua REST API.
- Thực hiện các việc: ship, build, run.
- Phần đầu: Từ Docker file ta có thể build ra một image, hoặc từ một repository ta có thể pull image về. Ngược lại, từ các images ta có thể push lên repository. Đó chính là Ship.
- Phần giữa: Từ images ta có thể run thành một container, hoặc từ container ta có thể commit để tạo ra images. Đó là phần Build.
- Phần cuối: Trong quá trình containers được chạy. Nó có thể liên kết với nhau (networks) và lưu trữ dữ liệu đâu đó ngoài container (volumes). Đó là phần Run

**b. Distribution Tools: công cụ quản lý phân tán các images**

- **Docker Registry:** là một mã nguồn mở, kho lưu trữ giúp bạn tự lưu trữ và quản lý các Docker images.
- **Docker Hub:** Được cung cấp bởi docker, mặc định Docker Client sẽ sử dụng Docker Hub.
- **Docker Trusted Registry:** Là công cụ quản lý và lưu trữ images có tính phí, giải pháp registry riêng tư, an toàn cho doanh nghiệp.

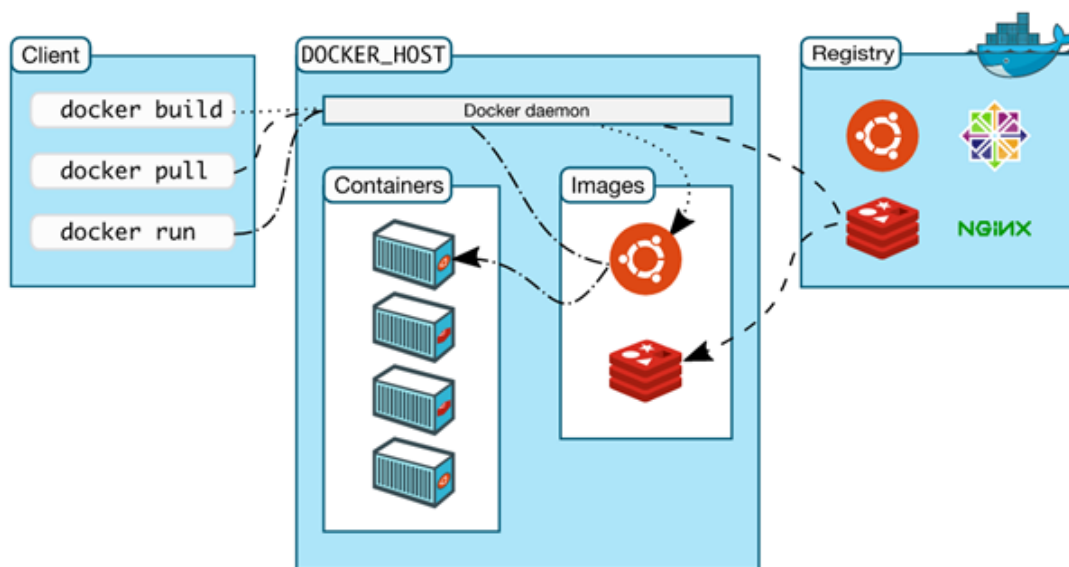
**c. Orchestration Tools: công cụ điều phối**

- **Docker Machine:** Cho phép tạo và cấu hình Docker Engine trên máy tính của bạn, trên các dịch vụ cloud, ... Docker tạo server và cài đặt lên chúng, tiếp đó cấu hình để Docker Client có thể giao tiếp được với Docker server.
- **Docker Compose:** Là tool cho việc định nghĩa và chạy multi-container qua file YAML, phù hợp cho pipeline ETL với nhiều dịch vụ (Spark, Kafka).
- **Docker Swarm:** Là tool giúp phân cụm cho Docker containers.

#### d. Thành phần khác

- **Dockerfile:** là một script chứa các lệnh tuần tự để xây dựng Docker image từ một image gốc.
- **Docker Toolbox:** Bộ công cụ hỗ trợ chạy Docker trên Windows/macOS bằng máy ảo Linux, cài đặt Docker Engine và các thành phần liên quan.

#### 1.2.2.2.2. Kiến trúc của Docker



*Hình 1.4. Kiến trúc Docker*

Docker sử dụng kiến trúc client-server.

#### **Docker client**

Là giao diện dòng lệnh người dùng sử dụng để gửi các lệnh đến Docker.

- **docker build:** dùng để xây dựng một Docker image từ Dockerfile
- **docker pull:** tải một image từ registry
- **docker run:** khởi chạy một container từ một image

Client giao tiếp trực tiếp với Docker Daemon trên Docker Host thông qua UNIX Socket hoặc TCP

### **Docker Host**

Nơi Docker Daemon và các tài nguyên chính của Docker được quản lý. Bao gồm ba thành phần chính:

- Docker Daemon: nhận lệnh từ Docker Client, quản lý trực tiếp các containers và images
- Container: được tạo và quản lý bởi Docker Daemon, được khởi tạo từ images
- Images: được quản lý bởi Docker Daemon, được tải từ Registry

### **Registry**

Là nơi lưu trữ các Docker images, nơi mà Docker Daemon tương tác để tải hoặc đẩy images

Luồng hoạt động

- Người dùng gửi lệnh qua Client
  - Người dùng nhập các lệnh thông qua Docker Client
  - Client gửi lệnh đến Docker Daemon qua REST API.
- Docker Daemon xử lý
  - Nếu lệnh là docker pull, daemon sẽ tải image từ Registry về
  - Nếu lệnh là docker run, daemon sử dụng images có sẵn để tạo container mới, rồi khởi chạy nó
- Quản lý
  - Docker Daemon tiếp tục quản lý các containers đang chạy

#### **1.2.2.3. Triển khai trong hệ thống**

Docker hỗ trợ triển khai MongoDB, Kafka, Mysql trong container, đảm bảo tính nhất quán cho pipeline.

### **1.2.3. Database**

Cơ sở dữ liệu giúp lưu trữ, quản lý, truy xuất dữ liệu một cách hiệu quả. Trong hệ thống, sử dụng hai hệ quản trị cơ sở dữ liệu phổ biến là MongoDB và MySQL.

#### **a. MongoDB:**

- MongoDB là một hệ quản trị cơ sở dữ liệu NoSQL mã nguồn mở đa nền tảng viết bằng C++. Bản ghi trong MongoDB được lưu trữ dạng một dữ liệu văn bản, là một cấu trúc dữ liệu bao gồm các cặp giá trị và trường tương tự như các đối tượng JSON.



- Đặc điểm:
  - Lưu trữ dữ liệu dạng văn bản
  - Có khả năng mở rộng ngang
  - Hiệu suất cao với dữ liệu không cấu trúc
- Ứng dụng trong hệ thống
  - Lưu trữ dữ liệu log của Website Glamira, là điểm khởi đầu của đường ống dữ liệu.

## b. MySQL

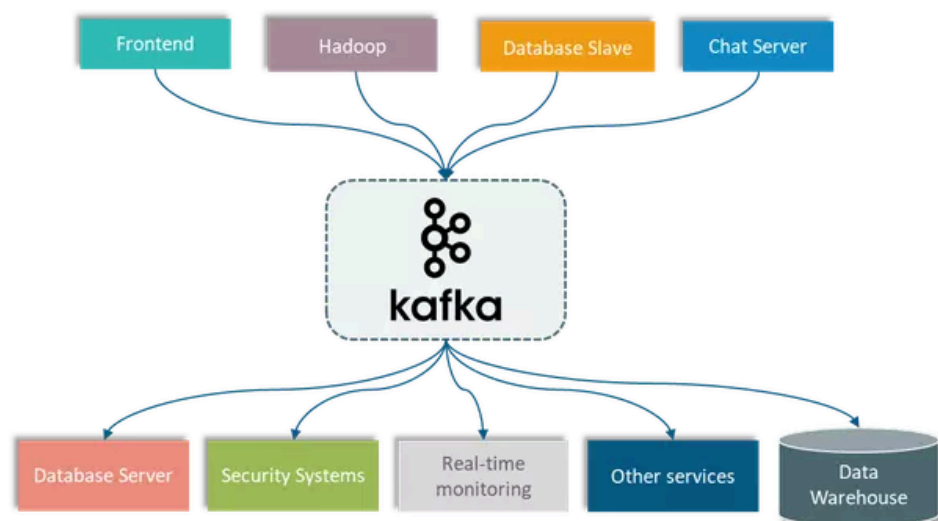
- MySQL là một hệ quản trị cơ sở dữ liệu quan hệ phổ biến, dựa trên ngôn ngữ SQL.
- Đặc điểm:
  - Lưu trữ dữ liệu có cấu trúc, dễ dàng thực hiện các truy vấn phức tạp
  - Hỗ trợ tính toán vẹn dữ liệu, bảo mật cao
- Ứng dụng trong hệ thống
  - Là điểm hứng dữ liệu từ Spark đẩy vào
  - Đảm bảo dữ liệu được lưu trữ có cấu trúc.

## 1.2.4. Kafka

### 1.2.4.1. Giới thiệu về Kafka

#### a. Kafka là gì?

Kafka là một nền tảng streaming dữ liệu phân tán, xử lý lượng lớn dữ liệu theo thời gian thực. Nó hoạt động như một hệ thống message queue hoặc distributed log, cho phép ứng dụng gửi, lưu trữ và xử lý dữ liệu một cách hiệu quả.



### *Hình 1.5. Hệ sinh thái của Kafka*

#### **b. Ưu điểm**

Sử dụng Apache Kafka mang lại nhiều lợi ích cho việc xây dựng và triển khai các hệ thống xử lý dữ liệu thời gian thực và streaming.

- **Xử lý dữ liệu thời gian thực**

Được thiết kế để xử lý dữ liệu thời gian thực và streaming.

- **Khả năng mở rộng dễ dàng**

Kafka có khả năng mở rộng theo chiều ngang một cách dễ dàng bằng cách thêm các node mới vào cluster mà không cần thời gian ngừng hoạt động. Điều này cho phép Kafka xử lý khối lượng dữ liệu khổng lồ và lưu lượng truy cập tăng đột biến, đảm bảo dữ liệu không bị mất mát.

- **Dữ liệu đa dạng**

Kafka không chỉ hỗ trợ dữ liệu thông thường mà còn có khả năng xử lý dữ liệu đa dạng như dữ liệu logs, ...

- **Lưu trữ dữ liệu lâu dài**

Kafka có khả năng lưu trữ dữ liệu lâu dài, cho phép truy xuất lại những dữ liệu quan trọng trong tương lai.

- **Khả năng chia lượng lớn**

Nhờ sử dụng phân vùng, Kafka có khả năng xử lý dữ liệu lớn hiệu quả

#### **c. Nhược điểm**

- **Độ phức tạp trong cài đặt ban đầu cũng như quản lý**

Kafka là một hệ thống phức tạp với nhiều thành phần, việc cài đặt và cấu hình ban đầu có thể phức tạp với người bắt đầu sử dụng

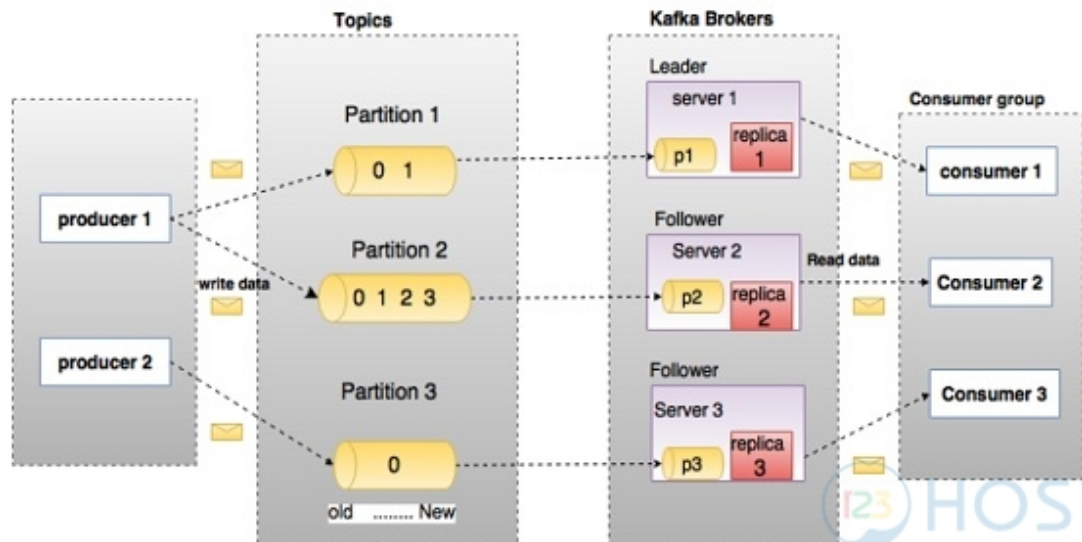
- **Yêu cầu tài nguyên cao**

Kafka yêu cầu một số lượng tài nguyên tương đối lớn như RAM, CPU để hoạt động tốt

- **Khó khăn trong việc sửa đổi dữ liệu**

Do thiết kế bất biến của Kafka, việc sửa đổi hoặc xóa dữ liệu đã được ghi vào topic là rất khó.

#### **1.2.4.2. Tổng quan về kiến trúc Kafka**



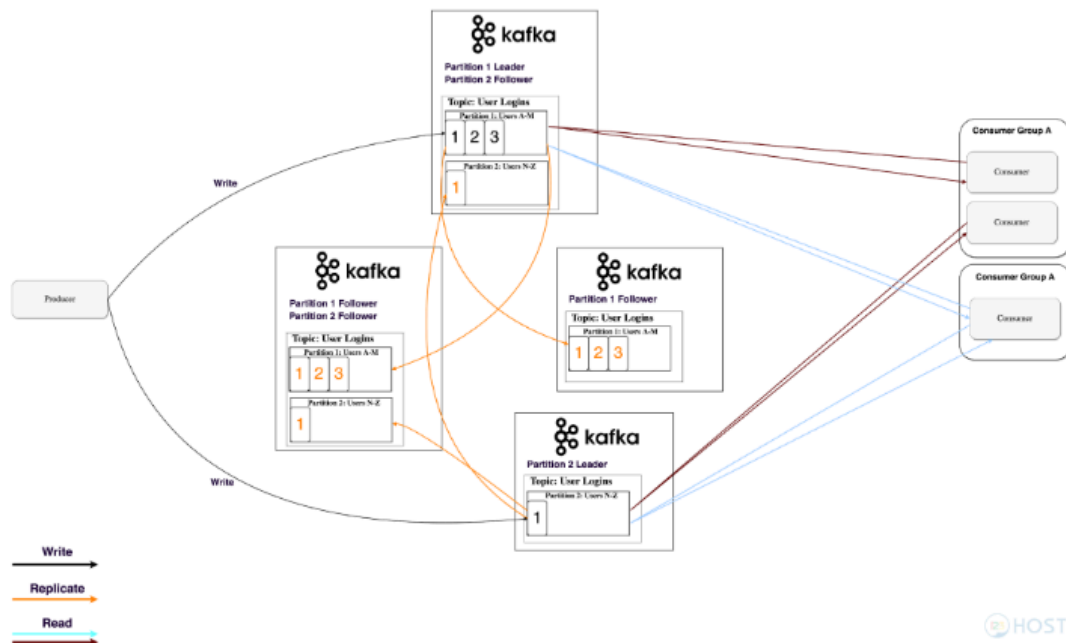
*Hình 1.6. Kiến trúc Kafka*

### Các thuật ngữ:

- Topic
  - Kafka gom các thông điệp cùng loại lại thành một danh mục gọi là topic.
- Partition:
  - Mỗi topic được chia thành nhiều partition để phân tán dữ liệu và tăng khả năng xử lý song song. Partition là đơn vị lưu trữ vật lý của topic trên các broker
  - Ý nghĩa: Partition giúp Kafka mở rộng quy mô bằng cách phân phối dữ liệu trên nhiều brokers và cho phép nhiều consumer đọc dữ liệu song song
- Partition offset:
  - Mỗi thông điệp được lưu trên partition sẽ được gán với một giá trị theo thứ tự tăng dần gọi là offset
  - Giống như index của mảng
- Replicas of Partition
  - Là bản sao của partition, được lưu trữ trên các broker khác nhau. Chỉ được dùng để tránh mất dữ liệu, không dùng để đọc bản ghi
- Broker
  - Là một Kafka server
  - Nếu ra chạy đồng thời nhiều broker, thì ta sẽ gọi đó là một Kafka cluster
  - Vai trò:
    - Lưu trữ partition của các topic

- Xử lý message từ producers và consumers
  - Phối hợp với các broker khác thông qua Zookeeper
- Ví dụ: Nếu cluster có 3 broker:
  - Broker 1 lưu partition 1 và 2
  - Broker 2 lưu partition 3
  - Broker 3 lưu replica của partition 1,2,3
- Producers:
  - Ứng dụng truyền messages đến cho Kafka Server (Broker, node) thông qua các Topic
  - Producer truyền messages đến Broker thì Broker sẽ đưa messages đó đến cuối hàng đợi của một partition
  - Producer có thể chọn gửi cho partition mong muốn
- Consumers
  - Ứng dụng đọc messages từ Broker thông qua các Topic
  - Có thể đọc từ 1 hay nhiều topic
  - Theo dõi offset để biết vị trí đọc hiện tại
  - Nhiều consumer có thể cùng đọc từ một topic (consumer group)
- Leader
  - Là broker chịu trách nhiệm chính cho việc đọc/ghi dữ liệu của một partition
- Follower
  - Là các replica của partition, lưu trữ bản sao và đồng bộ với leader
  - Nếu leader bị lỗi, Zookeeper sẽ chọn một follower để trở thành leader mới
- Zookeeper: được dùng để quản lý và điều phối các broker trong cluster
  - Lưu trữ metadata
  - Phát hiện lỗi broker và thông báo cho producer/consumer
  - Thực hiện leader election khi leader của một partition bị lỗi

#### **1.2.4.3. Cách thức hoạt động**



*Hình 1.7. Cách thức hoạt động của Kafka*

a. Producer gửi messages

- Gửi đến một topic trong Kafka, topic này được chia thành nhiều partition để dễ xử lý
- Mỗi partition có một leader (broker chịu trách nhiệm chính) và các follower (các broker khác chứa bản sao của partition)

b. Partition và thứ tự messages

- Mỗi partition giống như một hàng đợi, lưu trữ messages theo thứ tự đến (dựa trên offset)
- Kafka đảm bảo rằng các messages trong cùng một partition được xử lý theo đúng thứ tự

c. Consumer đọc messages

- Đọc messages từ topic để xử lý, đọc những gì nó muốn
- Theo dõi offset để biết mình đọc đến đâu trong partition
- Kafka lưu trữ messages trong thời gian dài. Consumer có thể đọc lại dữ liệu cũ nếu cần

d. Consumer Group và vấn đề trùng lặp/thứ tự

- Một consumer có thể đọc nhiều partition
- Nhưng một partition chỉ đọc được bởi một consumer duy nhất thuộc một group

e. Vai trò của Zookeeper trong việc tìm Leader (quản lý thông tin)

- Lưu trữ metadata của cluster

- Khi producer muốn gửi messages, nó hỏi một broker bất kì
- Broker này đã giao tiếp với Zookeeper trước đó, nên nó biết broker nào là leader của partition cần gửi
- Broker trả lời producer, và producer gửi messages đến đúng leader

## 1.2.5. Apache Spark

### 1.2.5.1. Giới thiệu về Apache Spark

#### a. Apache Spark là gì ?



*Hình 1.8. Hình ảnh về Apache Spark*

Apache Spark là một hệ thống xử lý phân tán mã nguồn mở được sử dụng cho các khối lượng công việc dữ liệu lớn. Spark cho phép xử lý dữ liệu theo thời gian thực, vừa nhận dữ liệu từ các nguồn khác nhau đồng thời thực hiện ngay việc xử lý trên dữ liệu vừa nhận được ( Spark Streaming).

#### b. Ưu điểm

Apache Spark mang lại một số lợi ích, đặc biệt là về tốc độ xử lý cao, hỗ trợ phân tích phức tạp và tính dễ sử dụng.

- **Tốc độ và hiệu suất cao**

Một lợi thế chính của Apache Spark là khả năng xử lý với tốc độ và hiệu suất cao, đặc biệt là khi so sánh với Hadoop MapReduce. Điều đó đến từ khả năng lưu trữ dữ liệu trong bộ nhớ RAM thay vì liên tục ghi và đọc dữ liệu vào ra khỏi đĩa.

- **Hỗ trợ đa ngôn ngữ**

Mặc dù được viết bằng Scala, nhưng nó được mở rộng để hỗ trợ cho các ngôn ngữ khác như Java, Python và R. Làm cho Spark trở nên linh hoạt hơn và dễ dàng học tập cũng như thực hành.

- **Khả năng phân tích nâng cao**

Nó kết hợp với Spark SQL để xử lý dữ liệu có cấu trúc, MLlib để học máy, GraphX để xử lý đồ thị và Spark Streaming. Cho phép người dùng thực hiện các tác vụ phức tạp.

- **Hệ sinh thái đa dạng**

Nó tương thích với nhiều hệ thống và tích hợp với nhiều nền tảng khác nhau.



*Hình 1.9. Hệ sinh thái của Apache Spark*

### c. Nhược điểm

Mặc dù khả năng của Spark rất ấn tượng, nhưng vẫn có những điểm hạn chế.

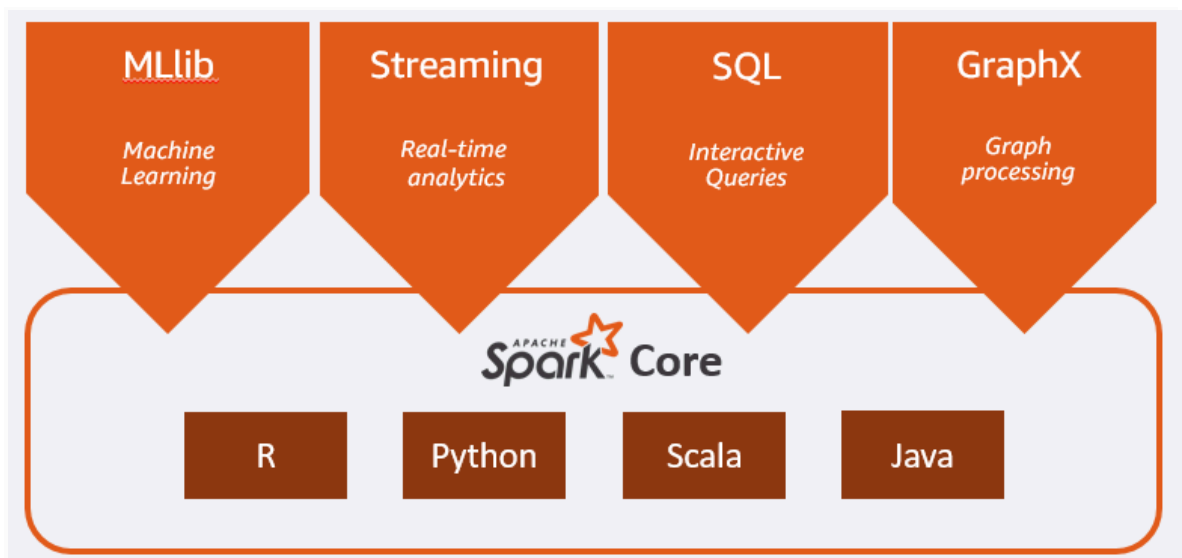
- **Tiêu thụ bộ nhớ**

Spark tận dụng rất nhiều RAM cho các phép tính trong bộ nhớ, cung cấp khả năng xử lý dữ liệu với tốc độ cao. Đặc điểm này dẫn đến mức tiêu thụ bộ nhớ cao, dẫn đến chi phí vận hành sẽ lớn.

- **Xử lý thời gian thực còn hạn chế**

Thực tế Apache Spark không có khả năng xử lý thời gian thực một cách thực sự. Bởi vì thực chất Spark Streaming phụ thuộc vào các lô nhỏ, xử lý các nhóm nhỏ được thu nhập trong một khoảng thời gian xác định trước.

#### 1.2.5.2. Thành phần của Apache Spark



*Hình 1.10. Các thành phần của Apache Spark*

**Spark Core:** làm nền móng cho nền tảng, thực hiện quản lý bộ nhớ, phục hồi lỗi, lên lịch, phân phối và giám sát tác vụ; tương tác với hệ thống lưu trữ. Cung cấp API cho Java, Scala, Python, R, ẩn phức tạp xử lý phân tán qua các toán tử cấp cao.

**MLlib:** là một nền tảng học máy phân tán.

**Spark Streaming:** được sử dụng để xử lý luồng dữ liệu liên tục.

**Spark SQL:** cung cấp SchemaRDD nhằm hỗ trợ cho cả kiểu dữ liệu có cấu trúc và dữ liệu nửa cấu trúc.

**GraphX:** khung phân tán cho ETL, phân tích và tính toán đồ thị lặp. Cung cấp API linh hoạt và bộ thuật toán đồ thị, hỗ trợ xây dựng, chuyển đổi cấu trúc đồ thị quy mô lớn.

### 1.2.5.3. Kiến trúc Apache Spark

#### 1.2.5.3.1. RDD và DAG

Về bản chất, kiến trúc của Apache Spark được thiết kế cho điện toán phân tán trong bộ nhớ, giúp xử lý dữ liệu lớn với độ đáng kinh ngạc

Hai khái niệm chính trong kiến trúc Spark là Resilient Distributed Dataset (RDD) và Directed Acyclic Graph (DAG).

##### a. Resilient Distributed Dataset (RDD)

RDD là một cấu trúc dữ liệu trừu tượng trong Spark, đại diện cho một tập hợp các đối tượng không thể thay đổi, được phân vùng và phân tán trên các node trong cluster. RDD được thiết kế để xử lý song song và đảm bảo khả năng chịu lỗi.



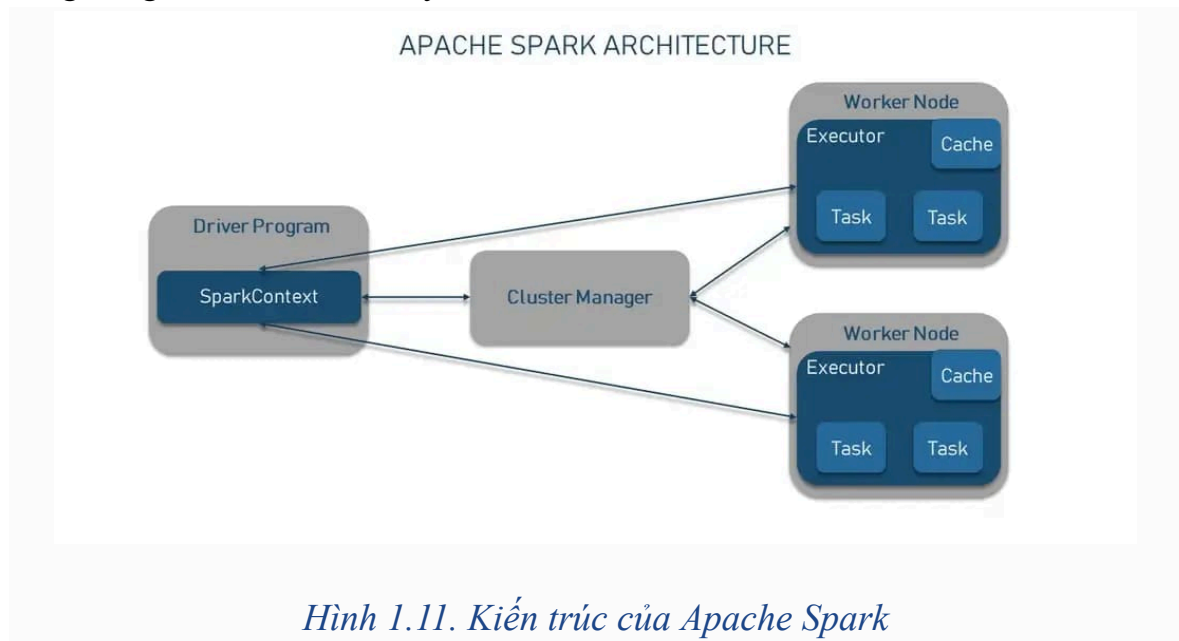
- Khả năng phục hồi (Resilient): dữ liệu có thể phục hồi sau lỗi thông qua cơ chế tái tạo dữ liệu. Bằng cách áp dụng lại các phép biến đổi từ dữ liệu gốc.
- Khả năng phân tán (Distributed): RDD được chia thành các phân vùng, mỗi phân vùng được lưu trữ và xử lý trên một node khác nhau trong cluster. Nhờ đó, Spark có thể thực hiện các tác vụ song song.
- Bộ dữ liệu (Dataset): RDD có thể chứa bất kì loại dữ liệu nào và có thể được tạo từ dữ liệu được lưu trữ trong hệ thống tệp cục bộ.

### b. Directed Acyclic Graph (DAG)

DAG là một đồ thị có hướng không chứa chu trình, Spark dùng để biểu diễn trình tự các phép biến đổi áp dụng lên RDD.

#### 1.2.5.3.2. Kiến trúc Master-Slave

Kiến trúc của Apache Spark dựa trên mô hình master-slave, trong đó có một master node điều phối công việc, và nhiều slave node thực thi các tác vụ. Mô hình này đảm bảo quản lý tài nguyên một cách có hiệu quả và có thể thực thi song song trên một cụm máy tính.



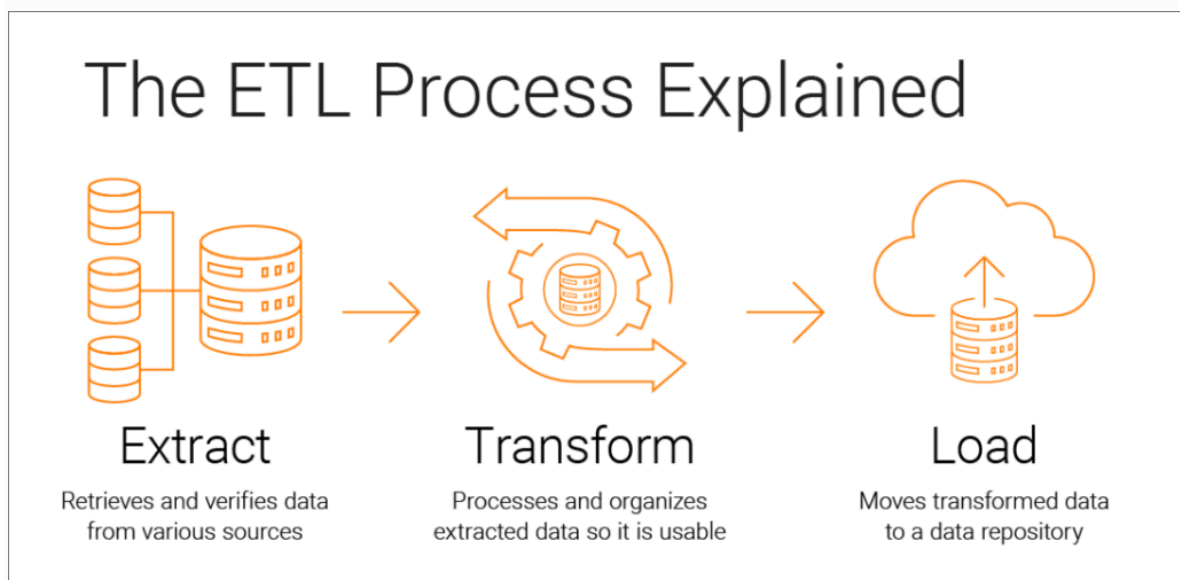
*Hình 1.11. Kiến trúc của Apache Spark*

- Driver Program: Là master node, chịu trách nhiệm chạy hàm chính của Spark và tạo SparkContext, đóng vai trò là điểm vào cho tất cả các chức năng của Spark. SparkContext quản lý việc giao tiếp với Cluster Manager, phân vùng công việc thành các tasks, và phân bổ các tasks này cho các worker nodes.
- Cluster Manager: là thành phần trung gian, chịu trách nhiệm phân bổ tài nguyên trong cụm.
- Worker Node: Là các slave nodes trong cụm, chịu trách nhiệm thực thi các tác vụ được giao từ Driver

### Cơ chế hoạt động:

- Driver Program khởi tạo ứng dụng Spark, tiếp đó tạo SparkContext. Và xây dựng DGA dựa trên các phép biến đổi RDD.
- Sau đó, SparkContext giao tiếp với Cluster Manager. Nó yêu cầu tài nguyên CPU và bộ nhớ, để phân bổ tới các Worker Node.
- Sau khi phân bổ tài nguyên, Cluster Manager khởi tạo các Executor trên Worker Node. Và cung cấp thông tin về tài nguyên cho Driver.
- Cuối cùng, Driver gửi các tasks tới các Executor thông qua Task Scheduler. Các Executor thực thi các tasks trên các phân vùng của RDD, rồi trả kết quả về cho Driver.

### 1.3. Quy trình ETL (Extract, Transform, Loading)



*Hình 1.12. Quy trình ETL*

#### 1.3.1. Định nghĩa

Quy trình ETL, viết tắt của Extract, Transform, và Load là một quá trình quan trọng trong việc lấy và xử lý dữ liệu từ các nguồn RDBMS khác nhau để nạp chúng vào hệ thống Data Warehouse.

Quá trình này bao gồm ba bước chính, bao gồm:

- **Extract:** Đây là bước đầu tiên, trong đó dữ liệu được thu thập từ nhiều nguồn khác nhau, chẳng hạn như cơ sở dữ liệu và hệ thống khác. Quá trình này đảm bảo rằng tất cả dữ liệu cần thiết được lấy ra để chuẩn bị cho các bước tiếp theo.

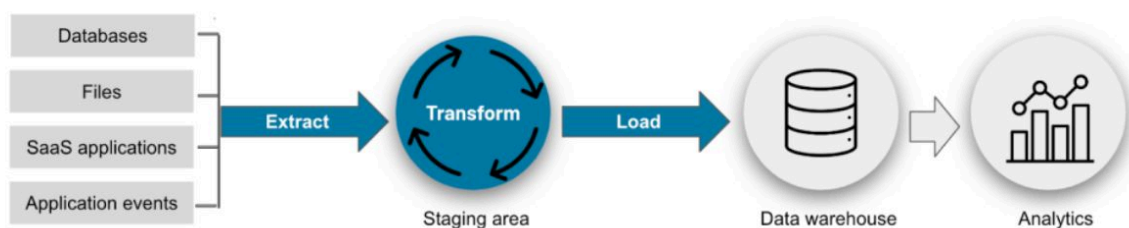
- **Transform:** Sau khi dữ liệu đã được trích xuất, nó cần phải được biến đổi để phù hợp với biểu đồ dữ liệu của hệ thống đích. Điều này có thể bao gồm việc nối chuỗi, tính toán, hoặc các biến đổi dữ liệu khác dựa trên quy tắc hoặc bảng tra cứu. Trong quá trình này, dữ liệu được làm mới và chuẩn hóa để đảm bảo tính nhất quán và chính xác.
- **Load:** Sau khi dữ liệu đã được biến đổi, nó được ghi vào cơ sở dữ liệu đích, chuẩn bị cho việc truy vấn và phân tích. Quá trình này thường là quá trình cuối cùng trong quy trình ETL và có thể thực hiện thông qua việc sao chép dữ liệu trực tiếp vào cơ sở dữ liệu đích trước khi thực hiện bất kỳ biến đổi nào.

Nói chung, ETL là quá trình quan trọng để di chuyển dữ liệu từ nguồn đến đích và đảm bảo rằng dữ liệu đó đã được xử lý và chuẩn bị tốt trước khi sử dụng.

### 1.3.2. Cách thức hoạt động

Quy trình ETL (Extract, Transform, Load) là một phương pháp chuyển đổi dữ liệu từ hệ thống nguồn sang hệ thống đích trong các vòng lặp thường xuyên. Quy trình ETL bao gồm ba giai đoạn quan trọng:

1. Trích xuất dữ liệu có liên quan từ cơ sở dữ liệu nguồn
2. Chuyển đổi dữ liệu để phù hợp hơn cho việc phân tích
3. Tải dữ liệu vào cơ sở dữ liệu đích



*Hình 1.12. Cách thức hoạt động ETL*

#### a. Extract - Giai đoạn trích xuất

Giai đoạn đầu tiên của quá trình ETL được gọi là Extract - Trích xuất. Đây là bước quan trọng nhất trong quy trình, liên quan trực tiếp đến việc lấy dữ liệu từ nhiều nguồn khác nhau.

Trong bước đầu này, dữ liệu có thể tồn tại ở nhiều dạng, từ dữ liệu không có cấu trúc đến dữ liệu có cấu trúc, và chúng cần được thu thập và hợp nhất vào

một kho lưu trữ duy nhất. Dữ liệu thô có thể xuất phát từ nhiều nguồn đa dạng, bao gồm:

- Hệ thống lưu trữ dữ liệu.
- Hệ thống quản lý thông tin khách hàng (CRM).
- Thiết bị và ứng dụng di động.
- Ứng dụng tiếp thị và quản lý bán hàng.
- Cơ sở dữ liệu hiện có.
- Các công cụ phân tích.
- Kho dữ liệu.

## **b. Transform - Giai đoạn chuyển đổi**

Giai đoạn thứ hai của quá trình ETL được gọi là Transform - Biến đổi. Trong quá trình biến đổi dữ liệu, ETL thực hiện việc chuyển đổi và tổng hợp dữ liệu thô trong khu vực lưu trữ tạm thời để sẵn sàng cho việc nhập dữ liệu vào kho lưu trữ chính. Các bước biến đổi dữ liệu này có thể liên quan đến nhiều loại chuyển đổi dữ liệu khác nhau, bao gồm:

- **Làm sạch dữ liệu**

Đây là quá trình loại bỏ lỗi và ánh xạ dữ liệu nguồn sang định dạng dữ liệu đích.

- **Chống trùng lặp dữ liệu**

Đây là quá trình xác định và loại bỏ các bản ghi trùng lặp trong dữ liệu.

- **Sửa đổi định dạng dữ liệu**

Quá trình này chuyển đổi các đơn vị đo lường, đơn vị thời gian và định dạng dữ liệu khác nhau thành một định dạng thống nhất.

- **Chuyển đổi dữ liệu nâng cao**

Quá trình này sử dụng các quy tắc kinh doanh để tối ưu hóa dữ liệu để phân tích dễ dàng hơn.

- **Gộp ghép**

Trong quá trình chuẩn bị dữ liệu, gộp ghép liên kết các dữ liệu giống nhau từ các nguồn dữ liệu khác nhau.

- **Chia tách**

Chuyển đổi một cột hoặc thuộc tính dữ liệu thành nhiều cột trong hệ thống đích.

- **Tổng hợp**

Tổng hợp cải thiện chất lượng dữ liệu bằng cách giảm số lượng giá trị dữ liệu thành một tập dữ liệu nhỏ hơn.

- **Mã hóa**

Bảo vệ dữ liệu nhạy cảm bằng cách thêm mã hóa trước khi luồng dữ liệu được truyền đến cơ sở dữ liệu đích để đảm bảo tuân thủ luật dữ liệu hoặc quyền riêng tư của dữ liệu.

### **c. Load - Giai đoạn tải**

Giai đoạn cuối của quá trình ETL được gọi là Load - Tải.

Trong quá trình tải dữ liệu, các công cụ ETL (Extract, Transform, Load) di chuyển dữ liệu đã biến đổi từ khu vực lưu trữ tạm thời sang kho dữ liệu đích. Đối với hầu hết các tổ chức sử dụng ETL, quy trình này được tự động hóa, được xác định rõ ràng, diễn ra liên tục và theo lịch trình định sẵn. Dưới đây là hai phương pháp chính để thực hiện quá trình tải dữ liệu:

- **Tải hoàn toàn**

Ở chế độ này, toàn bộ dữ liệu từ nguồn được chuyển đổi và tải vào kho dữ liệu. Quá trình tải hoàn toàn thường diễn ra trong lần đầu tiên bạn chuyển dữ liệu từ hệ thống nguồn vào kho dữ liệu.

- **Tải tăng dần**

Trong quá trình tải tăng dần, công cụ ETL tải sự thay đổi giữa hệ thống đích và nguồn theo khoảng thời gian đều đặn. Công cụ này duy trì thông tin về ngày trích xuất cuối cùng để đảm bảo rằng chỉ có các bản ghi thay đổi sau ngày này mới được tải. Có hai cách để thực hiện tải tăng dần:

- **Tải tăng dần theo luồng**

Đối với khối lượng dữ liệu nhỏ, ta có thể truyền các thay đổi liên tục qua đường ống dữ liệu đến kho dữ liệu đích. Khi tốc độ dữ liệu tăng lên hàng triệu sự kiện mỗi giây, lúc này có thể sử dụng xử lý luồng sự kiện để theo dõi và xử lý dữ liệu trực tiếp để đưa ra quyết định kịp thời hơn.

- **Tải gia tăng theo hàng loạt**

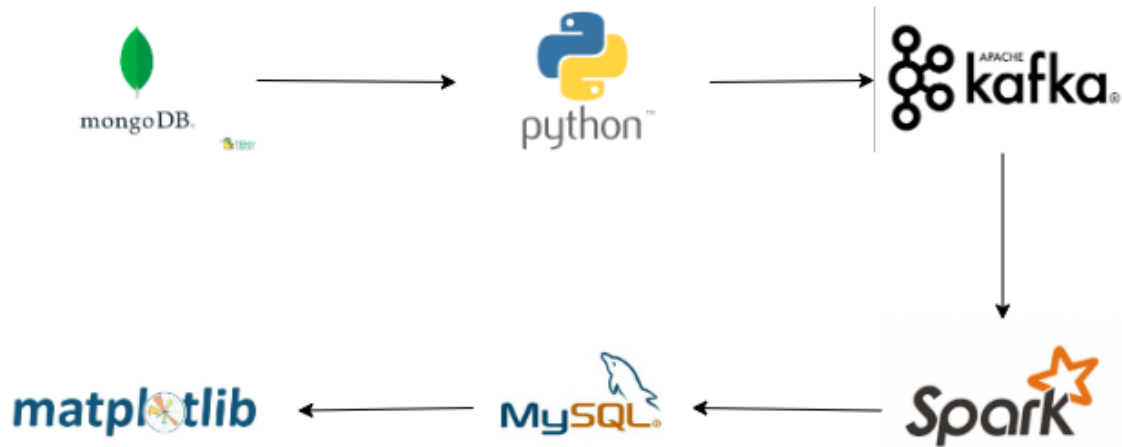
Nếu khối lượng dữ liệu lớn, ta có thể thu thập các thay đổi dữ liệu và tải chúng thành từng lô theo định kỳ. Trong khoảng thời gian định kỳ này, không có thay đổi nào được phép xảy ra trên hệ thống nguồn hoặc hệ thống đích để đảm bảo tính nhất quán trong quá trình đồng bộ hóa dữ liệu.

### **1.3.3. Áp dụng vào hệ thống**

- Extract - Giai đoạn trích xuất: Sử dụng python để thu nhập dữ liệu từ mongoDB
- Transform - Giai đoạn chuyển đổi: Sử dụng Python và Apache Spark để thực hiện các phép biến đổi phức tạp trên dữ liệu.

- Load - Giai đoạn tải: MySQL được triển khai như một Data Warehouse, kho lưu trữ cuối cùng, đảm bảo tính toàn vẹn và khả năng truy vấn hiệu quả cho dữ liệu đã xử lý.

#### 1.4. Mô tả hệ thống



*Hình 1.13. Hệ thống data pipeline*

Hệ thống data pipeline được thiết kế để xử lý khối dữ liệu lớn từ website thương mại điện tử Glamira. Hệ thống tích hợp các công nghệ hiện đại như MongoDB, Apache Kafka, Apache Spark, và MySQL, Matplotlib.

##### 1.4.1. Kiến trúc tổng quan

- **Nguồn dữ liệu:** Dữ liệu log (bao gồm log server, sự kiện người dùng, và dữ liệu giao dịch) được thu thập từ website Glamira và lưu trữ ban đầu trong MongoDB dưới dạng JSON.
- **Trích xuất dữ liệu ban đầu:** Python được sử dụng để trích xuất dữ liệu từ MongoDB.
- **Truyền tải dữ liệu:** Apache Kafka được sử dụng để truyền tải dữ liệu sau khi được Python xử lý tới Spark.
- **Xử lý dữ liệu:** Apache Spark chịu trách nhiệm xử lý dữ liệu đã được truyền từ Kafka.
- **Lưu trữ dữ liệu:** MySQL được sử dụng như là Data Warehouse, chứa dữ liệu đã được xử lý từ Spark.
- **Trực quan hóa dữ liệu:** Matplotlib là một thư viện trực quan hóa dữ liệu trong Python, được sử dụng để tạo các biểu đồ hỗ trợ cho việc phân tích dữ liệu.

- **Môi trường triển khai:** MongoDB, Kafka, MySQL được đóng gói trong Docker container.

#### 1.4.2. Quy trình ETL

- **Extract:** Python kết nối với MongoDB để trích xuất dữ liệu log.
- **Transform:** Spark thực hiện các bước làm sạch dữ liệu.
- **Load:** Dữ liệu đã xử lý được đẩy vào MySQL dưới dạng bảng quan hệ, tối ưu cho truy vấn và lưu trữ lâu dài.

#### 1.4.3. Đặc điểm nổi bật

- **Tính phân tán:** Sử dụng Kafka và Spark để xử lý dữ liệu lớn một cách hiệu quả, hỗ trợ mở rộng theo chiều ngang.
- **Thời gian thực:** Kafka đảm bảo luồng dữ liệu liên tục, cho phép phân tích gần thời gian thực.
- **Tính linh hoạt:** Docker và Linux cung cấp môi trường triển khai nhất quán, dễ dàng tích hợp với các hệ thống khác của Glamira.
- **Khả năng mở rộng:** Hệ thống có thể xử lý khối lượng dữ liệu tăng trưởng bằng cách thêm node Kafka hoặc cụm Spark.

#### 1.5. Kết luận

Qua chương 1, chúng ta đã có cái nhìn tổng quan về hệ thống data pipeline, với khả năng thu thập, xử lý, làm sạch và phân tích dữ liệu log từ hệ thống thương mại điện tử.

Những khía cạnh cơ bản được giới thiệu trong chương này sẽ là nền tảng để phát triển và triển khai chi tiết các thành phần của hệ thống trong các chương tiếp theo.

## CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

### 2.1. Phân tích dữ liệu cần xử lý

#### 2.1.1. Mô tả dữ liệu

##### a. Dữ liệu mẫu:

```
{
  _id: ObjectId('5ed8c9b634103036e28df298'),
  time_stamp: 1591265719,
  ip: '109.208.234.125',
  user_agent: 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14) AppleWebKit/605.1.15',
  resolution: '1680x1050',
  user_id_db: '',
  device_id: 'f059f067-ff73-44cd-9a7d-a20173f817b4',
  api_version: '1.0',
  store_id: '12',
  local_time: '2020-06-04 12:15:15',
  show_recommendation: null,
  current_url: 'https://www.glamira.fr/glamira-bagues-gratia.html?alloy=red_white',
  referrer_url: '',
  email_address: '',
  recommendation: false,
  utm_source: false,
  utm_medium: false,
  collection: 'view_product_detail',
  product_id: '90736',
  option: [
    {
      option_label: 'alloy',
      option_id: '326941',
      value_label: '',
      value_id: '3176136'
    },
    {
      option_label: 'diamond',
      option_id: '326940',
      value_label: '',
      value_id: '3176099'
    }
  ]
},
```

*Hình 2.1. Dữ liệu mẫu*

##### b. Tất cả các trường dữ liệu

```
{ field: '_id', types: ['objectId'] }
{ field: 'ip', types: ['string'] },
{ field: 'email_address', types: ['null', 'string'] },
{ field: 'currency', types: ['string', 'null'] },
{ field: 'order_id', types: ['int', 'double', 'string'] },
{ field: 'utm_source', types: ['string', 'null', 'bool'] },
{ field: 'collection', types: ['string'] },
```



```

{ field: 'collect_id', types: ['string'] },
{ field: 'cat_id', types: ['null', 'string'] },
{ field: 'recommendation_clicked_position', types: ['int', 'null'] },
{ field: 'viewing_product_id', types: ['string'] },
{ field: 'referrer_url', types: ['string'] },
{ field: 'utm_medium', types: ['bool', 'null', 'string'] },
{ field: 'key_search', types: ['null', 'string'] },
{ field: 'recommendation_product_position', types: ['int', 'string', 'null'] },
{ field: 'local_time', types: ['string', 'null'] },
{ field: 'recommendation', types: ['bool', 'null'] },
{ field: 'resolution', types: ['string', 'null'] },
{ field: 'show_recommendation', types: ['string', 'null'] },
{ field: 'user_agent', types: ['string'] },
{ field: 'recommendation_product_id', types: ['string', 'null'] },
{ field: 'device_id', types: ['string'] },
{ field: 'product_id', types: ['string'] },
{ field: 'store_id', types: ['string'] },
{ field: 'time_stamp', types: ['int'] },
{ field: 'user_id_db', types: ['string'] },
{ field: 'price', types: ['string', 'null'] },
{ field: 'is_paypal', types: ['null', 'bool'] },
{ field: 'api_version', types: ['string'] },
{ field: 'current_url', types: ['string'] },
{ field: 'option', types: ['array', 'object'] },
{ field: 'cart_products', types: ['array'] },

```

### c. Mô tả chi tiết dữ liệu

- `_id` (objectId): Khóa chính, định danh duy nhất cho mỗi bản ghi.
- `ip` (string): Địa chỉ IP của người dùng, dùng để theo dõi nguồn truy cập.
- `email_address` (null, string): Email người dùng, dùng cho nhận diện hoặc tiếp thị.
- `currency` (string, null): Loại tiền tệ (USD, EUR), dùng cho giá và thanh toán.
- `order_id` (int, double, string): Mã đơn hàng, theo dõi giao dịch.
- `utm_source` (string, null, bool): Nguồn tiếp thị (google, facebook), theo dõi lưu lượng truy cập.
- `collection` (string): Loại sự kiện (view\_product\_detail, checkout\_success), phân loại hành vi người dùng.

- `collect_id` (string): Mã danh mục hoặc nhóm sự kiện, dùng để phân loại.
- `cat_id` (null, string): Mã danh mục sản phẩm (ring, necklace).
- `recommendation_clicked_position` (int, null): Vị trí sản phẩm đề xuất được nhấp vào.
- `viewing_product_id` (string): ID sản phẩm đang xem.
- `referrer_url` (string): URL nguồn dẫn người dùng đến website.
- `utm_medium` (bool, null, string): Phương tiện tiếp thị (cpc, email).
- `key_search` (null, string): Từ khóa tìm kiếm người dùng nhập.
- `recommendation_product_position` (int, string, null): Vị trí sản phẩm trong danh sách đề xuất.
- `local_time` (string, null): Thời gian địa phương của người dùng.
- `recommendation` (bool, null): Cờ cho biết hành động liên quan đến đề xuất.
- `resolution` (string, null): Độ phân giải màn hình thiết bị.
- `show_recommendation` (string, null): Trạng thái hiển thị hệ thống đề xuất.
- `user_agent` (string): Thông tin trình duyệt/thiết bị của người dùng.
- `recommendation_product_id` (string, null): ID sản phẩm được đề xuất.
- `device_id` (string): Mã định danh thiết bị người dùng.
- `product_id` (string): ID sản phẩm liên quan đến hành động (giỏ hàng, thanh toán).
- `store_id` (string): Mã cửa hàng hoặc chi nhánh.
- `time_stamp` (int): Thời gian sự kiện (Unix timestamp).
- `user_id_db` (string): ID người dùng trong cơ sở dữ liệu.
- `price` (string, null): Giá sản phẩm hoặc đơn hàng.
- `is_paypal` (null, bool): Cờ cho biết giao dịch dùng PayPal.
- `api_version` (string): Phiên bản API ghi dữ liệu.
- `current_url` (string): URL trang hiện tại người dùng đang truy cập.
- `option` (array, object): Tùy chọn sản phẩm (kích thước, màu sắc).
- `cart_products` (array): Danh sách sản phẩm trong giỏ hàng.

### 2.1.2. Những dữ liệu cần lấy ra để phân tích

#### Dữ liệu định danh

- `_id.$oid` – ID duy nhất của bản ghi (giữ để theo dõi dữ liệu).

## Dữ liệu về hành vi người dùng trên trang web

- **current\_url** – Trang web người dùng đang xem.
- **referrer\_url** – Nguồn gốc trước khi truy cập trang hiện tại.
- **collection** – Loại hành động người dùng thực hiện

## Dữ liệu về thời gian

- **time\_stamp** – Dấu thời gian Unix (giúp phân tích theo ngày, giờ).

## Dữ liệu về giỏ hàng

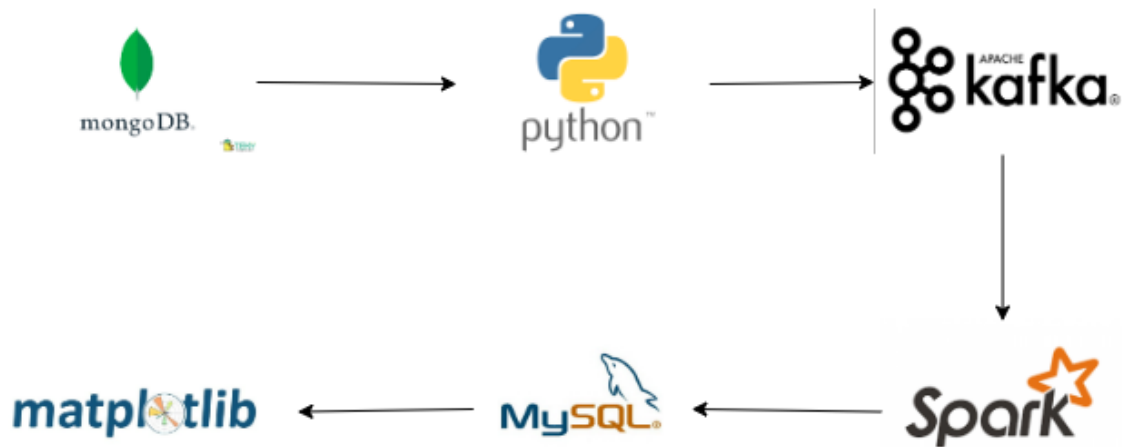
- **cart\_products** - chứa danh sách các sản phẩm mà người dùng đã thêm vào giỏ hàng trong một sự kiện cụ thể

Mỗi phần tử trong mảng **cart\_products** là một đối tượng (StructType) với các trường con:

- **product\_id** (IntegerType): ID duy nhất của sản phẩm trong giỏ hàng.
- **amount** (IntegerType): Số lượng sản phẩm được thêm vào giỏ hàng.
- **price** (StringType): Giá của sản phẩm, thường ở định dạng chuỗi (ví dụ: "100.00").
- **currency** (StringType): Loại tiền tệ của giá (ví dụ: "USD", "EUR").
- **option** (ArrayType): Mảng các tùy chọn tùy chỉnh của sản phẩm, mỗi tùy chọn chứa:
  - **option\_label** (StringType): Tên tùy chọn (ví dụ: "Color", "Size").
  - **option\_id** (IntegerType): ID của tùy chọn.
  - **value\_label** (StringType): Giá trị của tùy chọn (ví dụ: "Red", "Small").
  - **value\_id** (IntegerType): ID của giá trị tùy chọn.

## 2.2. Thiết kế hệ thống

Mục tiêu chính là thiết kế và triển khai hệ thống data pipeline xử lý dữ liệu lớn.



*Hình 2.2. Hệ thống data pipeline*

Mục tiêu chính là thiết kế và triển khai hệ thống data pipeline xử lý dữ liệu lớn.

### 2.2.1. Quản lý và kiểm tra dữ liệu trong MongoDB

#### a. Nhập dữ liệu từ tệp vào MongoDB

##### Mô tả quy trình

Để nhập dữ liệu từ file JSON vào MongoDB, sử dụng công cụ mongorestore. Công cụ này hỗ trợ nhập dữ liệu từ các định dạng phổ biến vào collection trong MongoDB một cách nhanh chóng và hiệu quả.

Thực hiện nhập dữ liệu: chạy câu lệnh trong MongoDB Shell

```
(base) nguyenphuc@phuc2108:~$ mongorestore --db Glamira --collection summary /home/nguyenphuc/Documents/GlamiraUserFlowInsightsProject/glamira_uhl_oct2019_nov2019/dump/countly/summary.bson
```

*Hình 2.3. Đưa file chứa dữ liệu lên MongoDB*

- Glamira là tên cơ sở dữ liệu được tạo trong MongoDB
- summary là tên collection được tạo trong MongoDB
- Dữ liệu được nhập từ tệp dữ liệu trong máy tính.

Kết quả về sau khi dữ liệu được đưa vào thành công

```
2025-03-18T20:42:41.288+0700 finished restoring Glamira.summary (41432473 documents, 0 failures)
2025-03-18T20:42:41.289+0700 restoring indexes for collection Glamira.summary from metadata
2025-03-18T20:42:41.289+0700 index: &idx.IndexDocument{Options:primitive.M{"name":"time_stamp_1", "ns":"c
2025-03-18T20:42:41.289+0700 index: &idx.IndexDocument{Options:primitive.M{"name":"device_id_1", "ns":"cou
2025-03-18T20:44:54.320+0700 41432473 document(s) restored successfully. 0 document(s) failed to restore.
```

*Hình 2.4. Kết quả trả về sau khi đưa dữ liệu*

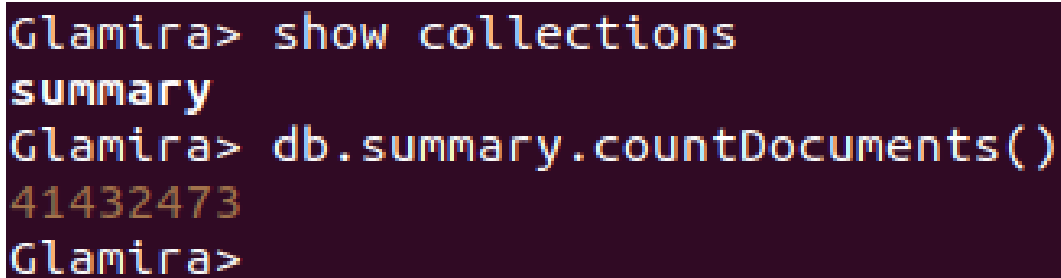
## **b. Kiểm tra dữ liệu trong MongoDB**

### **Mô tả quy trình**

Để đảm bảo dữ liệu đã được nhập chính xác, cần kiểm tra cả số lượng bản ghi và nội dung dữ liệu trong collection.

Kiểm tra số lượng bản ghi:

- Dùng câu lệnh truy vấn `db.collection.count()` trong MongoDB Shell để xác định tổng số bản ghi trong collection.
- Chạy câu lệnh trong MongoDB Shell



```
Glamira> show collections
summary
Glamira> db.summary.countDocuments()
41432473
Glamira>
```

*Hình 2.5. Kiểm tra số lượng bản ghi*

Số lượng bản ghi trong MongoDB là hơn 41 triệu bản. Việc kiểm tra số lượng bản ghi đảm bảo tính toàn vẹn dữ liệu, giúp phát hiện lỗi như mất mát dữ liệu, trước khi tiến hành các bước xử lý phức tạp hơn.

Kiểm tra dữ liệu cụ thể:

Thực hiện các truy vấn có cấu trúc để kiểm tra nội dung dữ liệu.

Chạy câu lệnh trong MongoDB Shell: `db.summary().find().limit(5).pretty()`

```
Glamira> db.summary.find().limit(5).pretty()
[
  {
    _id: ObjectId('5ed8cb2bc671fc36b74653ad'),
    time_stamp: 1591266092,
    ip: '37.170.17.183',
    user_agent: 'Mozilla/5.0 (iPhone; CPU iPhone OS 13_4_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/13.0 Mobile/15E148 Safari/604.1',
    resolution: '375x667',
    user_id_db: '502567',
    device_id: 'beb2cacb-20af-4f05-9c03-c98e54a1b71a',
    api_version: '1.0',
    store_id: '12',
    local_time: '2020-06-04 12:21:27',
    show_recommendation: 'false',
    current_url: 'https://www.glamira.fr/glamira-pendant-viktor.html?alloy=yellow-375',
    referrer_url: 'https://www.glamira.fr/men-s-necklaces/',
    email_address: 'pereira.vivien@yahoo.fr',
    recommendation: false,
    utm_source: false,
    utm_medium: false,
    collection: 'view_product_detail',
    product_id: '110474',
    option: [
      {
        option_label: 'alloy',
        option_id: '332084',
        value_label: '',
        value_id: '3279318'
      },
      {
        option_label: 'diamond',
        option_id: '',
        value_label: '',
        value_id: ''
      }
    ]
  }
],
```

*Hình 2.6. Kiểm tra dữ liệu đã đưa vào*

Kết quả trả về: dữ liệu được hiển thị dưới dạng bản ghi JSON.

## 2.2.2. Dùng python để lấy những dữ liệu cần thiết

### 2.2.2.1. Lý do sử dụng Python

**Câu hỏi:** Tại sao cần sử dụng Python để lấy dữ liệu từ MongoDB trước khi đẩy lên Kafka thay vì đẩy trực tiếp từ MongoDB lên Kafka?

**Lý do:** Thay vì đẩy hết 41 triệu bản ghi lên Kafka, có thể khiến cho hệ thống bị tắc nghẽn với lượng dữ liệu quá lớn. Thay vào đó, sử dụng Python như trạm trung chuyển, lấy những dữ liệu cần thiết từ MongoDB rồi mới đẩy lên Kafka. Việc này giúp tiết kiệm tài nguyên cho hệ thống.

### 2.2.2.2. Kết nối Python với MongoDB

#### 2.2.2.2.1. Mục tiêu

Thiết lập kết nối từ Python đến MongoDB để truy vấn dữ liệu từ collection summary, thực hiện tiền xử lý, và chuẩn bị dữ liệu trước khi đẩy lên Kafka. Việc kết nối sử dụng thư viện pymongo, đảm bảo khả năng truy vấn dữ liệu hiệu quả, xử lý lỗi, và bảo mật thông tin kết nối.

#### 2.2.2.2.2. Quy trình kết nối

1. Cài đặt thư viện pymongo: Cài đặt pymongo để tương tác với MongoDB.
2. Cấu hình URI kết nối: Sử dụng URI chứa thông tin host, port, database, và thông tin xác thực
3. Khởi tạo client MongoDB: Tạo đối tượng MongoClient để kết nối đến MongoDB.
4. Kiểm tra kết nối: Thử truy cập database và collection để xác nhận kết nối thành công.
5. Quản lý tài nguyên: Đóng kết nối sau khi hoàn tất để tránh rò rỉ tài nguyên.

#### 2.2.2.3. Xử lý dữ liệu

- Lọc dữ liệu: Chỉ lấy các trường cần thiết (\_id, time\_stamp, current\_url, referrer\_url, collection, cart\_products).
- Chuyển đổi định dạng: Chuyển ObjectId thành chuỗi để đảm bảo tương thích với JSON khi gửi qua Kafka.
- Giới hạn số lượng: Giới hạn 500000 bản ghi mỗi lần xử lý để tránh quá tải.

#### 2.2.3. Đẩy dữ liệu từ python lên kafka

##### 2.2.3.1. Mô tả

Dữ liệu sau khi xử lý được đẩy lên Kafka topic **product\_views** sử dụng thư viện kafka-python. Kafka được chọn để truyền dữ liệu thời gian thực do khả năng xử lý luồng dữ liệu lớn và độ trễ thấp.

##### 2.2.3.2. Lý do sử dụng Kafka

- Xử lý thời gian thực: Kafka hỗ trợ truyền dữ liệu với độ trễ thấp, phù hợp với các ứng dụng phân tích hành vi người dùng theo thời gian thực.
- Khả năng mở rộng: Kafka cho phép thêm partition và consumer để xử lý khối lượng dữ liệu lớn.
- Độ bền dữ liệu: Kafka lưu trữ dữ liệu trong một khoảng thời gian dài, cho phép xử lý lại nếu cần.
- Tích hợp với Spark: Kafka tích hợp tốt với Spark Streaming, cho phép đọc dữ liệu dưới dạng stream để phân tích.

##### 2.2.3.3. Quy trình thực hiện

1. Khởi tạo KafkaMessageProducer với cấu hình kết nối tới Kafka broker (localhost:9092).
2. Gửi từng bản ghi đã được làm sạch dưới dạng JSON đến topic product\_views.
3. Sử dụng flush() để đảm bảo tất cả tin nhắn được gửi trước khi kết thúc.

#### 2.2.3.4. Kết quả trả về

Sau khi chạy file main.py

```
/home/nguyenphuc/anaconda3/bin/python /home/nguyenphuc/Documents/GlamiraUser
Connected to MongoDB: Glamira
summary collection already exists, skipping creation
Collections: ['summary']
summary collection contains documents
Validated schema for summary collection
Processed 500000 records
Processed 1000000 records
Processed 1500000 records
Processed 2000000 records
Processed 2500000 records
Processed 3000000 records
Processed 3500000 records
```

*Hình 2.7. Kết quả trả về sau khi chạy file main.py*

Dữ liệu sau khi được python phân tích và đẩy lên Kafka:



```
{
  "_id": "5ed70192bb30d63718bdda75",
  "time_stamp": 1591148947,
  "current_url": "https://www.glamira.co.uk/checkout/onepage/success/",
  "referrer_url": "https://www.glamira.co.uk/customcheckout/onepage/review/",
  "collection": "checkout_success",
  "cart_products": [
    {
      "product_id": 92705,
      "amount": 1,
      "price": "591.00",
      "currency": "£",
      "option": [
        {
          "option_label": "alloy",
          "option_id": 129312,
          "value_label": "Weißgold 585",
          "value_id": 963173
        }
      ]
    }
  ]
}
```

*Hình 2.8. Dữ liệu sau khi được python xử lý*

## 2.2.4. Dùng spark để lấy dữ liệu từ kafka và phân tích

### 2.2.4.1. Mô tả

Apache Spark Streaming được sử dụng để đọc dữ liệu từ Kafka topic `product_views`, thực hiện phân tích doanh số theo sản phẩm và loại tiền tệ, sau đó chuẩn bị dữ liệu để lưu vào MySQL. Spark được chọn do khả năng xử lý dữ liệu lớn và hỗ trợ streaming.

### 2.2.4.2. Lý do sử dụng Spark

- Xử lý dữ liệu lớn: Spark hỗ trợ xử lý dữ liệu lớn với hiệu suất cao nhờ cơ chế phân tán và in-memory computing.
- Hỗ trợ streaming: Spark Streaming tích hợp tốt với Kafka, cho phép đọc và xử lý dữ liệu theo thời gian thực.
- Tính linh hoạt: Spark SQL cung cấp các phép toán như `explode`, `pivot`, và `regexp_replace` để xử lý dữ liệu phức tạp (như mảng `cart_products`).
- Tích hợp với MySQL: Spark JDBC cho phép ghi dữ liệu trực tiếp vào MySQL với hiệu suất cao.

### 2.2.4.3. Quy trình thực hiện

1. Tạo `SparkSession` với cấu hình tối ưu từ `spark_config.py`.
2. Đọc dữ liệu từ Kafka topic `product_views` dưới dạng stream và parse JSON theo schema xác định.

3. Lọc các sự kiện checkout\_success để tập trung vào giao dịch hoàn tất.
4. Xử lý mảng cart\_products bằng explode để lấy thông tin sản phẩm.
5. Làm sạch giá bằng regexp\_replace và tính tổng doanh số
6. Pivot dữ liệu theo product\_id và các loại tiền tệ Áp dụng watermark (20 năm) để xử lý dữ liệu trễ.

### 2.2.5. Đẩy dữ liệu đã phân tích từ Spark lên mysql

#### 2.2.5.1. Mô tả

Kết quả phân tích doanh số theo sản phẩm và tiền tệ được lưu vào bảng currency\_totals trong MySQL sử dụng Spark JDBC. MySQL được chọn vì khả năng hỗ trợ truy vấn SQL nhanh và tích hợp tốt với các công cụ BI.

#### 2.2.5.2. Lý do sử dụng MySQL

- Hiệu suất truy vấn: MySQL cung cấp hiệu suất cao cho các truy vấn báo cáo, đặc biệt với bảng có chỉ mục chính như product\_id
- Tích hợp với BI: MySQL dễ dàng tích hợp với các công cụ như Tableau, Power BI, Lookerstudio để trực quan hóa dữ liệu.
- Tính ổn định: MySQL là cơ sở dữ liệu quan hệ đáng tin cậy, phù hợp để lưu trữ dữ liệu đã phân tích.
- Hỗ trợ JDBC: Spark JDBC cho phép ghi dữ liệu trực tiếp từ Spark vào MySQL với hiệu suất cao.

#### 2.2.5.3. Quy trình thực hiện

1. Tạo bảng currency\_totals trong MySQL với schema phù hợp.
2. Đổi tên cột trong DataFrame để khớp với schema bảng.
3. Ghi dữ liệu theo từng batch vào MySQL với chế độ append.

#### 2.2.5.4. Kết quả trả về

Sau khi chạy file spark\_kafka\_consumer.py

```
/home/nguyenphuc/anaconda3/bin/python /home/nguyenphuc/Documents/GlamiraUserFlowInsightsProject/GlamiraUserFlowInsights/MyGlamira/src/spark_kafka_consumer.py
25/05/24 22:51:22 WARN Utils: Your hostname, phuc2108 resolves to a loopback address: 127.0.1.1; using 192.168.74.89 instead (on interface wlp0s20f3)
25/05/24 22:51:22 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
25/05/24 22:51:22 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
INFO: __main__: Successfully wrote all records to MySQL table 'currency_totals'
INFO: py4j.clientserver: Closing down clientserver connection

Process finished with exit code 0
```

*Hình 2.9. Kết quả trả về sau khi chạy file spark\_kafka\_consumer.py*

- Lỗi: Nếu có lỗi (ví dụ: MySQL không kết nối được), log lỗi như Failed to write to MySQL: Connection refused.
- Dữ liệu trong MySQL: Bảng currency\_totals chứa các bản ghi với product\_id và tổng doanh số theo từng loại tiền tệ, sẵn sàng cho truy vấn báo cáo.

## 2.2.6. Trực quan hóa dữ liệu

### 2.2.6.1. Mô tả

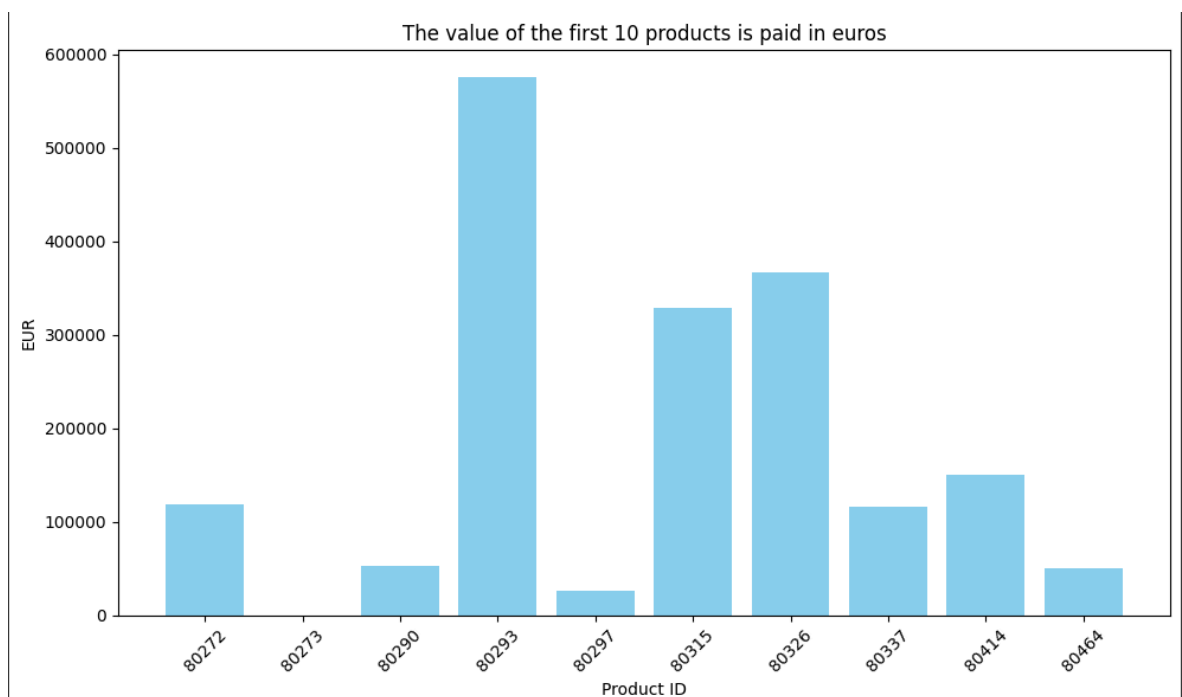
Trực quan hóa dữ liệu chuyển đổi kết quả phân tích từ bảng `currency_totals` trong MySQL thành các biểu đồ trực quan, hỗ trợ Glamira đưa ra quyết định kinh doanh.

### 2.2.6.2. Các công cụ trực quan hóa

- Python (Matplotlib): Tạo biểu đồ nhanh trong môi trường local, tích hợp tốt với Spark và Pandas.
- Chart.js: Hiển thị biểu đồ tương tác trên web, phù hợp với dashboard BI.

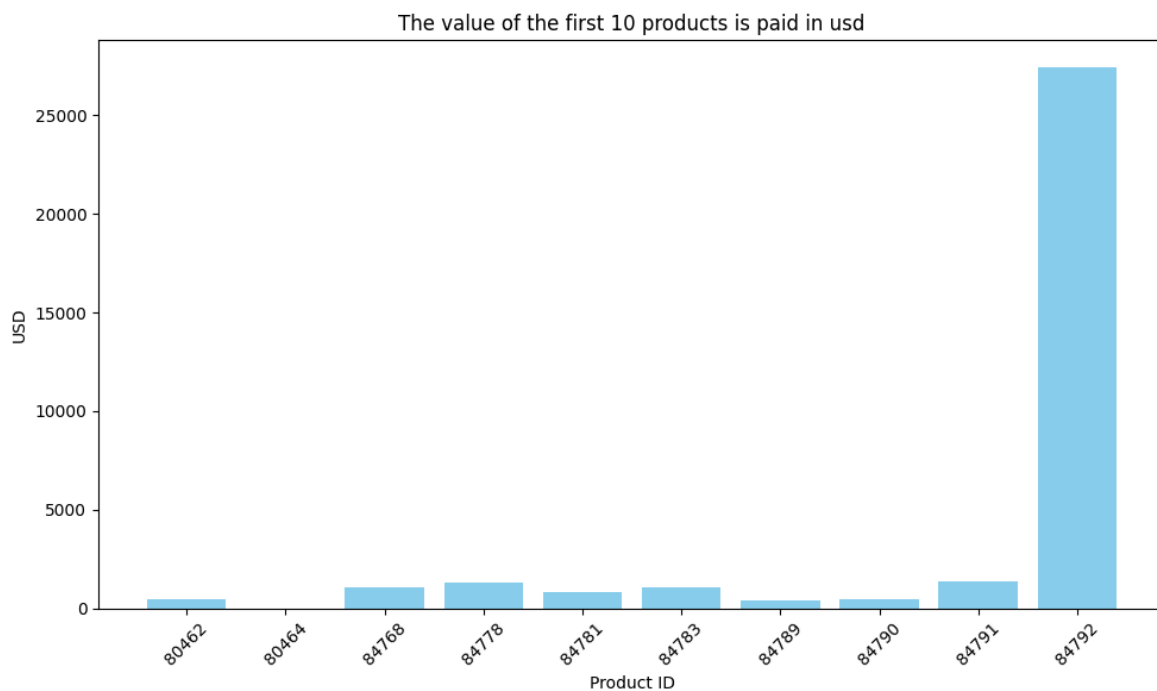
### 2.2.6.3. Các biểu đồ sau khi trực quan hóa dữ liệu

Biểu đồ cột hiển thị 10 sản phẩm đầu tiên được trả theo Euro



*Hình 2.10. Giá trị 10 sản phẩm đầu tiên được trả theo Euro*

Biểu đồ cột hiển thị 10 sản phẩm đầu tiên được trả theo USD



*Hình 2.11. Giá trị 10 sản phẩm đầu tiên được trả theo USD*

### 2.3. Kết luận

Chương này đã trình bày chi tiết quá trình phân tích và thiết kế hệ thống data pipeline để xử lý dữ liệu log từ website thương mại điện tử Glamira. Các bước từ phân tích dữ liệu (mô tả cấu trúc, trường dữ liệu, và lựa chọn dữ liệu cần thiết) đến thiết kế hệ thống (tích hợp MongoDB, Kafka, Spark, và MySQL) đã được thực hiện nhằm đảm bảo tính toàn vẹn, hiệu suất cao, và khả năng mở rộng của pipeline.

## CHƯƠNG 3: CÀI ĐẶT CHƯƠNG TRÌNH

### 3.1. Cài đặt Docker

#### 3.1.1. Cài đặt Docker trên Linux

Để cài đặt Docker Community Edition trên hệ điều hành Ubuntu, ta thực hiện các bước sau:

##### 3.1.1.1 Thêm khóa GPG chính thức của Docker

Khóa GPG được sử dụng để xác minh tính toàn vẹn và nguồn gốc của các gói cài đặt từ kho lưu trữ Docker. Điều này đảm bảo rằng phần mềm được tải về không bị giả mạo.

- Sử dụng câu lệnh:  
**curl -fsSL <https://download.docker.com/linux/ubuntu/gpg> | sudo apt-key add**

##### 3.1.1.2. Thêm kho lưu trữ Docker

Để cài đặt, cần thêm kho lưu trữ chính thức của Docker vào danh sách phần mềm của hệ thống Ubuntu.

- Sử dụng câu lệnh:  
**sudo add-apt-repository "deb [arch=amd64] <https://download.docker.com/linux/ubuntu> \$(lsb\_release -cs) stable"**

##### 3.1.1.3. Cập nhật danh sách gói phần mềm

Sau khi thêm kho lưu trữ, ta cần phải cập nhật lại danh sách gói để hệ thống nhận diện được gói đã cài đặt mới từ Docker

- Sử dụng câu lệnh:  
**sudo apt-get update**

##### 3.1.1.4. Kiểm tra chính sách gói

Trước khi cài đặt Docker, cần kiểm tra các phiên bản Docker khả dụng có sẵn để đảm bảo hệ thống cài đặt được phiên bản phù hợp

- Sử dụng câu lệnh:  
**apt-cache policy docker-ce**

##### 3.1.1.5. Cài đặt Docker Community Edition

Sau khi trải qua các bước trên, tiến hành cài đặt Docker Community Edition, phiên bản miễn phí và mã nguồn mở của Docker.

- Sử dụng câu lệnh: **sudo apt-get install -y docker-ce**

### 3.1.2. Cài đặt Docker Compose

Docker Compose được cài đặt để hỗ trợ quản lý các ứng dụng đa container. Cho phép định nghĩa và chạy nhiều container.

Các bước cài đặt bao gồm:

#### 3.1.2.1. Tải xuống tệp thực thi Docker Compose

Tải phiên bản 1.28.0 của Docker Compose từ kho Github chính thức.

- Sử dụng câu lệnh:  
**sudo curl -L**  
**"https://github.com/docker/compose/releases/download/1.28.0/docker-compose-\$(uname -s)-\$(uname -m)" -o**  
**/usr/local/bin/docker-compose**

#### 3.1.2.2. Cấp quyền thực thi

Để Docker Compose có thể chạy, cần cấp quyền thực thi cho tệp vừa tải về.

- Sử dụng câu lệnh:  
**sudo chmod +x /usr/local/bin/docker-compose**

### 3.1.3. Cấu hình Quyền Truy cập Docker

Theo như mặc định, chỉ những người dùng root hoặc những người có quyền sudo mới có thể chạy các lệnh trên Docker. Để cấp quyền truy cập cho người dùng thông thường mà không phải vào root hay dùng câu lệnh sudo, cần thêm người dùng vào nhóm Docker.

#### 3.1.3.1. Thêm người dùng hiện tại vào nhóm docker

- Sử dụng câu lệnh:  
**sudo usermod -aG docker \${USER}**  
Sử dụng câu lệnh:

#### 3.1.3.2. Đăng nhập lại để áp dụng thay đổi

Để áp dụng sau khi thêm, cần đăng nhập lại.

- Sử dụng câu lệnh:  
**su - \${USER}**

#### 3.1.3.3. Xác minh quyền nhóm

Kiểm tra xem người dùng đã được thêm vào docker hay chưa.

- Sử dụng câu lệnh:

**id -nG**

Lệnh này sẽ liệt kê tất cả các nhóm mà người dùng hiện tại thuộc về.

### 3.1.3.4. Cấp quyền cho người dùng cụ thể

- Sử dụng câu lệnh:

**sudo usermod -aG docker ubuntu**

### 3.1.4. Kiểm tra sau khi cài đặt

Để đảm bảo Docker và Docker Compose đã được cài đặt và hoạt động, thực hiện các bước kiểm tra sau:

#### 3.1.4.1. Kiểm tra phiên bản

Kiểm tra phiên bản của Docker và Docker Compose:

```
(base) nguyenphuc@phuc2108:~$ docker --version
Docker version 28.0.4, build b8034c0
(base) nguyenphuc@phuc2108:~$ docker-compose --version
docker-compose version 1.28.0, build d02a7b1a
```

*Hình 3.1. Phiên bản của docker và docker compose*

Lệnh trả về thông tin phiên bản của Docker là version 28.0.4, Docker Compose là version 1.28.0.

#### 3.1.4.2. Kiểm tra trạng thái dịch vụ Docker

```
(base) nguyenphuc@phuc2108:~$ systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2025-05-22 08:43:56 +07; 14h ago
     TriggeredBy: ● docker.socket
    Docs: https://docs.docker.com
   Main PID: 1848 (dockerd)
      Tasks: 36
     Memory: 48.0M
        CPU: 9.261s
   CGroup: /system.slice/docker.service
           └─ 1848 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
             └─ 51224 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 27017 -container-ip 172.17.0.2 -container-port 27017 -use-listen-fd
                └─ 51230 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 27017 -container-ip 172.17.0.2 -container-port 27017 -use-listen-fd
```

*Hình 3.2. Trạng thái dịch vụ của Docker*

Docker đã hiện trạng thái active. Xác nhận rằng Docker daemon đang hoạt động và sẵn sàng.

## 3.2. Cài đặt các container trên Docker

Phần này trình bày quy trình cài đặt, cấu hình và kiểm tra hoạt động của các Container trên Docker. Quy trình được thiết kế để đảm bảo triển khai các môi trường ổn định, an toàn và hiệu quả, phù hợp với các yêu cầu cho hệ thống xử lý dữ liệu lớn.

### 3.2.1. Cài đặt MongoDB

Để triển khai container MongoDB trên Docker, thực hiện các bước sau:

#### 3.2.1.1. Cài đặt Container MongoDB

Sử dụng lệnh sau để khởi chạy container MongoDB với các tham số cấu hình cụ thể:

- **docker run -d \**  
    **--name mongodb \**  
    **-p 27017:27017 \**  
    **-v mongo\_data:/data/db \**  
    **--memory="5g" \**  
    **-e MONGO\_INITDB\_ROOT\_USERNAME=nguyenphuc \**  
    **-e MONGO\_INITDB\_ROOT\_PASSWORD=123456 \**  
    **mongo:8.0.6 \**  
    **mongod --wiredTigerCacheSizeGB=4**

Giải thích các tham số:

- **-d**: Chạy container ở chế độ nền (detached mode).
- **--name mongodb**: Đặt tên container là mongodb.
- **-p 27017:27017**: Ánh xạ cổng 27017 của container với cổng 27017 trên máy chủ.
- **-v mongo\_data:/data/db**: Gắn kết volume mongo\_data để lưu trữ dữ liệu bền vững.
- **--memory="5g"**: Giới hạn bộ nhớ của container ở mức 5GB.
- **-e MONGO\_INITDB\_ROOT\_USERNAME=nguyenphuc**: Thiết lập tên người dùng quản trị.
- **-e MONGO\_INITDB\_ROOT\_PASSWORD=123456**: Thiết lập mật khẩu quản trị.
- **mongo:8.0.6**: Sử dụng hình ảnh MongoDB phiên bản 8.0.6.



- `mongod --wiredTigerCacheSizeGB=4`: Cấu hình kích thước bộ đệm WiredTiger là 4GB để tối ưu hóa hiệu suất.

### 3.2.1.2. Kiểm tra Container MongoDB

Để xác minh container MongoDB đã được triển khai và hoạt động đúng cách, thực hiện khởi động Container và kiểm tra trạng thái:

```
(base) nguyenphuc@phuc2108:~$ docker start mongodb
mongodb
(base) nguyenphuc@phuc2108:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED
df63e1641396   mongo:8.0.6   "docker-entrypoint.s..." 2 weeks ago
```

*Hình 3.3. Khởi động Container MongoDB và kiểm tra*

Container MongoDB đã xuất hiện trong danh sách

### 3.2.1.3. Thực thi các câu lệnh kiểm tra

Để tương tác với cơ sở dữ liệu MongoDB, thực hiện các bước sau:

- Truy cập vào MongoDB shell:  
**`docker exec -it mongodb mongosh -u nguyenphuc -p 123456 --authenticationDatabase admin`**

```
(base) nguyenphuc@phuc2108:~$ docker exec -it mongodb mongosh -u nguyenphuc -p 123456 --authenticationDatabase admin
Current Mongosh Log ID: 682fe86c7bd14782cd861df
Connecting to:
  mongodb://<credentials>@127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&authSource=admin&appName=mongosh+2.5.0
Using MongoDB:
  8.0.6
Using Mongosh:
  2.5.0

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

-----
The server generated these startup warnings when booting
2025-05-23T03:43:36.485+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/pr
2025-05-23T03:43:36.776+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
2025-05-23T03:43:36.776+00:00: For customers running the current memory allocator, we suggest changing the contents of the following sysfsFile
2025-05-23T03:43:36.776+00:00: We suggest setting the contents of sysfsFile to 0.
2025-05-23T03:43:36.776+00:00: vm.max_map_count is too low
2025-05-23T03:43:36.776+00:00: We suggest setting swappiness to 0 or 1, as swapping can cause performance problems.
-----
test> █
```

*Hình 3.4. Truy cập vào MongoDB shell*

- Kiểm tra danh sách cơ sở dữ liệu

```
test> show dbs;
Glamira          11.08 GiB
admin            100.00 KiB
config           72.00 KiB
local            72.00 KiB
photo_sharing    120.00 KiB
```

*Hình 3.5. Danh sách cơ sở dữ liệu*

### 3.2.2. Cài đặt MySQL

Để triển khai container MySQL trên Docker, thực hiện các bước sau:

#### 3.2.2.1. Cài đặt Container MySQL

Tải image MySQL chính thức

Sử dụng image `mysql/mysql-server:latest` từ Docker Hub để đảm bảo phiên bản mới nhất, được tối ưu hóa và bảo mật.

- Sử dụng câu lệnh:  
**`docker pull mysql/mysql-server:latest`**

Chạy container MySQL

Khởi chạy container với các tham số phù hợp để ánh xạ cổng, đặt tên container, và thiết lập chính sách khởi động lại.

- Sử dụng câu lệnh:  
**`docker run -p 3307:3306 --name=mysql1 -d --restart unless-stopped mysql/mysql-server:latest`**
- Giải thích:
  - `-p 3307:3306`: Ánh xạ cổng 3306 của container (cổng mặc định của MySQL) với cổng 3307 trên máy chủ để tránh xung đột với các dịch vụ khác.
  - `mysql1`: Đặt tên container là `mysql1`.
  - `--restart unless-stopped`: Tự động khởi động lại container trừ khi bị dừng thủ công.
  - `mysql/mysql-server:latest`: Sử dụng image MySQL mới nhất.

#### 3.2.2.2. Kiểm tra Container MySQL

Để xác minh container MongoDB đã được triển khai và hoạt động đúng cách, thực hiện khởi động Container và kiểm tra trạng thái:

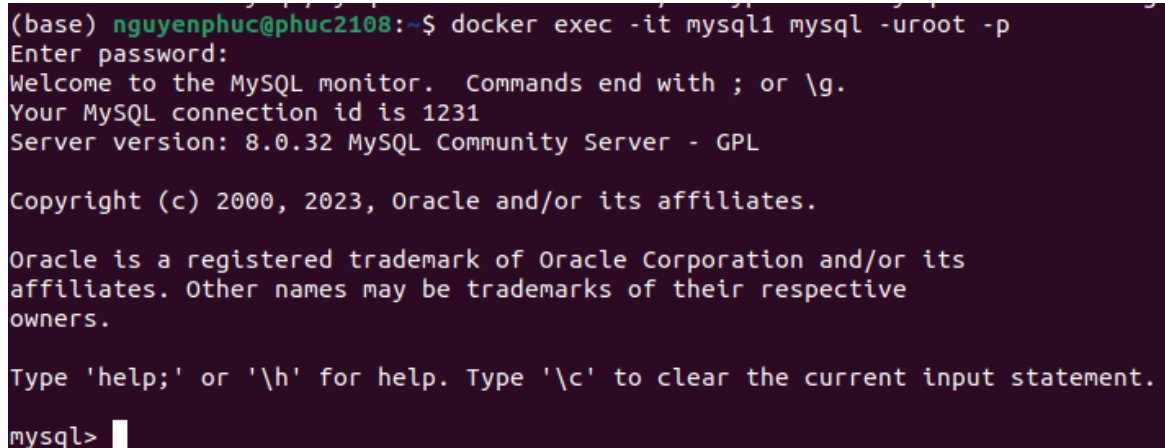
```
(base) nguyenphuc@phuc2108:~$ docker start mysql1
mysql1
(base) nguyenphuc@phuc2108:~$ docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED        STATUS
951376cc1963   mysql/mysql-server:latest "/entrypoint.sh mysql..." 7 weeks ago    Up 12 hours (healthy)
(base) nguyenphuc@phuc2108:~$
```

*Hình 3.6. Khởi động Container MySQL và kiểm tra*

#### 3.2.2.3. Thực thi các câu lệnh kiểm tra

Để tương tác với cơ sở dữ liệu MySQL, thực hiện các bước sau:

Truy cập vào Mysql shell:



```
(base) nguyenphuc@phuc2108:~$ docker exec -it mysql1 mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1231
Server version: 8.0.32 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> █
```

*Hình 3.7. Truy cập vào MySQL shell*

Sau đó ta thực hiện các câu lệnh để kiểm tra cơ sở dữ liệu trong MySQL

### **3.2.3. Cài đặt Kafka**

Cài đặt Kafka trên Docker yêu cầu chạy hai container: một cho ZooKeeper và một cho Kafka broker

#### **3.2.2.1. Cài đặt Container ZooKeeper và Kafka Broker**

Viết file docker-compose:

Tạo file docker-compose.yml để định nghĩa và quản lý các container ZooKeeper và Kafka.

```

1 version: '2'
2 services:
3   zookeeper:
4     image: wurstmeister/zookeeper
5     ports:
6       - "2181:2181"
7     restart: unless-stopped
8     environment:
9       ZOOKEEPER_CLIENT_PORT: 2181
10      ZOOKEEPER_TICK_TIME: 2000
11     mem_limit: 1g
12     cpus: 1
13   kafka:
14     image: wurstmeister/kafka
15     ports:
16       - "9092:9092"
17     environment:
18       KAFKA_ADVERTISED_HOST_NAME: localhost
19       KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
20       KAFKA_BROKER_ID: 1
21       KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
22       KAFKA_DEFAULT_REPLICATION_FACTOR: 1
23       KAFKA_NUM_PARTITIONS: 3
24       KAFKA_HEAP_OPTS: "-Xmx2g -Xms2g"
25     volumes:
26       - /var/run/docker.sock:/var/run/docker.sock
27     restart: unless-stopped
28     mem_limit: 4g
29     cpus: 2
30     depends_on:
31       - zookeeper

```

*Hình 3.8. File docker-compose*

Chạy file docker-compose:

- Sử dụng câu lệnh:  
**docker-compose up -d**

Lệnh này khởi chạy cả hai Container Zookeeper và Kafka.

### 3.2.2.2. Kiểm tra Container

#### 3.2.2.2.1. Kiểm tra trạng thái container

```

(base) nguyenphuc@phuc2108:~$ docker start kafka-docker_kafka_1
kafka-docker_kafka_1
(base) nguyenphuc@phuc2108:~$ docker start kafka-docker_zookeeper_1
kafka-docker_zookeeper_1
(base) nguyenphuc@phuc2108:~$ docker ps

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
890db412e63b	wurstmeister/kafka	"start-kafka.sh"	2 weeks ago	Up 10 seconds
5c5b5fb69f4e	wurstmeister/zookeeper	"/bin/sh -c '/usr/sb..."	2 weeks ago	Up 4 seconds

### *Hình 3.9. Khởi động các Container và kiểm tra*

Cả hai container zookeeper và kafka đã được khởi tạo

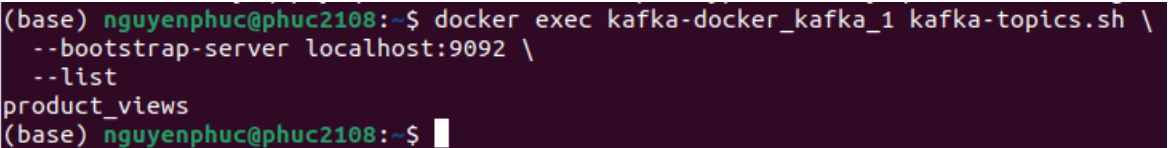
#### **3.2.2.2.2. Tạo topic cho Kafka**

Tạo topic product\_views để lưu trữ dữ liệu log trong Kafka.

- Sử dụng câu lệnh:

```
docker exec kafka-docker_kafka_1 kafka-topics.sh \  
--bootstrap-server localhost:9092 \  
--create \  
--topic product_views \  
--partitions 1 \  
--replication-factor 1
```

Kiểm tra topic



```
(base) nguyenphuc@phuc2108:~$ docker exec kafka-docker_kafka_1 kafka-topics.sh \
--bootstrap-server localhost:9092 \
--list
product_views
(base) nguyenphuc@phuc2108:~$
```

### *Hình 3.10. Kiểm tra topic*

Kafka đã tạo topic product\_views với một partition và một bản sao. Topic này sẵn sàng để nhận dữ liệu từ producer và cung cấp dữ liệu cho consumer.

## **3.3. Cài đặt PySpark**

Pyspark là thư viện Python của Apache Spark, được sử dụng để xử lý dữ liệu lớn trong hệ thống data pipeline.

### **3.3.1. Yêu cầu hệ thống**

Trước khi cài đặt PySpark, hệ thống cần đáp ứng các yêu cầu sau:

- **Hệ điều hành:** Ubuntu 22.04 (các phiên bản Linux khác như CentOS, Debian cũng tương thích).
- **Python:** Phiên bản 3.6 trở lên.
- **Java:** JDK 8, 11 hoặc 17 (khuyến nghị JDK 11).
- **Apache Spark:** Phiên bản mới nhất

- **Bộ nhớ RAM:** Tối thiểu 4GB (khuyến nghị 8GB trở lên để xử lý dữ liệu lớn).
- **Công cụ:** wget, tar, và trình chỉnh sửa văn bản (nano hoặc vim).

### 3.3.2. Quy trình cài đặt

Quy trình cài đặt được thực hiện theo các bước sau:

#### 3.3.2.1. Cài đặt Java

##### 3.3.2.1.1. Kiểm tra Java

PySpark yêu cầu Java để chạy Apache Spark.

- Kiểm tra xem Java đã được cài đặt chưa bằng lệnh:

```
java -version
```

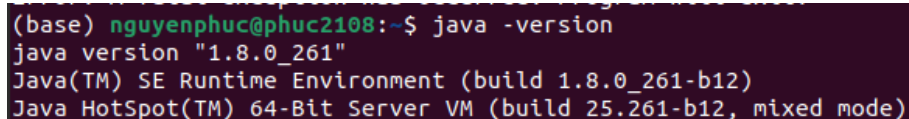
Nếu Java chưa được cài đặt, tiến hành cài đặt OpenJDK11.

##### 3.3.2.1.2. Cài đặt OpenJDK 11

- Sử dụng câu lệnh:

```
sudo apt install openjdk-11-jdk
```

##### 3.3.2.1.3. Xác nhận cài đặt:



```
(base) nguyenphuc@phuc2108:~$ java -version  
java version "1.8.0_261"  
Java(TM) SE Runtime Environment (build 1.8.0_261-b12)  
Java HotSpot(TM) 64-Bit Server VM (build 25.261-b12, mixed mode)
```

Hình 3.11. Kiểm tra phiên bản java sau khi cài đặt

#### 3.3.2.2. Cài đặt Python

##### 3.3.2.2.1. Kiểm tra Python

- Kiểm tra xem Python đã được cài đặt chưa bằng lệnh:

```
java -version
```

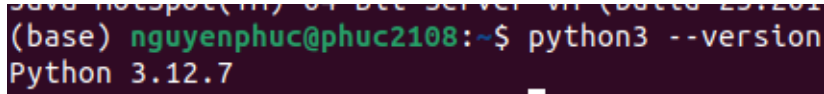
Nếu Python chưa được cài đặt, tiến hành cài đặt.

##### 3.3.2.2.2. Cài đặt Python

- Sử dụng câu lệnh:

**sudo apt install python3 python3-pip**

#### 3.3.2.2.3. Xác nhận cài đặt



```
(base) nguyenphuc@phuc2108:~$ python3 --version
Python 3.12.7
```

*Hình 3.12. Kiểm tra phiên bản python sau khi cài đặt*

#### 3.3.2.3. Tải và cài đặt Apache Spark

##### 3.3.2.3.1. Tải Apache Spark từ trang chính thức:

Tải phiên bản spark 3.5.5, vì đây là phiên bản ổn định, tương thích với các yêu cầu của dự án.

- Sử dụng câu lệnh:

**wget**  
**<https://archive.apache.org/dist/spark/spark-3.5.5/spark-3.5.5-bin-hadoop3.tgz>**

##### 3.3.2.3.2. Giải nén file tải về:

Giải nén tạo ra một thư mục chứa các thành phần của Spark

- Sử dụng câu lệnh:

**tar -xzf spark-3.5.5-bin-hadoop3.tgz**

##### 3.3.2.3.3. Di chuyển thư mục Spark đến /opt/spark

Thư mục opt thường được sử dụng để lưu trữ các phần mềm không thuộc kho lưu trữ mặc định của Ubuntu

- Sử dụng câu lệnh:

**sudo mv spark-3.5.5-bin-hadoop3 /opt/spark**

##### 3.3.2.3.4. Thiết lập biến môi trường

Thiết lập biến môi trường giúp hệ thống nhận diện các lệnh Spark và Pyspark.

- Sử dụng các câu lệnh:
  - `echo "export SPARK_HOME=/opt/spark" >> ~/.bashrc`
  - `echo "export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin" >> ~/.bashrc`
  - `echo "export PYSPARK_PYTHON=/usr/bin/python3" >> ~/.bashrc`

### 3.3.2.3.5. Tải lại cấu hình

Giúp cập nhật môi trường ngay lập tức

- Sử dụng câu lệnh: `source ~/.bashrc`

### 3.3.2.4. Kiểm tra cài đặt

Sau khi cài đặt và cấu hình. cần kiểm tra xem Pyspark đã được cài đặt đúng cách và hoạt động hay chưa.

```
(base) nguyenphuc@phuc2108:~$ pyspark
Python 3.12.7 | packaged by Anaconda, Inc. | (main, Oct 4 2024, 13:27:36) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
25/05/27 19:45:59 WARN Utils: Your hostname, phuc2108 resolves to a loopback address: 127.0.1.1; using 192.168.74.89 instead (on interface wlp0s20f3)
25/05/27 19:45:59 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/05/27 19:46:00 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to

  ____      __
 / ___ |__ /  __/
/ /___/ __/  /
/____/____/_/

version 3.5.5

Using Python version 3.12.7 (main, Oct 4 2024 13:27:36)
Spark context Web UI available at http://phuc2108.lan:4040
Spark context available as 'sc' (master = local[*], app id = local-1748317560827).
SparkSession available as 'spark'.
>>>
```

*Hình 3.13. Kiểm tra phiên bản pyspark sau khi cài đặt*

Kết quả trả về: phiên bản Spark hiện tại là 3.5.5. Spark đã được cài đặt và sẵn sàng để xử lý dữ liệu.

## 3.4. Kết luận

Ở chương này, dự án đã hoàn thành các bước cài đặt môi trường triển khai hệ thống data pipeline.

Trong phần tiếp theo sẽ nói về những gì đạt được cũng như hạn chế còn tồn đọng, đồng thời định hướng phát triển hệ thống trong tương lai.



## **PHẦN KẾT LUẬN**

### **1. Tóm tắt các nội dung đã thực hiện**

- Data pipeline đã hoàn thành được những mục tiêu ban đầu đặt ra
- Dữ liệu đã được chu chuyển và xử lý qua từng giai đoạn
  - Đưa dữ liệu lên MongoDB
  - Lấy những dữ liệu cần thiết bằng Python và đưa qua Kafka
  - Spark lấy dữ liệu từ Kafka và xử lý sâu hơn
  - Dữ liệu sau khi xử lý được đưa lên MySQL

### **2. Những điểm hạn chế của hệ thống**

- Các tính năng còn cơ bản, chưa phát triển các chức năng nâng cao.
- Chưa tích hợp các kỹ thuật phức tạp như machine learning hoặc deep learning.

### **3. Hướng phát triển trong tương lai**

- Tích hợp machine learning vào hệ thống
- Thiết kế pipeline theo kiến trúc microservices để dễ dàng tích hợp công cụ mới hoặc thay đổi cấu trúc mà không ảnh hưởng toàn hệ thống.

## TÀI LIỆU THAM KHẢO

- [1] <https://viblo.asia/newest>
- [2] <https://www.altexsoft.com/blog/apache-spark-pros-cons/>
- [3] [https://www.tutorialspoint.com/apache\\_spark/](https://www.tutorialspoint.com/apache_spark/)
- [4] <https://www.mongodb.com/docs/>
- [5] <https://aws.amazon.com/vi/what-is/etl/>
- [6] <https://www.mysql.com/>