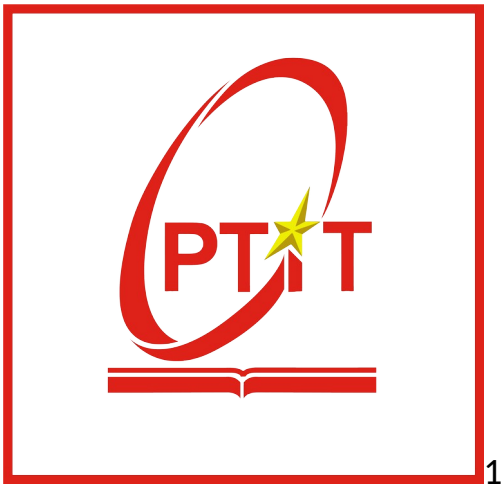
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



1

Báo cáo hàng tuần

Môn học: Thực tập cơ sở

Giảng viên: Kim Ngọc Bách

Họ và tên: Nguyễn Hữu Phúc

Mã SV: B22DCAT224

Lớp: E22CQCN04-B

Báo cáo tuần 10: Em bắt đầu viết báo cáo cho dự án

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 thiết kế và triển khai một **data pipeline** hiện đại, có khả năng thu thập, xử lý, làm sạch và phân tích dữ liệu log từ hệ thống thương mại điện tử. Mục tiêu chính bao gồm:

- **Đảm bảo tính toàn vẹn dữ liệu**: Xây dựng một pipeline mạnh mẽ, đảm bảo dữ liệu được truyền tải và xử lý mà không bị mất mát, sai lệch hoặc trùng lặp.
- **Tối ưu hóa hiệu suất xử lý**: Tăng tốc độ xử lý dữ liệu lớn, giảm độ trễ trong việc cung cấp thông tin để hỗ trợ ra quyết định kinh doanh theo thời gian thực.
- **Ứng dụng công nghệ tiên tiến**: Tích hợp các công nghệ và framework hiện đại như Apache Spark, Apache Kafka và các giải pháp lưu trữ phân tán để xây dựng một hệ thống linh hoạt, có khả năng mở rộng.
- **Tạo giá trị kinh doanh**: Cung cấp các insight giá trị từ dữ liệu log, hỗ trợ doanh nghiệp tối ưu hóa chiến lược tiếp thị, cải thiện trải nghiệm khách hàng và nâng cao hiệu quả vận hành.

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 như máy chủ web, hệ thống theo dõi hành vi người dùng và các API liên quan.
- **Xử lý và làm sạch dữ liệu**: Thực hiện các quy trình làm sạch dữ liệu, loại bỏ nhiễu, 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 (data warehouse) hoặc hồ dữ liệu (data lake) để hỗ trợ truy vấn hiệu quả và phân tích sâu.
- **Tích hợp hệ thống**: Đảm bảo pipeline tích hợp tốt với các hệ thống hiện có của Glamira, bao gồm các công cụ BI (Business Intelligence) và các ứng dụng kinh doanh khác.

1.1.3. Các bài toán cần giải quyết

1.1.3.1. Xử lí 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 hoặc Dask để phân chia khối lượng công việc, giảm thời gian xử lý.
- Khả năng mở rộng (Scalability): 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 (Reliability): Đảm bảo dữ liệu được xử lý không bị mất mát hoặc hỏng hóc, sử dụng các cơ chế như checkpointing và fault tolerance.
- 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/CSV/log.
- 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.

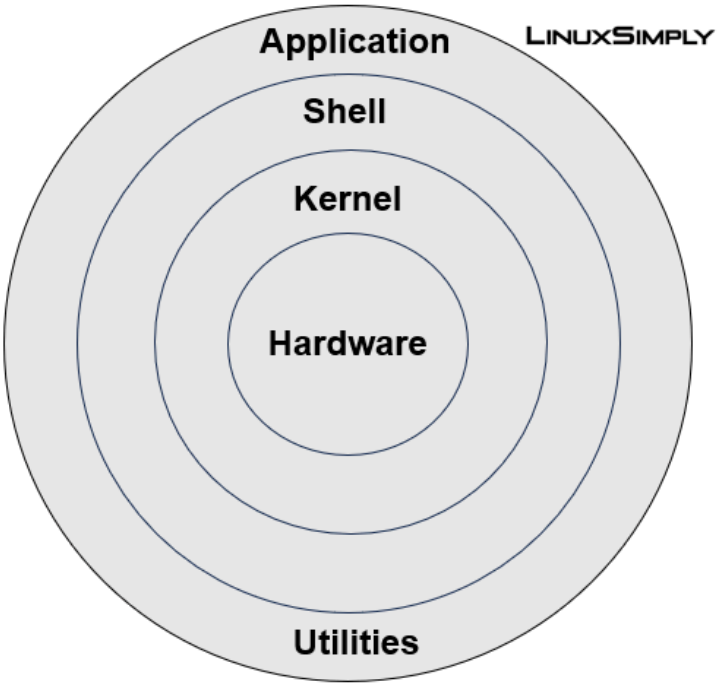
b. Ưu điểm

- 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 đẻ 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



– Kernel: Đây là phần quan trọng và được ví như trái tim của HĐH, phần kernel chứa các module, thư viện để quản lý và giao tiếp với phần cứng và các ứng dụng.
– Shell: Shell là một chương trình 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.
– Applications: Là các ứng dụng và tiện ích mà người dùng cài đặt trên Server. Ví dụ: ftp, samba, Proxy, …

1.2.2. Docker

1.2.2.1. Giới thiệu về Docker

a. Docker là gì

Docker là một nền tàng mã nguồn mở cung cấp các công cụ để các developers, adminsystems 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 lí dữ liệu lớn

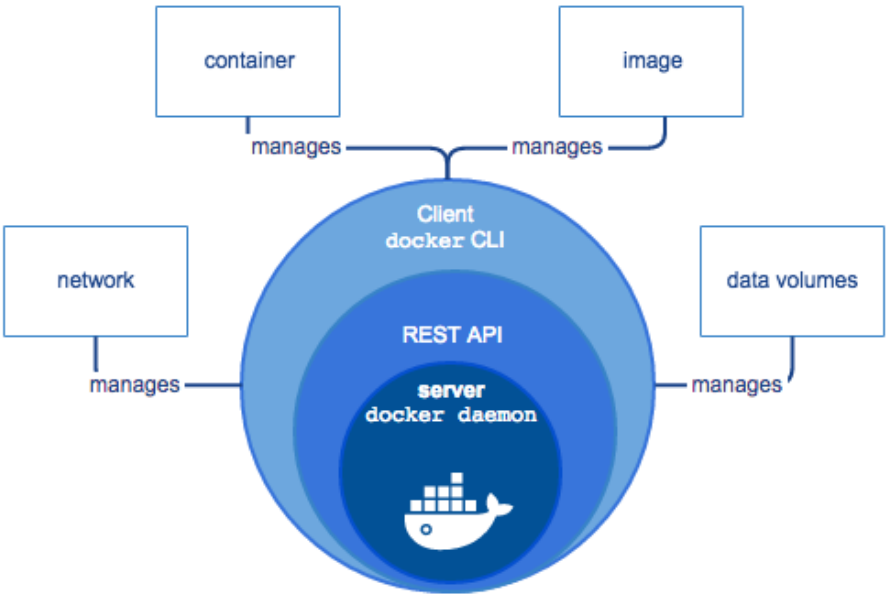
c. Nhược điểm

- **Phức tạp trong việc quản lý**
  Quản lý nhiều containers đòi hỏi kinh nghiệm của người sử dụng

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



Docker engine là một ứng dụng client-server. có 2 phiên bản phổ biến:

- Docker Community Edition (CE): Là phiên bản miễn phí và chủ yếu dựa vào các sản phẩm nguồn mở khác.
- Docker Enterprise(EE): Khi sử dụng phiên bản này bạn sẽ nhận được sự support của nhà phát hành, có thêm các tính năng quản lý và security.

Các thành phần chính của dockẻ engine gồm có:

- Server hay còn được gọi là docker daemon: chịu trách nhiệm tạo, quản lý các Docker objects như images, containers, networks, volume.

- REST API: docker daemon cung cấp các api cho Client sử dụng để thao tác với Docker
- Client là thành phần đầu cuối cung cấp một tập hợp các câu lệnh sử dụng api để người dùng thao tác với Docker.

b. Distribution Tools

- **Docker Registry**: Kho lưu trữ Docker image, hỗ trợ public/private (ví dụ: Docker Hub, AWS ECR).
- **Docker Hub**: Dịch vụ SaaS lưu trữ >100,000 image công khai, hỗ trợ registry public/private để chia sẻ và quản lý image cho pipeline (như Spark, Kafka).
- **Docker Trusted Registry**: Giải pháp registry riêng tư, an toàn cho doanh nghiệp.

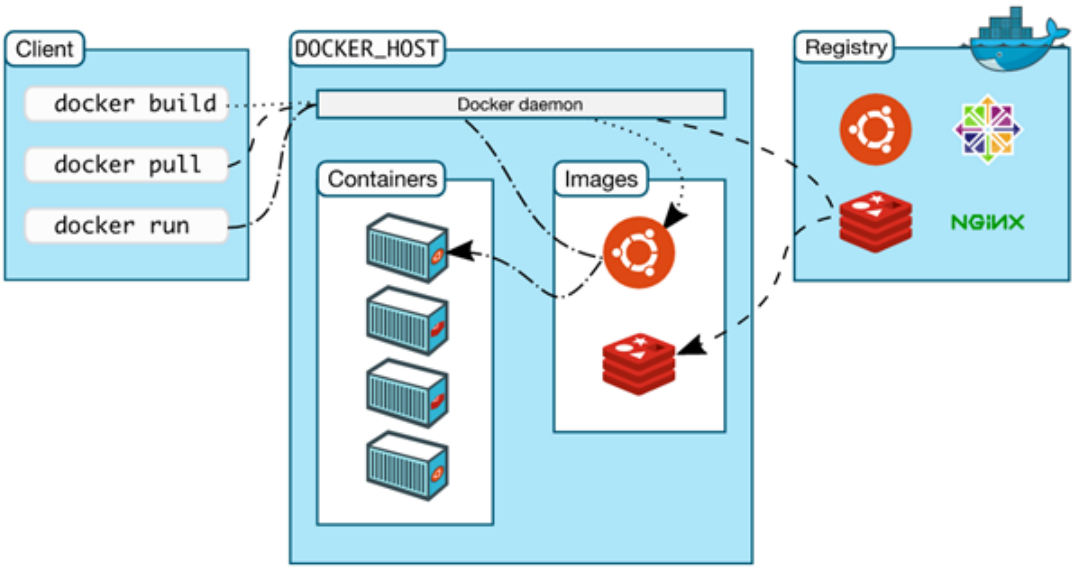
c. Orchestration Tools

- **Docker Machine**: Tạo và cấu hình Docker Engine trên local, cloud (AWS, Azure, GCP) hoặc data center (VMware, OpenStack), tự động cài đặt và kết nối Docker Client với Engine.
- **Docker Compose**: Định nghĩa và chạy ứng dụng multi-container qua file YAML, phù hợp cho pipeline ETL với nhiều dịch vụ (Spark, Kafka).
- **Docker Swarm**: Tạo cluster từ nhiều Docker Engine, hoạt động như một virtual Engine duy nhất, hỗ trợ cân bằng tải và mở rộng container.

d. Thành phần khác

- **Dockerfile**: Script chứa các lệnh tuần tự để build image từ image gốc (như Ubuntu + Spark), tự động hóa tạo image cho pipeline.
- **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



Docker sử dụng mô hình client-server, với các thành phần chính:

- Docker Daemon (dockerd):
  - Quản lý container, image, network, volume qua Docker API.
  - Có thể giao tiếp với các daemon khác để quản lý service.
- Docker Client:
  - CLI (docker run, docker pull) gửi lệnh đến daemon qua REST API (UNIX socket hoặc mạng).
  - Hỗ trợ giao tiếp với nhiều daemon.
- Docker Registry:
  - Lưu trữ image (Docker Hub hoặc registry riêng như AWS ECR).
  - Hỗ trợ docker pull/push để tải/lưu image.
- Docker Image:
  - Template chỉ đọc, đóng gói ứng dụng (ví dụ: Spark, Kafka) và phụ thuộc.
  - Tạo từ Dockerfile (như Ubuntu + Apache).
- Docker Container:
  - Thực thể chạy từ image, cô lập bằng cgroups/namespaces.
  - Có thể kết nối network, volume, hoặc tạo image mới từ trạng thái container.
- Docker Volume:
  - Lưu trữ dữ liệu độc lập, phù hợp cho log Glamira.
- Docker Network:

    - Cung cấp private network để container (Spark, Kafka) giao tiếp.
- Docker Service:
    - Mở rộng container qua swarm cluster, cân bằng tải trên nhiều daemon.

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

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
- Ứng dụng trong hệ thống
    - Lưu trữ dữ liệu log của Website Glamira, là điểm khởi đầu của đường ống dữ liệu.

b. MySQL

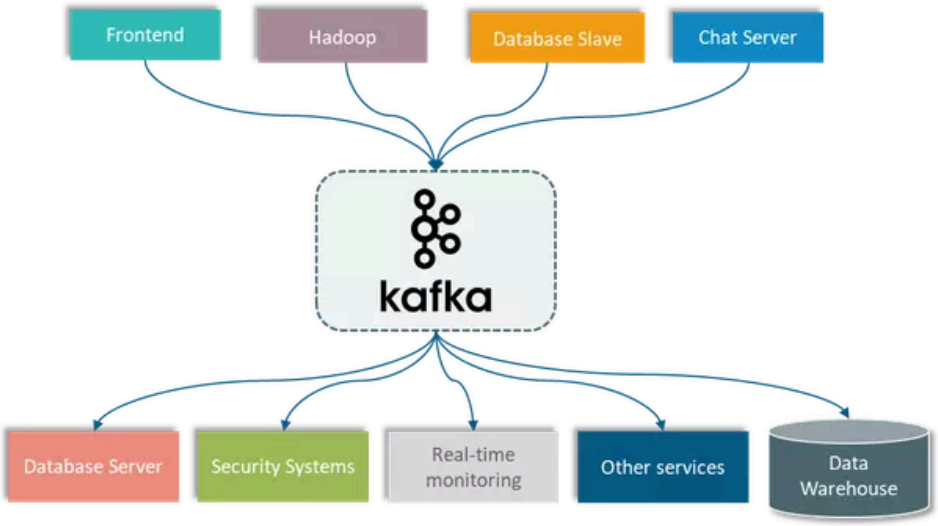
- MySQL là một hệ quản trị cơ sở dữ liệu quan hệ phổ biến, dựa trên ngôn ngữ SQL.
- Đặc điểm:
    - Lưu trữ dự liệu có cấu trúc, dễ dàng thực hiện các truy vấn phức tạp
    - Hỗ trợ tính toàn vẹn dữ liệu, bảo mật cao
- Ứng dụng trong hệ thống
    - Là điểm hứng dữ liệu từ Spark đẩy vào

1.2.4. Kiến trúc Kafka

1.2.4.1. Giới thiệu về Kafka

a. Kafka là gì?

Kafka là một nền tảng streaming dữ liệu phân tán, xử lí 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 lí dữ liệu một cách hiệu quả.



b. Ưu điểm

- **Độ tin cậy cao**

  Kafka được thiết kế với kiến trúc phân tán, phân chia, đồng bộ và chịu lỗi. Điều này đảm bảo rằng hệ thống có thể hoạt động ổn định ngay cả khi một số thành phần gặp sự cố.

- **Khả năng mở rộng**

  Kafka có khả năng mở rộng theo chiều ngang một cách dễ dàng bằng cách thêm các node mới vào cụm 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 mà không làm giảm hiệu suất.

- **Độ bền**

  Thông điệp trong Kafka được lưu trữ trên ổ đĩa và có thể được giữ lại trong thời gian dài, tùy thuộc vào cấu hình retention period.

- **Hiệu suất cao**

Kafka nổi bật với khả năng xử lý thông lượng cực lớn, phù hợp cho các ứng dụng yêu cầu publish-subscribe thông điệp với tốc độ cao. Nó có thể xử lý hàng triệu thông điệp mỗi giây với độ trễ thấp (low latency). Ngay cả khi lưu trữ hàng terabyte dữ liệu, Kafka vẫn duy trì được hiệu suất ổn định nhờ vào cơ chế đọc/ghi tuần tự trên ổ đĩa và tối ưu hóa việc sử dụng bộ nhớ.

- **Hệ sinh thái mạnh mẽ**

  Kafka tích hợp tốt với nhiều công cụ và nền tảng khác trong hệ sinh thái dữ liệu lớn, như Apache Hadoop, Apache Spark, Apache Flink, và các công cụ ETL (Extract, Transform, Load). Ngoài ra, Kafka Connect và Kafka Streams cung cấp các API mạnh mẽ để xây dựng các pipeline dữ liệu phức tạp và xử lý luồng dữ liệu theo thời gian thực.

c. Nhược điểm

- **Độ phức tạp trong triển khai và quản lý**

  Kafka là một hệ thống phức tạp với nhiều thành phần, bao gồm các broker, ZooKeeper, và các consumer/producer. Việc thiết lập, cấu hình, và quản lý một cụm Kafka đòi hỏi kiến thức chuyên sâu và đội ngũ vận hành có kinh nghiệm. Các vấn đề như cân bằng tải (load balancing), tối ưu hóa hiệu suất, hoặc xử lý lỗi có thể trở nên khó khăn đối với các tổ chức không có chuyên môn.

- **Yêu cầu tài nguyên cao**

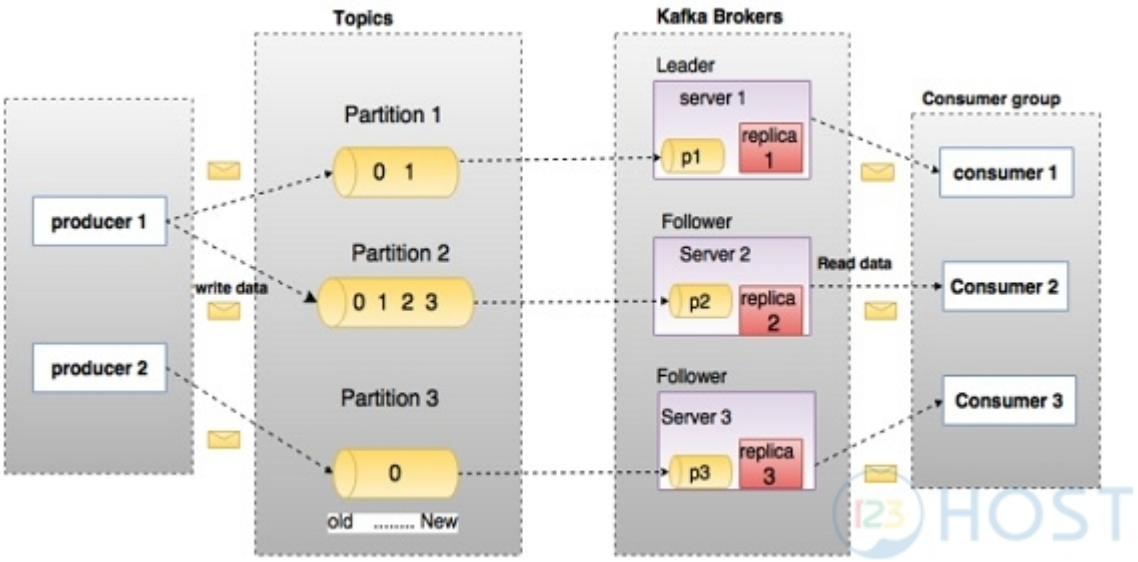
  Kafka yêu cầu phần cứng mạnh mẽ để đạt được hiệu suất tối ưu, đặc biệt là khi xử lý khối lượng dữ liệu lớn. Các cụm Kafka thường cần nhiều RAM, CPU, và dung lượng ổ đĩa, dẫn đến chi phí vận hành cao. Ngoài ra, ZooKeeper, một thành phần quan trọng trong kiến trúc Kafka, cũng cần được quản lý cẩn thận để đảm bảo hoạt động ổn định.

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

  Do thiết kế bất biến (immutable) của Kafka, việc sửa đổi hoặc xóa dữ liệu đã được ghi vào topic là rất khó. Điều này có thể gây bất

tiện trong các trường hợp cần chỉnh sửa dữ liệu hoặc tuân thủ các quy định về bảo mật dữ liệu (như GDPR)

1.2.4.2. Tổng quan về kiến trúc Kafka

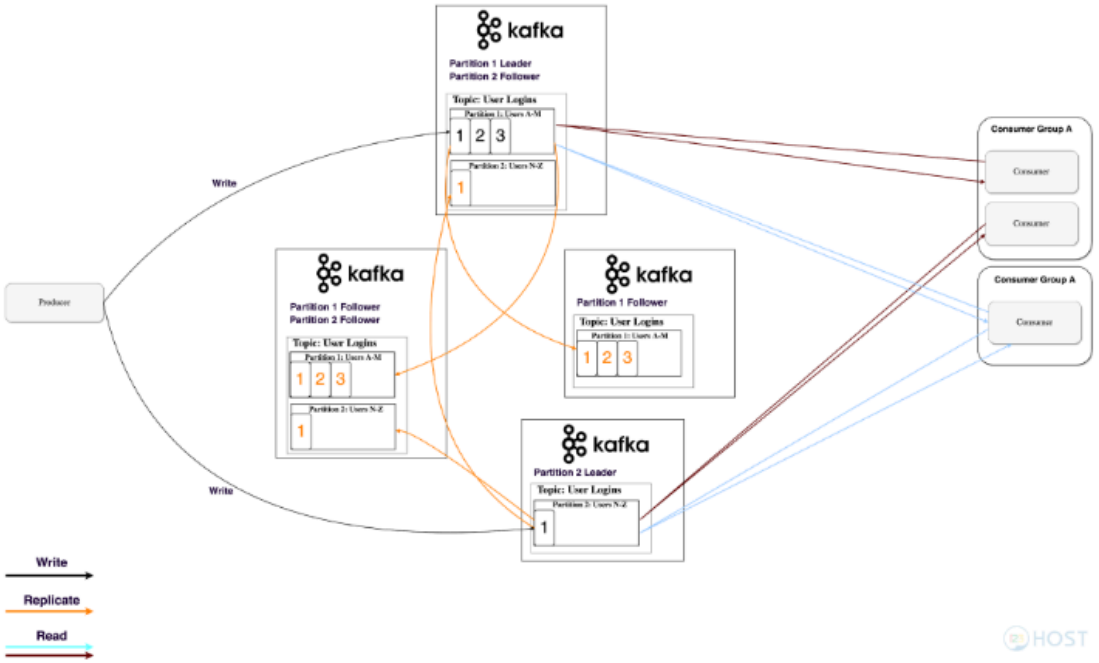


Các thuật ngữ:

- 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 lí 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 lí 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 par 1,2,3
- Producers:
    - Ứng dụng truyền messages đến cho Kaka 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 lí 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



a. Producer gửi messages

- Gửi đến một topic trong Kafka, topic này được chia thành nhiều partition để dễ xử lí
- 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 lí theo đúng thứ tự

c. Consumer đọc messages

- Đọc messages từ topic để xử lí, 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 lí 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

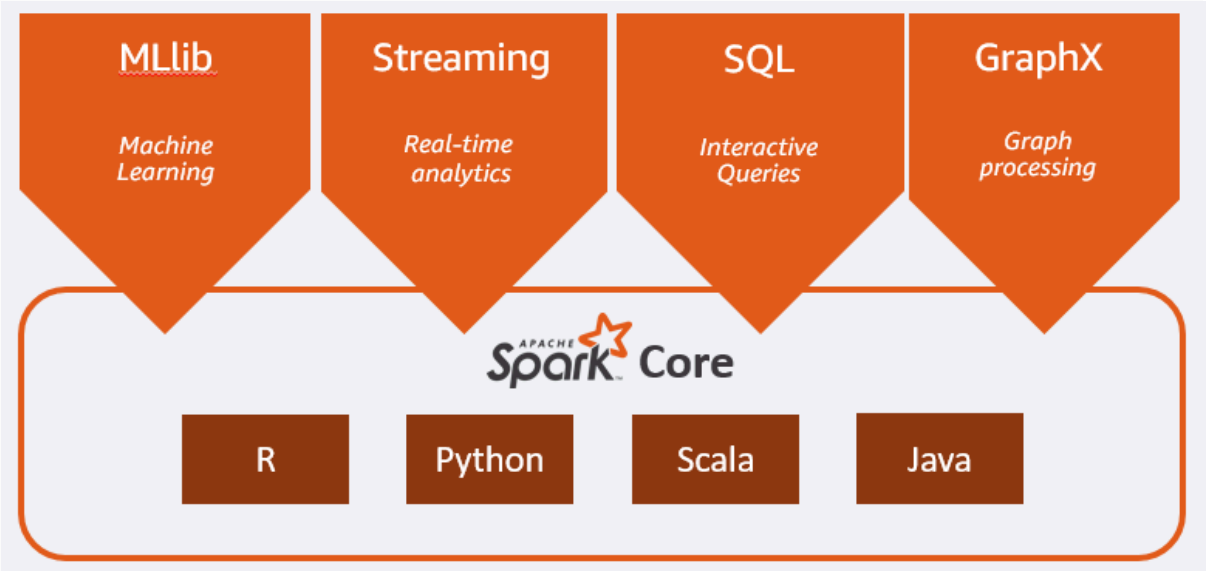
1.2.5. Apache Spark

a. Apache Spark là gì?

- Apache Spark là một hệ thống xử lý phân tán nguồn mở được sử dụng cho các khối lượng công việc dữ liệu lớn. Hệ thống này sử dụng khả năng ghi vào bộ nhớ đệm nằm trong bộ nhớ và thực thi truy vấn tối ưu hóa nhằm giúp truy vấn phân tích nhanh dữ liệu có kích thước bất kỳ.
- 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. Thành phần của Apache Spark

- Spark Core làm nền móng cho nền tảng
- Spark SQL cho các truy vấn tương tác
- Spark Streaming để phân tích theo thời gian thực
- Spark MLlib dành cho máy học
- Spark GraphX dành cho xử lý đồ thị



Spark Core

Nền tảng cốt lõi: 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

Máy học: Thư viện thuật toán cho phân loại, hồi quy, phân cụm, lọc cộng tác, khai thác mẫu. Hỗ trợ đào tạo mô hình trên dữ liệu Hadoop, tích hợp R/Python và triển khai qua Java/Scala. Tối ưu cho điện toán trong bộ nhớ, đảm bảo tốc độ cao.

Spark Streaming

Thời gian thực: Xử lý luồng dữ liệu theo lô nhỏ, tận dụng Spark Core để phân tích liên tục. Sử dụng cùng mã cho xử lý lô và luồng, tăng năng suất. Hỗ trợ nguồn dữ liệu như Kafka, Flume, HDFS, Twitter, ZeroMQ.

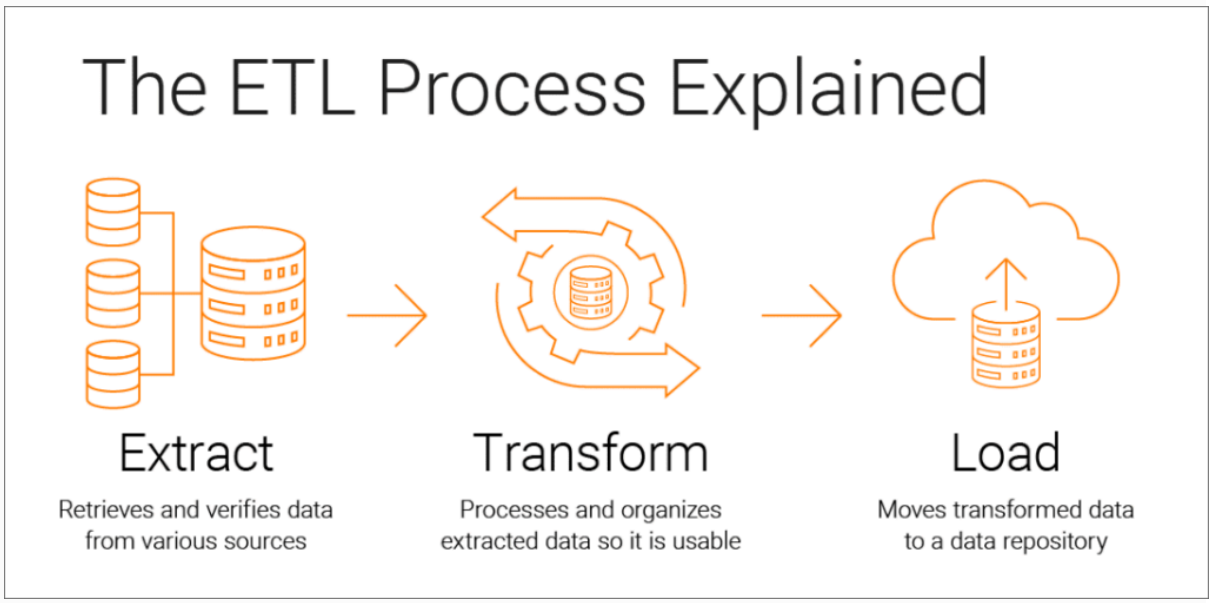
Spark SQL

Truy vấn tương tác: Truy vấn phân tán tốc độ cao, nhanh hơn MapReduce tới 100 lần. Tích hợp SQL, Hive QL, API (Scala, Java, Python, R) và trình tối ưu hóa. Hỗ trợ JDBC, ODBC, JSON, Hive, Parquet, Redshift, Cassandra, MongoDB, v.v.

GraphX

Xử lý đồ thị: 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.

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



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:

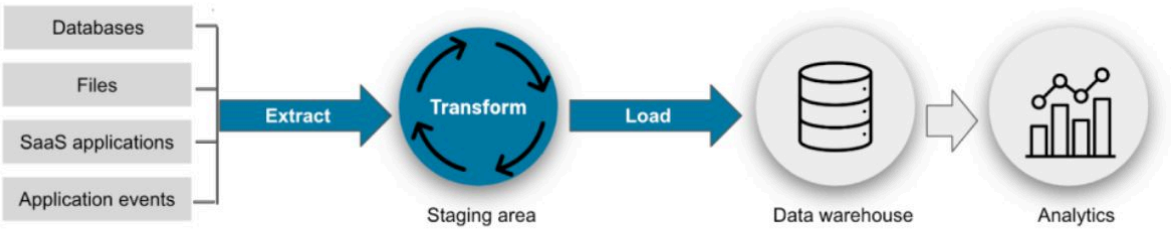
- **Extract:** Đây là bước đầu tiên, trong đó dữ liệu được thu thập từ nhiều nguồn khác nhau, chẳng hạn như cơ sở dữ liệu và hệ thống khác. Quá trình này đảm bảo rằng tất cả dữ liệu cần thiết được lấy ra để chuẩn bị cho các bước tiếp theo.
- **Transform:** Sau khi dữ liệu đã được trích xuất, nó cần phải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

1.3.2. Cách thức hoạt động

Quá 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



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. Ví dụ, áp dụng các quy tắc để tính toán các giá trị mới dựa trên giá trị hiện có.

- **Dẫn xuất**

  Đây là việc áp dụng quy tắc kinh doanh vào dữ liệu để tính toán các giá trị mới dựa trên giá trị hiện có. Ví dụ, chuyển đổi doanh thu thành lợi nhuận bằng cách trừ đi chi phí hoặc tính tổng chi phí mua hàng.
- **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. Ví dụ, tính tổng chi phí mua hàng từ các nhà cung cấp khác nhau và giữ lại tổng cuối trong hệ thống đích.
- **Chia tách**
  Chuyển đổi một cột hoặc thuộc tính dữ liệu thành nhiều cột trong hệ thống đích. Ví dụ, chia tách tên khách hàng thành các cột họ, tên đệm và tên riêng.
- **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. Ví dụ, tóm tắt giá trị hóa đơn của đơn đặt hàng khách hàng trong một khoảng thời gian nhất định để xây dựng chỉ số giá trị lâu dài của khách hàng (CLV).

- **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 delta (hoặc 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, bạn 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, bạn 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 bạn có khối lượng dữ liệu lớn, bạn 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 lấy dữ liệu từ mongoDB
- Transform - Giai đoạn chuyển đổi: Sử dụng Python và Spark
- Load - Giai đoạn tải: MySQL được sử dụng như một cơ sở dữ liệu quan hệ để lưu trữ dữ liệu đã xử lý.

1.4. Mô tả hệ thống

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.