**Trung tâm nghiên cứu và phát triển  
điện thoại di động Samsung Việt Nam**

**Báo cáo thực tập**

**Đề tài T27: Realtime Data/Message stream transform for Bigdata collection system**

|  |  |
| --- | --- |
| **Sinh viên thực hiện:** | Đinh Mạnh Hùng *(dinhhung2910@gmail.com)* |
| **Trường:** | Đại học Bách Khoa Hà Nội |
| **Người hướng dẫn:** | Phạm Hà Vân *(havan.pham@samsung.com)*  Đỗ Văn Thanh *(thanh.dv3@samsung.com)* |
| **HÀ NỘI, 6/2021** | |

**BÁO CÁO THỰC TẬP**

**1. Thông tin sinh viên**

Họ và tên sinh viên: Đinh Mạnh Hùng CMND: 145891497

Số điện thoại: 0966145823 Email: [dinhhung2910@gmail.com](mailto:dinhhung2910@gmail.com)

Lớp: CNTT2.01 - K61 Trường: ĐH Bách Khoa Hà Nội

**2. Nội dung của báo cáo thực tập**

Tìm hiểu về chuyển đổi dữ liệu thời gian thực (real-time data transforming), hàng đợi thông điệp (message queue). Từ đó ứng dụng xây dựng một hệ thống xử lý dữ liệu trực tuyến thời gian thực sử dụng Apache kafka và Apache flink.

**3. Các nhiệm vụ chính của báo cáo thực tập**

* Tìm hiểu về hàng đợi thông điệp, chuyển đổi dữ liệu thời gian thực.
* Tìm hiểu các nền tảng Apache Kafka, Apache Flink.
* Xây dựng hệ thống vận dụng các lý thuyết trên.

.

Giáo viên hướng dẫn

Ký và ghi rõ họ tên

**Tóm tắt nội dung báo cáo**

Trong những năm gần đây, với sự bùng nổ của các thiết bị và mạng internet, lượng thông tin được sinh ra trong các tác vụ hằng ngày đang trở nên ngày càng lớn với tốc độ ngày một nhanh, đặt ra nhiều thách thức về việc lưu trữ cũng như xử lý chúng.

Trên thực tế, bài toán xử lý dữ liệu lớn là một lĩnh vực vô cùng rộng lớn. Do vậy, trong bài báo cáo này em xin tập trung vào một khía cạnh quan trọng của xử lý dữ liệu lớn, đó là xử lý và biến đổi dữ liệu luồng theo thời gian thực. Nhiều nền tảng đã được tạo ra để giải quyết vấn đề này, tiêu biểu nhất trong số đó phải kể đến hệ sinh thái của Apache với Apache Hadoop, Apache Spark, Apache Kafka…

Thông qua việc tìm hiểu cơ sở lý thuyết và các nền tảng kể trên, em đã xây dựng được

Nội dung của bài báo cáo được trình bày qua 4 chương:

**Chương 1:** Cơ sở lý thuyết

**Chương 2:** Cơ sở công nghệ

**Chương 3:** Xây dựng hệ thống demo

**Chương 4:** Kết luận

**MỤC LỤC**

[CHƯƠNG 1. CƠ SỞ LÝ THUYẾT 1](#_Toc73927589)

[1.1 Giới thiệu bài toán 1](#_Toc73927590)

[1.1.1 Khái niệm luồng dữ liệu (data stream) 1](#_Toc73927591)

[1.1.2 Các cách tiếp cận trước đây 1](#_Toc73927592)

[1.2 Xử lý luồng dữ liệu theo thời gian thực (real-time datastream processing) 2](#_Toc73927593)

[1.2.1 Khái niệm 2](#_Toc73927594)

[1.2.2 Đặc điểm 3](#_Toc73927595)

[1.2.3 Các thành phần của hệ thống xử lý luồng dữ liệu 3](#_Toc73927596)

[CHƯƠNG 2. CƠ SỞ CÔNG NGHỆ 6](#_Toc73927597)

[2.1 Apache Kafka 6](#_Toc73927598)

[2.1.1 Giới thiệu về Kafka 6](#_Toc73927599)

[2.1.2 Hoạt động 7](#_Toc73927600)

[2.2 Apache Flink 10](#_Toc73927601)

[2.2.1 Giới thiệu về Apache Flink 10](#_Toc73927602)

[2.2.2 Luồng hoạt động 11](#_Toc73927603)

[2.3 Các công nghệ khác 11](#_Toc73927604)

[2.3.1 Docker 12](#_Toc73927605)

[2.3.2 MongoDB 12](#_Toc73927606)

[2.3.3 NodeJS 13](#_Toc73927607)

[2.3.4 ReactJS 13](#_Toc73927608)

[2.3.5 Elasticsearch & Kibana 14](#_Toc73927609)

[CHƯƠNG 3. CÀI ĐẶT THỰC NGHIỆM 15](#_Toc73927610)

[3.1 Tổng quan 15](#_Toc73927611)

[3.2 Cài đặt môi trường với docker 15](#_Toc73927612)

[3.2.1 Build docker image cho flink consumer 15](#_Toc73927613)

[3.2.2 Cấu hình docker-compose cho các docker container 16](#_Toc73927614)

[3.3 Cài đặt MongoDB 18](#_Toc73927615)

[3.4 Xây dựng ứng dụng web với vai trò producer 19](#_Toc73927616)

[3.5 Cấu hình Apache Kafka 20](#_Toc73927617)

[3.6 Lập trình ứng dụng Flink Consumer 21](#_Toc73927618)

[3.7 Cấu hình Kibana để visualize dữ liệu 22](#_Toc73927619)

[CHƯƠNG 4. KẾT LUẬN 24](#_Toc73927620)

[4.1 Kết quả đạt được 24](#_Toc73927621)

[4.2 Định hướng phát triển: 24](#_Toc73927622)

[TÀI LIỆU THAM KHẢO 25](#_Toc73927623)

**DANH MỤC HÌNH VẼ**

[Hình 1: Sơ đồ phương pháp xử lý dữ liệu theo lô 2](#_Toc73927624)

[Hình 2: Sơ đồ phương pháp xử lý dữ liệu theo luồng 3](#_Toc73927625)

[Hình 3: Sơ đồ tuần tự minh họa message broker 4](#_Toc73927626)

[Hình 4: Vị trí của các công cụ ETL trong hệ thống xử lý dữ liệu thời gian thực 5](#_Toc73927627)

[Hình 5: Sơ đồ cấu trúc Kafka 7](#_Toc73927628)

[Hình 6: Minh họa topic và partition 8](#_Toc73927629)

[Hình 7: Minh họa replica của partition 8](#_Toc73927630)

[Hình 8: Minh họa consumer group 9](#_Toc73927631)

[Hình 9: Sơ đồ luồng hoạt động của Flink 11](#_Toc73927632)

[Hình 10: Minh họa cơ chế hoạt động của Docker 12](#_Toc73927633)

[Hình 11: MongoDB 12](#_Toc73927634)

[Hình 12: NodeJS 13](#_Toc73927635)

[Hình 13: Các tính năng của ReactJS 13](#_Toc73927636)

[Hình 14: Một số dạng biểu đồ được hỗ trợ bởi Kibana 14](#_Toc73927637)

[Hình 15: Sơ đồ hệ thống demo 15](#_Toc73927638)

[Hình 16: Sơ đồ thiết kế cơ sở dữ liệu 19](#_Toc73927639)

[Hình 17: Giao diện web cho người dùng 19](#_Toc73927640)

[Hình 18: Danh sách các topic trên Kafka 20](#_Toc73927641)

[Hình 19: Danh sách các Broker 21](#_Toc73927642)

[Hình 20: Chọn index user-tracking trong Kibana 22](#_Toc73927643)

[Hình 21: Dữ liệu đã được enrich 23](#_Toc73927644)

[Hình 22: Tạo biểu đồ tại dashboard 23](#_Toc73927645)

# CƠ SỞ LÝ THUYẾT

## Giới thiệu bài toán

### Khái niệm luồng dữ liệu (data stream)

Từ những năm cuối thế kỷ XX, sự ra đời của Internet đã thực sự thay đổi cách con người tiếp nhận và xử lý thông tin. Internet cho phép chúng ta khả năng truy cập, tìm kiếm thông tin, giao tiếp gần như ngay lập tức tới mọi nơi trên thế giới. Cho đến những năm đầu thế kỷ XXI, khi giá thành các thiết bị máy tính ngày càng rẻ, Internet dần trở nên dễ tiếp cận, thì sự kết nối của con người ngày càng mạnh mẽ. Kết quả là một lượng dữ liệu khổng lồ được sinh ra, cùng với đó là nhiều khái niệm mới ra đời, ví dụ như “luồng dữ liệu”.

Luồng dữ liệu được dùng để chỉ những dữ liệu được sinh ra một cách liên tục, với một khối lượng lớn và tốc độ sinh nhanh. Luồng dữ liệu thường bao gồm bản ghi nhật ký của các sự kiện khi nó diễn ra, ví dụ sự kiện người dùng nhấn vào 1 link trên trang web, hay việc cảm biến ghi lại nhiệt độ hiện tại.

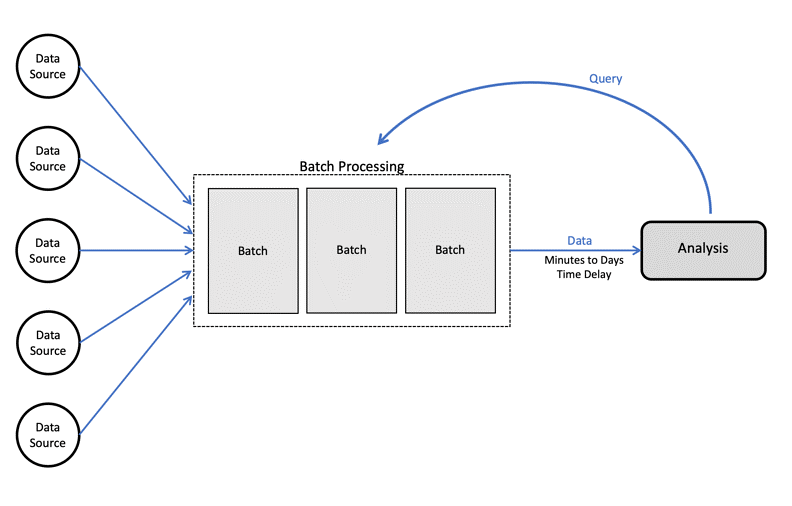
Một vài nguồn luồng dữ liệu có thể kể đến như:

* Các cảm biến IoT
* Nhật ký bảo mật (security logs) của server
* Quảng cáo thời gian thực

Dữ liệu sinh ra trong các trường hợp trên thường ở dạng không có cấu trúc, hoặc bán cấu trúc, ví dụ như JSON, XML, … hoặc có thể chỉ là chuỗi kí tự. Chính vì tính thiếu cấu trúc của luồng dữ liệu và cũng như khối lượng khổng lồ của chúng nên ta không thể làm việc trực tiếp với những dữ liệu đó, mà thường phải thông qua một quá trình tiền xử lý, phân tích cú pháp và tái cấu trúc trước khi có thể sử dụng.

### Các cách tiếp cận trước đây

Trước khi việc xử lý dữ liệu luồng ra đời, phương pháp phổ biến được sử dụng là xử lý theo lô (batch processing). Theo phương pháp đó, toàn bộ dữ liệu được lưu trữ trong một cơ sở dữ liệu hoặc một hệ thống file phân tán dưới dạng các tệp hoặc bản ghi, các ứng dụng sẽ thực hiện việc tính toán dựa trên các bản ghi này. Các công cụ xử lý dữ liệu theo lô được xây dựng để xử lý một tập dữ liệu với kích thước hữu hạn. Do đó, để xử lý được dữ liệu mới một cách liên tục, cần có một ứng dụng chuyên tính toán lượng dữ liệu mới được sinh ra theo mỗi chu kỳ, ví dụ theo phút, theo giờ,…



Hình : Sơ đồ phương pháp xử lý dữ liệu theo lô

Mô hình này vẫn tồn tại và hoạt động cho đến ngày nay và vẫn hiệu quả với các ứng dụng quy mô nhỏ. Tuy nhiên, với việc không xử lý dữ liệu mới ngay từ khi nó được sinh ra, ta có thể thấy rõ một số hạn chế của nó:

* Độ trễ cao: Những dữ liệu mới chỉ được xử lý sau một khoảng thời gian nhất định, làm mất đi giá trị của dữ liệu.
* Dữ liệu bị chia thành từng lô: Trong một số trường hợp, một sự kiện có thể bị chia nhỏ, bắt đầu ở lô này nhưng lại kết thúc ở lô khác. Dẫn đến việc khó xử lý những sự kiện như vậy.

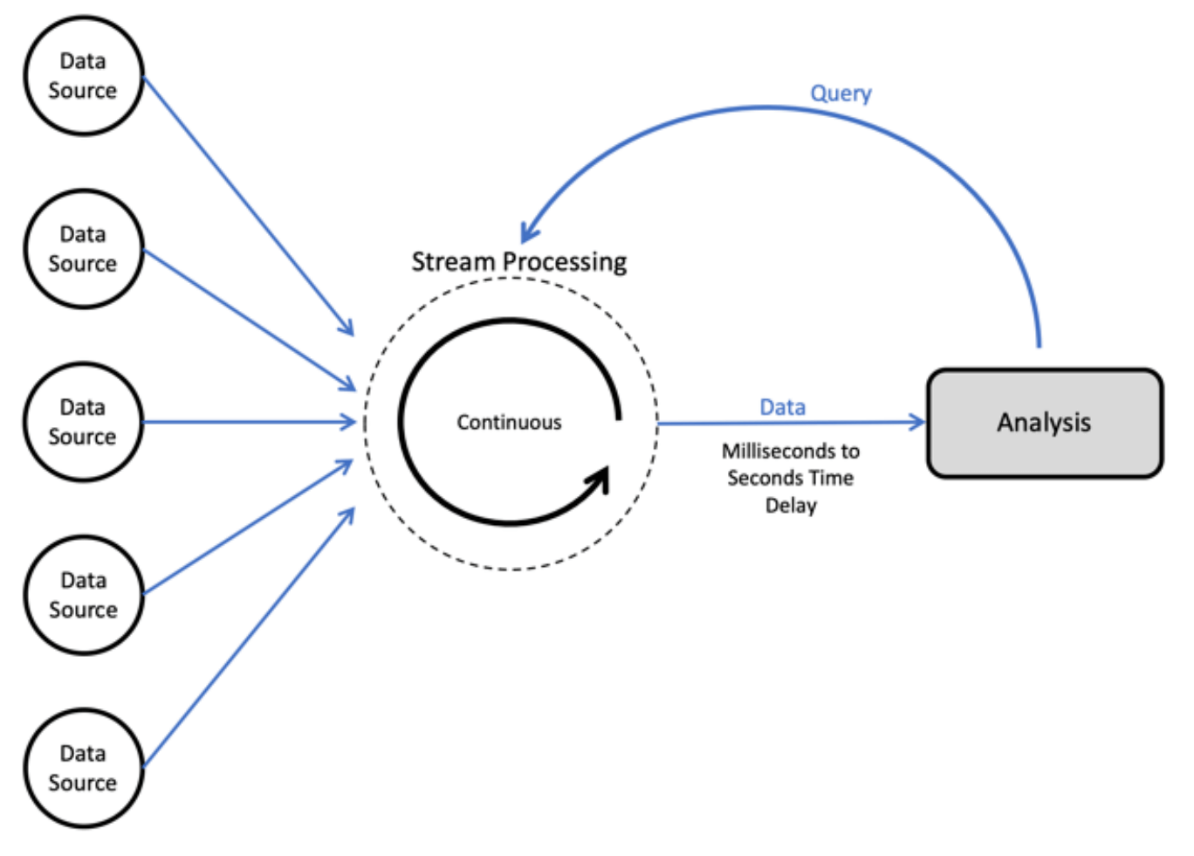
## Xử lý luồng dữ liệu theo thời gian thực (real-time datastream processing)

### Khái niệm

Khác với cách xử lý theo lô truyền thống, phương án xử lý luồng dữ liệu theo thời gian thực nhắm đến việc tiếp nhận dữ liệu ngay khi nó được sinh ra, lưu vào bộ nhớ rồi tích hợp thêm các công cụ xử lý dữ liệu tùy theo từng usecase cụ thể.

Đặc điểm của luồng dữ liệu là thiếu tính cấu trúc, và thường có khối lượng lớn (có thể lên đến hàng Terabytes), do đó dữ liệu đòi hỏi một quá trình tiền xử lý và ETL hợp lý để có thể sử dụng được.

Trên thực tế, khó có thể chỉ sử dụng một cơ sở dữ liệu hay một công cụ ETL đơn lẻ mà có thể giải quyết được bài toán xử lý luồng dữ liệu thời gian thực. Ta sẽ cần xây dựng cả một hệ thống với nhiều khối, mỗi khối đảm nhận một nhiệm vụ chuyên biệt.



Hình : Sơ đồ phương pháp xử lý dữ liệu theo luồng

### Đặc điểm

So với cách xử lý theo lô truyền thống, phương án xử lý luồng dữ liệu theo thời gian thực có một số lợi thế:

* Có khả năng làm việc với luồng sự kiện không có kết thúc.
* Có khả năng xử lý dữ liệu theo thời gian thực
* Nhận biết được mô hình trong dữ liệu kiểu chuỗi thời gian
* Dễ dàng mở rộng.

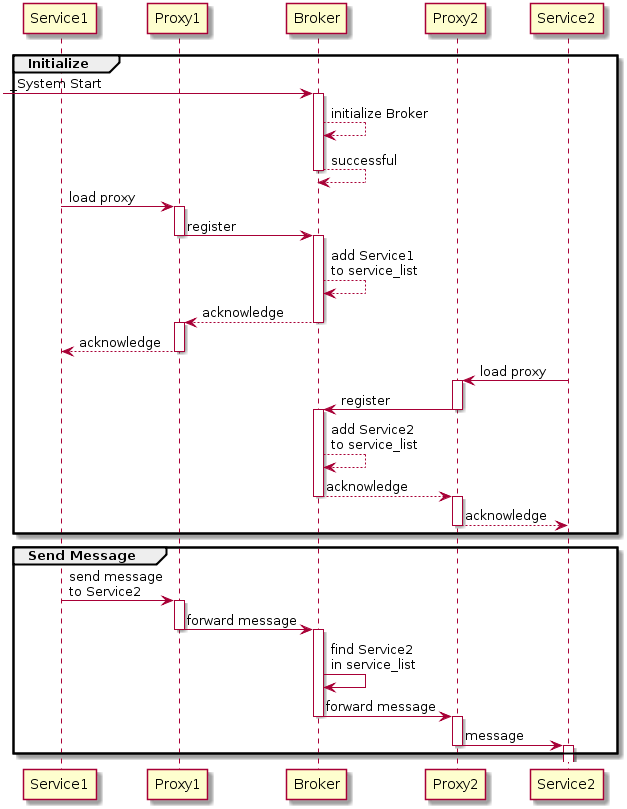
### Các thành phần của hệ thống xử lý luồng dữ liệu

#### Message broker

Đây là thành phần nhận dữ liệu từ nguồn (gọi là producer), biến đổi nó thành một dạng thông điệp chuẩn, rồi thực hiện stream ra ngoài tới các nền tảng khác. Những thành phần lắng nghe message broker và sử dụng các thông điệp được gửi đi từ message broker được gọi là các consumer.

Message broker có thể xác nhận, lưu trữ, định tuyến và gửi thông điệp đến các đích đến thích hợp. Nó đóng vai trò trung gian giữa các ứng dụng, cho phép các ứng dụng producer, consumer hoạt động bình thường mà không cần quan tâm đến các ứng dụng khác. Từ đó, tăng tính linh hoạt và tăng khả năng mở rộng của toàn bộ hệ thống.

Các message broker thường có khả năng đạt được thông lượng cao (đến hàng triệu thông điệp mỗi giây) chỉ với một lượng tài nguyên hạn chế, rất phù hợp với các trường hợp xử lý luồng dữ liệu.



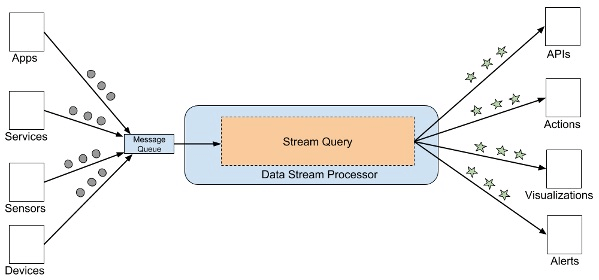
Hình 3: Sơ đồ tuần tự minh họa message broker

Một số nền tảng message broker phổ biến hiện nay có thể kể đến như RabbitMQ, Apache Kafka,…

#### Các công cụ ETL

Luồng dữ liệu từ các message broker cần được tổng hợp, biến đổi hoặc tái cấu trúc lại trước khi những dữ liệu đó được sử dụng bởi các ứng dụng thông thường khác.

Công việc này được thực hiện bởi một công cụ hoặc nền tảng ETL. Khi nhận một truy vấn từ người dùng, nó tìm nạp các sự kiện từ hàng đợi thông điệp rồi áp dụng vào truy vấn, sử dụng các phép join, hoặc các phép biến đổi trên dữ liệu để tạo ra một kết quả, có thể lại một tác vụ cụ thể hoặc cũng có thể là sinh ra một luồng dữ liệu khác.



Hình 4: Vị trí của các công cụ ETL trong hệ thống xử lý dữ liệu thời gian thực

#### Các thành phần khác:

* Data analytics engine: Sau khi dữ liệu được biến đổi để có thể sử dụng được, cần có các thành phần phân tích, đánh giá hoặc visualize dữ liệu để có thể sử dụng dữ liệu ở một mức độ chuyên sâu hơn.
* Bộ lưu trữ dữ liệu: Dữ liệu sau khi được biến đổi hoặc phân tích cần được lưu lại trên một bộ nhớ dài hạn. Đó có thể là các database, …

# CƠ SỞ CÔNG NGHỆ

## Apache Kafka

### Giới thiệu về Kafka

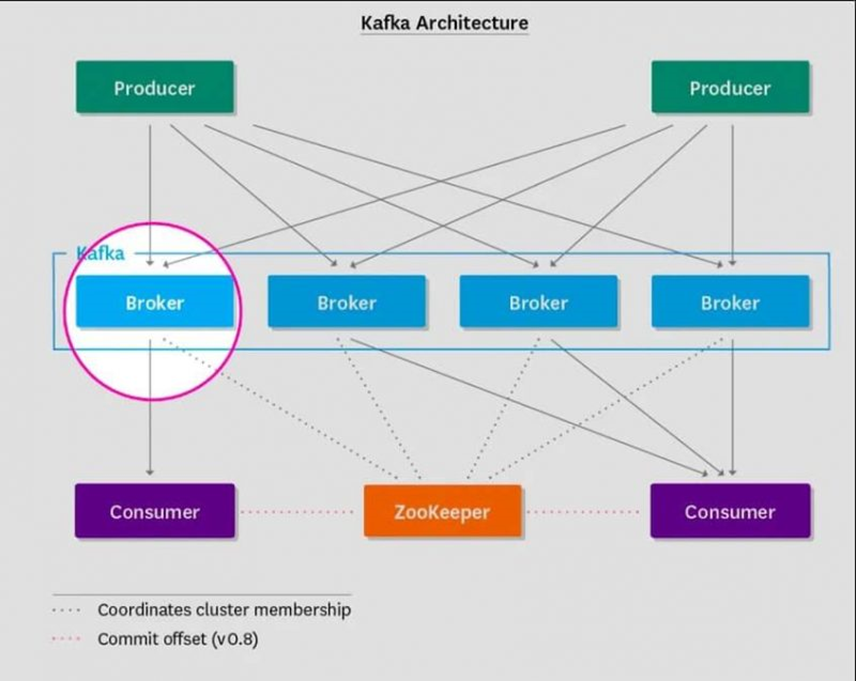


Apache Kafka là một nền tảng streaming trực tuyến mã nguồn mở được phát triển bởi Apache Software Foundation. Đầu tiên kafka đã được phát triển dưới dạng một ứng dụng hàng đợi nhắn tin tại LinkedIn. Dần dần, trong quá trình phát triển, nhiều chức năng khác đã được tích hợp vào Kafka. Ngày nay, Kafka đã không đơn thuần chỉ còn là một ứng dụng hàng đợi tin nhắn mà đã trở thành một nền tảng mạnh mẽ cho việc xử lý dữ liệu streaming, với các usecase đa dạng.

Một số đặc điểm của Kafka:

* Nó có khả năng mở rộng cực kỳ dễ dàng. Để mở rộng, chúng ta chỉ cần thêm một vài nút vào cụm Kafka hiện tại.
* Kafka còn có khả năng xử lý một lượng lớn dữ liệu trong một đơn vị thời gian với độ trễ tương đối thấp.
* Kafka được viết bằng Java và Scala, tuy nhiên nó vẫn có thể tương thích với các ngôn ngữ lập trình khác.
* So với các ứng dụng message broker truyền thống khác như RabbitMQ, Kafka có một số yếu tố khác biệt: Kafka duy trì các thông điệp trong một khoảng thời gian (mặc định là 7 ngày) sau khi thông điệp được tiếp nhận bới các consumer. Trong khi RabbitMQ xóa thông điệp ngay khi nó nhận được xác nhận từ phía consumer. Do đó dẫn đến một điểm khác biệt nữa là trong khi RabbitMQ chủ động gửi thông điệp đến cho Consumer kết hợp với việc theo dõi consumer thì Kafka lưu lại các thông điệp để các consumer tự lấy về.

### Hoạt động



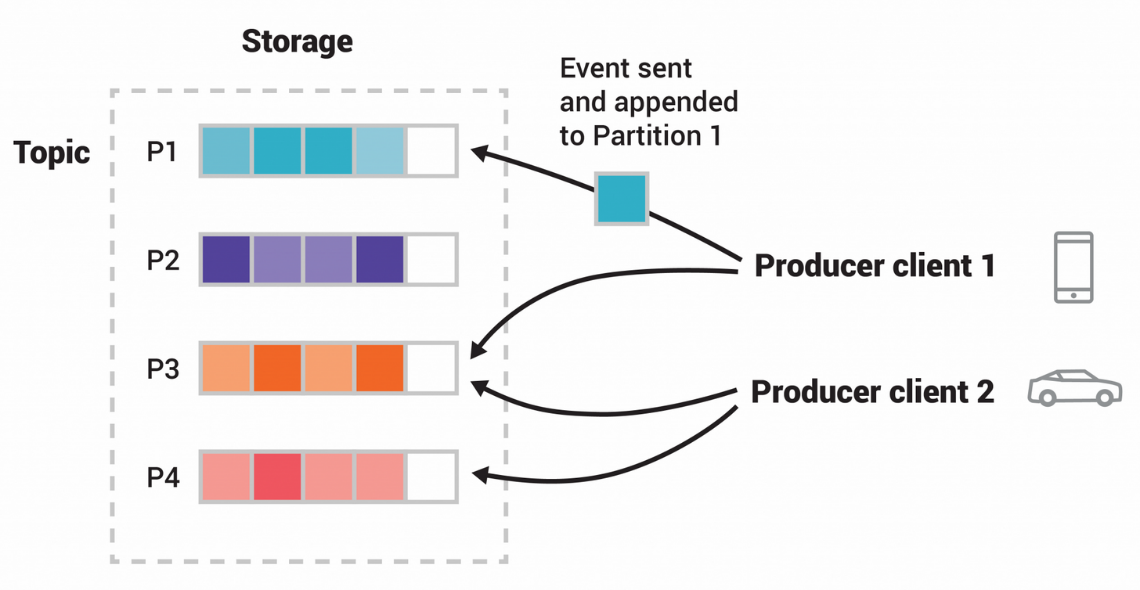
Hình : Sơ đồ cấu trúc Kafka

#### Producer và consumer

Kafka được xây dựng dựa trên mô hình publish/subcribe, tương tự như bất kỳ hệ thống message nào khác. Các ứng dụng (đóng vai trò là producer) gửi các thông điệp (message, records) tới một node kafka (broker) và nói rằng những thông điệp này sẽ được xử lý bởi các ứng dụng khác gọi là consumers. Các thông điệp được gửi tới kafka node sẽ được lưu trữ trong một nơi gọi là topic và sau đó consumer có thể đăng ký tới topic đó và lắng nghe những thông điệp này. Thông điệp có thể là bất cứ thông tin gì như giá trị cảm biến, hành động người dùng,…

#### Topic

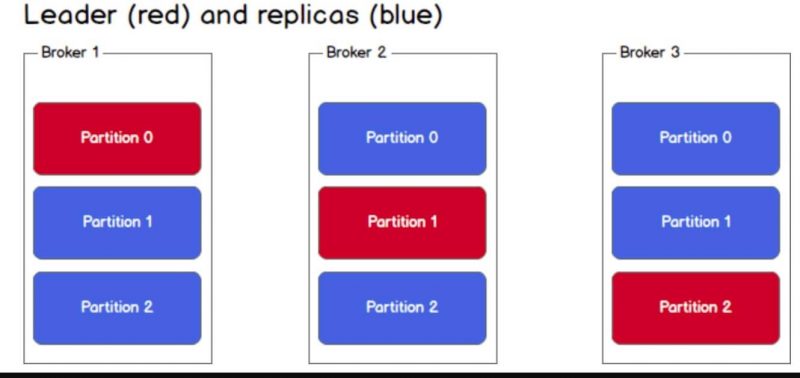
Topic có thể được xem như là tên của một danh mục mà các thông điệp sẽ được lưu trữ và được đẩy vào.



Hình : Minh họa topic và partition

Topics trong kafka có thể có kích cỡ rất lớn, như vậy không nên lưu trữ tất cả dữ liệu của một topic trên một node, dữ liệu cần đươc phân chia ra thành nhiều partition sẽ giúp bảo toàn dữ liệu cũng như xử lý dữ liệu dễ dàng hơn. Partitions cho phép chúng ta thực hiện subcribe song song tới một topic cụ thể bằng cách phân chia dữ liệu trong một topic cụ thể ra cho nhiều broker khác nhau (**kafka node**), mỗi partition có thể được đặt trên một máy riêng biệt – cho phép nhiều consumer đọc dữ liệu từ một topic diễn ra một cách song song. Các topic có cùng một event key sẽ được ghi vào cùng một partition và đảm bảo rằng bất kỳ consumer nào cũng có thể đọc được event theo đúng thứ tự nó được đẩy vào topic.

Để tăng tính khả dụng (**availability**) của partition, mỗi partition cũng có giá trị replicas của riêng nó, đó là số bản copy của mỗi partition. Trong số những bản copy partition này, sẽ có một bản copy được bầu chọn làm leader, trong khi những bản copy khác chỉ thực hiện đồng bộ dữ liệu với partition leader.



Hình : Minh họa replica của partition

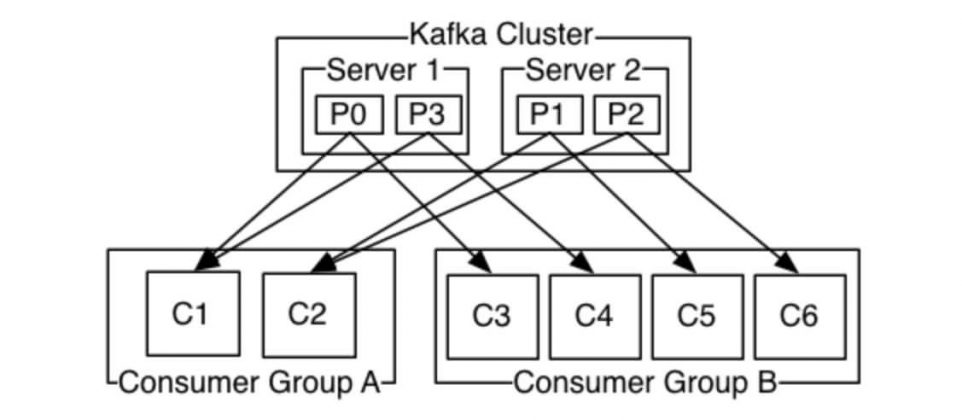
Tất cả các hành động ghi và đọc tới một topic sẽ đều phải đi qua partition leader tương ứng và leader sẽ phối hợp để cập nhật dữ liệu mới tới các replica parition khác. Nếu leader bị hỏng, một trong các replica partition sẽ đảm nhận vai trò là một leader mới.

Để consumer/producer biết được leader nằm ở partition nào thì thông tin về các partition cần được lưu lại. Kafka lưu trữ những thông tin như vậy là metadata trong một dịch vụ gọi là Zookeeper.

#### Consumer group

Consumer cũng được tổ chức thành các consumer groups cho một topic cụ thể – mỗi consumer bên trong group đọc message từ một partition duy nhất, để tránh việc có 2 consumer cùng xử lý đọc cùng một message 2 lần và toàn bộ group xử lý tất cả các message từ toàn bộ topic.

* Khi số consumer > số partition, khi đó một số consumer sẽ ở chế độ rảnh rỗi bởi vì chúng không có partition nào để xử lý.
* Khi số partition > số consumer, khi đó consumer sẽ nhận các message từ nhiều partition.
* Khi số consumer = số partition, mỗi consumer sẽ đọc message theo thứ tự từ 1 partition.



Hình : Minh họa consumer group

Kafka có cơ chế lưu giữ lại message trong một khoảng thời gian. Bản thân consumer sẽ thăm dò xem Kafa có message nào mới hay không và cho Kafka biết những record nào chúng muốn đọc. Điều này cho phép chúng tăng/giảm offset mà consumer muốn, do đó nó có thể đọc lại các message đã được đọc rồi và tái xử lý các sự kiện trong trường hợp gặp sự cố.

#### Kafka APIs

Kafka cũng cấp 5 API cốt lõi cho ngôn ngữ Java và Scala:

* Admin API: Quản lý và kiểm tra các topic, broker và các đối tượng Kafka khác.
* Producer API: đẩy một luồng sự kiện (event stream) cho một hoặc nhiều topic trong kafka
* Consumer API: Đăng ký một hoặc nhiều chủ đề và xử lý luồng sự kiện được tạo ra trong chủ đề.
* Kafka Streams API: Triển khai các ứng dụng xử lý luồng. API này cung cấp các chức năng cấp cao hơn để xử lý các luồng sự kiện, bao gồm các phép biến đổi, kết hợp, windowing, xử lý dựa trên thời gian sự kiện,… Đầu vào được đọc từ một hoặc nhiều topics và tạo đầu ra là một hoặc nhiều topic khác, chuyển đổi hiệu quả luồng đầu vào thành luồng đầu ra.
* Kafka Connect API: Xây dựng và chạy các trình kết nối nhập/xuất dữ liệu tới các hệ thống bên ngoài.

## Apache Flink

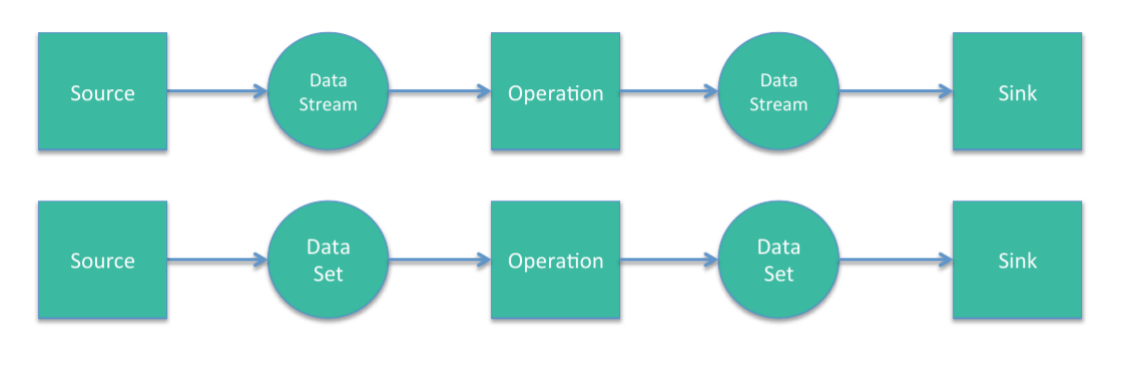
### Giới thiệu về Apache Flink

Apache Flink là một framework mã nguồn mở được sử dụng để xử lý dữ liệu phân tán trên quy mô lớn. Flink chủ yếu được sử dụng như một công cụ xử lý dữ liệu streaming nhưng cũng có thể được sử dụng như một công cụ xử lý theo lô. Flink được phát triển từ 2011 với ngôn ngữ lập trình Java và Scala. Hiện nay các thư viện API cho Flink đã có trên hầu hết các ngôn ngữ khác. Flink tích hợp với tất cả các trình quản lý tài nguyên cụm phổ biến như Hadoop YARN, … đồng thời cũng có thể hoạt động như một cụm độc lập.

Các tính năng chính của Flink:

* Có thể xử lý cả dữ liệu streaming và dữ liệu theo lô.
* Nó có thể xử lý dữ liệu với tốc độ rất nhanh.
* Có các API cho Java, Scala, Python, …
* Cung cấp API cho các operator phổ biến và rất dễ dàng cho các lập trình viên sử dụng.
* Khả năng chịu lỗi cao, nếu một nút, một ứng dùng hoặc phần cứng bị lỗi nó không ảnh hưởng đến cụm.
* Có thể dễ dàng tích hợp với Apache Hadoop, Apache MapReduce, Apache Spark, HBase và các công cụ big data khác.
* Quản lý in-memory có thể tùy chỉnh để tính toán tốt hơn.
* Có khả năng mở rộng (scalable) cao, có thể mở rộng tới hàng nghìn node trong một cụm.

### Luồng hoạt động



Hình : Sơ đồ luồng hoạt động của Flink

Một chương trình Flink bao gồm 3 giai đoạn:

* Nhập dữ liệu từ nguồn. Các nguồn dữ liệu có thể là:
  + Collections
  + Files (TextInputFormat, CSVInputFormat)
  + SocketInputFormat
  + KafkaInputFormat
  + …
* Thực hiện các thao tác trên dữ liệu. Các thao tác phổ biến là:
  + Map: Biến đổi một phần tử thành một phần tử khác
  + FlatMap: Lấy một phần tử và biến đổi thành một tập hợp phần tử khác.
  + Filter: Lọc lấy các phần tử thỏa mãn một hàm kiểm tra nào đó.
  + Reduce: Gộp một nhóm các phần tử thành một phần tử duy nhất bằng cách lặp đi lặp lại việc gộp 2 phần tử thành 1. Reduce có thể áp dụng trên toàn bộ tập dữ liệu hoặc trên một nhóm nhỏ của tập dữ liệu.
  + Cross: Tích đề các 2 tập dữ liệu, tạo thành một cặp phần tử mới
  + Union: Giao 2 tập dữ liệu.
  + …
* Xuất dữ liệu. Đích đến có thể là:
  + File
  + SocketOutputFormat
  + KafkaOutputFormat
  + Database
  + …

## Các công nghệ khác

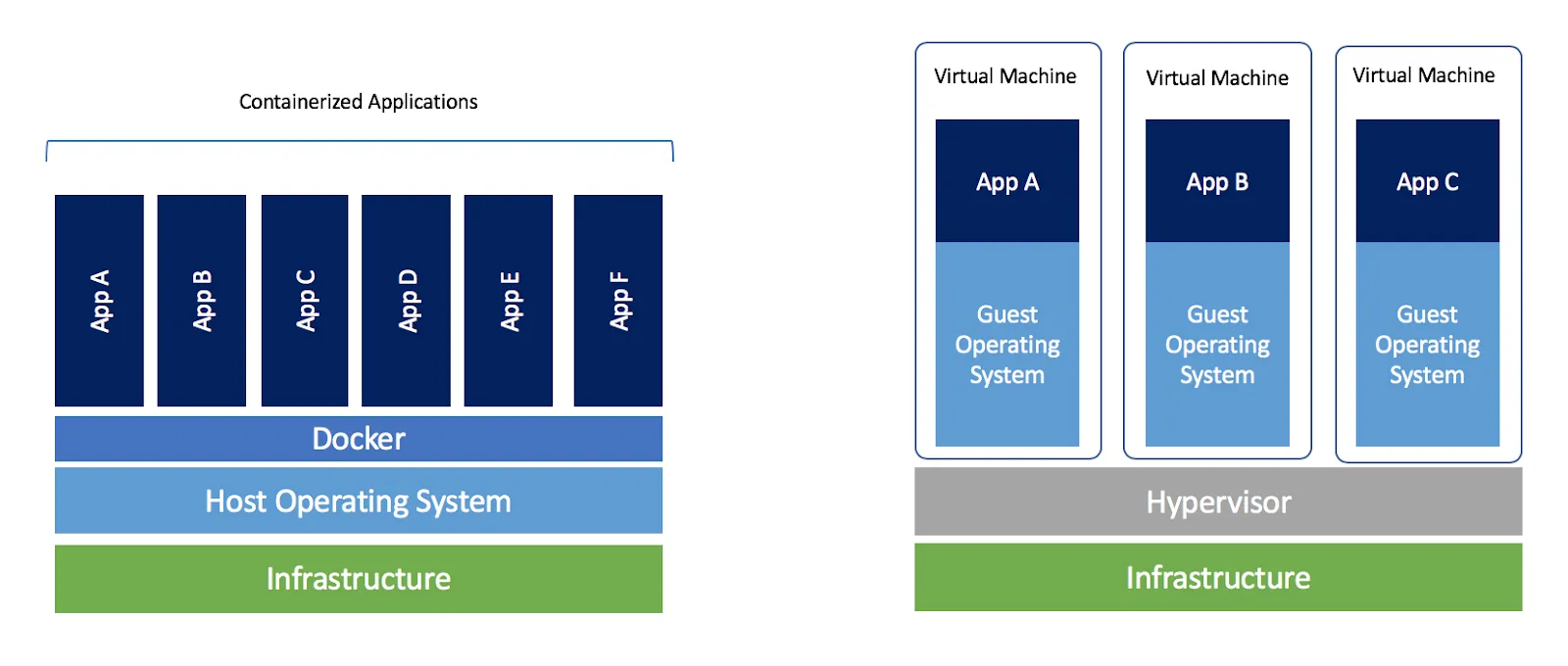
Ngoài Kafka và Flink, em còn sử dụng các công nghệ sau đây để hoàn thành đề tài thực tập.

### Docker

Docker là một nền tảng để cung cấp cách để building, deploying và running ứng dụng dễ dàng hơn bằng cách sử dụng các containers (trên nền tảng ảo hóa).

Các containers cho phép lập trình viên đóng gói một ứng dụng với tất cả các phần cần thiết, chẳng hạn như thư viện và các phụ thuộc khác, và gói tất cả ra dưới dạng một package.

Bằng cách đó, nhờ vào container, ta có thể giả lập một hệ thống các máy ảo với một chi phí nhỏ hơn nhiều so với sử dụng Virtual Machine.



Hình : Minh họa cơ chế hoạt động của Docker

### MongoDB

MongoDB là một database hướng tài liệu (document), một dạng NoSQL database. Vì thế, MongoDB sẽ tránh cấu trúc table-based của cơ sở dữ liệu quan hệ để thích ứng với các tài liệu như JSON có một schema rất linh hoạt gọi là BSON. MongoDB sử dụng lưu trữ dữ liệu dưới dạng Document JSON nên mỗi một collection sẽ các các kích cỡ và các document khác nhau. Các dữ liệu được lưu trữ trong document kiểu JSON nên truy vấn sẽ rất nhanh.



Hình : MongoDB

### NodeJS

NodeJS là một nền tảng được xây dựng trên V8 JavaScript Engine – trình thông dịch thực thi mã JavaScript, giúp xây dựng các ứng dụng web một cách đơn giản và dễ dàng mở rộng.

NodeJS được viết bằng JavaScript với cộng đồng người dùng lớn mạnh. NodeJS cũng có tốc độ xử lý nhanh nhờ cơ chế xử lý bất đồng độ (non-blocking), NodeJS có thể xử lý hàng ngàn kết nối cùng lúc mà không gặp bất cứ khó khăn nào.

Icon

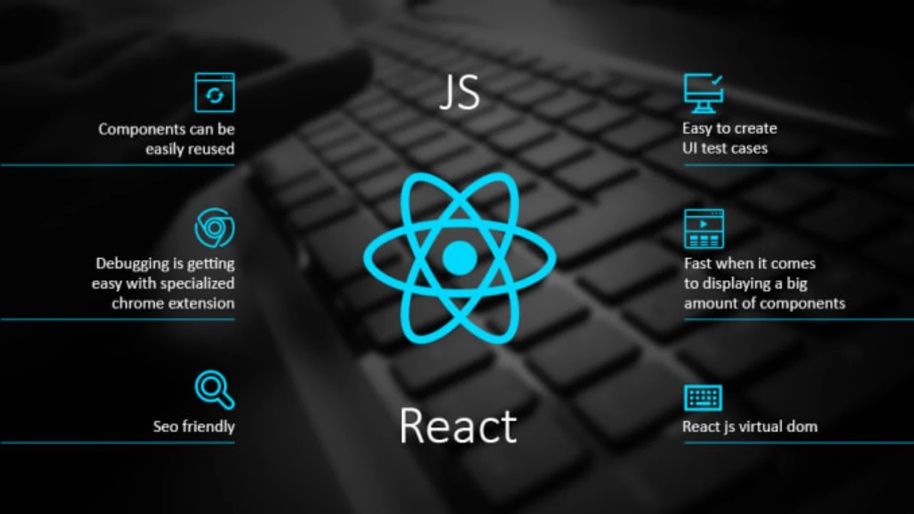
Description automatically generated

Hình : NodeJS

### ReactJS

Reactjs là một thư viện Javascript mã nguồn mở hỗ trợ xây dựng các thành phần giao diện nhanh gọn và tiện lợi. Khác với HTML truyền thống, Reactjs làm việc như một thư viện cho phép nhúng HTML vào javascript thông qua JSX. Qua đó ta có thể dễ dàng lồng các đoạn HTML vào trong JSX làm cho các component dễ hiểu và dễ sử dụng hơn.

Ngoài ra, ReactJS khuyến khích lập trình viên chia nhỏ các thành phần trên giao diện web thành từng component riêng biệt, giúp tăng tính tái sử dụng code cũng như giúp tăng tốc độ phát triển ứng dụng web.



Hình : Các tính năng của ReactJS

### Elasticsearch & Kibana

Elasticsearch là một search engine, được kế thừa từ Lucene Apache, thực chất hoặt động như 1 web server, có khả năng tìm kiếm nhanh chóng (near realtime) thông qua giao thức RESTful. Elasticsearch có khả năng phân tích và thống kê dữ liệu.

Kibana là một nền tảng phân tích hiển thị dữ liệu từ Elasticsearch một cách trực quan dễ sử dụng, Mặc định Kibana cung cấp nhiều tùy chọn để visualize cũng như phân tích dữ liệu.

Graphical user interface, application

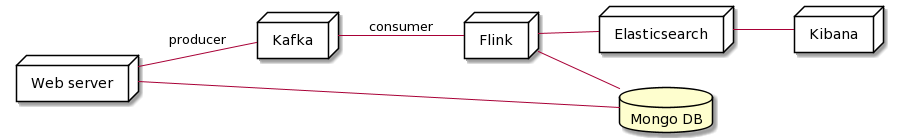
Description automatically generated

Hình : Một số dạng biểu đồ được hỗ trợ bởi Kibana

# CÀI ĐẶT THỰC NGHIỆM

## Tổng quan

Ứng dụng một trong những usecase của datastream đó là theo dõi và phân tích hành vi người dùng. Cụ thể, em tạo ra một trang web danh sách sản phẩm và ghi nhận các hành động hover vào sản phẩm của người dùng. Sau đó, các hành động này được phân tích và visualize để phục vụ cho quá trình đánh giá.



Hình 15: Sơ đồ hệ thống demo

Hệ thống gồm các thành phần sau:

* Cơ sở dữ liệu MongoDB: lưu trữ thông tin về người dùng, sản phẩm,…
* Web server: Ứng dụng web tương tác với người dùng, ghi nhận hành vi người dùng. Đóng vai trò producer để gửi các hành vi đó tới Kafka
* Kafka: message broker
* Flink: Đóng vai trò consumer, biến đổi dữ liệu bằng cách truy cập vào database để enrich data.
* Elasticsearch & Kibana: Chứa dữ liệu và visualize dữ liệu.

## Cài đặt môi trường với docker

### Build docker image cho flink consumer

FROM openjdk:8u151-jdk-alpine3.7

# Install Bash

RUN apk add --no-cache bash libc6-compat

# Copy resources

WORKDIR /

COPY wait-for-it.sh wait-for-it.sh

COPY target/kafka-spark-flink-example-1.0-SNAPSHOT-jar-with-dependencies.jar kafka-spark-flink-example.jar

COPY startup.sh startup.sh

# Wait for Zookeeper, Kafka, MongoDB and

# elasticsearch to be available and run application

CMD ./startup.sh

Chạy lệnh build, gắn tag cho image này là kafka-flink:

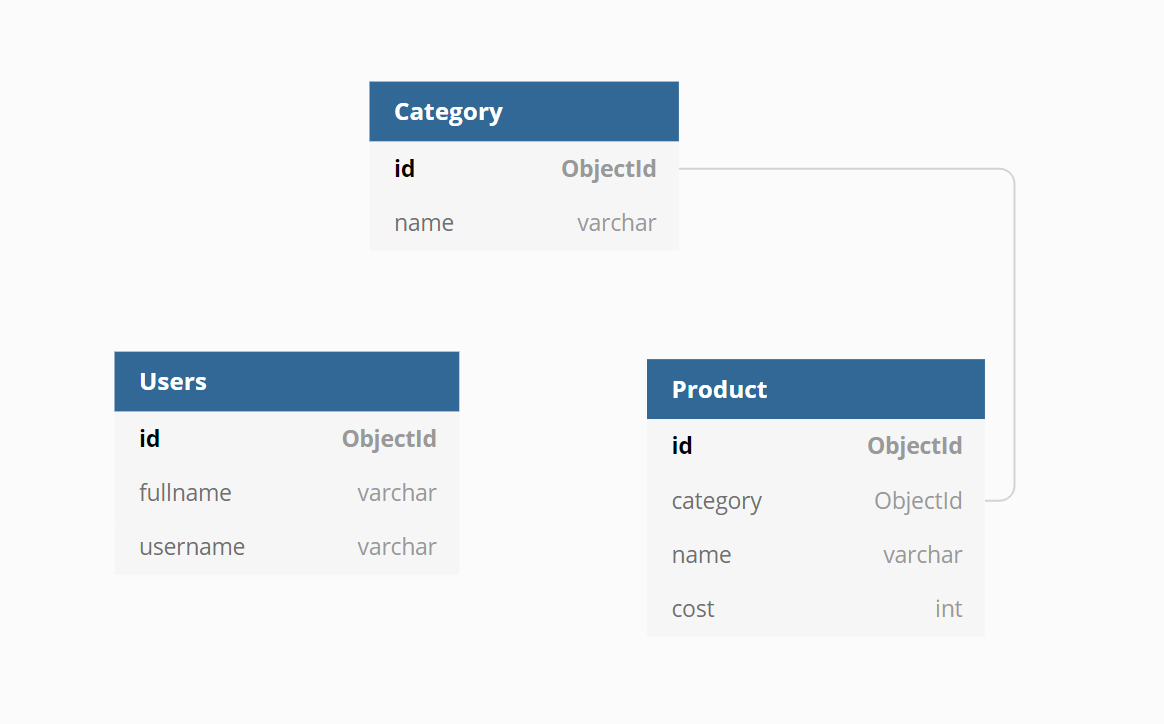
$ docker build -t kafka-flink .

### Cấu hình docker-compose cho các docker container

|  |
| --- |
| version: '3.6'  networks:    bridge:      driver: bridge  services:    # Cấu hình cho database    # Port 27017    mongo:      image: mongo      restart: always      ports:        - 27017:27017      volumes:        - ./collections:/data/db/      environment:        MONGO\_INITDB\_ROOT\_USERNAME: root        MONGO\_INITDB\_ROOT\_PASSWORD: incorrect      networks:        bridge:          aliases:            - mongo      # Giao diện web truy cập mongodb    # Port 27018    mongo-express:      image: mongo-express      depends\_on:        - mongo      restart: always      ports:        - 27018:8081      environment:        ME\_CONFIG\_MONGODB\_ADMINUSERNAME: root        ME\_CONFIG\_MONGODB\_ADMINPASSWORD: incorrect    # Cấu hình cho zookeeper    zookeeper:      image: confluentinc/cp-zookeeper:latest      environment:        ZOOKEEPER\_CLIENT\_PORT: 32181        ZOOKEEPER\_TICK\_TIME: 2000      networks:        bridge:          aliases:            - zookeeper      # Cấu hình cho kafka    # Listen cho các docker container khác tại địa chỉ kafka:9092    # Listen cho các ứng dụng bên ngoài docker (ví dụ dùng khi dev) tại localhost:29092    kafka:      image: wurstmeister/kafka:latest      ports:        - 29092:29092        - 32181:32181      depends\_on:        - zookeeper      environment:        KAFKA\_BROKER\_ID: 1        KAFKA\_ADVERTISED\_HOST\_NAME: 0.0.0.0        KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:32181        KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1        JMX\_PORT: 9999        KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT        KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT\_HOST://localhost:29092        KAFKA\_LISTENERS: PLAINTEXT://:9092,PLAINTEXT\_HOST://:29092        KAFKA\_INTER\_BROKER\_LISTENER\_NAME: PLAINTEXT      networks:        bridge:          aliases:            - kafka    # Giao diện web quản lý cho kafka    kafka-manager:      image: sheepkiller/kafka-manager:latest      environment:        ZK\_HOSTS: "zookeeper:32181"        JMX\_PORT: 9999      ports:        - 9000:9000      networks:        - bridge    # Kafka consumer flink    # Sử dụng docker image vừa được build    kafka-consumer-flink:          image: kafka-flink          depends\_on:            - kafka            - elasticsearch          environment:            EXAMPLE\_GOAL: "consumer.flink"            EXAMPLE\_KAFKA\_TOPIC: "example"            EXAMPLE\_KAFKA\_SERVER: "kafka:9092"            EXAMPLE\_ZOOKEEPER\_SERVER: "zookeeper:32181"            EXAMPLE\_ELASTIC\_SERVER: "elasticsearch"            EXAMPLE\_MONGODB\_SERVER: "mongo"          networks:            - bridge    # Index data    elasticsearch:          image: docker.elastic.co/elasticsearch/elasticsearch:7.6.0          environment:            - cluster.name=docker-cluster            - bootstrap.memory\_lock=true            - "ES\_JAVA\_OPTS=-Xms512m -Xmx512m"            - discovery.type=single-node          ports:            - "9200:9200"            - "9300:9300"          ulimits:            memlock:              soft: -1              hard: -1            nofile:              soft: 65536              hard: 65536          networks:            bridge:              aliases:                - elasticsearch    # Visualize data    kibana:          image: docker.elastic.co/kibana/kibana:7.6.0          ports:            - "5601:5601"          networks:            - bridge |

## Cài đặt MongoDB

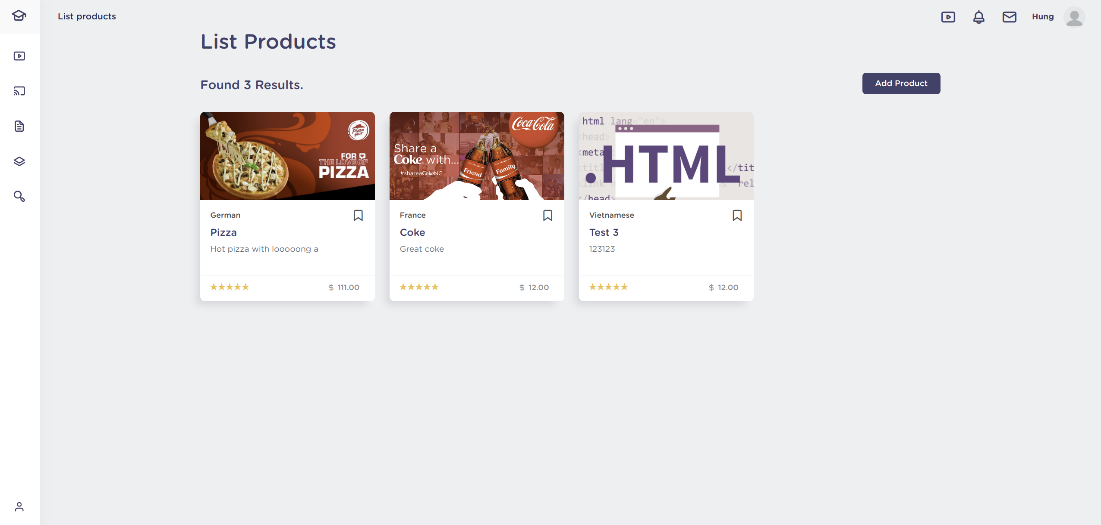
Tạo các collection sau trong MongoDB:



Hình : Sơ đồ thiết kế cơ sở dữ liệu

## Xây dựng ứng dụng web với vai trò producer

Hiện tại, ứng dụng web cho người dùng được xây dựng với một usecase cơ bản là liệt kê danh sách sản phẩm. Tại đây, ta sẽ theo dõi xem vị trí chuột của user hover vào sản phẩm nào với thời gian bao lâu, bằng cách bắt sự kiện mouseenter và mouseleave từ phía người dùng rồi gửi lên web server.



Hình 17: Giao diện web cho người dùng

Phía web server đóng vai trò là producer, với kết nối như sau:

|  |
| --- |
| const init = () => {    client = new kafka.KafkaClient({      kafkaHost: 'localhost:29092',    });    producer = new Producer(client);    const topicsToCreate = [{      topic: 'example',      partitions: 1,      replicationFactor: 1,    }];    producer.on('ready', function() {      console.log('Producer is ready');      producer.createTopics(topicsToCreate, (err, result) => {      });    });  }; |

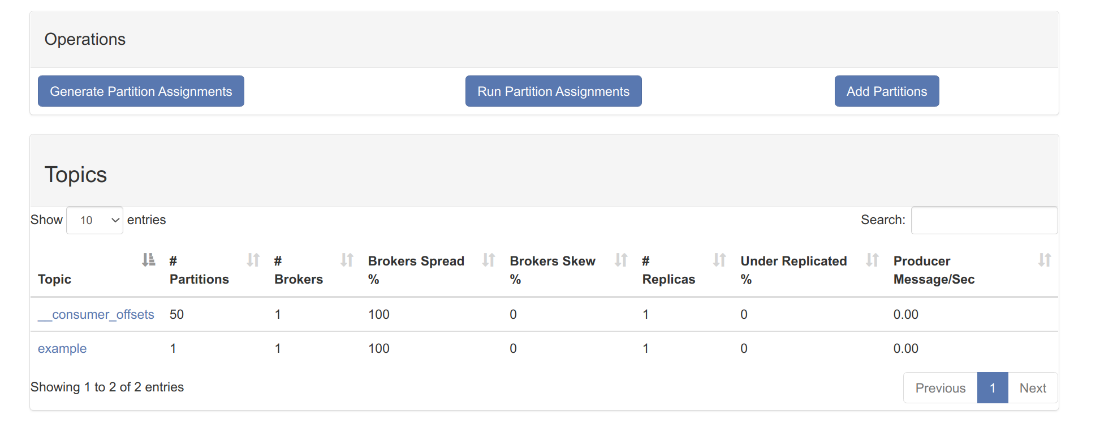
Producer khởi tạo topic Example với cấu hình số replication là 1, số partitions là 1.

Web server tiếp nhận request từ web client về sự kiện di chuyển chuột của người dùng rồi gửi thông điệp tới kafka với cấu trúc như sau:

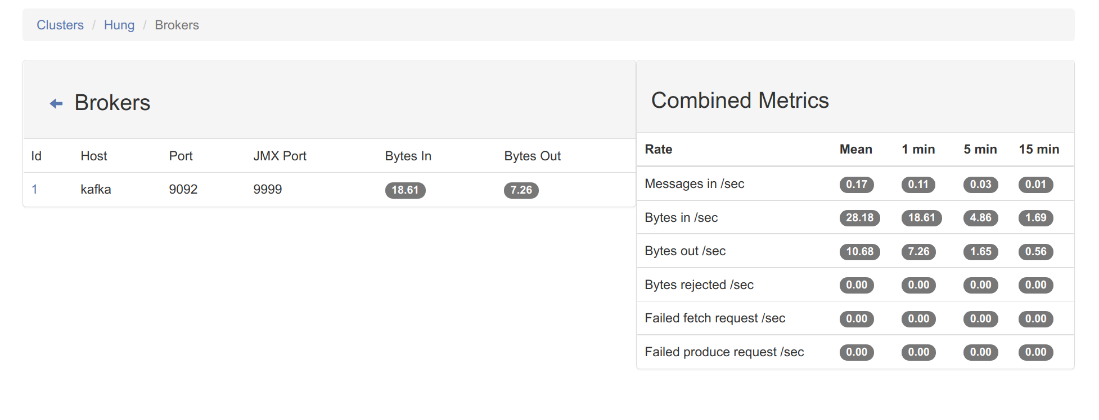
|  |
| --- |
| <UserId> <Timestamp> <ProductId> <HoverTime> |

## Cấu hình Apache Kafka

Cấu hình zookeeper server của kafka là zookeeper:32181. Sử dụng các cấu hình mặc định của kafka. Khi truy cập vào giao diện web của Kafka Manager ta thấy các thông tin như sau:



Hình 18: Danh sách các topic trên Kafka



Hình 19: Danh sách các Broker

## Lập trình ứng dụng Flink Consumer

* Khởi tạo kết nối tới Kafka broker:

|  |
| --- |
| // Create execution environment          StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();          // Properties          final Properties props = new Properties();          props.put(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, Commons.EXAMPLE\_KAFKA\_SERVER);          props.put(ConsumerConfig.GROUP\_ID\_CONFIG, "FlinkConsumerGroup");          DataStream<String> messageStream = env.addSource(new FlinkKafkaConsumer010<>(Commons.EXAMPLE\_KAFKA\_TOPIC, new SimpleStringSchema(), props)); |

* Sử dụng thao tác Map, biến đổi dữ liệu đầu vào từ dạng String thành dạng LogObject, thực hiện query từ database để enrich data tại bước này.

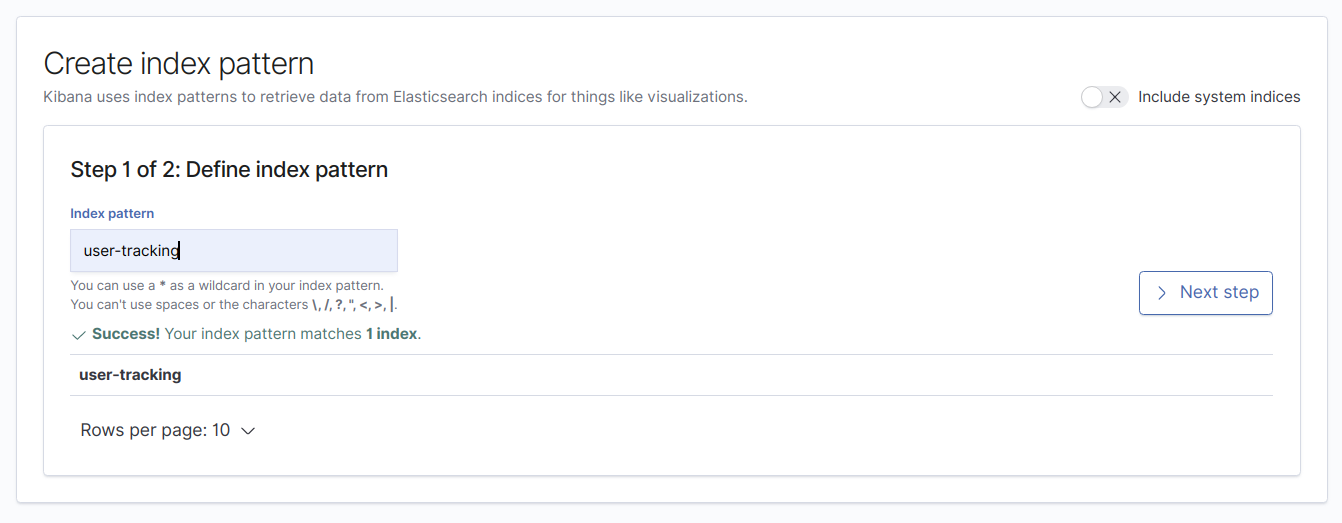
|  |
| --- |
| DataStream<LogObject> parsedStream = messageStream              .map(en -> new LogObject(en))              .filter(en -> !en.isEnter); |

* Thêm elasticsearch thành output source. Datastream sẽ được đẩy lên elasticsearch với index là “user-tracking”

|  |
| --- |
| List<HttpHost> httpHosts = new ArrayList<>();          httpHosts.add(new HttpHost(Commons.EXAMPLE\_ELASTIC\_SERVER, 9200, "http"));          // use a ElasticsearchSink.Builder to create an ElasticsearchSink          ElasticsearchSink.Builder<LogObject> esSinkBuilder = new ElasticsearchSink.Builder<>(              httpHosts,              new ElasticsearchSinkFunction<LogObject>() {                  public IndexRequest createIndexRequest(LogObject element) {                      Map<String, Object> json = new HashMap<>();                      json.put("productId", element.productId);                      json.put("userId", element.userId);                      json.put("timestamp", element.timestamp.toString());                      json.put("uptime", element.length);                      json.put("username", element.username);                      json.put("productName", element.productName);                      json.put("categoryId", element.categoryId);                      json.put("categoryName", element.categoryName);                      return Requests.indexRequest()                              .index("user-tracking")                              .type("my-type")                              .source(json);                  }                    @Override                  public void process(LogObject element, RuntimeContext ctx, RequestIndexer indexer) {                      indexer.add(createIndexRequest(element));                  }              }          );            esSinkBuilder.setBulkFlushMaxActions(1);          parsedStream.addSink(esSinkBuilder.build()); |

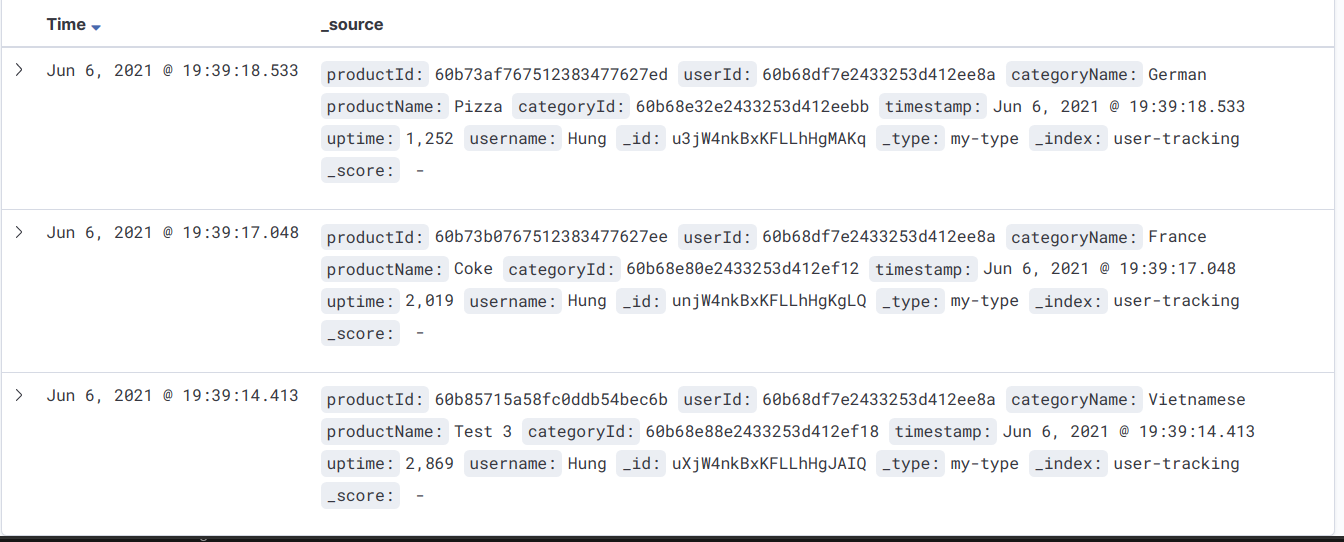
## Cấu hình Kibana để visualize dữ liệu

* Chọn index user-tracking



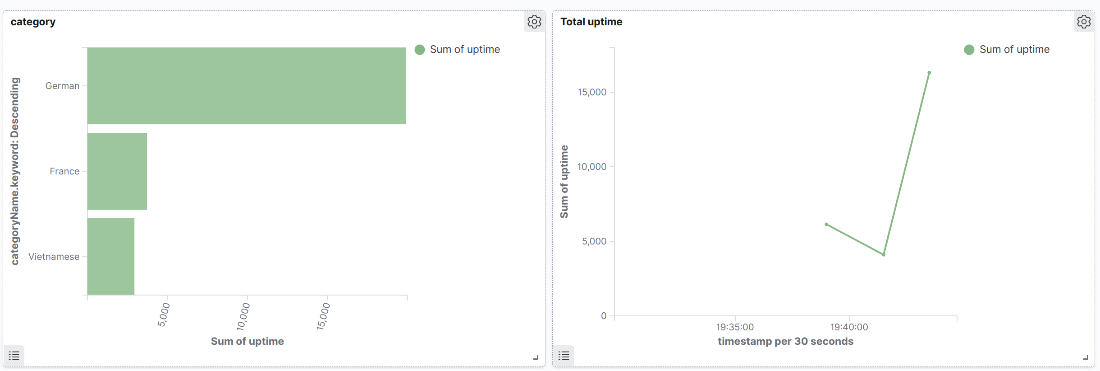
Hình 20: Chọn index user-tracking trong Kibana

Tại đây ta có thể quan sát thấy dữ liệu dã được enrich, thêm thông tin về các trường username, productname, categoryId, categoryName



Hình 21: Dữ liệu đã được enrich

Tại dashboard của Kibana, ta có thể thêm các biểu đồ để visualize dữ liệu.



Hình 22: Tạo biểu đồ tại dashboard

Ví dụ: Bên trái là biểu đồ so sánh thời gian hover vào sản phẩm theo category, bên phải là biểu đồ thể hiện tổng thời gian hover vào sản phẩm của user theo thời gian.

# KẾT LUẬN

## Kết quả đạt được

Một số kết quả sau quá trình làm đề tài:

* Tìm hiểu được cách dùng cơ bản của một số ứng dụng phân tích dữ liệu lớn như Kafka, Flink, Elasticsearch…
* Xây dựng và deploy thành công một hệ thống theo dõi phân tích hoạt động người dùng thông qua docker.

## Định hướng phát triển:

* Tích hợp thêm nhiều usecase hơn vào ứng dụng web cho user.
* Sử dụng thêm nhiều đánh giá phức tạp hơn
* Cài đặt các thành phần Kafka, Flink trên nhiều nút hơn.

# TÀI LIỆU THAM KHẢO

1. Todd P., Gwen S. & Neha N. (2017). *Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale*. O'Reilly Media.
2. Fabian H. & Vasiliki K. (2019). *Stream Processing with Apache Flink: Fundamentals, Implementation, and Operation of Streaming Applications*. O'Reilly Media.
3. Eran L. (2021). *4 Key Components of a Streaming Data Architecture.* <https://www.upsolver.com/blog/streaming-data-architecture-key-components>
4. Sibanjan D. (2020). *Apache Flink: The Next Gen Big Data Analytics Framework*. https://dzone.com/articles/apache-flink-the-4g-of-big-data