CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập – Tự do – Hạnh phúc

BÁO CÁO KẾT QUẢ CÔNG VIỆC

Từ 06/07/2024 đến 10/07/2024

Họ và tên	:	Trần Danh Linh
Chức vụ	:	Thực tập sinh
Bộ phận công tác	:	Trung tâm Dịch vụ Truyền hình

HÀ NỘI, 7/2024

Mục lục

I. Elastic-cluster

1. Nhiệm vụ của các node trong cluster

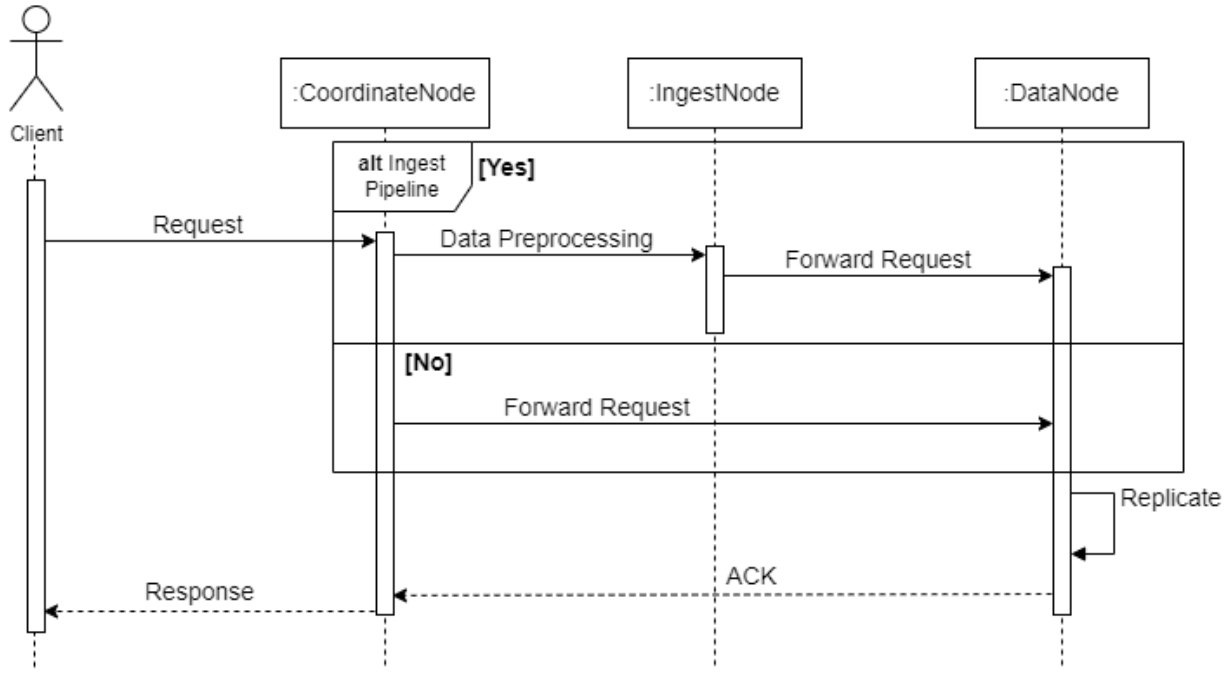
- Elastic lưu trữ và xử lý dữ liệu thông qua các cluster. Trong đó, mỗi cluster bao gồm các node, mỗi node thì sẽ bao gồm các shard. Trong đó, có 1 primary shard và có thể có nhiều replica shard tùy thuộc vào cài đặt ban đầu. Số lượng replica shard chỉ có thể được khởi tạo ngay khi tạo cluster, nếu không sẽ có thể phải đánh index lại tất cả dữ liệu từ đầu.
- Khi 1 node bị lỗi và primary shard bị hỏng thì 1 replica shard sẽ được chọn làm primary. Còn nếu replica shard bị hỏng thì primary sẽ chỉ việc tạo lại replica shard mới.

- Mỗi nút có một vai trò đặc biệt thực hiện các chức năng khác nhau, bao gồm:
    - Master Node: Chịu trách nhiệm thực hiện các tác vụ quản lý trên cluster như: thêm, xóa chỉ mục, theo dõi hoạt động của các node khác và quyết định phân mảnh nào sẽ được lưu trữ ở node nào.
    - Data Node: Chịu trách nhiệm lưu trữ và quản lý dữ liệu trong cluster. Chúng sẽ thực hiện xử lý các yêu cầu chỉ mục, lưu trữ các dữ liệu trong phân mảnh và thực hiện truy vấn dữ liệu. Mỗi data node lưu trữ các dữ liệu trong các phân mảnh chính và bản sao, phân phối dữ liệu trên toàn cluster để có khả năng mở rộng và chịu lỗi. Ngoài ra, có các vai trò riêng biệt cho data node như: Data Content, Data Hot, Data Warm, Data Cold, Data Frozen.
    - Ingest Node: Thực hiện việc tiền xử lý dữ liệu trước khi được đánh chỉ mục. Chúng giúp giảm tải các tác vụ xử lý cho data node, từ đó gia tang hiệu suất cho cluster.

- Coordinating-Only Node: Không lưu trữ dữ liệu hay tham gia vào quá trình bầu chọn master node. Nhiệm vụ của chúng là hoạt động như một proxy cho các yêu cầu của client, phân phối các yêu cầu tìm kiếm và lập chỉ mục đến cho các node thích hợp.
- Machine Learning Node: Thực hiện việc xử lý các yêu cầu Machine Learning API.
- Remote Eligible Node: Phục vụ cho các remote cluster, trong đó, các chỉ mục được sao chép liên cụm và được truy vấn liên cụm.
- Transform Node: Thực hiện xử lý các yêu cầu Transform API để chuyển đổi chỉ mục, cung cấp thông tin chi tiết và phân tích dữ liệu tóm tắt.

2. Cơ chế truy vấn/lưu dữ liệu

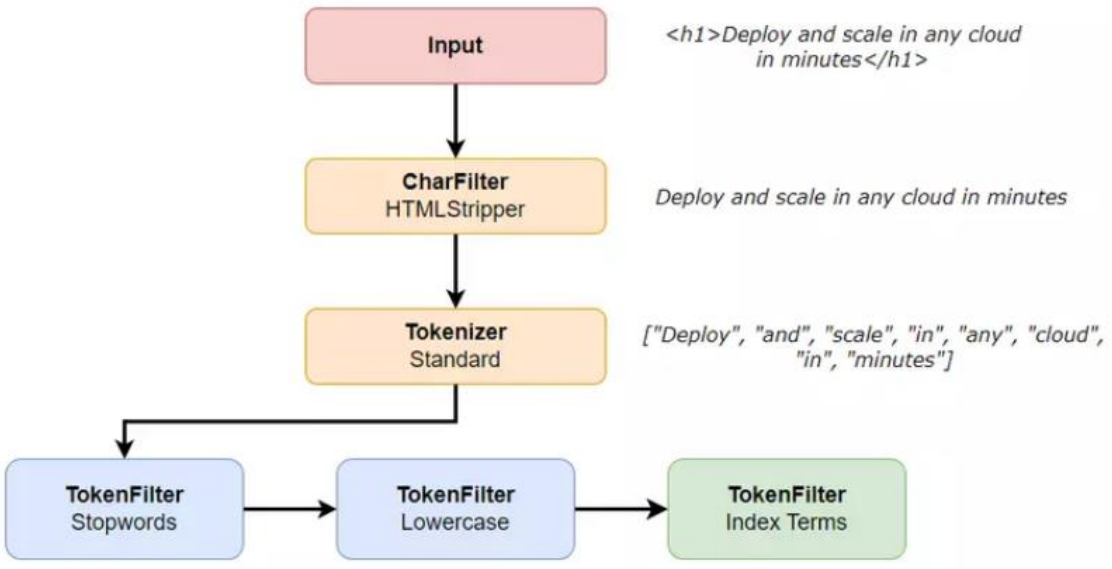
a. Cách thức hoạt động



Hình 1: Biểu đồ luồng hoạt động của Elastic Search

- Do master node đóng vai trò quản lý cluster nên không tham gia trực tiếp vào quá trình truyền, nhận dữ liệu với Client.
- Trong quá trình ghi, dữ liệu sẽ được ghi vào primary shard rồi primary shard sẽ tạo ra các bản sao của dữ liệu lên các replica shard.
- Trong quá trình đọc, dữ liệu có thể được đọc từ primary shard hoặc replica shard.
- Elasticsearch sử dụng cơ chế **Inverted Index** (Là một chỉ mục cơ sở dữ liệu lưu trữ ánh xạ từ nội dung, chẳng hạn như từ hoặc số, đến các vị trí của nó trong bảng hoặc trong tài liệu hoặc tập hợp tài liệu) để lưu trữ và truy vấn dữ liệu.

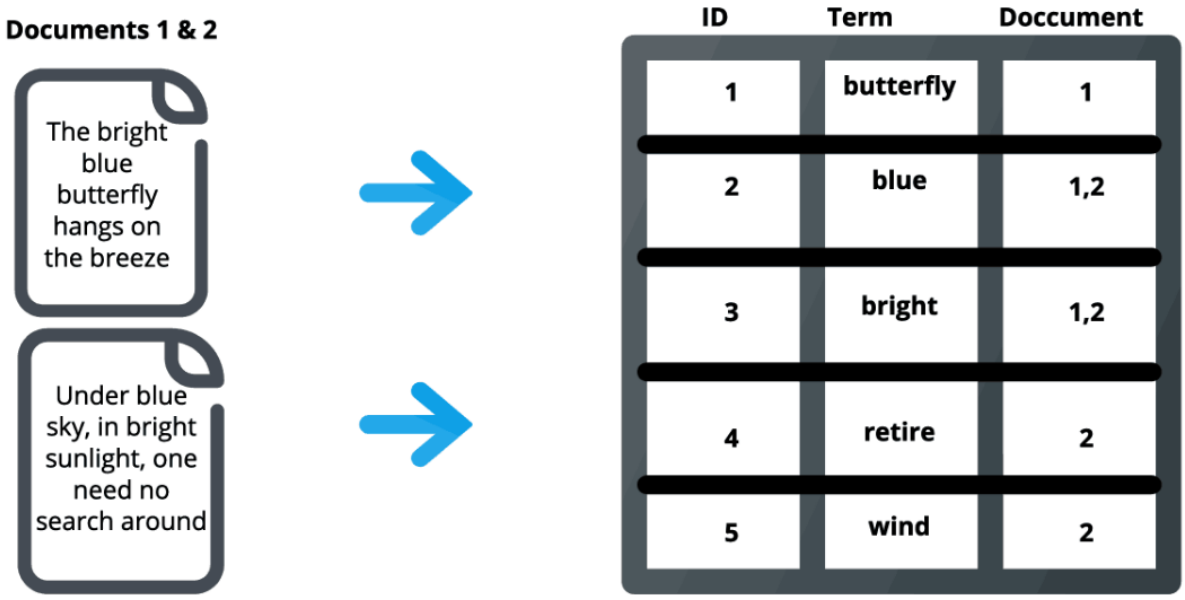
b. Cơ chế lưu dữ liệu:



Hình 2. Mô tả quá trình Analyzer

- Dữ liệu được gửi đến Elasticsearch dưới dạng JSON document.
- Elasticsearch sử dụng quy trình Analyzer để phân tích văn bản thành các term (từ khóa) riêng lẻ.
- Tokenizer trong Analyzer sẽ tách văn bản thành các term dựa trên bộ quy tắc.

- Các term được lưu trữ trong Inverted Index cùng với thông tin về document chứa term đó (document ID).
- Ví dụ các term được lưu trữ trong Inverted Index:



Hình 3: Ví dụ về term được lưu trữ trong Inverted Index

c. Cơ chế truy vấn dữ liệu:

1. Yêu cầu tìm kiếm (query) được gửi đến Elasticsearch, bao gồm các term cần tìm kiếm.
2. Elasticsearch sử dụng Inverted Index để tìm kiếm các document chứa các term đó.
3. Elasticsearch tính toán relevance score (điểm liên quan) cho mỗi document dựa trên số lần xuất hiện của term.
4. Elasticsearch trả về danh sách các document phù hợp nhất với yêu cầu tìm kiếm, được sắp xếp theo relevance score.

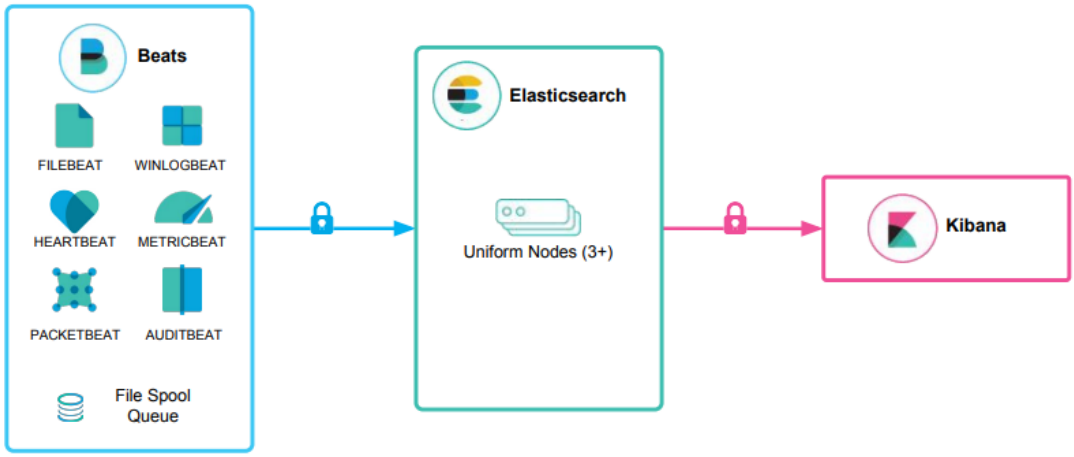
d. Các loại truy vấn:

Elasticsearch hỗ trợ nhiều loại truy vấn khác nhau, bao gồm:

- Match query:
    - Là truy vấn chuẩn để thực hiện full text query. Bao gồm truy vấn kết hợp và truy vấn cụm từ hoặc gần đúng.
    - Match query trả về các document chứa ít nhất 1 trong các từ trong truy vấn.
- Match Phrase Query: Tìm kiếm các document chứa cụm từ trong truy vấn.
- Match Phrase Prefix Query: Tìm kiếm các document khớp với tiền tố trong truy vấn.
- Multi Match Query: Tương tự như Match Query nhưng cho phép tìm kiếm trên nhiều trường.
- Fuzzy Search:
    - Cho phép tìm kiếm gần đúng, dựa trên khoảng cách Levenshtein.
    - Giúp người dùng dễ dàng tiếp cận được với nội dung hơn, khi mà họ có thể tìm thấy được những thứ cần thiết, ngay cả khi họ không nhớ được chính xác nội dung mình muốn tìm kiếm là gì.

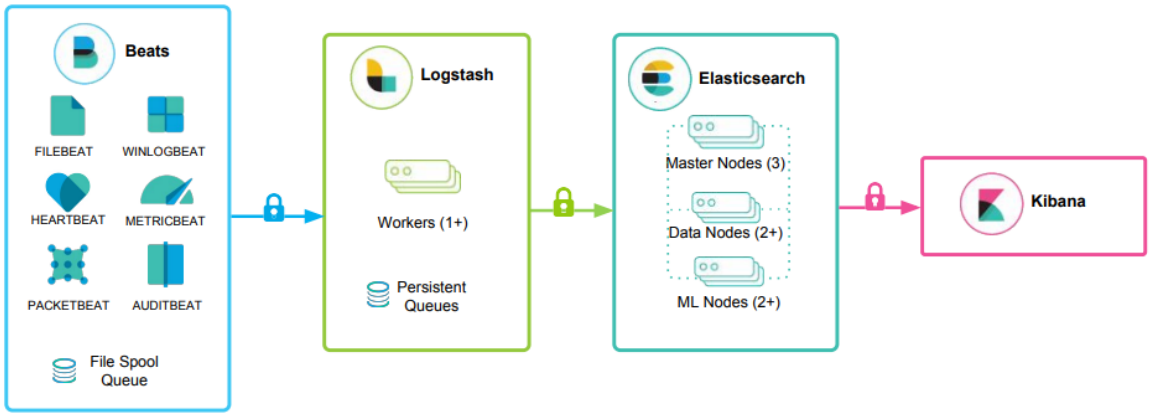
3. Các mô hình triển khai trong thực tế

- Mô hình cơ bản:



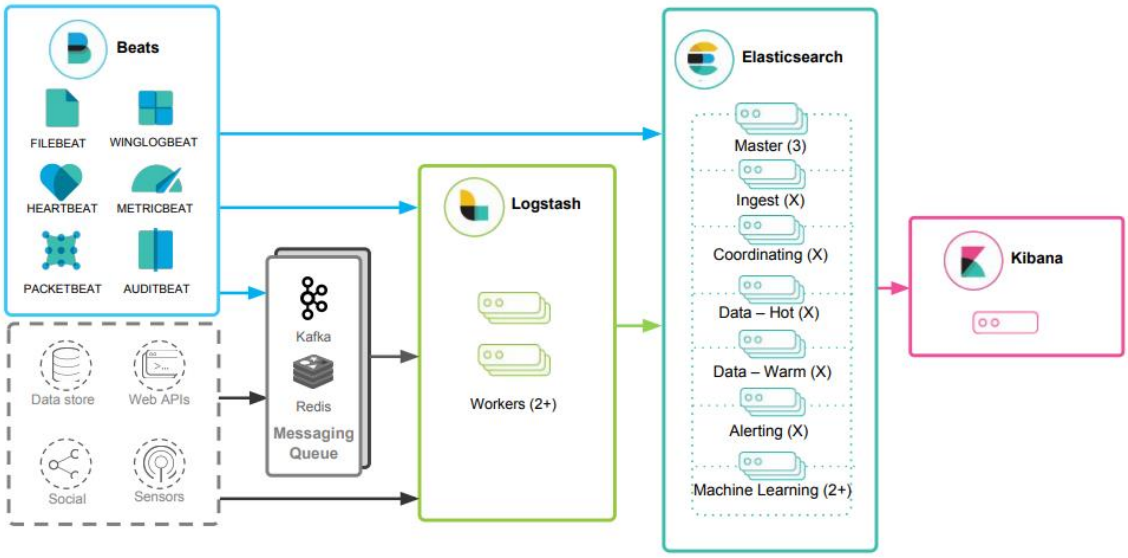
Hình 4: Mô hình triển khai Elapstic Search cơ bản

- Mô hình xử lý nâng cao và có khả năng phục hồi



Hình 5: Mô hình triển khai Elapstic nâng cao và có khả năng phục hồi

- Mô hình xử lý đầu vào linh hoạt



Hình 6: Mô hình triển khai Elapstic xử lý đầu vào linh hoạt

II. Redis-cluster

1. Nhiệm vụ các node trong mô hình cluster

- Mỗi Redis Cluster yêu cầu 2 cổng TCP được mở cho mỗi nút: một cổng cho clients và một cổng cho kết nối giữa các nút trong cụm.
- Redis Cluster sử dụng 16384 hash slot để phân bố dữ liệu trên các nút. Mỗi nút trong cụm chịu trách nhiệm cho một tập hợp con các hash slot.
- Redis Cluster sử dụng cấu hình master-slave để tăng khả năng sẵn sàng. Mỗi master node có thể có một hoặc nhiều slave node để sao chép dữ liệu.
- Các node trong cluster liên tục tương tác với nhau để đảm bảo các master node luôn hoạt động. Nếu một master node gặp sự cố thì một slave node sẽ được chọn để thay thế để đảm bảo tính liên tục của hệ thống.

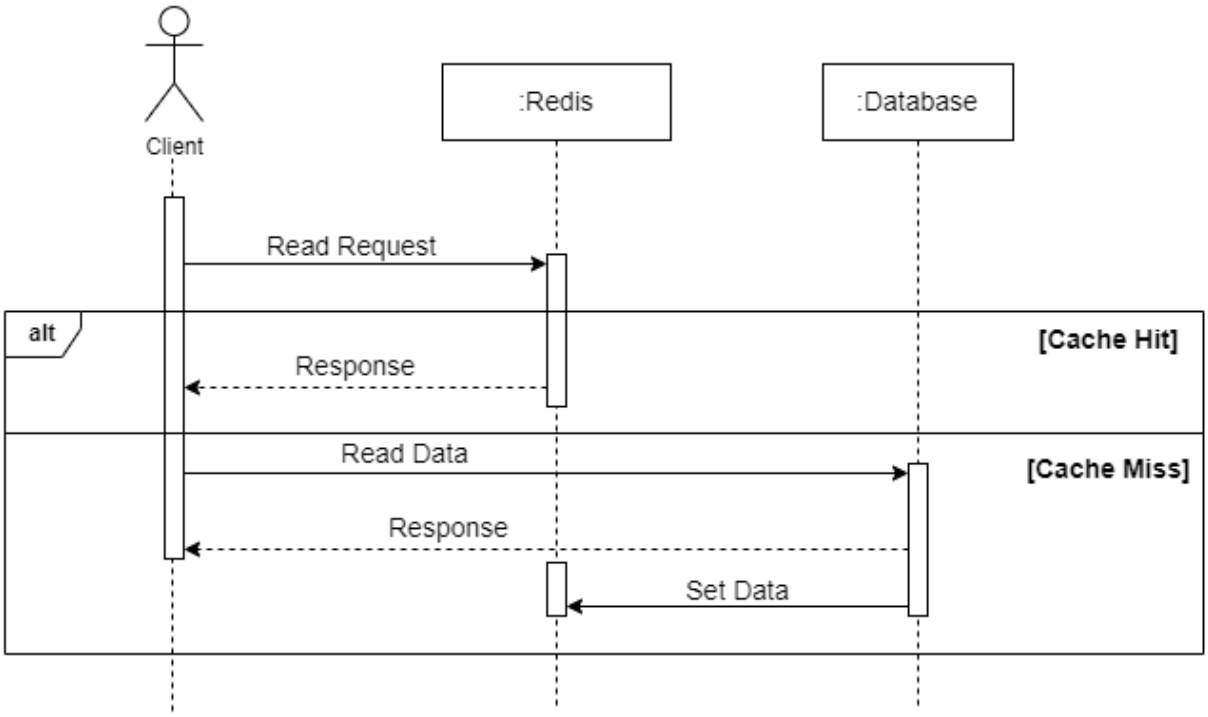
- Quá trình lựa chọn slave node thay thế được thực hiện bởi một số lượng node nhất định trong cluster và có thể được cấu hình. Quá trình này sử dụng cơ chế "Gossiping", cho phép các nút trao đổi thông tin về trạng thái của nhau.

2. Cơ chế truy vấn/lưu dữ liệu

a. Cách thức hoạt động của Redis

- Redis thường được dùng như một bộ cache trước khi truy cập vào database để cải thiện hiệu năng hệ thống
- Dữ liệu thường được lưu trong Redis:
    - Dữ liệu ít thay đổi nhưng được truy cập thường xuyên
    - Dữ liệu ít quan trọng và vẫn đang trong quá trình phát triển
- Khi client muốn ghi hoặc đọc dữ liệu, nó sẽ gửi yêu cầu đến một nút bất kỳ trong cluster.
- Nút nhận yêu cầu sẽ sử dụng hàm băm để tính toán hash slot tương ứng với key của dữ liệu.
- Dữ liệu được lưu trữ trong một hash slot duy nhất, dựa trên kết quả của phép modulo giữa giá trị băm của key và tổng số hash slot.
- Nếu hash slot nằm trong dải quản lý của nút nhận yêu cầu, nút đó sẽ trực tiếp xử lý yêu cầu.
- Nếu hash slot thuộc về nút khác, nút nhận yêu cầu sẽ chuyển hướng client đến nút chính xác để truy cập dữ liệu.

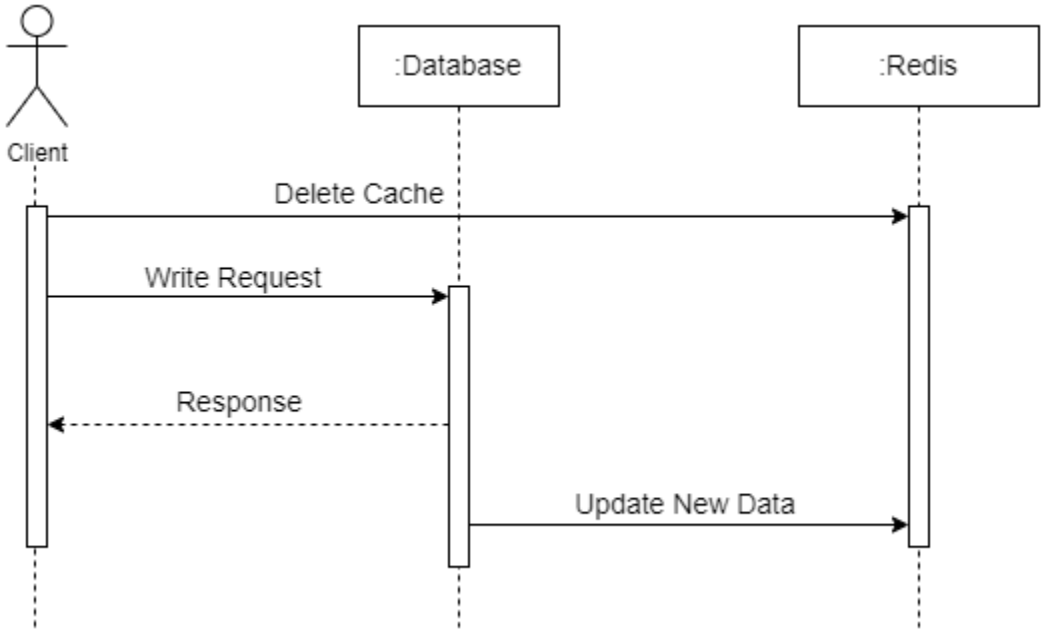
b. Cách thức truy vấn dữ liệu



Hình 7: Biểu đồ luồng hoạt động truy vấn dữ liệu của Redis

- Khi 1 yêu cầu đọc dữ liệu được gửi từ người dùng, hệ thống sẽ kiểm tra xem trong Redis đã có lưu trữ dữ liệu đó chưa:
    - Nếu đã có trong Redis thì hệ thống sẽ lấy dữ liệu đó ra từ Redis và gửi lại cho người dùng.
        - Nếu không có trong Redis thì hệ thống sẽ lấy dữ liệu đó từ trong Database. Sau đó, database sẽ ghi lại dữ liệu vừa được yêu cầu vào trong Redis. Hệ thống gửi lại dữ liệu cho người dùng.

c. Cách thức lưu dữ liệu



Hình 8: Biểu đồ luồng hoạt động lưu dữ liệu của Redis

- Khi có yêu cầu ghi dữ liệu từ người dùng, hệ thống sẽ ghi vào trong database trước. Sau đó, sẽ xóa dữ liệu cũ trong Redis đi, rồi ghi lại dữ liệu mới vào trong Redis.

3. Các mô hình triển khai trong thực tế

- Triển khai Redis trên Kubernetes: Mang lại khả năng mở rộng dễ dàng, tính sẵn sàng cao, hỗ trợ cân bằng tải, triển khai và quản lý đơn giản, giám sát và ghi lại nhật ký.
- Triển khai Redis thông qua AWS.

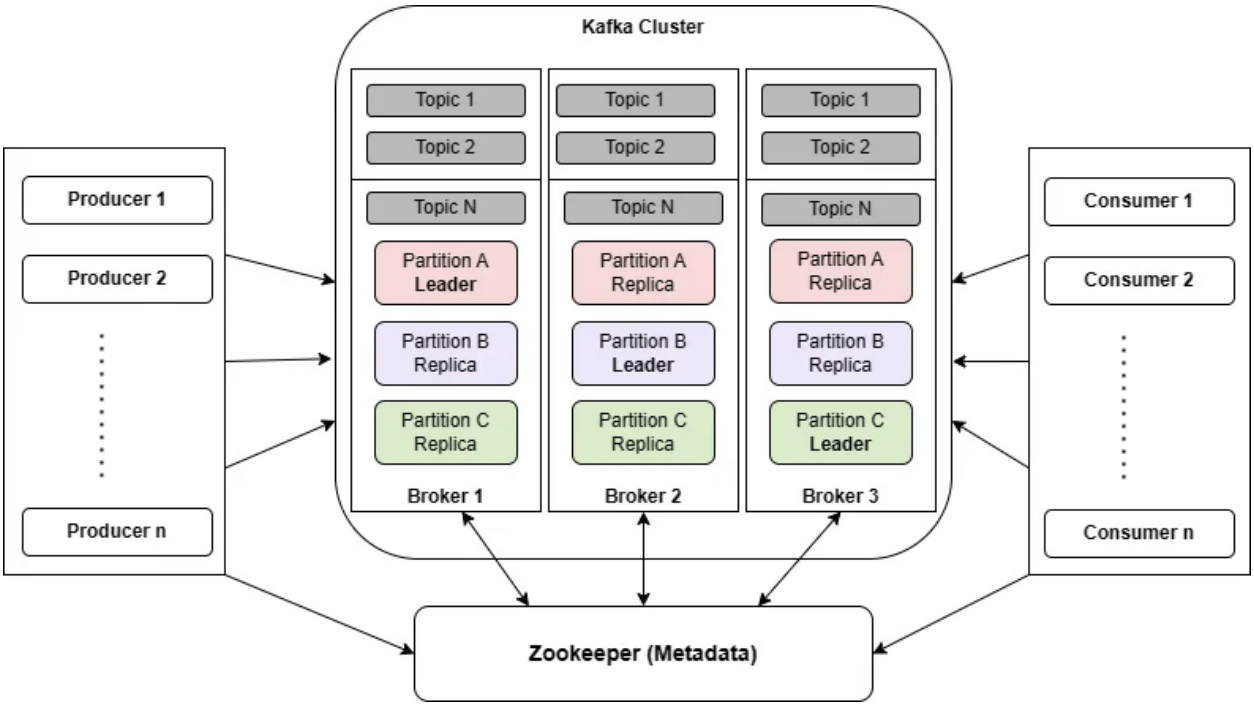
III. Maria Database và Kafka

1. MariaDB

- MariaDB là hệ quản trị cơ sở dữ liệu miễn phí được phát triển từ hệ quản trị cơ sở dữ liệu mã nguồn mở MySQL.
- MariaDB được hình thành dựa trên nền tảng của MySQL, vì thế nó kế thừa được hầu hết các chức năng cơ bản cần thiết của MySQL. Bên cạnh đó, MariaDB cũng phát triển thêm nhiều tính năng mới và có sự nâng cấp hơn về cơ chế lưu trữ, tối ưu máy chủ.
- MariaDB khắc phục những hạn chế của MySQL, bổ sung thêm nhiều Engine hơn, kết hợp cả SQL và NoSQL.

2. Kafka

- Kafka được tạo ra để giải quyết những thách thức trong việc xử lý lượng dữ liệu khổng lồ trong thời gian thực, cho phép các ứng dụng publish, subscribe, store và xử lý các luồng bản ghi (streaming event) một cách hiệu quả.
- Kafka tái định nghĩa các tổ chức xử lý luồng dữ liệu và xử lý sự kiện.
- Kafka có kiến trúc phân tán, khả năng chịu lỗi và khả năng mở rộng.



Hình 9: Mô hình cấu trúc của Kafka

- Event/Record/Message:
    - Ghi lại các sự kiện xảy ra trong hệ thống. Việt đọc và ghi dữ liệu trong Kafka thực hiện thông qua event.
    - Mỗi event chứa một key (khoá), value (giá trị) và metadata (nếu có).
    - Ví dụ về 1 event có chứa key, value và metadata (timestamp) như sau:

```
{
    key: "Violet",
    value: "Made a payment of $100 to Alex",
    timestamp: "Jun. 25, 2023, at 2:06 p.m."
}
```

Hình 10: Ví dụ về 1 event trong Kafka

- Topic: Nơi lưu trữ và tổ chức các event.
- Cluster:
    - Kafka hoạt động dưới dạng 1 Cluster bao gồm 1 hoặc nhiều server có thể mở rộng trên nhiều data center hoặc cloud.
    - Trong mỗi cluster sẽ có 1 broker hoạt động như 1 cluster controller chịu trách nhiệm chỉ định phân vùng cho các broker và theo dõi lỗi của các broker.
- Broker: Là 1 server trong cluster chịu trách nhiệm quản lý bộ lưu trữ, xử lý các yêu cầu đọc, ghi cũng như sao chép dữ liệu trên toàn cluster.
- Partition:
    - Là đơn vị cơ bản của tính song song và phân phối trong Kafka.
    - Các topic được chia thành các partition.
    - Mỗi partition được lưu trữ trên 1 broker duy nhất.
    - Nhiều partition cho phép mở rộng quy mô theo chiều ngang và cải thiện hiệu suất.
    - Mỗi event trong 1 partition đều được gán offset duy nhất, bắt đầu từ 0, để xác định vị trí của event trong partition.
    - Mỗi partition có thể có 1 hoặc nhiều bản sao trên các broker khác nhau -> Ngăn ngừa mất mát dữ liệu trong trường hợp broker bị lỗi.
- Producer: Là các ứng dụng khách có khả năng publish event vào topic.

- 1 producer sẽ publish event vào 1 topic cụ thể.
- Theo mặc định, producer không quan tâm tới event được ghi ở partition nào mà sẽ publish đều event trên tất cả partition của một topic.
- Trong vài trường hợp, 1 producer sẽ gửi trực tiếp event tới các partition cụ thể.
- Producer kết nối bằng kết nối 2 chiều tới Broker thông qua giao thức mạng TCP.

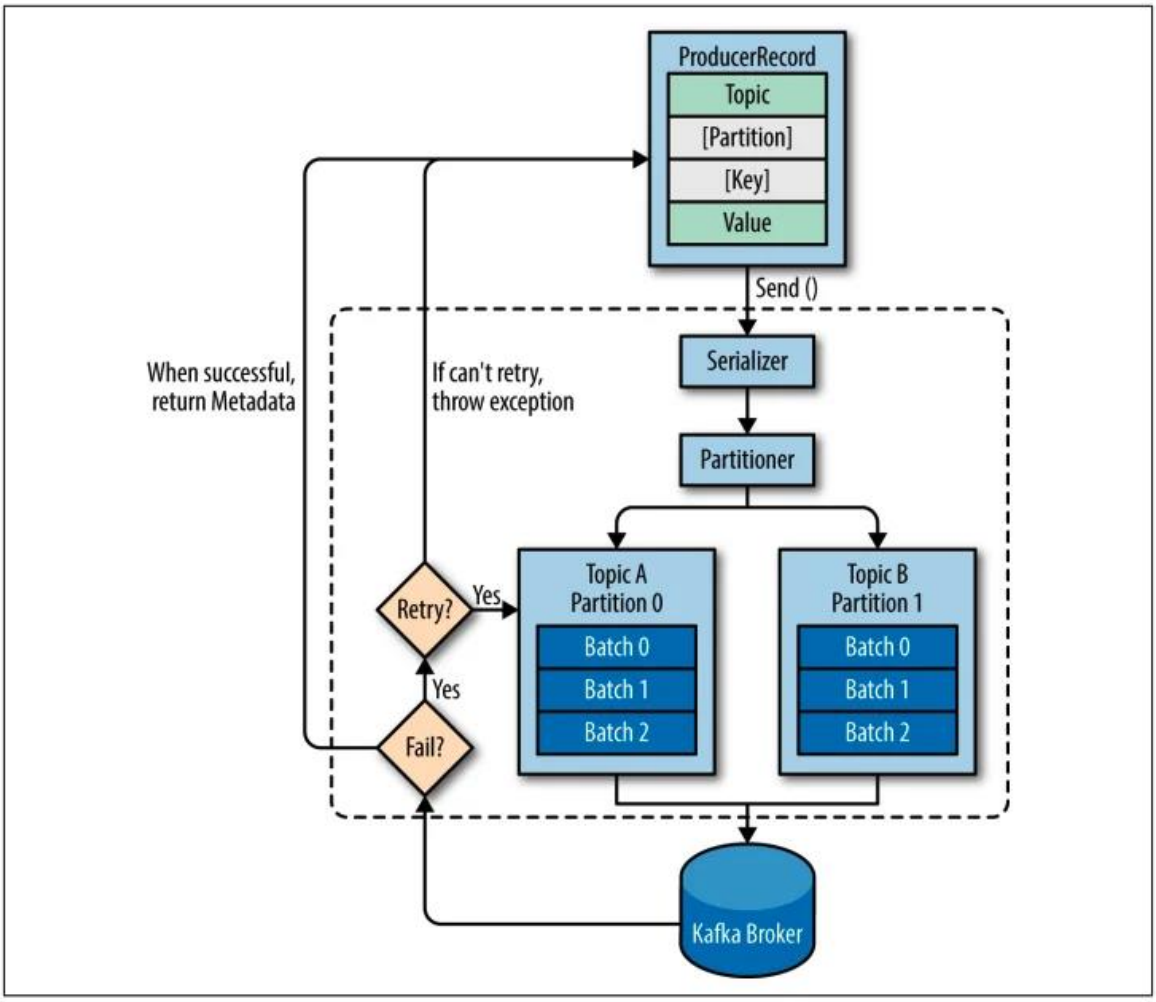


Figure 3-1. High-level overview of Kafka producer components

Hình 11: Mô tả tổng quan các thành phần trong Producer

- Quá trình gửi event từ Producer đến Broker gồm 4 bước:

- Bước 1: Producer publish event tới Kafka bằng cách tạo một ProducerRecord, trong đó bắt buộc có topic và value, và không bắc buộc có partition và key.
- Bước 2: Trước khi gửi ProducerRecord qua network, producer sẽ serialize (tuần tự hoá) key và value thành ByteArrays (mảng các bytes).
- Bước 3: Sau bước Serializer, dữ liệu được gửi tới một Partitioner. Nếu chúng ta chỉ định sẵn partition thì partitioner sẽ trả về partition được chỉ định, còn không thì partitioner sẽ chọn một partition dựa vào ProducerRecord key.
- Bước 4: Broker xử lý event và trả về kết quả cho Producer
    - Khi biết event cần được gửi tới topic và partition nào, producer sẽ thêm event vào một lô các bản ghi. Từ đó, chúng sẽ được gửi đi cùng topic và partition. Một thread độc lập chịu trách nhiệm gửi các lô bản ghi tới Broker phù hợp.
    - Broker sẽ gửi lại một response khi nhận được các event. Nếu event được ghi thành công vào Kafka, Broker sẽ gửi lại một object chứa RecordMetadata bao gồm topic, partition và offset của record bên trong partition. Còn nếu không thành công thì broker trả về lỗi. Khi producer nhận được lỗi, event sẽ được gửi lại (retry) vài lần trước khi bỏ cuộc và trả về lỗi.

- Consumer: Là các ứng dụng khách có khả năng subscribe vào 1 hoặc nhiều topic.
    - Đọc các bản ghi theo thứ tự chúng được tạo ra.
    - Đọc dữ liệu theo thời gian thực hoặc theo tốc độ riêng của chúng.

- Consumer kết nối bằng kết nối 2 chiều tới Broker thông qua giao thức mạng TCP.
- Consumer hoạt động trong một consumer group, làm việc cùng nhau để xử lý dữ liệu từ các partition -> Cung cấp khả năng mở rộng theo chiều ngang và cho phép nhiều phiên bản của cùng một ứng dụng xử lý dữ liệu đồng thời.
- Khi 1 consumer group đọc các event từ các partition, có 3 trường hợp xảy ra:
  - Trường hợp 1: Số consumer nhỏ hơn số partition thì 1 consumer có thể đọc các event từ nhiều hơn 1 partition.
  - Trường hợp 2: Số consumer bằng số partition thì 1 consumer sẽ đọc các event từ 1 partition
  - Trường hợp 3: Số consumer lớn hơn số partition thì số consumer thừa ra sẽ không được đọc event nào -> Nên tránh trường hợp này để tránh event bị bỏ sót hoặc chưa được đọc.

- Zookeeper: Được sử dụng để quản lý, lưu trữ metadata của cluster và điều phối các consumer.

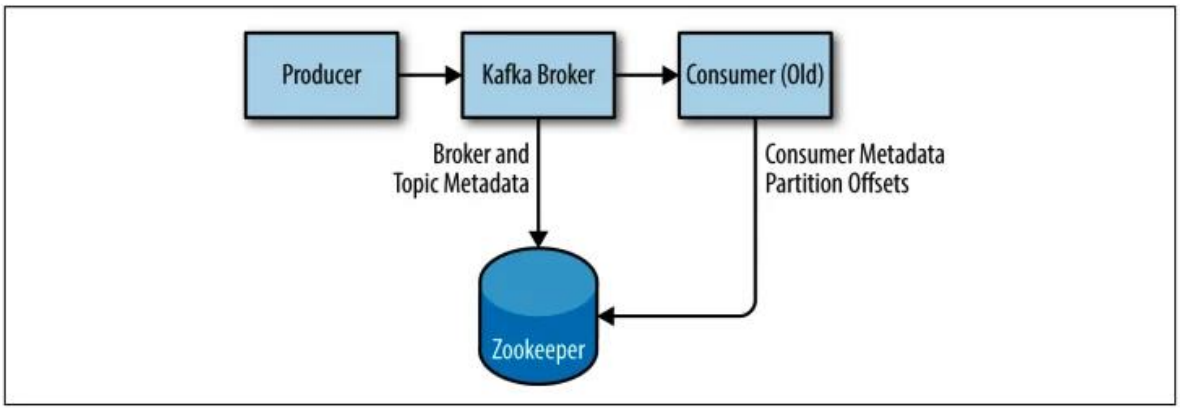


Figure 2-1. Kafka and Zookeeper

Hình 12: Mô tả hoạt động của Zookeeper trong Kafka

- Một số ứng dụng của Kafka:
    - **Truyền tải dữ liệu:** Kafka được sử dụng để truyền tải dữ liệu từ các nguồn khác nhau đến các đích, giúp các hệ thống khác nhau có thể giao tiếp và chia sẻ dữ liệu một cách hiệu quả
    - **Xử lý luồng sự kiện:** Kafka hỗ trợ xử lý luồng sự kiện theo thời gian thực, cho phép các ứng dụng phân tích dữ liệu ngay khi nó được tạo ra. Điều này rất hữu ích trong các hệ thống giám sát, phát hiện gian lận, và các ứng dụng yêu cầu phản ứng nhanh với sự kiện.
    - **Tích hợp dữ liệu:** Kafka có thể được sử dụng để tích hợp dữ liệu từ nhiều nguồn khác nhau vào một kho dữ liệu trung tâm, giúp việc phân tích và báo cáo trở nên dễ dàng hơn.
    - **Hệ thống phân tán:** Với khả năng mở rộng và tính chịu lỗi cao, Kafka là lựa chọn lý tưởng cho các hệ thống phân tán yêu cầu khả năng chịu tải cao và độ tin cậy.
    - **Quản lý logs và giám sát:** Kafka có thể được sử dụng để thu thập, lưu trữ và phân tích log từ nhiều hệ thống khác nhau, giúp việc giám sát và quản lý hệ thống trở nên hiệu quả hơn.
    - **Microservices:** Trong kiến trúc microservices, Kafka thường được sử dụng để truyền tải thông tin giữa các dịch vụ, đảm bảo tính nhất quán và đồng bộ dữ liệu.