ElasticSearch

[**I. Introduction**](#_qzp195y3gwzi) **2**

[1.Giới thiệu chung](#_xza2g94llnws) 2

[2. Data model](#_5p8j1rqx4hmq) 3

[3. Mục đích](#_yy59k1y0opv1) 3

[4. Khả năng sử dụng](#_8rdj4npblm2x) 3

[5. Advantages](#_bewmpiin4t7g) 4

[6. Disadvantages](#_x3p4bwr6gsx) 4

[7. Security](#_6yqj7z7c4ou3) 5

[8. Ứng dụng nổi tiếng đã sử dụng HQT CSDL Elasticsearch](#_y7lu3bhhzubx) 5

[9. Elasticsearch có thể tích hợp được với tất cả các ứng dụng có sử dụng ngôn ngữ](#_l0hzicvqjo9l) 5

[**II. Data model / Type**](#_h8y1j9i1j56) **6**

[1. Mô hình dữ liệu](#_rgnd7rguv7nw) 6

[2. Khái niệm cơ bản](#_ihheioaglghy) 7

[3. Lưu trữ](#_8zpss0lm1rlw) 8

[**III. Query Language**](#_b8ft8qdq6lst) **8**

[**IV. Integrity Mode**](#_7zx3zvnpa9hs) **10**

[1. Giới thiệu chung về mô hình dữ liệu](#_whabiip66u18) 10

[2. Mô hình ứng dụng](#_zf86meu62usc) 10

[3. ACID-CAP-BASE](#_qdwf57tor9) 11

[**V. File Organization, Index, Hashing**](#_fcto136vx6t6) **13**

[1. Vấn đề tổ chức dữ liệu](#_bgz7zs534wb5) 13

[2. Index](#_x0adqsgyegl7) 19

[**VI. Concurrency Control, Conflict, Deadlock**](#_cnjgwdmhpdea) **31**

[1. Vấn đề xử lý đồng thời](#_w1ypkhm2xmt5) 31

[2. Tương tranh](#_om5uu07xmc0v) 34

[**VI. Access Control**](#_aqb1z8blnzll) **34**

[**VII. Backup, recovery, partition, sharding, replication, cluster**](#_col12xkksvzx) **38**

[1. Backup](#_alapr2hle4w7) 38

[2. Recovery](#_pn3jhfadi57b) 40

[3. Partition](#_pa4x04ndasz5) 41

[4. Sharding](#_g8hyzpifdnsz) 42

[5. Replication](#_to90cs6d8sd6) 42

[6. Cluster](#_stbnrpihukep) 43

[**VIII. Cài đặt, demo**](#_l76pdqx76vfg) **45**

# 

# 

# **I. Introduction**

## **1.Giới thiệu chung**

* Elasticsearch là một công cụ tìm kiếm dựa trên nền tảng Apache Lucene. Nó cung cấp một bộ máy tìm kiếm dạng phân tán, có đầy đủ công cụ với một giao diện web HTTP có hỗ trợ dữ liệu JSON. Elasticsearch được phát triển bằng Java và được phát hành dạng nguồn mở theo giấy phép Apache.
* Elasticsearch thực chất hoạt động như 1 web server, có khả năng tìm kiếm nhanh chóng (near real time) thông qua giao thức RESTful.
* Elasticsearch là 1 hệ thống phân tán và có khả năng mở rộng tuyệt vời (horizontal scalability). Lắp thêm node cho nó là nó tự động auto mở rộng cho bạn.

## **2. Data model**

* Elasticsearch là một hệ NoSQL database.
* Elasticsearch là một hệ document store database.

## **3. Mục đích**

* Công cụ hỗ trợ cho những app có chức năng search hoặc yêu cầu phức tạp.
* Elasticsearch có khả năng phân tích và thống kê dữ liệu.
* Elasticsearch chạy trên server riêng và đồng thời giao tiếp thông qua RESTful do vậy nên nó không phụ thuộc vào client viết bằng gì hay hệ thống hiện tại của bạn viết bằng gì. Nên việc tích hợp nó vào hệ thống bạn là dễ dàng, bạn chỉ cần gửi request http lên là nó trả về kết quả.

## **4. Khả năng sử dụng**

* Khi bạn chạy một web bán hàng online, bạn cho phép khách hàng có thể tìm kiếm tìm kiếm sản phẩm của bạn. Trong trường hợp này bạn có thể sử dụng ElasticSearch để lưu trữ toàn bộ danh mục sản phẩm, cung cấp các đề xuất tìm kiếm. Lưu trữ các khoảng không quảng cáo và làm đầy chúng khi cần thiết.
* Bạn muốn có một tập log hoặc có tập dữ liệu trao đổi và bạn muốn phân tích chúng thành các data dưới dạng xu hướng, thống kê, tóm tắt, hoặc không phải các loại trên. Trong trường hợp này bạn có thể sử dụng Logstash (một phần của ElasticSearch) để thu thập, tổng hợp và phân tích cú pháp dữ liệu của bạn, sau đó chuyển dữ liệu từ Logstash vào ElasticSearch. Lúc này, bạn có thể search hoặc tổng hợp thông tin theo cách mình muốn.
* Bạn có nhu cầu phân tích, hoặc kinh doanh mà muốn điều tra, phân tích, hoặc cái nhìn trực quan hoặc đặt câu hỏi quảng cáo cho một dữ liệu rất lớn. Trường hợp này , bạn cần sử dụng ElasticSearch để lưu trữ data của bạn và sau đó sử dụng Kibana (một phần của Elasticsearch/ Logstash/Kibana stack) để xây dựng bảng điều tra , điều đó giúp bạn có thể trực quan hóa data của mình, đây cũng là điều rất quan trọng. Thêm vào đó, bạn có thể sử dụng chức năng tổng hợp của Elasticsearch để xây dựng nhũng câu truy vấn thông minh phù hợp với yêu cầu bài toán.

## **5. Advantages**

* Tìm kiếm dữ liệu rất nhanh chóng, mạnh mẽ dựa trên Apache Lucene ( near - real time searching).
* Có khả năng phân tích dữ liệu (Analysis data).
* Khả năng mở rộng theo chiều ngang tuyệt vời.
* Hỗ trợ tìm kiếm mờ (fuzzy), tức là từ khóa tìm kiếm có thể bị sai lỗi chính tả hay không đúng cú pháp thì vẫn có khả năng elasticsearch trả về kết quả tốt.
* Hỗ trợ Structured Query DSL (Domain-Specific Language ), cung cấp việc đặc tả những câu truy vấn phức tạp một cách cụ thể và rõ ràng bằng JSON.
* Hỗ trợ nhiều Elasticsearch client như Java, PhP, Javascript, Ruby, .NET, Python.

## **6. Disadvantages**

* Elasticsearch được thiết kế cho mục đích search, do vậy với những nhiệm vụ khác ngoài search như CRUD thì Elasticsearch kém thế hơn so với những database khác như Mongodb, Mysql …. Do vậy người ta ít khi dùng Elasticsearch làm database chính, mà thường kết hợp nó với 1 database khác.
* Trong Elasticsearch không có khái niệm database transaction , tức là nó sẽ không đảm bảo được toàn vẹn dữ liệu trong các hoạt động Insert, Update, Delete. Tức khi chúng ta thực hiện thay đổi nhiều bản ghi nếu xảy ra lỗi thì sẽ làm cho logic của mình bị sai hay dẫn tới mất mát dữ liệu. Đây cũng là 1 phần khiến Elasticsearch không nên là database chính.
* Không thích hợp với những hệ thống thường xuyên cập nhật dữ liệu. Sẽ rất tốn kém cho việc đánh index dữ liệu.

## **7. Security**

* Các tính năng bảo mật của Elasticsearch cung cấp cho người dùng quyền truy cập phù hợp. Các nhóm công nghệ thông tin, vận hành và ứng dụng có thể tận dụng các khả năng phù hợp này để quản lý người dung có thiện chí và đẩy lùi các tác nhân độc hại.

## **8. Ứng dụng nổi tiếng đã sử dụng HQT CSDL Elasticsearch**

* Mozilla, Quora.
* Sound Cloud, GitHub.
* Stack Exchange.
* Center for Open Science.
* Reverb, Netflix.
* Người dùng chỉ truy cập trên dữ liệu của Elasticsearch.
* Người quản trị thì truy cập trực tiếp trên Database SQL (MySQL) để cập nhật thêm xóa sửa và nó tự động cập nhật sang ElasticSearch.

## **9. Elasticsearch có thể tích hợp được với tất cả các ứng dụng có sử dụng ngôn ngữ**

* Java
* JavaScript
* Groovy
* .NET
* PHP
* Perl
* Python
* Ruby

# **II. Data model / Type**

## **1. Mô hình dữ liệu**

* Cơ sở dữ liệu NoSQL là Cơ sở dữ liệu được xây dựng dành riêng cho mô hình dữ liệu và có sơ đồ linh hoạt để xây dựng các ứng dụng hiện đại. Cơ sở dữ liệu NoSQL được công nhận rộng rãi vì khả năng dễ phát triển, chức năng cũng như hiệu năng ở quy mô lớn. Các Cơ sở dữ liệu này sử dụng nhiều mô hình dữ liệu đa dạng, trong đó có văn bản, đồ thị, khóa – giá trị, trong bộ nhớ và tìm kiếm.
* Document stores, hay còn gọi là hệ thống database hướng document (document-oriented) là một kiểu database mà ở đó data được tổ chức một cách tự do không theo một lược đồ nào cả (schema-free).

Điều này có nghĩa là:

* Mỗi bản ghi không cần phải có cấu trúc cố định, các bản ghi khác nhau có thể có nhiều cột khác nhau.
* Loại dữ liệu trong mỗi cột giữa các bản ghi cũng có thể khác nhau.
* Một cột có thể có nhiều hơn một giá trị (mảng – array).
* Các bản ghi có thể có cấu trúc lồng nhau (trong một bản ghi chứa một hoặc nhiều bản ghi khác – nested structure).

## **2. Khái niệm cơ bản**

* **RESTful API** là một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web (thiết kế Web services) để tiện cho việc quản lý các resource. Nó chú trọng vào tài nguyên hệ thống (tệp văn bản, ảnh, âm thanh, video, hoặc dữ liệu động…), bao gồm các trạng thái tài nguyên được định dạng và được truyền tải qua HTTP.
* **Node** là trung tâm hoạt động của Elasticsearch. Là nơi lưu trữ dữ liệu, tham gia thực hiện đánh index của cluster cũng như thực hiện các thao tác tìm kiếm. Mỗi node được định danh bằng 1 **unique name**.
* **Cluster** là tập hợp các nodes hoạt động cùng với nhau, chia sẻ cùng thuộc tính cluster.name. Chính vì thế Cluster sẽ được xác định bằng 1 'unique name'. Việc định danh các cluster trùng tên sẽ gây nên lỗi cho các node vì vậy khi setup cần hết sức chú ý điểm này. Mỗi cluster có một node chính (master), được lựa chọn một cách tự động và có thể thay thế nếu sự cố xảy ra. Một cluster có thể gồm 1 hoặc nhiều nodes. Các nodes có thể hoạt động trên cùng 1 server. Tuy nhiên trong thực tế, một cluster sẽ gồm nhiều nodes hoạt động trên các server khác nhau để đảm bảo nếu 1 server gặp sự cố thì server khác (node khác) có thể hoạt động đầy đủ chức năng so với khi có 2 servers. Các node có thể tìm thấy nhau để hoạt động trên cùng 1 cluster qua giao thức unicast. Chức năng chính của Cluster đó chính là quyết định xem shards nào được phân bổ cho node nào và khi nào thì di chuyển các Cluster để cân bằng lại Cluster.
* **Index** trong Elasticsearch là cấu trúc inverted index, được thiết kế để cho phép tìm kiếm full-text. Cách thức của nó khá đơn giản, các văn bản được phân tách ra thành từng từ có nghĩa sau đó sẽ được map xem thuộc văn bản nào.
* **Document** là một JSON object với một số dữ liệu. Đây là basic information unit trong Elasticsearch. Hiểu 1 cách cơ bản thì đây là đơn vị nhỏ nhất để lưu trữ dữ liệu trong Elasticsearch.
* **Shard** là đối tượng của Lucene , là tập con các documents của 1 Index. Một Index có thể được chia thành nhiều Shard. Mỗi node bao gồm nhiều Shard. Chính vì thế Shard mà là đối tượng nhỏ nhất, hoạt động ở mức thấp nhất, đóng vai trò lưu trữ dữ liệu. Chúng ta gần như không bao giờ làm việc trực tiếp với các Shard vì Elasticsearch đã support toàn bộ việc giao tiếp cũng như tự động thay đổi các Shard khi cần thiết. Có 2 loại Shard là : primary shard và replica shard.
* **Primary Shard**: Primary Shard là sẽ lưu trữ dữ liệu và đánh index. Sau khi đánh xong dữ liệu sẽ được vận chuyển tới các Replica Shard. Mặc định của Elasticsearch là mỗi index sẽ có 5 Primary shard và với mỗi Primary shard thì sẽ đi kèm với 1 Replica Shard.
* **Replica Shard**: đúng như cái tên của nó, nó là nơi lưu trữ dữ liệu nhân bản của Primary Shard. Replica Shard có vai trò đảm bảo tính toàn vẹn của dữ liệu khi Primary Shard xảy ra vấn đề. Ngoài ra Replica Shard có thể giúp tăng cường tốc độ tìm kiếm vì chúng ta có thể setup lượng Replica Shard nhiều hơn mặc định của Elasticsearch.

## **3. Lưu trữ**

* Lưu trữ dữ liệu theo dạng JSON – JSON document storage.
* Sử dụng RESTful API để lưu trữ document.
* Mỗi document gửi tới Elasticsearch được lưu trữ qua một thuật toán và sau đó được gửi đến shard. Elasticsearch cố gắng để phân tán document thông qua các shard.
* Khi lưu trữ document, Elasticsearch tạo ra inverted index, map các thuật ngữ / từ khóa xuất hiện trong document này tới chính document đó.
* Một điều quan trọng khi lưu trữ document đó là được quyết định cách tốt nhất để lưu trữ chúng, giúp nâng cao tốc độ truy vấn. Khi thiết kế các giải pháp sử dụng Elasticsearch, điều đáng lưu tâm nhất khi lưu trữ document chính là: tôi sẽ truy vấn document này như thế nào? “First query” này tiếp cận với việc sử dụng cho tất cả các khả năng Elasticsearch indexing để thực hiện truy vấn cực kỳ nhanh chóng.

# **III. Query Language**

Elasticsearch cung cấp full Query DSL (Domain Specific Language) dựa trên JSON để định nghĩa truy vấn. Hãy nghĩ về Query DSL như một AST (Abstract Syntax Tree) của truy vấn, bao gồm 2 loại:

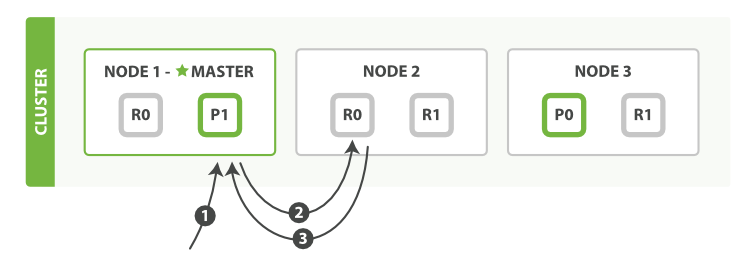
* Câu truy vấn đơn:

Các câu lệnh truy vấn đơn tìm kiếm một giá trị cụ thể trong một field cụ thể, ví dụ như math, term hay range. Những câu lệnh này có thể được sử dụng bởi chính nó.

* Câu lệnh truy vấn kép:

Các câu lệnh truy vấn kép bao gồm những câu vấn đơn khác hoặc truy vấn kép và được sử dụng

Quá trình lấy dữ liệu:



Hình trên mô phỏng quá trình lưu trình truy vấn dữ liệu theo id của document. Có thể chia thành 3 bước.

1. Request được gửi đến node master (Node 1). Tại đây xác định primary shard cho document sẽ là 0.
2. Do tất cả các node đều lưu dữ liệu, nên master node sẽ chọn ra 1 node và lấy dữ liệu ở shard số 0. Việc chọn này giúp giảm tập trung vào 1 node. Thuật toán Round-robin được sử dụng để các shard được chọn khác nhau ở mỗi request. Trong trường hợp này Node 2 được chọn.
3. Replica 0 ở Node 2 trả về kết quả cho master node.

Quá trình truy vấn dữ liệu:

1. Node nhận request (node trung gian) sẽ gửi broadcast request đó đến tất cả các node khác. Tại mỗi node này sẽ chỉ định shard thực hiện công việc tìm kiếm theo yêu cầu. Shard có thể là primary hoặc replica shard.
2. Mỗi shard sẽ thực hiện công việc tìm kiếm, trả về id và score của document. Trong dó score là giá trị dùng để sắp xếp. Danh sách gồm id, score này là một danh sách đã được sắp xếp, mang tính chất cục bộ.
3. Node trung gian sau khi nhận kết quả trả về từ các node khác sẽ thực hiện công việc sắp xếp toàn cục tất các document, dựa theo id và score. Cuối cùng trả ra kết quả về phía client.

Ví dụ: Chúng ta bắt đầu bằng một query đơn giản, lấy 10 giá trị từ giá trị thứ 100.

GET /\_search

{

“from”: 100,

“size”: 10

}

Tại mỗi shard sẽ trả về 110 bản ghi ( from + size ). Nếu có 3 primary shard thì node trung gian sẽ nhận tổng cộng 330 bản ghi. Sau đó nó sẽ thực hiện sắp xếp lại để được 1 danh sách 330 bản ghi đã sắp xếp. Cuối cùng dựa theo yêu cầu mà sẽ trả về document có vị trí từ 100 đến 109.

# **IV. Integrity Mode**

## **1. Giới thiệu chung về mô hình dữ liệu**

Elasticsearch là một cơ sở dữ liệu phân tán theo định hướng tài liệu. Toàn bộ biểu đồ đối tượng bạn muốn tìm kiếm cần được lập chỉ mục, vì vậy trước khi lập chỉ mục tài liệu của bạn, chúng phải được chuẩn hóa. Elasticsearch thiết kế ánh xạ và lưu trữ tài liệu theo cách được tối ưu hóa cho tìm kiếm và truy xuất. Chúng là tuyệt vời cho khối lượng viết một lần đọc nhiều lần. Giống như nhiều cơ sở dữ liệu định hướng tài liệu khác, Elasticsearch không có ràng buộc về dữ liệu.

## **2. Mô hình ứng dụng**

* Người dùng truy cập trực tiếp vào dữ liệu của ElasticSearch.
* ElasticSearch có thể được cài đặt đồng bộ với MySQL:
* Người dùng vẫn truy cập dữ liệu Elasticsearch.
* Dữ liệu được đồng bộ từ MySQL, admin update, edit dữ liệu trên MySQL rồi cập nhật sang Elasticsearch.

## **3. ACID-CAP-BASE**

**ACID Transaction Là Gì?**

* ACID là tính chất trong đó bao gồm tập hợp của 4 đặc tính khác nhau áp dụng cho một database transaction. Bốn đặc tính này gồm: atomicity , consistency (nhất quán), isolation (độc lập), và durability (bền vững).
* Elasticsearch không hỗ trợ các giao dịch ACID cho các thay đổi liên quan đến nhiều tài liệu, các thay đổi đối với các tài liệu riêng lẻ là ACIDic.
* Elasticsearch sử dụng công nghệ công cụ tìm kiếm có tên là [Lucene](https://lucene.apache.org/) . Lucene là một công nghệ truy xuất thông tin được xây dựng cho tốc độ, không phải dự phòng. Nó có một kiến ​​trúc hoàn toàn khác biệt mang lại cho nó hiệu năng nhanh chóng, với chi phí dễ bị mất dữ liệu hơn.
* Elasticsearch không tuân thủ ACID.

**CAP**

* Định lý CAP phát biểu như sau: Trong một hệ thống lưu trữ phân tán không thể đồng thời đảm bảo nhiều hơn hai trong ba tính chất sau:
* **Availability** (Sẵn sàng): Một yêu cầu được phản hồi lại không phải lỗi sẽ luôn được lưu trữ mới nhất.
* **Consistency** (Nhất quán): Mỗi lần đọc dữ liệu sẽ nhận được nội dung mới nhất hoặc lỗi.
* **Partition Tolerance** (Phân vùng): Hệ thống tiếp tục hoạt động bất chấp một lượng tùy ý các thông điệp / gói tin bị mất hoặc trì hoãn do trục trặc mạng giữa các nút.
* Elasticsearch, với tư cách là một kho lưu trữ dữ liệu phân tán, hỗ trợ định lý CAP, nơi người dùng có thể điều chỉnh sự cân bằng giữa tính nhất quán của dữ liệu trên các phân vùng, tính khả dụng của dữ liệu trong mỗi phân vùng và dung sai phân vùng của chỉ mục.
* Elasticsearch thường không được sử dụng như một thanh ghi tuyến tính hóa, vì vậy định lý CAP không thực sự được áp dụng. Điều đó nói rằng Elasticsearch gần đây đã đạt được một tính năng bổ sung thao tác so sánh và thiết lập và công việc để xác minh rằng điều này có ngữ nghĩa tuyến tính hóa đang diễn ra .Khi đối mặt với một phân vùng mạng đủ tệ, các hoạt động so sánh và thiết lập trong Elasticsearch sẽ duy trì tính nhất quán và hy sinh tính sẵn có.
* Đối với phân vùng mạng đủ tệ, các thao tác ghi trong Elasticsearch thường sẽ thích "tính nhất quán" hơn "tính khả dụng", theo nghĩa là không bao giờ bị mất ghi nhưng một số ghi có thể không được kiểm tra hoặc có thể thất bại. Các hoạt động đọc (tức là tìm kiếm) cung cấp các đảm bảo sẵn có mạnh mẽ hơn với chi phí nhất quán, theo nghĩa là một tìm kiếm đôi khi có thể trả về kết quả cũ hơn là thất bại. Đây thường là một lựa chọn thích hợp cho các công cụ tìm kiếm như Elasticsearch. Có các cơ chế để ngăn chặn kết quả *quá* cũ.

**BASE**

* Về cơ bản có sẵn (Basic Available) có thể đề cập đến sự sẵn có nhận thức của dữ liệu. Nếu một nút đơn bị lỗi, một phần dữ liệu sẽ không khả dụng, nhưng toàn bộ lớp dữ liệu vẫn hoạt động.
* Trạng thái mềm(Soft) : Tất cả những gì có thể tìm thấy là khái niệm dữ liệu cần làm mới thời gian. Nếu không làm mới, dữ liệu sẽ hết hạn hoặc bị xóa.
* Tính nhất quán cuối cùng (Eventual Consistency) cuối cùng có nghĩa là các bản cập nhật cuối cùng sẽ gợn qua tất cả các máy chủ, được cung cấp đủ thời gian.
* Nó được xây dựng trên ý tưởng rằng dữ liệu không phải lúc nào cũng phải ở trạng thái nhất quán. Nó tập trung nhiều hơn vào tính nhất quán cuối cùng và các hệ thống NoQuery đạt được các thuộc tính BASE cũng thường cố gắng để tăng tính khả dụng với chi phí nhất quán . Nó được kết nối mạnh mẽ với AP từ các thuộc tính CAP. Với tính khả dụng cơ bản, dữ liệu được phân phối trên một số nút có phân đoạn và sao chép, và nếu một lỗi xảy ra trên một nút, vẫn còn các nút khác chứa dữ liệu có thể truy cập. Đơn giản, điều đó có nghĩa là dịch vụ về cơ bản luôn sẵn sàng, thậm chí mặc dù toàn bộ dữ liệu là không.
* Khi cho phép đọc từ bản sao khái niệm Sự kiện. Với tính nhất quán cuối cùng, các phần chèn thêm và cập nhật không nhân rộng trực tiếp, nhưng cuối cùng họ sẽ được. Chính xác là khi nào, có thể cấu hình và lên đến nhà phát triển hệ thống. Ví dụ: nếu khối lượng công việc lớn, việc nhân rộng có thể bị hoãn lại cho đến khi khối lượng công việc thấp. Trạng thái mềm có nghĩa là trạng thái của hệ thống có thể thay đổi theo thời gian.
* Bất kể tương tác của người dùng, trạng thái của hệ thống có thể thay đổi, ví dụ như khi phân đoạn và bản sao của nó đạt đến một trạng thái nhất quán.

# **V. File Organization, Index, Hashing**

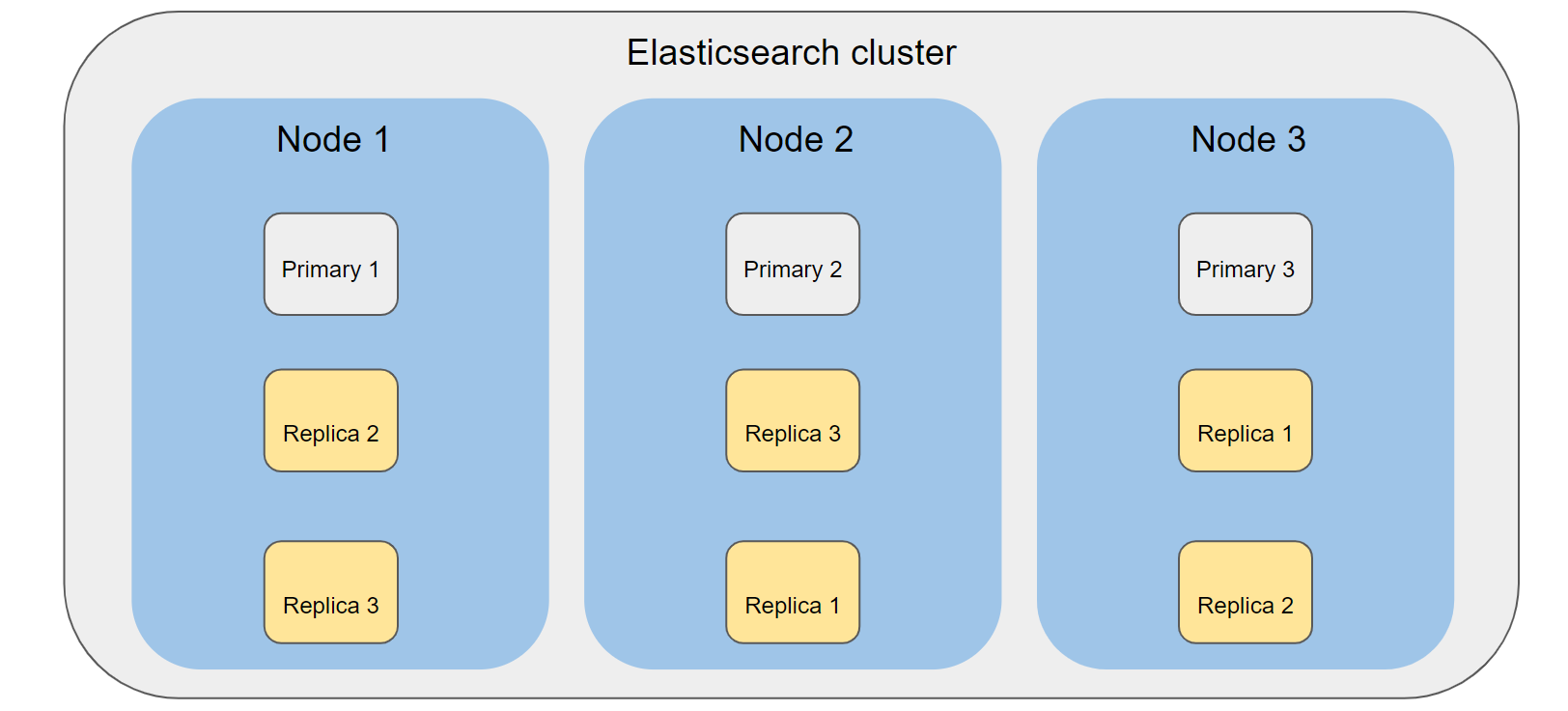
## **1. Vấn đề tổ chức dữ liệu**

**Tài liệu và index**

* Elasticsearch sử dụng "index" để lưu trữ và xử lý dữ liệu, nó tương tự một cơ sở dữ liệu thông thường. Dữ liệu sẽ được lưu trữ thành các "tài liệu" dưới dạng JSON. Mỗi một index có nhiều "type" (tương tự như bảng trong cơ sở dữ liệu), điều này sẽ giúp chúng ta phân tách các loại dữ liệu khác nhau khi lưu vào cùng một index.
* Mọi tài liệu được lưu vào trong cùng một type của một index sẽ có cấu trúc các trường và kiểu dữ liệu giống hệt nhau (giống như mỗi bảng có một schema trong các hệ cơ sở dữ liệu quan hệ)
* Mỗi một index của Elasticsearch có một hoặc nhiều shard (tuỳ cấu hình, mặc định là 5). Các shard này lại tồn tại trên các node khác nhau. Nội dung về shard và node này liên quan nhiều đến kiến trúc của Elasticsearch (sẽ tìm hiểu ở phần sau), cũng là một phần rất quan trọng giúp Elasticsearch có thể thực hiện công việc tìm kiếm một cách nhanh chóng.
* Shard cũng có hai loại là primary và replica, mỗi một primary shard sẽ có một vài (có thể là không có) shard replica (mặc định là 1). Elasticsearch sẽ đảm bảo là primary shard và replica shard sẽ tồn tại trên các node khác nhau.
* Một đặc điểm rất quan trọng, đó là số lượng primary shard cho một index không thể thay đổi được sau khi index đã được tạo. Do đó, trong thực tế, chúng ta sẽ phải cân nhắc rất nhiều yếu tố để quyết định xem cần bao nhiêu primary shard cho mỗi index. Trừ khi bạn muốn tạo lại index và thêm dữ liệu vào từ đầu.
* Elasticsearch hoạt động dựa trên Lucene của Apache. Vậy nó dựa trên như thế nào? Đây chính là lúc Lucene được sử dụng.
* Một shard thực chất là một Lucene index, đó mới là nơi thực sự lưu trữ dữ liệu, và bản thân shard cũng là một search engine. Là một Lucene index, nên một shard lại được tạo nên từ nhiều segment và mỗi segment là một inverted index với đầy đủ chức năng.
* Segment cho phép Lucene có thể thêm tài liệu vào index một cách dễ dàng mà không cần quá nhiều xử lý với index.
* Với mỗi yêu cầu tìm kiếm, mọi segment trong một index sẽ được sử dụng, và mỗi segment như vậy sẽ hoạt động một cách khá độc lập (điều này khiến nó sử dụng CPU và bộ nhớ). Điều đó có nghĩa là, nếu có quá nhiều segments, performance của thao tác tìm kiếm sẽ bị giảm đi, nhưng nếu ít segment quá thì lại không tận dụng hết được khả năng của máy chủ.
* Để xử lý vấn đề này, Lucene thực hiện một thao tác khá thông minh là "gộp" các segment nhỏ lại thành một segment lớn hơn. Tuy nhiên, điều này cũng có hai mặt: nếu thao tác gộp này được tiến hành không cẩn thận, Elasticsearch có thể rơi vào sai lầm là gộp tất cả segment vào làm một và bỏ phí rất nhiều tài nguyên (điều này cũng khiến performance giảm đi).

**Node và cluster**

* Node là cơ sở trong kiến trúc server của Elasticsearch, và nhiều node hợp với nhau tạo thành cluster.



**Lưu trữ dữ liệu và tìm kiếm**

* **Mô hình lưu trữ**:
* Elasticsearch sử dụng Lucene của Apache - một thử viên full text search viết bằng Java. Về bản chất, nó sử dụng một cấu trúc dữ liệu gọi là inverted index (chỉ mục đảo ngược) để có thể thực hiện tìm kiếm với hiệu suất cao.
* Tài liệu chính đơn vị cơ sở để quản lý dữ liệu trong Elasticsearch và inverted index được tạo ra bằng việc tokenize (thuật ngữ trong Elasticsearch) các khái niệm trong tài liệu. Bằng việc sử dụng kỹ thuật inverted index, một bảng chỉ mục các khái niệm và danh sách tài liệu liên quan đến khái niệm đó sẽ được tạo ra.
* Nó khá tương đồng với chỉ mục của một cuốn sách, nơi hiển thị một danh sách các khái niệm được sử dụng cùng với số trang mà nó xuất hiện. Trong Elasticsearch, khi nói rằng một tài liệu được index, thì chúng ta hiểu rằng, inverted index của tài liệu đó đã được tạo ra.
* Ví dụ, khi chúng ta cần tìm từ khoá "Việt Nam", Elasticsearch sẽ thực hiện tìm kiếm trên inverted index (rất nhanh do các từ được sắp xếp), tìm được từ khoá "Việt Nam", sau đó trả về ID của các tài liệu tương ứng. Để có thể tìm kiếm nâng cao hơn (ví dụ, tìm với từ khoá "Viet Nam"), thì quá trình phân tích và index tài liệu là rất quan trọng.
* Ngoài ra, quá trình tìm kiếm thực tế còn phức tạp hơn nữa, khi mà Elasticsearch còn phải đánh giá mức độ tương qua giữa những từ trong index với từ khoá cần tìm.
* **Tìm kiếm**: Quá trình tìm kiếm tài liệu sẽ diễn ra theo 2 giai đoạn.
* Giai đoạn query:
* Đầu tiên, truy vấn tìm kiếm sẽ đến node điều phối, ở đây, truy vấn này sẽ được chuyển tiếp đến mọi shard (cả primary và replica) trong index. Mỗi shard sẽ thực hiện thao tác tìm kiếm tương ứng với truy vấn một cách độc lập và trả về ID của các tài liệu có độ tương quan cao nhất (mặc định sẽ là 10 tài liệu có điểm tương quan cao nhất được trả về).
* Các ID này được trả về cho node điều phối, ở đây, nó sẽ được gộp với kết quả của các shard khác và sắp xếp lại để tìm ra những tài liệu có độ tương quan cao nhất.
* Giai đoạn fetch:
* Sau khi node điều phối gộp và sắp xếp lại kết quả nhận được từ các shard, nó sẽ thực hiện thao tác lấy thông tin của tài liệu đó. Các shard lại thực hiện công việc của mình và trả về tài liệu (toàn bộ tài liệu chứ không chỉ ID như giai đoạn trước) cho node điều phối.
* Sau khi quá trình lấy dữ liệu đã xong, kết quả sẽ được trả về cho client.
* Lưu ý rằng, để thực hiện tìm kiếm hiệu quả, mọi node trong cluster cần biết được trạng thái của cluster đó. Trạng thái của cluster sẽ bao gồm các thông tin như mỗi node có chứa index và shard nào. Nhờ đó, mọi node đều có thể trở thành node điều phối (và Elasticsearch sẽ thay đổi node điều phối trong trường hợp node được chọn không hoạt động).

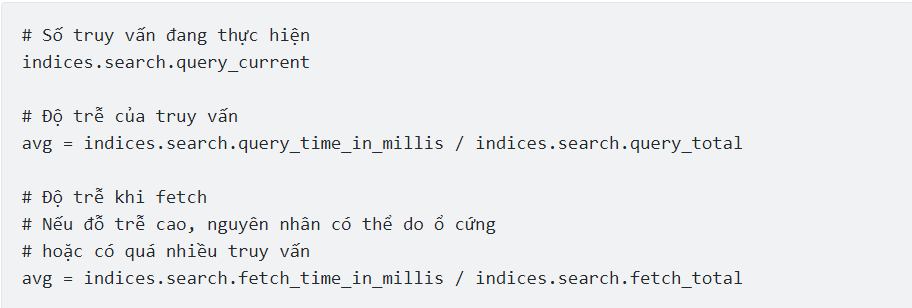
**Độ tương quan**

Độ tương quan được đánh giá bởi điểm mà Elasticsearch chấm cho từng tài liệu khi thực hiện thao tác tìm kiếm. Mặc định, thuật toán để đánh giá độ tương quan được sử dụng là thuật toán tf/idf (term frequency/inverse document frequency). Điểm tương quan cuối cùng chính là tổ hợp của điểm td-idf với một vài yếu tố đánh giá khác, như mức độ khớp của tài liệu với từ khoá.



**Đánh giá hiệu năng tìm kiếm**

Dựa trên quy trình thực hiện truy vấn tìm kiếm, chúng ta có thể dựa vào những thông số như sau để đánh giá hiệu năng của việc tìm kiếm, đồng thời cũng từ đó mà biết vấn đề nằm ở đâu (trong trường hợp việc tìm kiếm diễn ra lâu):



**Một số vấn đề hay gặp với Elasticsearch**

* **Cluster health**
* Đây là một vấn đề rất quan trọng nhưng nhiều khi không được quan tâm đúng mức. Với các cluster, chúng ta có thể kiểm tra trạng thái hoạt động của nó (gọi là cluster health) bằng cách:

****

* Kết quả trả về ở trên, chúng ta có thể dựa vào status để biết tình trạng hiện tại của cluster. Nó có thể là một trong 3 giá trị:
* green: Tất cả hoạt động bình thường.
* yellow: Tất cả primary shard đã hoạt động, vẫn còn replica shard chưa hoạt động.
* red: Có primary shard chưa hoạt động.
* Đây là thông số rất quan trọng, liên quan trực tiếp nên hiệu năng của Elasticsearch. Trạng thái red thường khó gặp trong thực tế, nhưng yellow thì rất nhiều.
* Cách giải quyết là thêm node cho Elasticsearch là xong. Lưu ý rằng, việc thêm node nhưng node này lại nằm trên các server vật lý khác nhau thì cần thêm một chút cầu hình nữa.
* **Quá nhiều tài liệu**
* Việc tìm kiếm của Elasticsearch được phân trang, thông qua hai giá trị from và size. Việc phân trang này diễn ra rất phổ biến, trên hầu hết các ứng dụng và thậm chí, trên giao diện, người dùng có thể dễ dàng nhảy từ trang này sang trang khác.
* Nhưng, Elasticsearch có những vấn đề của riêng nó liên quan đến phân trang, khi mà người dùng nhảy đến những trang quá lớn, nhất là trang cuối cùng.
* Số trang càng lớn, mức độ tính toán càng nhiều. Do đó, trên thực tế, chúng ta sẽ phải tìm cách giải quyết vấn đề này, dù tỉ lệ người dùng tìm đến các trang sâu như vậy là không nhiều.
* Một trong số những cách có thể áp dụng đó là sử dụng [scroll](https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-scroll.html) thay cho truy vấn search thông thường.

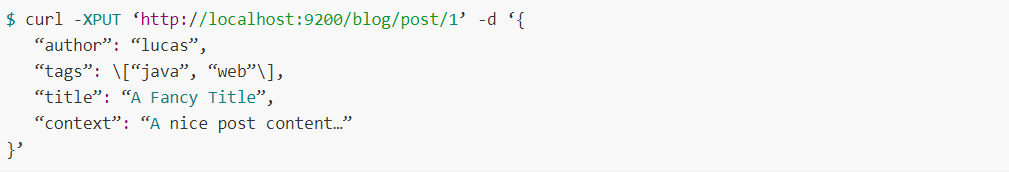
## **2. Index**

**Tổng quan về index**

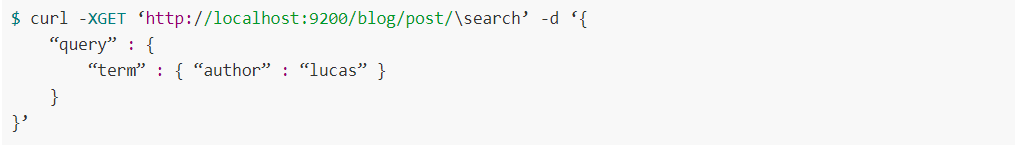
* Elasticsearch còn có những điểm khác giúp searching document hiệu quả hơn. Khi lưu trữ document trong Elasticsearch, nó tạo ra một số internal data structures làm cho query performance tốt hơn.
* Mỗi document gửi tới ES được lưu trữ qua một thuật toán và sau đó được gửi đến shard. Elasticsearch cố gắng để phân tán document thông qua các shard.
* Khi lưu trữ document, Elasticsearch tạo ra inverted index, map các thuật ngữ/từ khóa xuất hiện trong document này tới chính document đó.
* Khi sử dụng inverted index, nó có thể tìm kiếm thông qua terms như một binary tree (sử dụng thứ tự chữ cái) làm giảm thời gian tìm kiếm.
* Một điều quan trọng khi lưu trữ document đó là được quyết định cách tốt nhất để lưu trữ chúng giúp nâng cao tốc độ truy vấn. Khi thiết kế các giải pháp sử dụng ElasticSearch, điều đáng lưu tâm nhất khi lưu trữ document chính là: truy vấn document này như thế nào? “First query” này tiếp cận với việc sử dụng cho tất cả các khả năng ES indexing để thực hiện truy vấn cực kỳ nhanh chóng.

**Rest API**

* Elasticsearch sử dụng một REST API cho việc tìm kiếm và lưu trữ document. Dưới đây là một ví dụ về indexing (storing) a document:

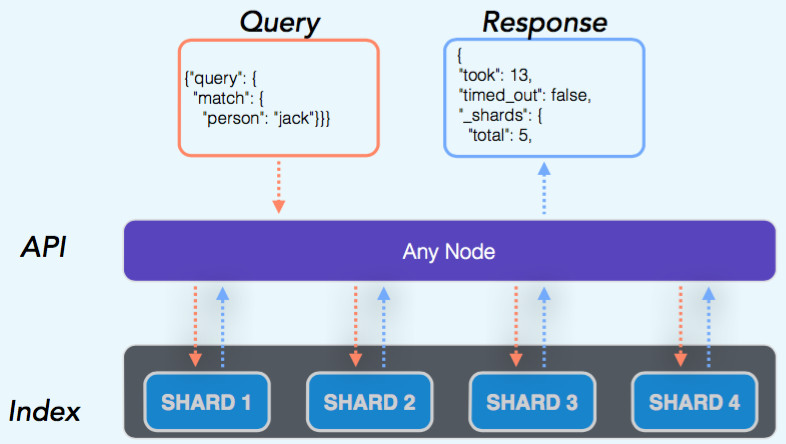


* “http://localhost:9200/” là địa chỉ của ES node. Với câu lệnh này, chúng ta đang tạo ra một blog post trong index “blog” với một type “post” và một “id=1”. Lưu ý rằng nó có thể sử dụng một truy vấn “POST” thay vì “PUT” để tự động tạo ra document id. Chính document là một JSON document bình thường.
* Dưới đây là request để lấy về một document:



* Nhìn vào sẽ rất dễ hiểu chúng ta đang truy vấn điều gì. Một query có thể được gửi bằng “GET” hoặc “POST” . “blog” xác định index nào mà ta đang truy vấn và “post” là document type (optional parameter). Các tham số “\_search” chỉ ra hành động mà chúng ta muốn thực hiện. Phần body của query sẽ xác định loại truy vấn và parameter. Trong trường hợp này, chúng ta đang tìm kiếm bất kỳ document mà giá trị của trường “author” có từ khóa “lucas”.
* Khi bạn gửi một query đến Elasticsearch cluster, query sẽ nhấn một trong các node và sau đó nó xác định shard nào nên được truy vấn. Sau đó, “query coordinator” node gửi query cho mỗi shard để thực hiện truy vấn song song. Sau khi lần lượt từng node trả lời các truy vấn với các kết quả từng phần, node sẽ merge kết quả và gửi trở lại cho user. Một trong những điều làm cho ES query document rất nhanh chính là nó đã được thực hiện truy vấn song song theo mặc định (đi qua shard).

Hình ảnh minh họa cách elasticsearch thực hiện truy vấn:



**Tuỳ biến mapping**

* Các trường có kiểu dữ liệu là string thì thường sẽ chứa text (và tìm kiếm thường là full text search). Vì vậy, để tìm kiếm hiệu quả, giá trị của những trường này cần được phân tích trước khi index. Khi tìm kiếm theo kiểu full text search thì từ khoá cũng sẽ được phân tích trước khi tiền hành tìm kiếm.
* Vì vậy, với các trường string, hai giá trị cực kỳ quan trọng để có thể tìm kiếm chính xác là index và analyzer. Với index, có 3 lựa chọn cho chúng ta:
* analyzed: phân tích text rồi sau đó index nó (dùng cho full text search).
* not\_analyzed: index trường với toàn bộ giá trị của nó mà không phân tích gì (không thể full text search).
* no: không index, trường này sẽ không dùng để tìm kiếm.
* Để phân tích một trường string, thì analyzer là rất quan trọng để làm điều này. Chúng ta cần cấu hình analyzer cho từng trường, và bộ phân tích này sẽ được sử dụng cho cả quá trình phân tích để index và phân tích từ khoá để tìm kiếm.
* Mặc định, Elasticsearch sử dụng standard analyzer, tuy nhiên bộ phân tích này thường kém hiệu quả với các ngôn ngữ không phải tiếng Anh. Ngoài standard analyzer, Elasticsearch còn tích hợp sẵn nhiều bộ phân tích khác nữa, như whitespace, simple và english. Tuy nhiên, chúng thường chỉ có tác dụng với tiếng Anh mà thôi.
* Với các ngôn ngữ khác, chúng ta cần sử dụng thêm plugin để có thể dùng các analyzer khác cho hiệu quả hơn trong quá trình tìm kiếm:

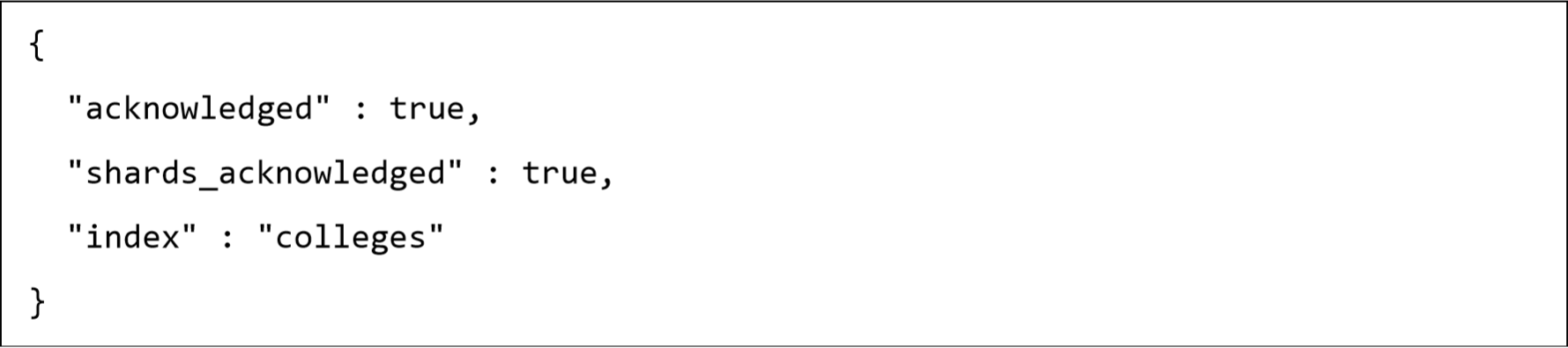


**Tạo index**

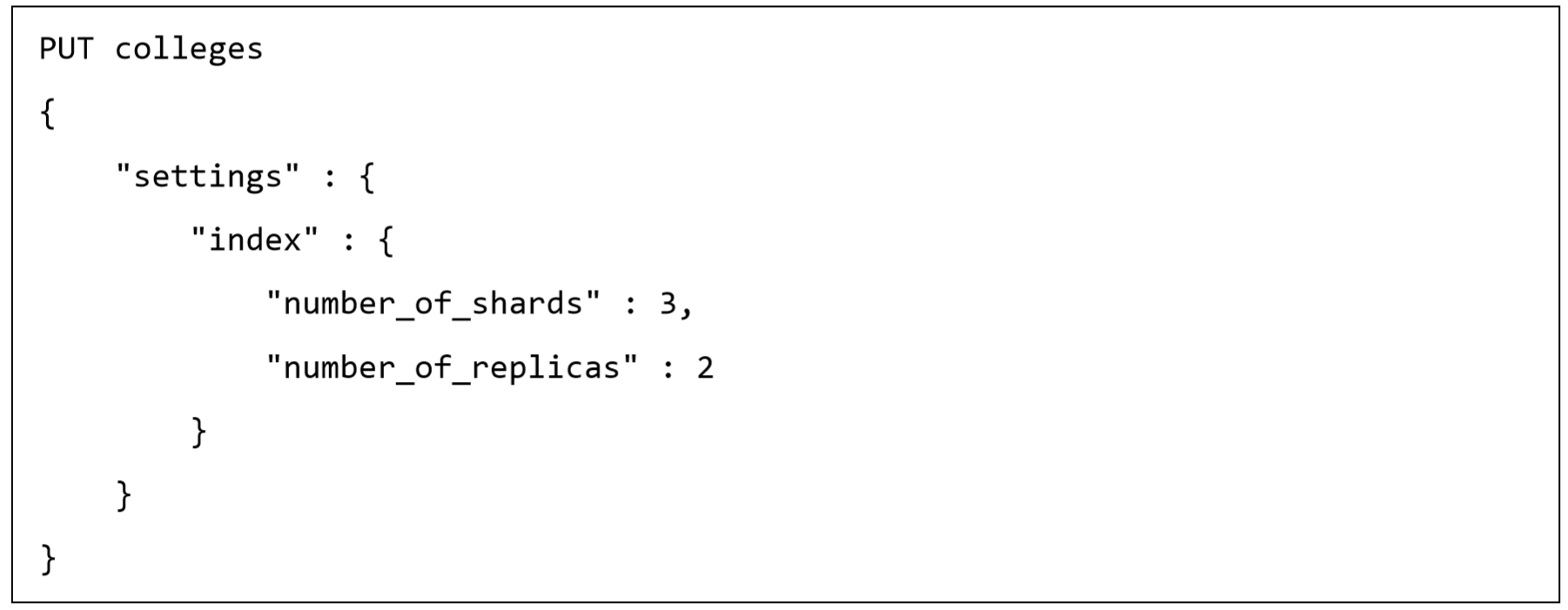
API này giúp tạo một Index. Một Index có thể được tạo tự động khi người dùng chuyển các đối tượng JSON đến bất kỳ Index nào hoặc nó có thể được tạo trước đó. Để tạo một Index, chỉ cần gửi yêu cầu PUT với các cài đặt, ánh xạ và bí danh hoặc chỉ một yêu cầu đơn giản không có phần thân.



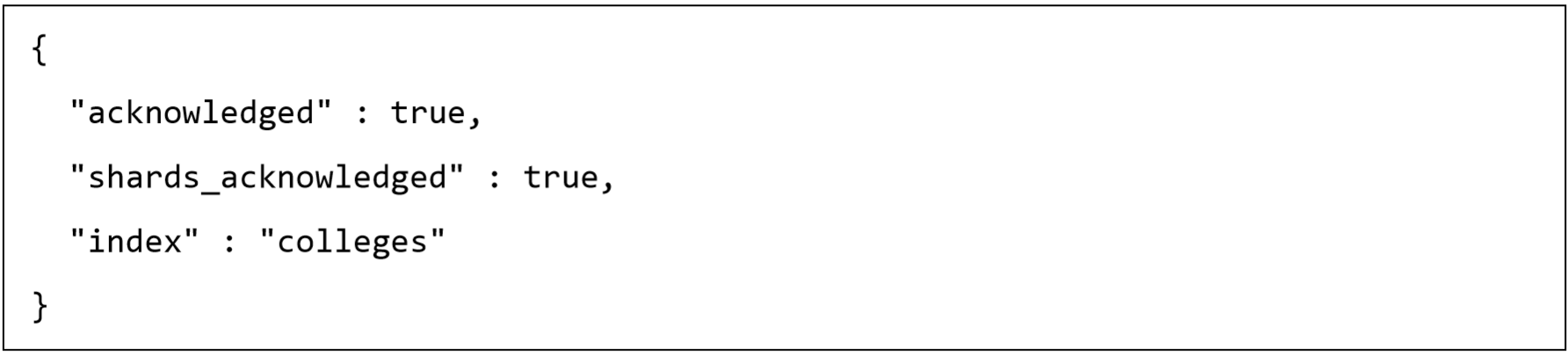
Khi chạy mã trên, chúng ta nhận được output như hình dưới đây:



Chúng ta cũng có thể thêm một số cài đặt cho lệnh trên:



Khi chạy mã trên, chúng ta nhận được output như hình dưới đây:



**Xóa index**

API này giúp xóa bất kỳ Index nào. Người dùng chỉ cần chuyển yêu cầu xóa với tên của Index cụ thể đó.

****

Có thể xóa tất cả các Index bằng cách sử dụng \_all hoặc \*.

**Nhận index**

API này có thể được gọi bằng cách chỉ gửi yêu cầu nhận đến một hoặc nhiều chỉ số. Điều này trả về thông tin về Index.



Khi chạy mã trên, chúng ta nhận được đầu ra như hình dưới đây:

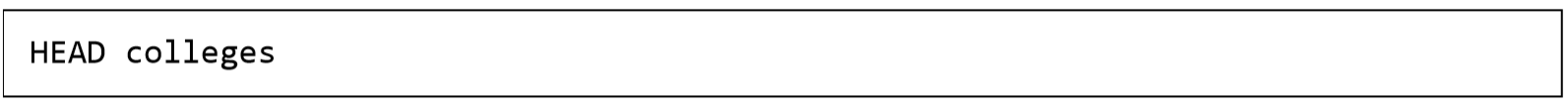




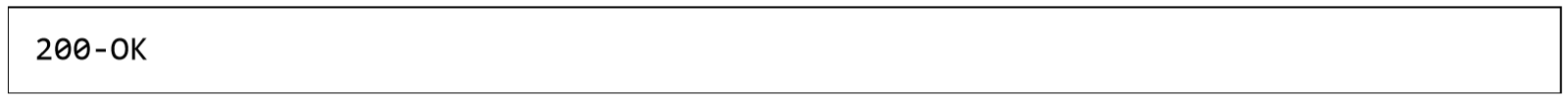
Có thể lấy thông tin của tất cả các chỉ số bằng cách sử dụng \_all hoặc \*.

**Index tồn tại**

Sự tồn tại của một Index có thể được xác định bằng cách gửi yêu cầu nhận tới Index đó. Nếu phản hồi HTTP là 200, nó tồn tại; nếu nó là 404, nó không tồn tại.

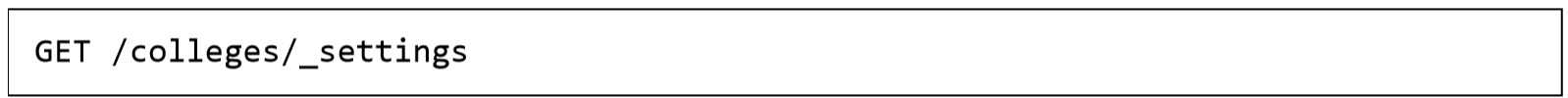


Khi chạy mã trên, chúng ta nhận được đầu ra như hình dưới đây:



**Cài đặt Index**

Có thể nhận cài đặt Index bằng cách chỉ thêm từ khóa \_sinstall ở cuối URL.



Khi chạy mã trên, chúng ta nhận được đầu ra như hình dưới đây:

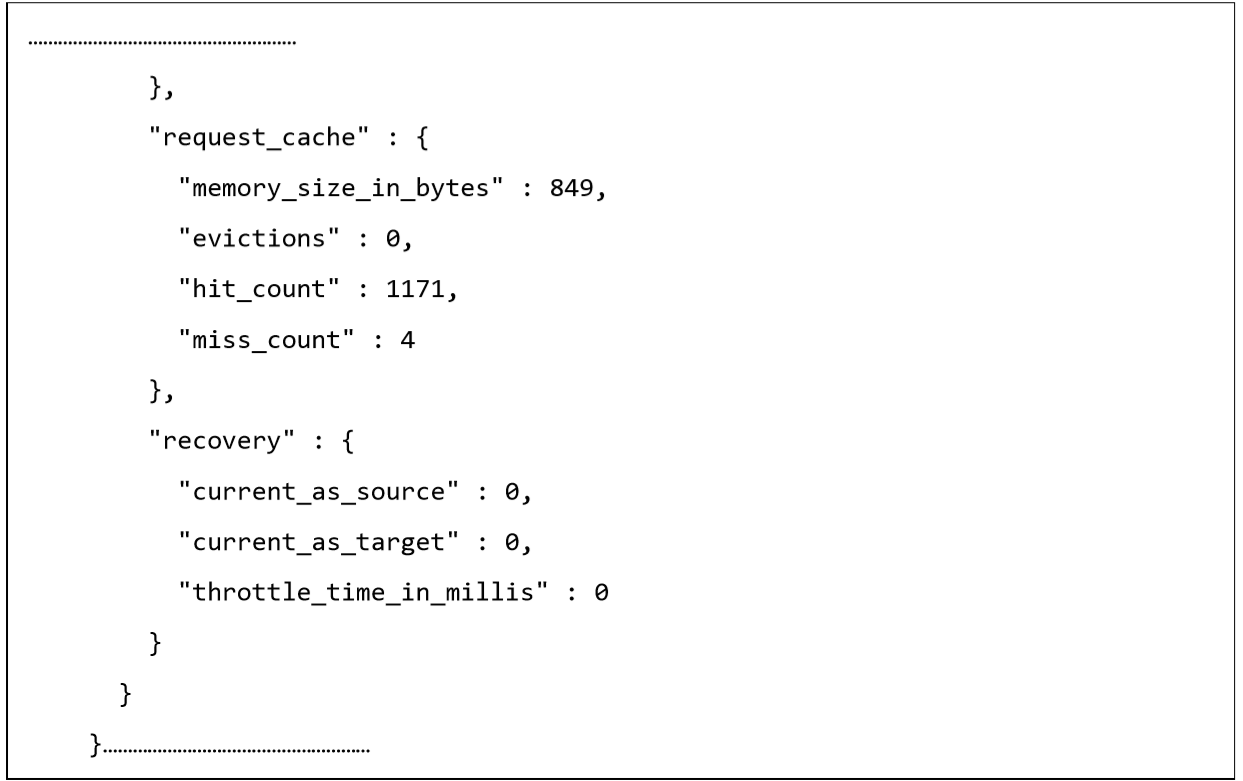


**Chỉ số thống kê**

API này giúp trích xuất số liệu thống kê về một Index cụ thể. Người dùng chỉ cần gửi yêu cầu nhận với URL Index và từ khóa “\_stats” ở cuối.



Khi chạy mã trên, chúng ta nhận được đầu ra như hình dưới đây:

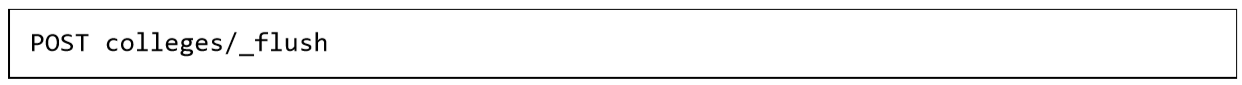


**Index Refresh**

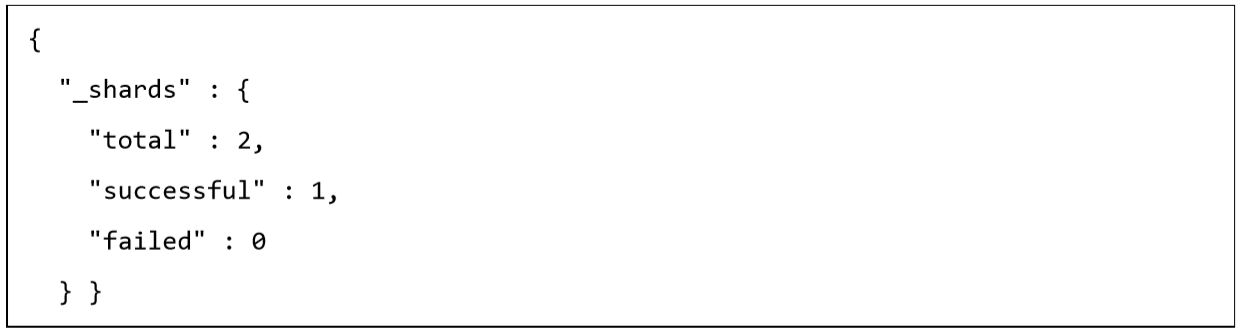
* Khi có một truy vấn yêu cầu index một tài liệu, nội dung đó sẽ được thêm vào translog và được lưu tạo vào buffer trên bộ nhớ trong. Khi tiến hành index refresh (mặc định mỗi giây một lần), tiến trình refresh sẽ tạo segment ngay trên bộ nhớ trong từ buffer trước đó. Tới lúc này, tài liệu cần index đã có thể tìm kiếm được.
* Sau đó, buffer trên bộ nhớ trong sẽ bị xoá đi. Dần dần quá trình này sẽ tạo ra một tập các segment, rồi các segment lại được gộp lại với nhau thành segment lớn hơn để đảm bảo việc sử dụng các tài nguyên của máy chủ (CPU, bộ nhớ, v.v...) được tối ưu.
* Quá trình index refresh là một hoạt động rất tốn kém, do đó nó thường được đặt lịch chạy định kỳ chứ không phải được kích hoạt ngay sau khi index.
* Nếu chúng ta cần index một lượng lớn tài liệu và không có nhu cầu gấp trong việc tìm kiếm ngay lập tức các tài liệu đó, chúng ta có thể tìm cách tối ưu hiệu năng của quá trình index bằng cách kéo giãn khoảng thời gian mà index refresh được thực hiện.

**Flush**

Quá trình xóa của một Index đảm bảo rằng bất kỳ dữ liệu nào hiện chỉ tồn tại trong nhật ký giao dịch cũng được duy trì vĩnh viễn trong Lucene. Điều này giúp giảm thời gian phục hồi vì dữ liệu đó không cần phải được lập chỉ mục lại từ nhật ký giao dịch sau khi Index Lucene được mở.



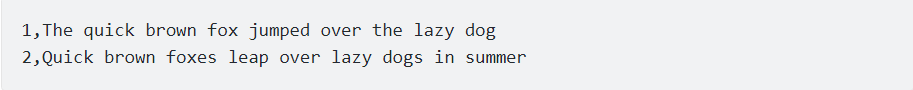
Output:



**Inverted Index**

Trong Elasticsearch , sử dụng một cấu trúc được gọi là inverted index . Nó được thiết kế để cho phép tìm kiếm full-text search.Khi sử dụng inverted index, nó có thể tìm kiếm thông qua terms như một binary tree (sử dụng thứ tự chữ cái) làm giảm thời gian tìm kiếm.Một điều quan trọng khi lưu trữ document đó là được quyết định cách tốt nhất để lưu trữ chúng giúp nâng cao tốc độ truy vấn. Khi thiết kế các giải pháp sử dụng ElasticSearch, điều đáng lưu tâm nhất khi lưu trữ document chính là: tôi sẽ truy vấn document này như thế nào? “First query” này tiếp cận với việc sử dụng cho tất cả các khả năng ES indexing để thực hiện truy vấn cực kỳ nhanh chóng. Cách thức của nó khá đơn giản, các văn bản được phân tách ra thành từng từ có nghĩa sau đó sẽ được map xem thuộc văn bản nào. Khi search tùy thuộc vào loại search sẽ đưa ra kết quả cụ thể.

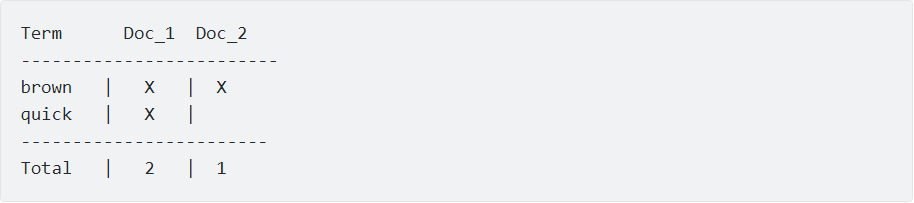
Ví dụ: Chúng ta có 2 văn bản cụ thể như sau:



Để tạo ra một inverted index, trước hết chúng ta sẽ phân chia nội dung của từng tài liệu thành các từ riêng biệt (chúng tôi gọi là terms), tạo một danh sách được sắp xếp của tất cả terms duy nhất, sau đó liệt kê tài liệu nào mà mỗi thuật ngữ xuất hiện. Kết quả như sau:

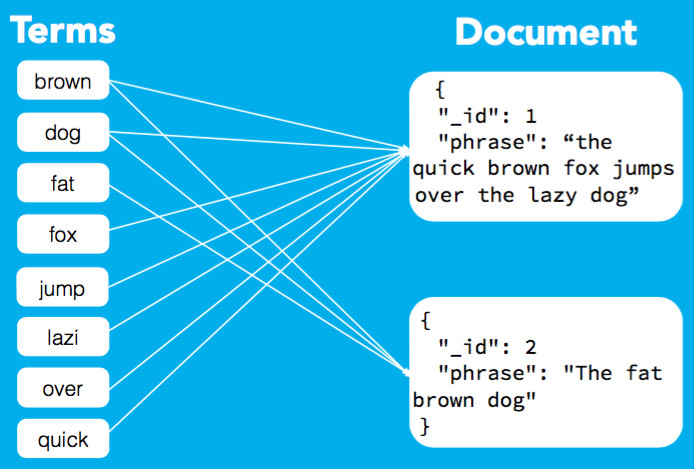


Bây giờ, nếu chúng ta muốn tìm kiếm màu quick brown, chúng ta chỉ cần tìm trong các tài liệu trong đó mỗi thuật ngữ có xuất xuất hiện hay không. Kết quả như sau:



Cả 2 đoạn văn bản đều thích hợp với từ khóa. Tuy nhiên có thể dễ dàng nhận ra rằng Doc\_1 chính xác hơn so với Doc\_2.

Hình ảnh minh họa:



# 

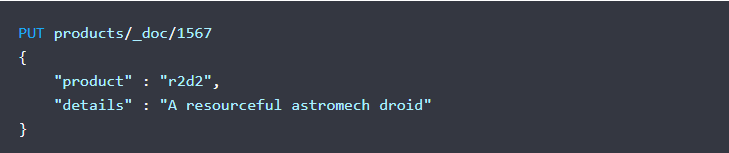
# **VI. Concurrency Control, Conflict**

## **1. Vấn đề xử lý đồng thời**

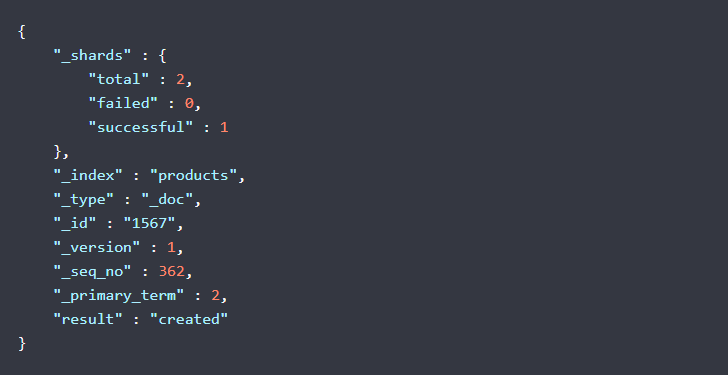
Elasticsearch được phân phối. Khi tài liệu được tạo, cập nhật hoặc xóa, phiên bản mới của tài liệu phải được sao chép sang các nút khác trong cụm. Elasticsearch cũng không đồng bộ và đồng thời, có nghĩa là các yêu cầu sao chép này được gửi song song và có thể đến đích của chúng ngoài chuỗi. Elasticsearch cần một cách để đảm bảo rằng phiên bản cũ hơn của tài liệu không bao giờ ghi đè lên phiên bản mới hơn.

Để đảm bảo phiên bản cũ hơn của tài liệu không ghi đè lên phiên bản mới hơn, mọi thao tác được thực hiện cho tài liệu đều được gán một số thứ tự bởi phân đoạn chính có tọa độ thay đổi. Số thứ tự được tăng lên với mỗi hoạt động và do đó các hoạt động mới hơn được đảm bảo có số thứ tự cao hơn các hoạt động cũ. Sau đó, Elasticsearch có thể sử dụng số thứ tự các thao tác để đảm bảo phiên bản tài liệu mới hơn không bao giờ bị ghi đè bởi một thay đổi có số thứ tự nhỏ hơn được gán cho nó.

Ví dụ, lệnh lập chỉ mục sau đây sẽ tạo một tài liệu và gán cho nó một số thứ tự ban đầu và thuật ngữ chính:



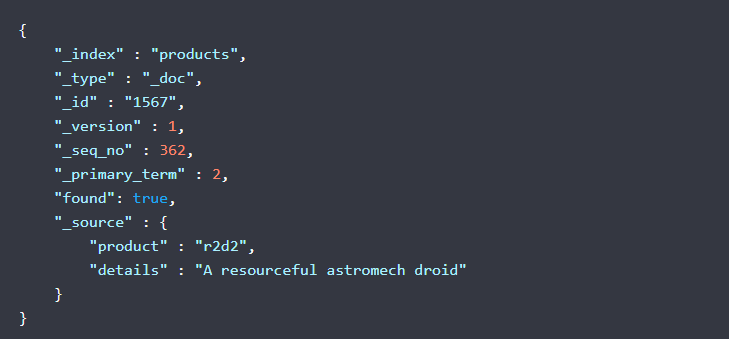
Chúng ta có thể thấy số thứ tự được chỉ định và thuật ngữ chính trong “\_seq\_no” và “\_primary\_term” các trường của phản hồi:



Elasticsearch theo dõi số thứ tự và thuật ngữ chính của hoạt động cuối cùng để thay đổi từng tài liệu mà nó lưu trữ. Số thứ tự và thuật ngữ chính được trả về trong “\_seq\_no” và “\_primary\_term” các trường trong phản hồi của [GET API](https://www.elastic.co/guide/en/elasticsearch/reference/6.7/docs-get.html):

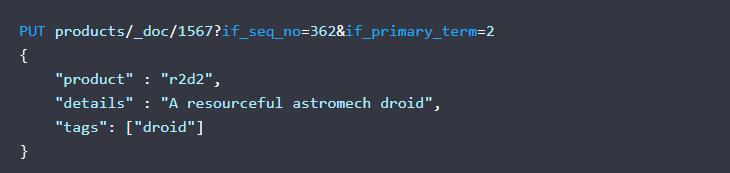


* Nhận về :



Số thứ tự và thuật ngữ chính xác định duy nhất một thay đổi. Bằng cách lưu ý số thứ tự và thuật ngữ chính được trả về, bạn có thể đảm bảo chỉ thay đổi tài liệu nếu không có thay đổi nào khác được thực hiện kể từ khi bạn truy xuất nó. Điều này được thực hiện bằng cách đặt if\_seq\_no và if\_primary\_term các tham số của [Index API](https://www.elastic.co/guide/en/elasticsearch/reference/6.7/docs-index_.html) hoặc [Delete API](https://www.elastic.co/guide/en/elasticsearch/reference/6.7/docs-delete.html) .

Ví dụ: cuộc gọi lập chỉ mục sau đây sẽ đảm bảo thêm thẻ vào tài liệu mà không mất bất kỳ thay đổi tiềm năng nào đối với mô tả hoặc bổ sung thẻ khác bởi API khác:

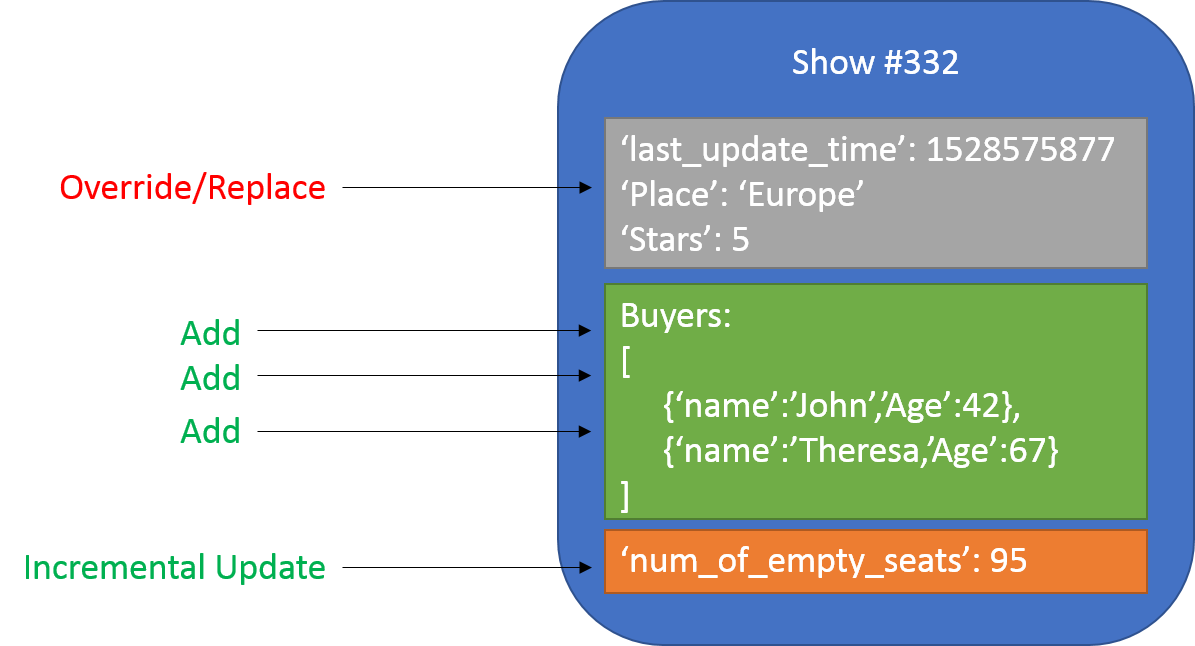


## **2. Tương tranh**

Elasticsearch sử dụng kiểm soát phiên bản nội bộ, để kiểm soát đồng thời lạc quan. Nó đảm bảo rằng chỉ một bản cập nhật được xử lý cùng một lúc cho một tài liệu. Nhưng điều đó không gây ra ảnh hưởng gì khi xử lý nhiều bản cập nhật cho cùng một tài liệu trong một số lượng lớn hoặc thậm chí tệ hơn - nhiều bản cập nhật cho cùng một tài liệu từ các bulks khác nhau.

ElasticSearch sử dụng cập nhật một phần. Trong trường hợp thêm tài liệu lồng nhau hoặc cập nhật gia tăng một số thuộc tính thì không xảy ra vấn đề gì. Trong trường hợp bản cập nhật ghi đè một số thuộc tính thì thứ tự không quan trọng, bởi vì lần ghi cuối cùng sẽ là bản chiến thắng.

Như trong hình minh họa này, có 3 nhóm khác nhau (xám, xanh lá cây, cam) trong các tài liệu, mỗi nhóm cập nhật một phần. Mặc dù các cập nhật bổ sung và gia tăng đồng thời vẫn ổn (không có xung đột có thể xảy ra), các yêu cầu ghi đè / thay thế đồng thời là nguy hiểm.



Giải pháp cho việc đó là sử dụng một số loại kiểm tra thời gian tức là chỉ cập nhật khi nó mới hơn.

# **VII. Access Control**

## **1. Xác thực người dùng**

Xác thực xác định một cá nhân. Để có quyền truy cập vào các tài nguyên bị hạn chế, người dùng phải chứng minh danh tính của họ, thông qua mật khẩu, thông tin đăng nhập hoặc một số phương tiện khác (thường được gọi là mã thông báo xác thực).

Elastic xác thực người dùng bằng cách xác định người dùng đằng sau các yêu cầu đánh vào cụm và xác minh rằng họ là người mà đã kê khai. Quá trình xác thực được xử lý bởi một hoặc nhiều dịch vụ xác thực được gọi là [*realms*](https://www.elastic.co/guide/en/elasticsearch/reference/7.5/realms.html).

Bạn có thể sử dụng hỗ trợ riêng để quản lý và xác thực người dùng hoặc tích hợp với các hệ thống quản lý người dùng bên ngoài như LDAP và Active Directory.

Các tính năng bảo mật Elastic stack cung cấp được built-in chẳng hạn như native, ldap, active\_directory, pki, file, saml, và oidc. Nếu không có built-in tích hợp nào đáp ứng nhu cầu của bạn, bạn cũng có thể xây dựng built-in tùy chỉnh của riêng mình và cắm nó vào Elastic stack. Khi các tính năng bảo mật được bật, tùy thuộc vào các lĩnh vực bạn đã định cấu hình, bạn phải đính kèm thông tin đăng nhập của người dùng với các yêu cầu được gửi tới Elasticsearch.

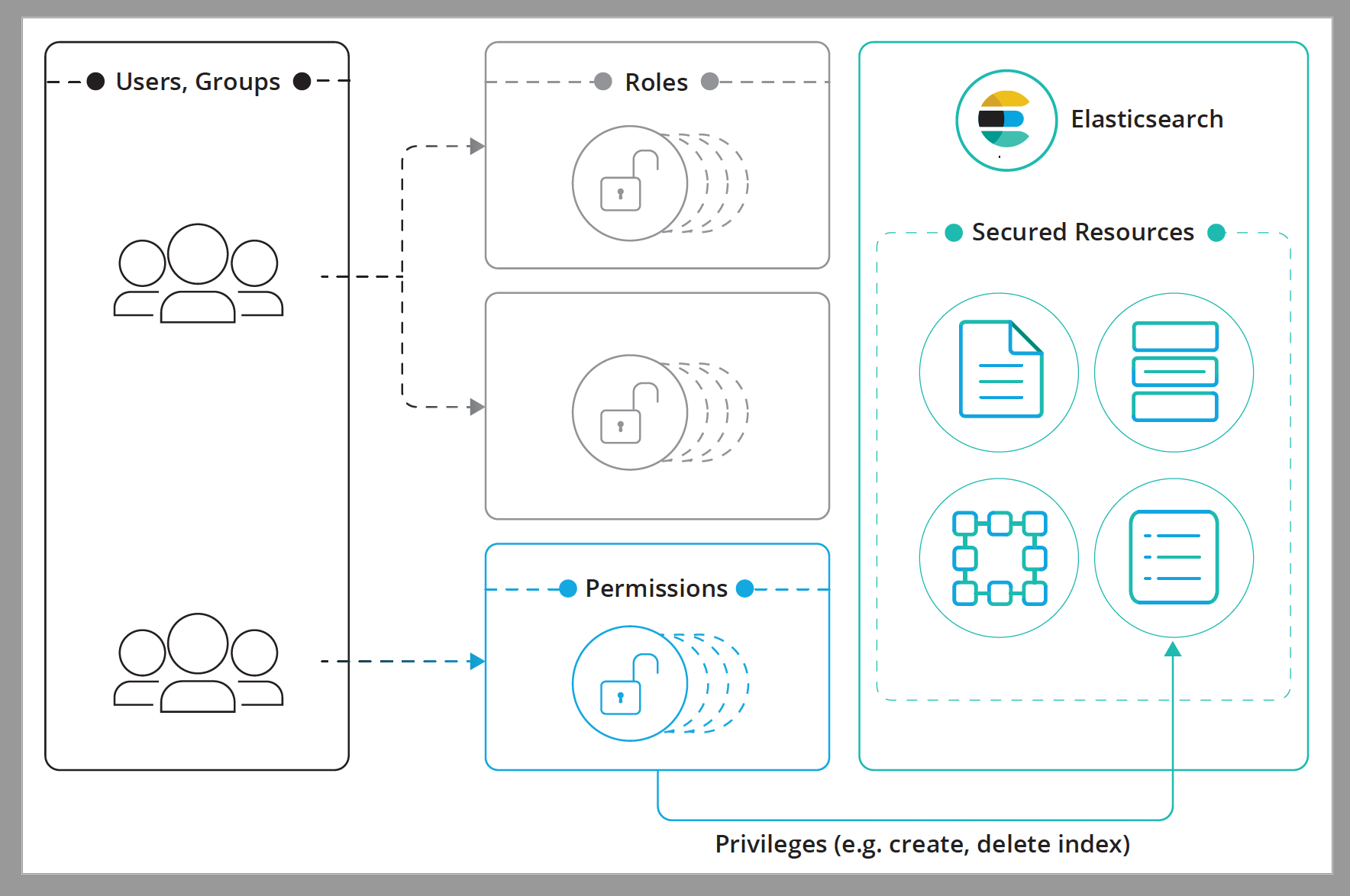
## **2. Kiểm soát truy cập dựa trên vai trò**

Hiện tại, Elastic stack có bảo mật X-Pack, một sản phẩm cung cấp bảo mật toàn diện cho Elasticsearch, bao gồm thông tin liên lạc được mã hóa, kiểm soát truy cập dựa trên vai trò.

Các tính năng bảo mật của Elastic stack thêm ủy quyền, đây là quá trình xác định xem người dùng đằng sau một yêu cầu đến có được phép thực hiện yêu cầu hay không. Quá trình này diễn ra sau khi người dùng được xác định và [xác thực](https://www.elastic.co/guide/en/elasticsearch/reference/7.5/setting-up-authentication.html) thành công.

Các tính năng bảo mật cung cấp cơ chế kiểm soát truy cập dựa trên vai trò (RBAC), cho phép bạn ủy quyền cho người dùng bằng cách gán đặc quyền cho vai trò và gán vai trò cho người dùng hoặc nhóm.

Quá trình ủy quyền xoay quanh các cấu trúc sau:



***Secured Resource***

Một nguồn tài nguyên mà truy cập bị hạn chế. Các chỉ số, bí danh, tài liệu, trường, người dùng và chính cụm Elasticsearch là tất cả các ví dụ về các đối tượng được bảo mật.

***Privilege***

Một nhóm được đặt tên gồm một hoặc nhiều hành động mà người dùng có thể thực thi đối với tài nguyên được bảo mật. Mỗi tài nguyên được bảo đảm có các đặc quyền riêng có sẵn. Ví dụ: read là một đặc quyền chỉ mục đại diện cho tất cả các hành động cho phép đọc dữ liệu được lập chỉ mục / lưu trữ.

***Permissions***

Một tập hợp một hoặc nhiều đặc quyền đối với tài nguyên được bảo đảm. Quyền có thể dễ dàng được mô tả bằng từ ngữ, đây là một vài ví dụ:

* read đặc quyền về products chỉ số
* manage đặc quyền trên cụm
* run\_as đặc quyền john người dùng
* read đặc quyền trên các tài liệu phù hợp với truy vấn X
* read đặc quyền trên credit\_card sân

***Role***

Một bộ quyền được đặt tên.

***User***

Người dùng xác thực.

***Group***

Một hoặc nhiều nhóm mà người dùng thuộc về. Các nhóm không được hỗ trợ trong một số lĩnh vực, chẳng hạn như các native, file hoặc PKI realms.

Một vai trò có một tên duy nhất và xác định một tập các quyền dịch thành đặc quyền trên các tài nguyên. Bạn có thể liên kết người dùng hoặc nhóm với số lượng vai trò tùy ý. Khi bạn ánh xạ vai trò thành các nhóm, vai trò của người dùng trong nhóm đó là sự kết hợp giữa các vai trò được gán cho nhóm đó và vai trò được gán cho người dùng đó. Tương tự, tổng số các quyền mà người dùng có được xác định bởi sự kết hợp của các quyền trong tất cả các vai trò của nó. Phương pháp gán vai trò cho người dùng khác nhau tùy thuộc vào lĩnh vực bạn sử dụng để xác thực người dùng.

## **3. Kiểm soát truy cập dựa trên thuộc tính**

Các tính năng bảo mật cũng cung cấp cơ chế kiểm soát truy cập dựa trên thuộc tính (ABAC), cho phép bạn sử dụng các thuộc tính để hạn chế quyền truy cập vào tài liệu trong các truy vấn và tổng hợp tìm kiếm. Ví dụ: bạn có thể gán các thuộc tính cho người dùng và tài liệu, sau đó thực hiện chính sách truy cập theo định nghĩa vai trò. Người dùng với vai trò đó chỉ có thể đọc một tài liệu cụ thể nếu họ có tất cả các thuộc tính bắt buộc.

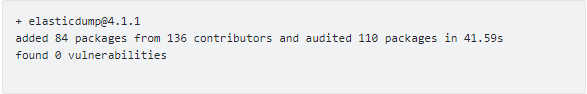
# **VIII. Backup, recovery, partition, sharding, replication, cluster**

## **1. Backup**

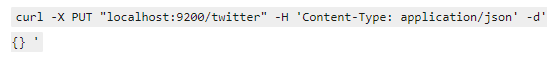
* Một Elasticsearch snapshot là một back up của cụm (cluster) Elasticsearch. Snapshot module cho phép bạn tạo nhiều snapshot của nhiều hơn một hoặc các chỉ số riêng lẻ hoặc một snapshot của toàn cụm. Snapshot của các chỉ số cụ thể hoặc toàn bộ cụm và lưu trữ trên hệ thống tệp được chia sẻ.
* Có nhiều loại kho khác nhau được hỗ trợ. Nếu bạn có một hệ thống tập tin chia sẻ, ví dụ:
* Một hệ thống tệp NFS có thể truy cập được bởi tất cả các nút Elasticsearch tại cùng một điểm gắn kết, sau đó bạn có thể sử dụng hệ thống đó để lưu trữ các chỉ số riêng lẻ cụ thể của mình hoặc toàn bộ cluster snapshot.
* Elasticsearch được thiết kế để chạy trong ác môi trường khác nhau, cũng hỗ trợ các snapshot và restore module khu lưu trữ đám mây khác nhau như:
* AWS
* Google Cloud Storage
* Azure Cloud etc
* Cài đặt:



* Sau khi cài đặt thành công:



* Tạo Index:



* Tạo mappings:



* Tạo data:

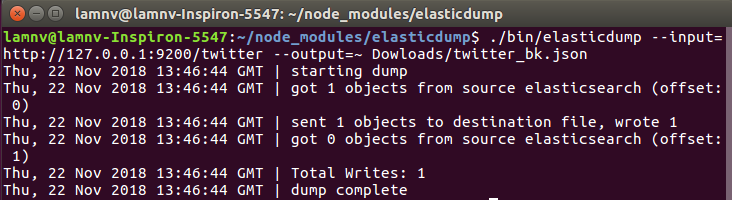


* Backup:
* Bước 1:



* Bước 2: Export ra file json dữ liệu từ 1 index(twitter)



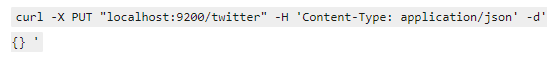


## **2. Recovery**

* Trả về thông tin về các phục hồi shard đang diễn ra và đã hoàn thành, tương tự như API index recovery. Đây là một cái nhìn nhỏ gọn hơn về API index recovery JSON.
* Recovery phân đoạn là quá trình đồng bộ phân đoạn bản sao từ phân đoạn chính. Sau khi hoàn thành, bản sao có sẵn để tìm kiếm.
* Recovery tự động xảy ra trong các quá trình sau:
* Node khởi động hoặc thất bại. Đây được gọi là local store recovery (khôi phục địa phương).
* Primary shard replication.
* Di dời một mảnh vỡ đến một node khác trong cùng một cluster.
* Snapshot restoration.
* Tham số đường dẫn:
* Index: (Tùy chọn, chuỗi) Danh sách được phân tách bằng dấu phẩy hoặc biểu thức ký tự đại diện của các tên chỉ mục được sử dụng để giới hạn yêu cầu.
* Tham số truy vấn:
* Active\_only: (Tùy chọn, boolean) Nếu “true”, phản hồi chỉ bao gồm các phục hồi phân đoạn đang diễn ra. Mặc định là “false”.
* Bytes: (Tùy chọn, đơn vị đo kích thước byte) Đơn vị được sử dụng để hiển thị giá trị byte.
* Detailed: (Tùy chọn, boolean) Nếu “true”, phản hồi bao gồm thông tin chi tiết về phục hồi phân đoạn. Mặc định là “false”.
* Format: (Tùy chọn, chuỗi) Phiên bản ngắn của tiêu đề chấp nhận HTTP. Các giá trị hợp lệ bao gồm JSON, YAML, …
* Help
* Index
* Recovery:
* Tiếp theo phần backup ở trên. Xóa index đã backup dữ liệu:



* Tạo index:

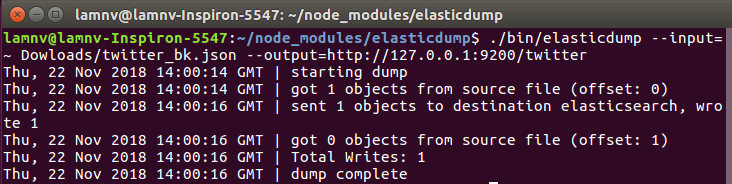


* Tạo mappings:



* Phục hồi dữ liệu đã sao lưu:





## **3. Partition**

* Trong Elasticsearch, type được định nghĩa cho các tài liệu có một tập các trường chung. Nó là một logical category/ partition của một index có ngữ nghĩa học hoàn toàn tùy thuộc vào người dùng. Bạn cũng có thể xác định nhiều type trong một index.
* Type: Có thể có nhiều Type trong cùng một Index. Ví dụ, một ứng dụng thương mại điện tử có thể sử dụng các sản phẩm cũ trong một Type và những sản phẩm mới trong một Type khác của cùng Index. Một Index có thẻ có nhiều Type, giống như có nhiều Table trong một Database.

## **4. Sharding**

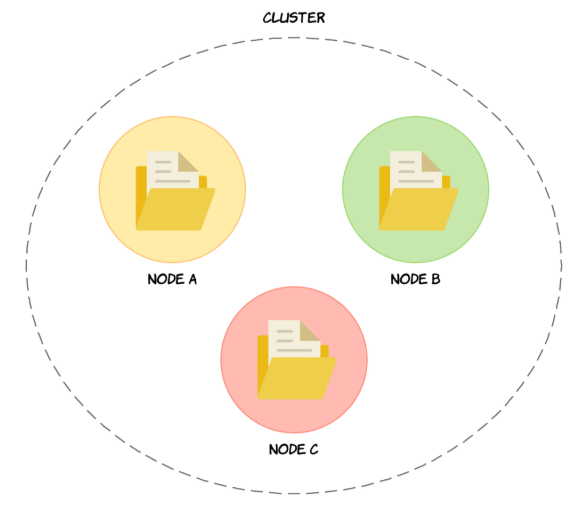
* Mỗi một index của Elasticsearch có một hay nhiều shard(tùy cấu hình, mặc định là 5). (Elasticsearch sử dụng index để lưu trữ và xử lý dữ liệu, nó tương tự một cơ sở dữ liệu thông thường).Các shard này tồn tại trên các node khác nhau. Nội dung về shard này liên quan nhiều đến kiến trúc của Elasticsearch, cũng là một phần quan trọng giúp cho Elasticsearch có thể thực hiện công việc tìm kiếm một cách nhanh chóng.
* Shard cũng có hai loại: “Primary” và “Replica”, mỗi một primary shard sẽ có một vài(có thể là không có) shard replica(mặc định là 1). Elasticsearch sẽ đảm bảo là primary shard và replica shard sẽ tồn tại trên các node khác nhau.
* Một đặc điểm rất quan trọng, đó là số lượng primary shard cho một index không thế thay đổi được sau khi index đã được tại. Do đó, trong thực tế, chúng ta sẽ phải cân nhắc rất nhiều yếu tố để quyết định xem cần bao nhiêu primary shard cho mỗi index. Trừ khi bạn muốn tạo lại index và thêm dữ liệu vào từ đầu.
* Một shard thực chất là một Lucene index(Elasticsearch hoạt động dựa trên Lucene của Apache), đó mới thực sự là nơi lưu dữ liệu, và bản thân shard cũng là một search engine. Là một Lucene index, nên một shard lại được tạo nên từ nhiều segment và mỗi segment là một inverted index với đầy đủ chức năng.
* Segment cho phép Lucene có thể thêm tài liệu vào index. Với mỗi yêu cầu tìm kiếm, mọi segment có trong một index sẽ được sử dụng và mỗi segment sẽ hoạt động độc lập. Nếu có nhiều segment thì performance của thao tác bị giảm đi còn nếu ít segment thì không tận dụng được khả năng của máy chủ. Vì thế Lucene gộp các segment nhỏ thành segment lớn. Tuy nhiên nếu không cẩn thận Elasticsearch có thể rơi vào sai lầm và bỏ phí nhiều tài nguyên dẫn đến performance giảm.

## **5. Replication**

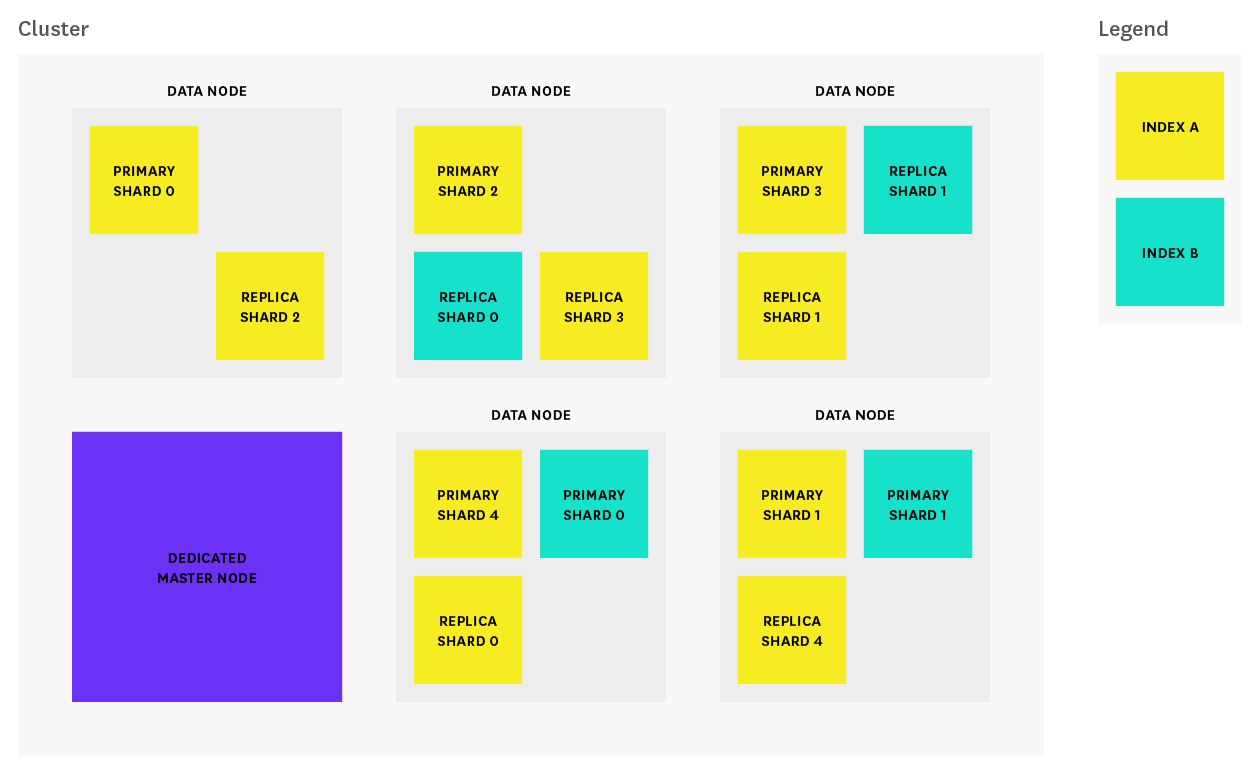
* Replica Shard là nơi lưu trữ dữ liệu nhân bản của Primary Shard.
* Replica Shard có vai trò đảm bảo tính toàn vẹn của dữ liệu khi primary shard xảy ra vấn đề. Ngoài ra replica shard có thể giúp tăng cường tốc độ tìm kiếm vì chúng ta có thể setup lượng replica shard nhiều hơn mặc định của Elasticsearch.

## **6. Cluster**

* Node là cơ sở trong kiến trúc server của Elasticsearch, và nhiều node hợp với nhau tạo thành cluster.



* Tập hợp các node hoạt động cùng với nhau tạo thành một cluster, mỗi node trong cluster chứa một phần dữ liệu của cluster đó (và toàn bộ dữ liệu của một cluster sẽ được chia ra cho các node).
* Mỗi node sẽ tham gia vào quá trình index và tìm kiếm dữ liệu trong cluster. Các truy vấn của người dùng đề thông qua HTTP request, và cluster sẽ chuyển những truy vấn đó cho các node của nó.
* Các node trong cùng một cluster hiểu được các node khác nên có thể chuyển tiếp truy vấn của người dùng cho một node khác. Việc này được thực hiện nội bộ bên trong cluster.
* Node có 3 loại khác nhau: master, data và client. Một cluster sẽ tự động chọn ra một node làm master từ các node của nó. Node master sẽ có nhiệm vụ điều phối công việc của cluster, ví dụ: Phân bố các shard, tạo/ xóa index,...Chỉ có node master mới có khả năng cập nhật trạng thái của cluster.
* Node data là node mà các shard được phân bố, có nhiệm vụ thực hiện các công việc như index tài liệu, tìm kiếm và thống kê. Node client hoạt động như một load balancer có nhiệm vụ định tuyến các truy vấn của người dùng như index hay tìm kiếm. Một cluster có thể không có node client.
* Kiến trúc của Elasticsearch:



* Cluster và node được định danh bởi tên (duy nhất). Mặc định, tên của cluster sẽ là “elasticsearch” còn node được gán tên là một UUID (Universally Unique Identifier - Mã định danh toàn cầu). Một cluster về mặt kỹ thuật là có bao nhiêu node cũng được, thậm chí 1 node cũng không sao. Kiến trúc Elasticsearch giúp nó có khả năng mở rộng theo chiều ngang một cách nhanh chóng mà không hề cần một sự thay đổi nào trong code.

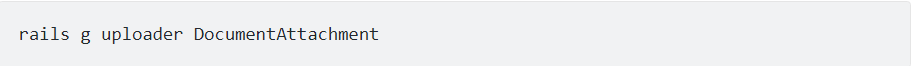
# **IX. Cài đặt, demo**

**Xây một ứng dụng nho nhỏ trên Rails**

Tiếp theo, chúng ta sẽ xây dựng một chức năng tìm kiếm nội dung file trên rails để kiểm chứng khả năng của Elasticsearch. Giữa Elasticsearch và rails chưa có một mối quan hệ nào (hay có thể nói là chưa thể giao tiếp với nhau), chính vì vậy chúng ta phải cài thêm gem "tire" Để upload file bạn muốn tìm kiếm thì bạn cũng cần gem "carrierwave" Cuối cùng là chạy lệnh:



Tạo thư mục document chứa file sẽ upload lên:



Tạo model chứa file:



Upload file bất kỳ vào bảng Document, ví dụ 2 file test\_1.pdf và test\_2.pdf :



Trong file document.rb:



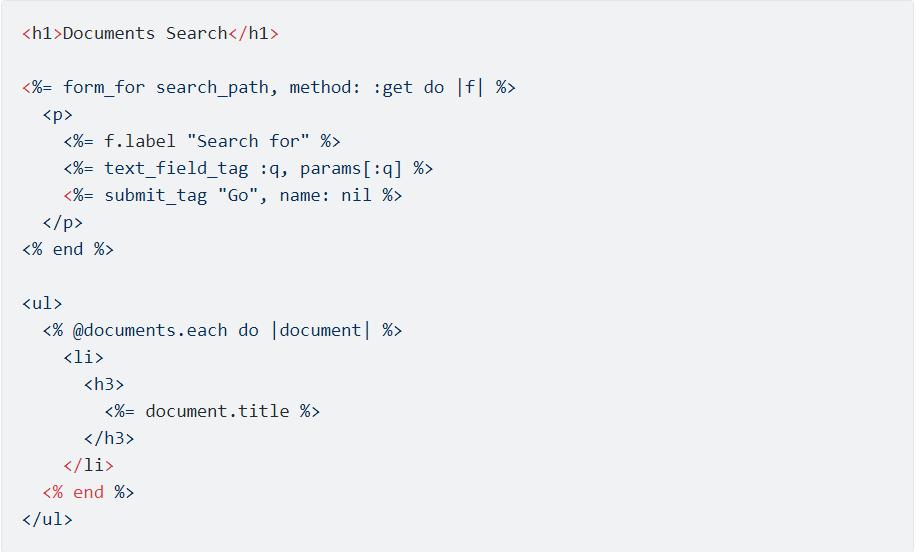
Tạo controller search:



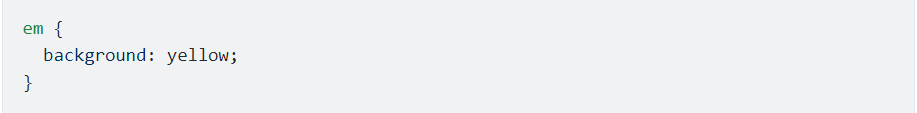
Trong file search\_controller.rb:



Tạo mới một file ở app/views/search/search.html.erb:



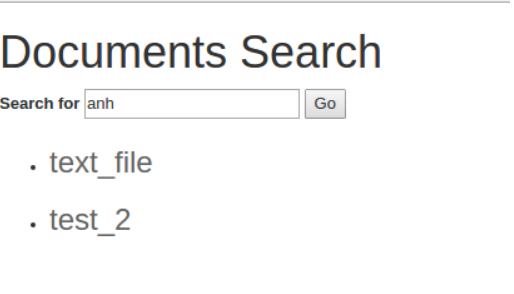
Thêm vào file app/assets/stylesheets/search.css.scss:



Cấu hình file config/routes.rb:



Vào đường dẫn <http://localhost:3000/search> và tiến hành tìm kiếm nội dung dựa trên 2 file test\_1.pdf và test\_2.pdf, ở đây tôi có sử dụng thêm file text\_file.txt:



Phân công công việc:

* Nguyễn Tuấn Anh: Introduction; Backup, recovery, partition, sharding, replication, cluster; Chỉnh sửa tổng quan.
* Cao Văn Cường: Concurrency Control, Conflict; Access Control; Chỉnh sửa tổng quan.
* Bùi Quang Minh: Data model/Type ; Query language; Chỉnh sửa tổng quan.
* Vũ Tuấn An: Integrity model; File organization, Index; Demo.