**TRƯỜNG ĐẠI HỌC THỦY LỢI**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO BÀI TẬP LỚN**

**MÔN PHÂN TÍCH DỮ LIỆU LỚN**

**ĐỀ TÀI: NGHIÊN CỨU VÀ ỨNG DỤNG ELASTICSEARCH TRONG LƯU TRỮ VÀ PHÂN TÍCH DỮ LIỆU LỚN VỀ TỐC ĐỘ GIAO THÔNG THỜI GIAN THỰC**

|  |  |  |
| --- | --- | --- |
| **Nhóm thực hiện:** | **12 – 64HTTT3** |  |
| **Thành viên nhóm:** | **Phan Thị Phương Anh** | **-** **2251061711** |
|  | **Đào Việt Chung** | **- 2251061728** |
|  | **Đặng Hữu Trưởng** | **- 2251061910** |
| **Giảng viên hướng dẫn:** | **ThS. Nguyễn Đắc Phương Thảo** |  |

**Hà Nội, 2025**

**MỤC LỤC**

[**CHƯƠNG I. TỔNG QUAN ĐỀ TÀI** 5](#_Toc213012924)

[**1.1. Bối cảnh và lý do chọn đề tài** 5](#_Toc213012925)

[**1.2. Mục tiêu nghiên cứu** 6](#_Toc213012926)

[**1.3. Phạm vi và đối tượng nghiên cứu** 6](#_Toc213012927)

[1.3.1. Đối tượng nghiên cứu 6](#_Toc213012928)

[1.3.2. Phạm vi nghiên cứu 7](#_Toc213012929)

[**1.4. Phương pháp nghiên cứu** 7](#_Toc213012930)

[**CHƯƠNG II. CƠ SỞ LÝ THUYẾT VỀ ELASTICSEARCH** 8](#_Toc213012931)

[**2.1. Tổng quan về Elasticsearch** 8](#_Toc213012932)

[**2.2. Kiến trúc tổng thể của hệ thống** 9](#_Toc213012933)

[2.2.1. Cluster 9](#_Toc213012934)

[2.2.2. Node và vai trò của các loại node 10](#_Toc213012935)

[2.2.3. Index, Document, Shard, Replica 11](#_Toc213012936)

[**2.3. Cơ chế lập chỉ mục và tìm kiếm** 13](#_Toc213012937)

[2.3.1. Quá trình Indexing 13](#_Toc213012938)

[2.3.2. Quá trình Searching 14](#_Toc213012939)

[**2.4. Cơ chế phân tán và chịu lỗi** 15](#_Toc213012940)

[2.4.1. Phân phối shard giữa các node 15](#_Toc213012941)

[2.4.2. Cơ chế replicate dữ liệu 16](#_Toc213012942)

[2.4.3. Bầu chọn master node 17](#_Toc213012943)

[**2.5. Ưu điểm, hạn chế và phạm vi ứng dụng của Elasticsearch** 18](#_Toc213012944)

[2.5.1. Ưu điểm 18](#_Toc213012945)

[2.5.2. Hạn chế 19](#_Toc213012946)

[2.5.3. Phạm vi ứng dụng 19](#_Toc213012947)

[**CHƯƠNG III. TRIỂN KHAI THỬ NGHIỆM HỆ THỐNG PHÂN TÁN** 20](#_Toc213012948)

[**3.1. Mục tiêu triển khai** 20](#_Toc213012949)

[**3.2. Mô hình hệ thống** 20](#_Toc213012950)

[3.2.1. Kiến trúc tổng thể cluster 20](#_Toc213012951)

[3.2.2. Vai trò của từng node 21](#_Toc213012952)

[**3.3. Cấu hình hệ thống** 22](#_Toc213012953)

[3.3.1. Thiết lập các node Elasticsearch 22](#_Toc213012954)

[3.3.2. Cấu hình bảo mật và kết nối Kibana 23](#_Toc213012955)

[**3.4. Thiết kế chỉ mục và phân shard** 25](#_Toc213012956)

[3.4.1. Nguyên tắc thiết kế 25](#_Toc213012957)

[3.4.2. Cấu hình chỉ mục 26](#_Toc213012958)

[**3.5. Nạp dữ liệu và xử lý (Data Ingestion)** 27](#_Toc213012959)

[3.5.1. Mô tả tập dữ liệu gốc 27](#_Toc213012960)

[3.5.2. Tiền xử lí dữ liệu 29](#_Toc213012961)

[3.5.3. Nạp dữ liệu 35](#_Toc213012962)

[**3.6. Thực hiện truy vấn và phân tích dữ liệu** 37](#_Toc213012963)

[3.6.1. Truy vấn tốc độ trung bình theo thời gian (ngày) 37](#_Toc213012964)

[3.6.2. Truy vấn tốc độ trung bình theo quận (borough) 38](#_Toc213012965)

[3.6.3. Truy vấn tỷ lệ đóng góp theo quận (số bản ghi) 39](#_Toc213012966)

[3.6.4. Truy vấn tốc độ trung bình toàn thành phố 40](#_Toc213012967)

[3.6.5. Truy vấn tốc độ & thời gian di chuyển theo tháng 41](#_Toc213012968)

[3.6.6. Truy vấn thời gian di chuyển trung bình theo quận 42](#_Toc213012969)

[**3.7. Thử nghiệm khả năng chịu lỗi của cluster** 43](#_Toc213012970)

[3.7.1. Bối cảnh hệ thống thử nghiệm 44](#_Toc213012971)

[3.7.2. Thử nghiệm mất một Data Node (node-2 hoặc node-3) 44](#_Toc213012972)

[3.7.3. Thử nghiệm mất nhiều node (hai node đồng thời) 46](#_Toc213012973)

[3.7.4. Quan sát bổ sung trong Kibana 48](#_Toc213012974)

[3.7.5. Kết luận 49](#_Toc213012975)

[**3.8. Trực quan hóa dữ liệu bằng Kibana** 50](#_Toc213012976)

[3.8.1. Biểu đồ tốc độ trung bình theo thời gian 50](#_Toc213012977)

[3.8.2. Biểu đồ so sánh tốc độ trung bình giữa các quận 51](#_Toc213012978)

[3.8.3. Biểu đồ tỷ lệ đóng góp theo quận 52](#_Toc213012979)

[3.8.4. Biểu đồ tốc độ trung bình toàn thành phố 53](#_Toc213012980)

[3.8.5. Biểu đồ tương quan giữa tốc độ và thời gian di chuyển theo thời gian 54](#_Toc213012981)

[3.8.6. Biểu đồ thời gian di chuyển trung bình theo quận 55](#_Toc213012982)

[**3.9. Đánh giá kết quả triển khai** 57](#_Toc213012983)

[**CHƯƠNG IV. KẾT LUẬN** 57](#_Toc213012984)

[**4.1. Kết luận chung** 57](#_Toc213012985)

[**4.2. Những kết quả nổi bật** 58](#_Toc213012986)

[**4.3. Hạn chế của nghiên cứu** 59](#_Toc213012987)

[**4.4. Kiến nghị và hướng phát triển tiếp theo** 59](#_Toc213012988)

[**TÀI LIỆU THAM KHẢO** 60](#_Toc213012989)

# **CHƯƠNG I. TỔNG QUAN ĐỀ TÀI**

## **1.1. Bối cảnh và lý do chọn đề tài**

Trong bối cảnh chuyển đổi số và Cách mạng công nghiệp 4.0 đang diễn ra mạnh mẽ trên toàn cầu, dữ liệu đã trở thành một trong những tài nguyên quý giá nhất của kỷ nguyên số. Cùng với sự phát triển của Internet, điện toán đám mây, trí tuệ nhân tạo và Internet vạn vật, khối lượng dữ liệu được tạo ra từ các hoạt động hàng ngày của con người, máy móc và hệ thống cảm biến đang tăng theo cấp số nhân. Theo các thống kê của IDC, đến năm 2025, thế giới sẽ tạo ra hơn 175 zettabyte dữ liệu, trong đó phần lớn đến từ các thiết bị kết nối và cảm biến thông minh.

Trong số các lĩnh vực đang khai thác dữ liệu lớn để tối ưu hóa vận hành, giao thông thông minh là một trong những lĩnh vực tiên phong. Ở các đô thị lớn như New York, Tokyo, Seoul, hay Singapore, hàng triệu phương tiện giao thông di chuyển liên tục mỗi ngày tạo ra một lượng thông tin khổng lồ về tốc độ di chuyển, mật độ giao thông, thời gian di chuyển và tình trạng tắc nghẽn. Việc thu thập và phân tích dữ liệu giao thông theo thời gian thực giúp các cơ quan quản lý theo dõi tình trạng lưu thông, phát hiện các điểm ùn tắc, tối ưu hóa đèn tín hiệu, đồng thời hỗ trợ người dân lựa chọn lộ trình tối ưu.

Tuy nhiên, bài toán xử lý dữ liệu giao thông thời gian thực đặt ra nhiều thách thức về mặt kỹ thuật. Dữ liệu thu thập được thường đa dạng về định dạng – có thể là dữ liệu cảm biến, dữ liệu GPS, dữ liệu video, hay dữ liệu văn bản mô tả sự cố. Ngoài ra, dữ liệu được sinh ra với tốc độ rất cao và có dung lượng khổng lồ, trong khi yêu cầu độ trễ thấp để phục vụ ra quyết định gần như tức thời. Các hệ quản trị cơ sở dữ liệu truyền thống (RDBMS) khó có thể đáp ứng được yêu cầu này do hạn chế về khả năng mở rộng và hiệu năng truy vấn.

Chính vì vậy, sự ra đời của các công cụ xử lý dữ liệu lớn như Hadoop, Spark, Kafka, Cassandra và Elasticsearch đã mở ra hướng tiếp cận mới. Trong đó, Elasticsearch nổi bật với khả năng lưu trữ, tìm kiếm và phân tích dữ liệu phi cấu trúc theo thời gian thực. Được xây dựng trên nền tảng Apache Lucene, Elasticsearch là một hệ thống tìm kiếm phân tán mã nguồn mở có khả năng mở rộng linh hoạt, hỗ trợ lưu trữ dữ liệu ở quy mô hàng tỷ bản ghi, đồng thời cho phép truy vấn, tổng hợp và trực quan hóa dữ liệu chỉ trong vài mili-giây. Đặc biệt, khi được kết hợp với Kibana, Logstash và Beats, Elasticsearch trở thành trung tâm của bộ công cụ Elastic Stack – một nền tảng mạnh mẽ phục vụ phân tích dữ liệu lớn theo thời gian thực trong nhiều lĩnh vực.

Trên thế giới, Elasticsearch đã được ứng dụng rộng rãi trong phân tích log hệ thống, giám sát an ninh mạng, quản lý dữ liệu IoT, và đặc biệt là trong các hệ thống giao thông thông minh. Dữ liệu về tốc độ giao thông là một ví dụ điển hình cho dạng dữ liệu thời gian thực có quy mô lớn và đòi hỏi khả năng xử lý tức thì. Thành phố New York cung cấp bộ dữ liệu mở “NYC Real-Time Traffic Speed Data” chứa thông tin về tốc độ trung bình, thời gian di chuyển, tình trạng giao thông trên hàng ngàn đoạn đường, được cập nhật theo từng phút. Đây là nguồn dữ liệu quý giá để nghiên cứu và thử nghiệm các giải pháp phân tích dữ liệu giao thông đô thị.

Xuất phát từ thực tế đó, nhóm tác giả quyết định lựa chọn đề tài: “Nghiên cứu và ứng dụng Elasticsearch trong lưu trữ và phân tích dữ liệu lớn về tốc độ giao thông thời gian thực”.

## **1.2. Mục tiêu nghiên cứu**

Mục tiêu của đề tài là nghiên cứu, triển khai và đánh giá công cụ Elasticsearch trong việc lưu trữ, tìm kiếm và phân tích dữ liệu lớn về tốc độ giao thông thời gian thực, thông qua bộ dữ liệu NYC Real-Time Traffic Speed Data. Đề tài hướng đến việc xây dựng một mô hình thử nghiệm hệ thống Elasticsearch phân tán có khả năng thu thập, lưu trữ, xử lý và trực quan hóa dữ liệu, từ đó đánh giá hiệu quả của công cụ trong việc xử lý dữ liệu lớn và phục vụ phân tích giao thông thông minh.

Cụ thể, đề tài tập trung vào các mục tiêu sau:

* Tìm hiểu lý thuyết, cấu trúc và cơ chế hoạt động của Elasticsearch trong hệ thống lưu trữ phân tán.
* Triển khai mô hình cụm (cluster) Elasticsearch gồm 3 node trên môi trường Windows local, nhằm mô phỏng khả năng phân tán dữ liệu và cơ chế chịu lỗi của hệ thống.
* Thực hiện nạp và xử lý bộ dữ liệu thực tế NYC Real-Time Traffic Speed Data.
* Xây dựng hệ thống tìm kiếm và phân tích tốc độ giao thông theo địa điểm, thời gian và khu vực thông qua Kibana Dashboard.
* Đánh giá hiệu năng, độ tin cậy và khả năng mở rộng của hệ thống Elasticsearch.

## **1.3. Phạm vi và đối tượng nghiên cứu**

### 1.3.1. Đối tượng nghiên cứu

Đối tượng nghiên cứu của đề tài là hệ thống Elasticsearch và khả năng ứng dụng của nó trong lưu trữ, tìm kiếm và phân tích dữ liệu lớn theo thời gian thực. Nghiên cứu tập trung làm rõ kiến trúc, cơ chế xử lý phân tán và hiệu năng truy vấn của Elasticsearch trong các bài toán yêu cầu tốc độ cao và dung lượng dữ liệu lớn. Bên cạnh đó, đề tài triển khai mô hình phân tích dữ liệu tốc độ giao thông tại New York thông qua các bước thu thập, lập chỉ mục, truy vấn và trực quan hóa dữ liệu, hướng tới phân tích mô tả để nhận diện xu hướng lưu thông giữa các khu vực. Mục tiêu cuối cùng là đánh giá tính phù hợp, hiệu quả và khả năng mở rộng của Elasticsearch trong các hệ thống Big Data hiện đại.

### 1.3.2. Phạm vi nghiên cứu

Đề tài được triển khai thử nghiệm trên môi trường Windows local, thiết lập một cụm (cluster) Elasticsearch gồm ba node để mô phỏng khả năng phân tán dữ liệu, mở rộng quy mô và chịu lỗi của hệ thống. Cấu hình này giúp người thực hiện quan sát trực tiếp quá trình phân phối dữ liệu, sao lưu bản sao (replica) và quản lý hoạt động của cluster trong môi trường phân tán thực tế.

Nguồn dữ liệu sử dụng trong đề tài là bộ dữ liệu NYC Real-Time Traffic Speed Data, được thu thập từ nền tảng Kaggle. Bộ dữ liệu này phản ánh chi tiết tốc độ di chuyển và tình trạng giao thông của thành phố New York trong từng khoảng thời gian. Đây là bộ dữ liệu phù hợp để kiểm chứng khả năng xử lý và phân tích dữ liệu có tính thời gian thực của Elasticsearch.

Phạm vi nội dung nghiên cứu tập trung vào các thao tác cốt lõi trong quá trình xử lý dữ liệu với Elasticsearch, bao gồm:

* Lập chỉ mục (indexing) dữ liệu giao thông đã được tiền xử lý;
* Truy vấn (querying) và tổng hợp (aggregation) dữ liệu để phân tích xu hướng;
* Trực quan hóa (visualization) kết quả phân tích thông qua công cụ Kibana;
* Đánh giá khả năng vận hành và hiệu năng của hệ thống trong việc xử lý dữ liệu lớn.

Đề tài không đi sâu vào các nội dung tích hợp hệ thống trí tuệ nhân tạo hay các mô hình học máy, mà chỉ dừng lại ở phân tích mô tả – nhằm nhận diện, tổng hợp và biểu diễn thông tin hiện có trong dữ liệu giao thông. Cách tiếp cận này giúp đảm bảo phạm vi nghiên cứu phù hợp với mục tiêu ứng dụng và điều kiện thực nghiệm, đồng thời tạo nền tảng cho các hướng mở rộng trong tương lai như phân tích dự báo hoặc ứng dụng học máy trong giao thông thông minh.

## **1.4. Phương pháp nghiên cứu**

Đề tài kết hợp giữa nghiên cứu lý thuyết và thực nghiệm mô phỏng nhằm đánh giá khả năng ứng dụng của Elasticsearch trong lưu trữ và phân tích dữ liệu lớn về tốc độ giao thông thời gian thực. Trên cơ sở nghiên cứu cơ chế lập chỉ mục, truy vấn, lưu trữ phân tán và khả năng chịu lỗi, nhóm đã triển khai cụm Elasticsearch ba node trên môi trường Windows local để mô phỏng hoạt động của hệ thống phân tán. Bộ dữ liệu NYC Real-Time Traffic Speed Data được tiền xử lý, lập chỉ mục và truy vấn nhằm phân tích biến động tốc độ giao thông theo khu vực và thời gian, đồng thời được trực quan hóa qua Kibana Dashboard để thể hiện xu hướng và mật độ lưu thông. Cuối cùng, nhóm tiến hành đánh giá hiệu năng qua các chỉ tiêu như độ trễ truy vấn, tốc độ lập chỉ mục và khả năng phục hồi khi mất node, qua đó khẳng định tính ổn định, khả năng mở rộng và độ tin cậy của Elasticsearch trong xử lý và phân tích dữ liệu lớn theo thời gian thực.

# **CHƯƠNG II. CƠ SỞ LÝ THUYẾT VỀ ELASTICSEARCH**

## **2.1. Tổng quan về Elasticsearch**

Elasticsearch là một công cụ tìm kiếm và phân tích dữ liệu phân tán mã nguồn mở, được phát triển bởi Shay Banon từ năm 2010 trên nền tảng Apache Lucene (ra đời năm 2004). Mục tiêu ban đầu của Elasticsearch là đơn giản hóa việc tìm kiếm toàn văn bản trong các ứng dụng web hiện đại.

Năm 2012, công ty Elasticsearch B.V. được thành lập để phát triển và thương mại hóa sản phẩm. Đến năm 2015, bộ công cụ ELK Stack (gồm Elasticsearch, Logstash, và Kibana) chính thức ra mắt, tạo nên một hệ sinh thái hoàn chỉnh cho lưu trữ, xử lý, tìm kiếm và trực quan hóa dữ liệu. Năm 2018, Beats được bổ sung, mở rộng khả năng thu thập dữ liệu nhẹ, đồng thời đổi tên thành Elastic Stack. Năm 2019, Elastic tích hợp thêm X-Pack, bao gồm các tính năng về bảo mật, học máy, giám sát hiệu năng và cảnh báo. Đến năm 2021, giấy phép mã nguồn được chuyển từ Apache 2.0 sang SSPL/ELv2, gây nhiều tranh luận trong cộng đồng mã nguồn mở. Tính đến năm 2025, Elasticsearch đã phát triển đến phiên bản 8.x, hỗ trợ các tính năng hiện đại như ESQL (Elasticsearch Query Language), vector search và AI-powered analytics, đánh dấu bước tiến mạnh mẽ trong xu hướng phân tích dữ liệu thông minh.

Về đặc điểm kỹ thuật, Elasticsearch nổi bật với kiến trúc shared-nothing – mỗi node hoạt động độc lập và không chia sẻ tài nguyên, giúp hệ thống dễ dàng mở rộng ngang. Giao tiếp được thực hiện thông qua RESTful API với dữ liệu phản hồi ở dạng JSON, thuận tiện cho tích hợp với nhiều ngôn ngữ lập trình. Hệ thống hỗ trợ indexing gần thời gian thực (near real-time) với độ trễ trung bình khoảng 1 giây, sử dụng inverted index kết hợp thuật toán BM25 để tối ưu hóa khả năng tìm kiếm toàn văn bản. Cơ chế sharding và replication tự động đảm bảo tính sẵn sàng cao và khả năng chịu lỗi. Ngoài ra, Elasticsearch còn hỗ trợ aggregation phức tạp, hoạt động theo cơ chế schema-less (tự động nhận dạng kiểu dữ liệu), và cho phép mở rộng quy mô hầu như không giới hạn.

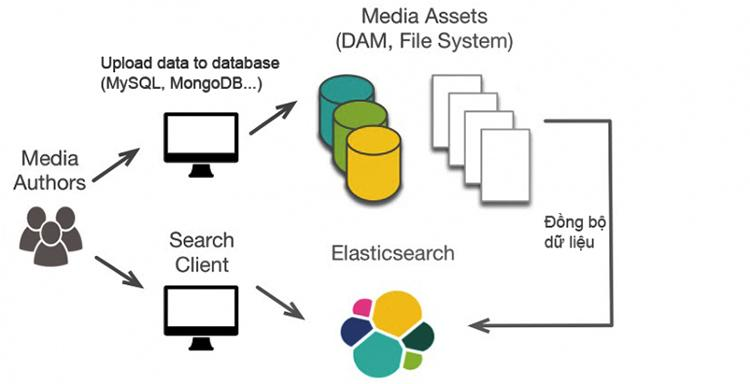
Trong lĩnh vực Big Data, Elasticsearch đóng vai trò quan trọng trong việc giải quyết ba đặc trưng cơ bản của dữ liệu lớn – Volume, Velocity và Variety.

* Với Volume, Elasticsearch có khả năng phân mảnh dữ liệu (sharding) để xử lý tập dữ liệu cực lớn.
* Với Velocity, hệ thống có thể xử lý hàng triệu bản ghi mỗi giây nhờ khả năng lập chỉ mục và truy vấn nhanh.
* Với Variety, Elasticsearch hỗ trợ nhiều định dạng dữ liệu khác nhau như log, JSON, văn bản, dữ liệu không gian hay chuỗi thời gian.

Nhờ đó, Elasticsearch thường được xem như lớp tìm kiếm và phân tích trung tâm trong kiến trúc Big Data hiện đại, dễ dàng tích hợp với các công cụ khác như Kafka, Spark/Hadoop, và Kibana.

Trên thế giới, nhiều tập đoàn lớn đã ứng dụng Elasticsearch ở quy mô toàn cầu: Netflix sử dụng để giám sát hàng tỷ sự kiện mỗi ngày, Uber dùng cho hệ thống phát hiện lỗi thời gian thực, còn Wikipedia triển khai Elasticsearch để cung cấp khả năng tìm kiếm tức thì trong hàng triệu bài viết.

Với xu hướng phát triển mạnh mẽ hướng tới AI-driven analytics, Cloud-native infrastructure và Observability, Elasticsearch hiện nay không chỉ là một công cụ tìm kiếm mà đã trở thành nền tảng cốt lõi giúp chuyển hóa dữ liệu lớn thành tri thức và giá trị kinh doanh, đóng vai trò quan trọng trong các hệ thống phân tích và ra quyết định thông minh.



## **2.2. Kiến trúc tổng thể của hệ thống**

Kiến trúc của Elasticsearch được thiết kế theo mô hình phân tán hoàn toàn và không chia sẻ tài nguyên, cho phép hệ thống mở rộng ngang vô hạn, chịu lỗi cao và tự động cân bằng tải. Toàn bộ hệ thống xoay quanh ba khái niệm cốt lõi: Cluster, Node, và Index (với Document, Shard, Replica). Kiến trúc này được xây dựng trên nền Apache Lucene ở tầng dưới, nhưng được mở rộng để hoạt động trong môi trường đám mây, cụm máy chủ lớn với hàng trăm node.

### 2.2.1. Cluster

Cluster là đơn vị tổ chức cao nhất trong kiến trúc Elasticsearch, được định nghĩa như một tập hợp các node hoạt động đồng bộ để lưu trữ dữ liệu, xử lý truy vấn và duy trì tính sẵn sàng của hệ thống; mỗi cluster có tên duy nhất (cấu hình qua cluster.name trong elasticsearch.yml, mặc định là elasticsearch) nhằm đảm bảo các node chỉ kết nối với nhau trong cùng một cụm, tránh xung đột với các cluster khác trên mạng.

Vai trò cốt lõi của cluster bao gồm:

* Quản lý trạng thái toàn cục (cluster state): lưu trữ thông tin về danh sách node, index, mapping, setting, và vị trí phân bổ shard — được đồng bộ trên tất cả các master-eligible node theo cơ chế gossip protocol;
* Tự động phát hiện và kết nối node (discovery): sử dụng Zen Discovery (trước phiên bản 7.0) hoặc seed hosts + unicast (phiên bản mới) để các node tự tìm nhau qua danh sách IP/port (cấu hình discovery.seed\_hosts), đồng thời hỗ trợ cross-cluster replication để liên kết nhiều cluster;
* Phân phối và cân bằng dữ liệu: tự động chia index thành primary shard, phân bổ đều trên các data node theo thuật toán round-robin, và rebalance khi có node mới gia nhập hoặc rời đi nhằm tối ưu tải;
* Chịu lỗi và phục hồi: liên tục giám sát sức khỏe node qua ping mechanism (mặc định mỗi 1 giây), phát hiện node chết → kích hoạt recovery để sao chép dữ liệu từ replica còn lại, đảm bảo không mất dữ liệu;
* Điều phối truy vấn phân tán: khi nhận query, cluster phân phối đến tất cả shard liên quan trên các node, sau đó tổng hợp kết quả tại coordinating node để trả về client.
* Cơ chế bầu chọn master dựa trên quorum để tránh split-brain: cần ít nhất (số master-eligible nodes / 2 + 1) node đồng thuận (cấu hình discovery.zen.minimum\_master\_nodes ở phiên bản cũ), đảm bảo chỉ có một master duy nhất quản lý cluster state tại một thời điểm.

Ví dụ: Trong một cluster sản xuất tên log-analysis-cluster gồm 12 node (3 master-eligible, 9 data node), khi node master hiện tại bị mất kết nối, 2 master-eligible node còn lại sẽ bầu chọn master mới trong vòng vài giây, đồng thời tự động di chuyển các primary shard từ node chết sang node sống để duy trì dịch vụ.

Nhờ kiến trúc này, cluster không chỉ mang lại khả năng mở rộng ngang vô hạn mà còn đảm bảo độ tin cậy cao trong môi trường Big Data, nơi dữ liệu có thể lên đến hàng petabyte và yêu cầu uptime 99.99%.

### 2.2.2. Node và vai trò của các loại node

Node là đơn vị thực thi cơ bản trong Elasticsearch, được định nghĩa như một instance Java đang chạy trên một máy chủ vật lý hoặc ảo, đại diện cho một tiến trình Elasticsearch độc lập với UUID duy nhất (tạo tự động khi khởi động) và tên tùy chỉnh (cấu hình qua node.name trong elasticsearch.yml); mỗi node tham gia cluster thông qua địa chỉ HTTP (mặc định port 9200) để nhận request từ client và transport (port 9300) để giao tiếp nội bộ với các node khác.

Vai trò cốt lõi của node bao gồm:

* Lưu trữ và xử lý dữ liệu: data node giữ các shard (primary/replica), thực hiện indexing, search, aggregation;
* Quản lý cluster: master-eligible node lưu trữ và đồng bộ cluster state, bầu chọn master, phân bổ shard, theo dõi sức khỏe toàn cụm;
* Tiền xử lý dữ liệu: ingest node chạy ingest pipeline (grok, date, geoip, remove) để biến đổi dữ liệu trước khi lập chỉ mục;
* Điều phối truy vấn: mọi node đều có thể đóng vai trò coordinating node — nhận query từ client, phân tán đến các data node chứa shard liên quan, tổng hợp kết quả theo relevance score, rồi trả về client.

Các loại node chuyên biệt được cấu hình qua node.roles:

* Master-eligible node ([master]): chỉ quản lý, không lưu dữ liệu lớn; yêu cầu CPU ổn định, RAM trung bình; ví dụ: 3 node riêng biệt trong production để tránh overload;
* Data node ([data]): lưu trữ shard, xử lý I/O nặng; cần RAM lớn (heap ≥50% RAM vật lý), SSD nhanh; phân loại thêm: data\_hot (dữ liệu mới, truy cập thường), data\_warm (dữ liệu cũ, ít truy cập), data\_cold (lưu trữ dài hạn), data\_frozen (tìm kiếm hiếm);
* Ingest node ([ingest]): tiền xử lý dữ liệu streaming; CPU cao để parse log phức tạp;
* Coordinating-only node (không khai báo role hoặc [ ]): chuyên điều phối, giảm tải cho data node; thường dùng trong cluster lớn.

Ví dụ cấu hình: trong cluster monitoring-cluster gồm 20 node, cấu hình 3 master-eligible (node.roles: [master]), 15 data node (node.roles: [data\_hot, data\_content]), 2 ingest node (node.roles: [ingest]); khi nhận 1 triệu log/giây từ Beats, ingest node parse IP → geoip, data node lập chỉ mục, coordinating node tổng hợp top 10 IP truy cập để hiển thị trên Kibana.

Nhờ phân vai trò rõ ràng, node giúp hệ thống tối ưu tài nguyên, dễ bảo trì và mở rộng linh hoạt, đảm bảo hiệu suất cao trong môi trường Big Data với hàng nghìn node và petabyte dữ liệu.

### 2.2.3. Index, Document, Shard, Replica

Trong hệ thống Elasticsearch, dữ liệu được tổ chức và quản lý theo cấu trúc phân cấp, bao gồm Cluster → Node → Index → Shard → Document. Trong đó, Index, Document, Shard và Replica là bốn thành phần cốt lõi đóng vai trò quyết định đến khả năng lưu trữ, truy vấn và mở rộng dữ liệu trong môi trường phân tán.

a) Index

Index là đơn vị quản lý dữ liệu cấp cao nhất trong Elasticsearch, tương đương với cơ sở dữ liệu (database) trong các hệ quản trị dữ liệu truyền thống. Mỗi index bao gồm một tập hợp các document có cùng cấu trúc, mục đích hoặc ngữ cảnh kinh doanh cụ thể.

Một index được định danh bằng tên duy nhất, chỉ bao gồm ký tự viết thường và các ký tự đặc biệt hợp lệ như “-”, “\_” hoặc “+”. Mỗi index được định nghĩa bởi hai nhóm thông tin chính: mapping và settings.

Mapping quy định cấu trúc của các trường dữ liệu, có thể ở dạng dynamic (Elasticsearch tự động suy luận kiểu dữ liệu) hoặc explicit (người dùng khai báo thủ công).

Settings cho phép người dùng cấu hình số lượng shard, replica, bộ phân tích analyzer, chu kỳ refresh interval, cũng như chính sách quản lý vòng đời dữ liệu (ILM).

Elasticsearch còn hỗ trợ các tính năng như alias (bí danh cho index), template (mẫu định nghĩa cấu trúc cho các index mới) và rollover (tự động tạo index mới khi đạt ngưỡng dữ liệu hoặc thời gian).

Về vai trò, index chịu trách nhiệm tổ chức và phân loại dữ liệu, kiểm soát truy cập ở cấp hệ thống thông qua RBAC, tối ưu hiệu năng truy vấn bằng cách điều chỉnh cấu hình shard/replica, và quản lý vòng đời dữ liệu tự động (hot → warm → cold → delete). Đây cũng là điểm kết nối chính giữa Elasticsearch với các công cụ trong Elastic Stack như Kibana, Logstash, Beats và REST API.

b) Document

Document là đơn vị dữ liệu nhỏ nhất trong Elasticsearch, tương đương với một bản ghi (record) trong cơ sở dữ liệu quan hệ. Mỗi document được lưu trữ ở dạng JSON và có một \_id duy nhất, được hệ thống tự sinh hoặc người dùng chỉ định.

Mỗi document bao gồm ba nhóm thông tin chính:

* \_source: chứa toàn bộ nội dung gốc của document.
* \_version: theo dõi lịch sử cập nhật và hỗ trợ cơ chế kiểm soát đồng thời lạc quan (optimistic concurrency control).
* Metadata fields như \_index, \_routing, \_parent giúp xác định vị trí lưu trữ và mối quan hệ giữa các document.

Elasticsearch cho phép lưu trữ các cấu trúc dữ liệu phức tạp như object, array, nested hoặc parent–child, nhờ đó có thể biểu diễn được dữ liệu bán cấu trúc và phi cấu trúc. Document chính là đối tượng mà người dùng thao tác trực tiếp thông qua các lệnh index, search, update, delete hoặc bulk, và là nguồn dữ liệu đầu vào cho các truy vấn Query DSL, phép tổng hợp aggregation hoặc script tùy chỉnh.

c) Shard

Shard là đơn vị lưu trữ vật lý nhỏ nhất của Elasticsearch, đại diện cho một phần dữ liệu của index. Mỗi shard thực chất là một Lucene index độc lập, chịu trách nhiệm lưu trữ và xử lý một tập con của các document trong index tương ứng.

Elasticsearch chia index thành nhiều shard để phân tán dữ liệu theo chiều ngang, giúp cải thiện khả năng mở rộng và xử lý song song. Có hai loại shard:

* Primary shard là bản gốc của dữ liệu, tiếp nhận toàn bộ các thao tác ghi.
* Replica shard là bản sao của primary shard, đảm bảo khả năng chịu lỗi và phân tải truy vấn đọc.

Số lượng primary shard được xác định tại thời điểm tạo index và không thể thay đổi trực tiếp sau đó. Mỗi shard chứa các segment file, translog và được refresh định kỳ để hiển thị dữ liệu mới. Elasticsearch còn hỗ trợ các cơ chế shard allocation filtering, delayed allocation và rebalancing để tối ưu phân phối dữ liệu giữa các node.

Trong thực tế, kích thước tối ưu cho mỗi shard thường nằm trong khoảng 20–50 GB. Với dữ liệu khoảng 13 GB như trong đề tài, việc sử dụng một primary shard là hợp lý nhất, đảm bảo hiệu năng cao và tránh lãng phí tài nguyên. Tuy nhiên, nếu hệ thống có nhu cầu mở rộng hoặc xử lý song song cao, có thể cấu hình ba primary shard, khi đó mỗi shard sẽ lưu khoảng 4–5 GB dữ liệu. Dù vậy, số lượng shard quá nhiều so với dung lượng thực tế sẽ làm tăng chi phí bộ nhớ và giảm hiệu quả tổng thể của hệ thống.

d) Replica

Replica là bản sao đồng bộ theo thời gian thực của primary shard, được dùng để đảm bảo tính sẵn sàng và an toàn dữ liệu. Hệ thống có thể thay đổi số lượng replica bất kỳ lúc nào bằng API hoặc cấu hình index.

Elasticsearch sử dụng cơ chế sao chép đồng bộ, nghĩa là thao tác ghi chỉ được xác nhận hoàn tất khi primary shard và tất cả các replica đều lưu thành công dữ liệu. Khi một primary shard gặp sự cố, hệ thống sẽ tự động thăng cấp một replica shard thành primary mới, bảo đảm hệ thống hoạt động liên tục mà không mất dữ liệu.

Replica cũng giúp tăng tốc độ truy vấn đọc, nhờ khả năng phân tải song song giữa primary và replica, và được phân bố trên các node khác nhau để giảm rủi ro mất mát dữ liệu. Ngoài ra, trong một số trường hợp, replica có thể được đặt trên node có cấu hình thấp hơn để tối ưu chi phí lưu trữ.

## **2.3. Cơ chế lập chỉ mục và tìm kiếm**

Cơ chế lập chỉ mục và tìm kiếm là hai thành phần cốt lõi tạo nên hiệu năng và khả năng mở rộng của Elasticsearch. Được xây dựng trên nền tảng Apache Lucene, Elasticsearch sử dụng cấu trúc inverted index để tối ưu hóa quá trình truy vấn văn bản toàn văn, đồng thời đảm bảo hiệu suất gần thời gian thực.

Quá trình indexing chịu trách nhiệm chuyển đổi dữ liệu thô thành cấu trúc lưu trữ tối ưu cho việc tìm kiếm, trong khi searching thực hiện phân tích, truy vấn và xếp hạng kết quả theo các mô hình xác suất (như BM25). Hai cơ chế này phối hợp chặt chẽ để đảm bảo độ chính xác, tốc độ phản hồi và tính phân tán trong môi trường dữ liệu lớn.

### 2.3.1. Quá trình Indexing

Quá trình indexing là giai đoạn tiền xử lý quan trọng, trong đó các document JSON được chuyển đổi thành các segment bất biến chứa inverted index, nhằm đảm bảo tính nhất quán giữa dữ liệu đầu vào và cấu trúc tìm kiếm. Cơ chế này giúp Elasticsearch hỗ trợ truy vấn gần thời gian thực (near real-time search) thông qua refresh và translog.

Cụ thể, mỗi document được gửi đến Elasticsearch qua HTTP RESTful API và ánh xạ vào primary shard dựa trên hàm băm của \_id. Trước khi lập chỉ mục, hệ thống áp dụng mapping để xác định kiểu dữ liệu và cấu hình analyzer cho từng trường (field). Trong quá trình phân tích văn bản, chuỗi dữ liệu được xử lý qua ba lớp: *character filters*, *tokenizer* và *token filters* (bao gồm chuẩn hóa chữ hoa–thường, loại bỏ stop words, stemming và đồng nghĩa).

Từ luồng token đầu ra, Elasticsearch (dựa trên Lucene) tạo inverted index gồm ba cấu phần:

* Term Dictionary (FST): lưu trữ danh sách từ duy nhất;
* Posting List: chứa thông tin [doc\_id, tần suất, vị trí] của từng term;
* Norms: lưu trọng số và độ dài trường phục vụ tính điểm (scoring).

Mỗi lần lập chỉ mục, hệ thống tạo thêm một segment mới và hợp nhất (merge) định kỳ để giảm I/O. Dữ liệu sau khi ghi vào primary shard được sao chép sang replica shard để bảo đảm tính sẵn sàng (high availability). Tất cả thay đổi được ghi vào translog như một write-ahead log nhằm duy trì độ bền (durability).

Sau cùng, cơ chế refresh được kích hoạt (mặc định mỗi 1 giây) để các segment mới trở nên khả kiến cho truy vấn, giúp Elasticsearch đạt được khả năng tìm kiếm gần thời gian thực mà vẫn giữ hiệu năng cao.

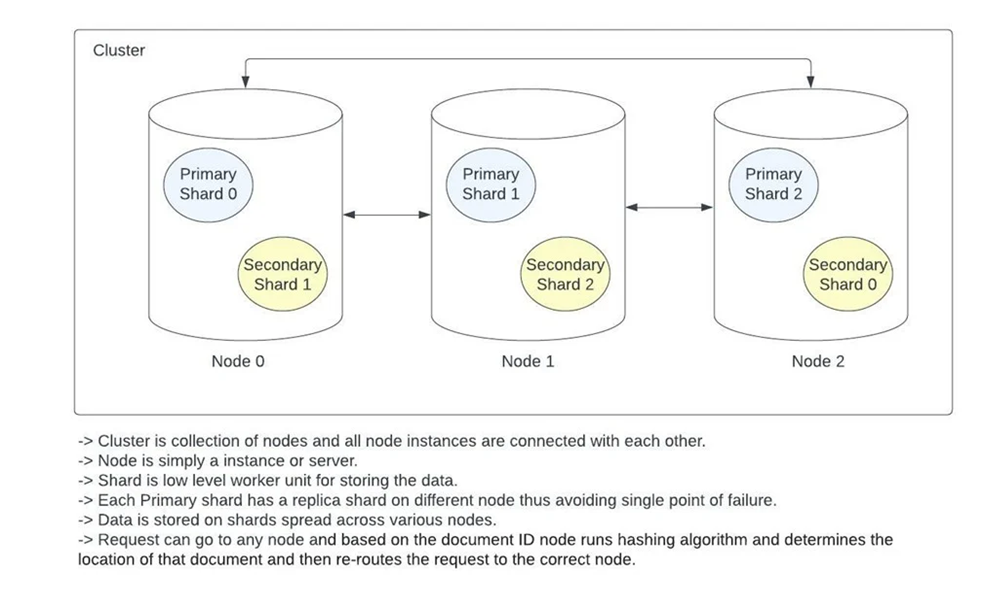
### 2.3.2. Quá trình Searching

Quá trình searching trong Elasticsearch là một hoạt động phân tán, được tối ưu để thực thi truy vấn với độ phức tạp gần O(1) cho các phép tra cứu theo term. Cơ chế này dựa trên mô hình scatter–gather, trong đó truy vấn được phân phối đến nhiều shard, xử lý song song, và kết quả được tổng hợp tại coordinating node.

Khi nhận truy vấn, Elasticsearch tiến hành phân tích và parse Query DSL (JSON) thành cây truy vấn logic. Các từ khóa truy vấn (query string) được xử lý qua cùng analyzer với dữ liệu để tạo ra term query tương ứng. Sau đó, trong scatter phase, coordinating node xác định các shard chứa dữ liệu liên quan và gửi truy vấn song song đến chúng. Mỗi shard tra cứu term trong term dictionary, kết hợp các posting list thông qua phép giao/hợp (bitset operations) và áp dụng bộ lọc (filter context) trước khi tính điểm liên quan (relevance scoring) bằng thuật toán BM25.

Trong gather phase, coordinating node thu thập kết quả top-K từ các shard, thực hiện xếp hạng toàn cục (global ranking) và gộp kết quả (merge) cho các phép aggregation. Kết quả cuối cùng được trả về ở dạng JSON, bao gồm danh sách kết quả (hits), các thống kê (aggregations) và thời gian truy vấn (took).

Để tối ưu hiệu suất, Elasticsearch sử dụng nhiều cơ chế cache như filter cache, shard request cache, và Query Profile API nhằm phân tích thời gian từng pha, giúp cải thiện tốc độ xử lý và hiệu năng của toàn hệ thống.



## **2.4. Cơ chế phân tán và chịu lỗi**

Cơ chế phân tán và chịu lỗi là nền tảng giúp Elasticsearch đạt được tính high availability (HA), scalability ngang và fault tolerance trong môi trường Big Data. Hệ thống hoạt động theo kiến trúc shared-nothing, mỗi node độc lập nhưng phối hợp chặt chẽ thông qua giao thức Zen Discovery và Raft-like consensus. Phần lý thuyết dưới đây phân tích chi tiết cách phân phối dữ liệu, đồng bộ replica và duy trì tính nhất quán của cluster ngay cả khi xảy ra sự cố node hoặc mạng.

### 2.4.1. Phân phối shard giữa các node

Cơ chế phân phối shard là nền tảng giúp Elasticsearch đạt được khả năng mở rộng ngang và cân bằng tải trong môi trường phân tán. Mỗi index được chia thành nhiều shard (bao gồm *primary* và *replica*), và các shard này được phân bố tự động hoặc thủ công trên các node trong cluster.

Quá trình định tuyến document sử dụng thuật toán băm (hash-based routing) theo công thức:

*shard\_num=hash(routing)modnumber\_of\_primary\_shards*

Nhờ đó, cùng một document luôn được gửi đến đúng *primary shard* tương ứng. Trạng thái phân bổ được quản lý tập trung bởi master node thông qua cluster state, chứa thông tin về node, index mapping, và vị trí shard (primary/replica).

Hệ thống còn hỗ trợ allocation awareness để tránh đặt primary và replica trong cùng vùng, đồng thời duy trì balanced shard allocation nhằm phân phối đều tài nguyên giữa các node. Khi có thay đổi (node mới tham gia, node rời cluster, hoặc chênh lệch số shard vượt ngưỡng), cơ chế rebalancing sẽ tự động di chuyển shard để khôi phục cân bằng.

Master node chịu trách nhiệm ra quyết định phân bổ và gửi lệnh allocation đến node đích, nơi sẽ thực hiện peer recovery hoặc snapshot restore nếu cần. Ngoài phân bổ tự động, người quản trị có thể điều khiển thủ công thông qua các tham số như cluster.routing.allocation.enable hoặc các bộ lọc include/exclude theo IP, tên, hoặc thuộc tính node.

Nhờ cơ chế này, Elasticsearch đảm bảo được tính sẵn sàng, hiệu năng ổn định, và khả năng tự điều chỉnh động khi quy mô dữ liệu hoặc cấu trúc cluster thay đổi.

### 2.4.2. Cơ chế replicate dữ liệu

Replication đảm bảo data redundancy, fault tolerance và read scalability. Mỗi primary shard có ít nhất một replica shard (mặc định = 1), được duy trì đồng bộ gần thời gian thực.

Cơ chế chi tiết:

* Replication model – Primary-Backup (Async với acknowledgment):
  + Write path:
    - Client gửi request đến coordinating node.
    - Coordinating node forward đến node chứa primary shard.
    - Primary shard:
      * Ghi vào in-memory buffer + translog.
      * Gửi replication request song song đến tất cả replica nodes.
    - Replica nodes:
      * Áp dụng thay đổi vào buffer + translog.
      * Trả về ACK khi fsync thành công.
    - Primary thu thập đủ quorum ACK → trả success cho client.
  + Consistency level:

required\_successful\_shards = + 1

* Translog và durability:
  + Mỗi shard (primary/replica) có translog riêng (append-only).
  + Mọi operation (index, delete, update) được ghi vào translog trước khi áp dụng vào Lucene.
  + index.translog.durability = request (fsync mỗi request) hoặc async (mặc định 5s).
* Recovery sau sự cố:
  + Node failure:
    - Master phát hiện qua ping timeout (Zen Discovery).
    - Tăng phiên bản cluster state → đánh dấu shard trên node chết là unassigned.
    - Tìm replica còn sống → promote thành primary.
    - Tạo replica mới trên node khác từ primary mới.
  + Recovery types:
    - Peer recovery: Copy segment files + replay translog từ nguồn.
    - Snapshot recovery: Từ repository (S3, HDFS).
  + Sequence ID: Mỗi operation có global sequence number → tránh duplicate khi replay.
* Read từ replica:
  + Search request có thể được route đến replica (round-robin).
  + Preference: \_primary, \_local, \_only\_nodes.

### 2.4.3. Bầu chọn master node

Master node là single point of coordination, không lưu dữ liệu, chỉ quản lý cluster state và allocation. Elasticsearch sử dụng gossip-based discovery + leader election để đảm bảo exactly one master tại mọi thời điểm.

Cơ chế chi tiết:

* Zen Discovery (pre-8.0) / Native Discovery (8.0+):
  + Unicast discovery: Node chỉ ping danh sách discovery.seed\_hosts.
  + Ping phase: Mỗi node gửi ping chứa: node ID, cluster name, master-eligible flag.
  + Join cluster: Node gửi join request đến master hiện tại.
* Master election (Raft-inspired):
  + Eligibility: Chỉ node có node.master: true (mặc định) tham gia.
  + Trigger:
    - Cluster bootstrap (lần đầu).
    - Master failure (ping timeout > discovery.zen.ping\_timeout).
  + Voting process:
    - Node phát hiện không có master → vào trạng thái candidate.
    - Gửi vote request đến tất cả master-eligible nodes.
    - Node nhận vote request:
      * So sánh term (phiên bản bầu cử).
      * Vote cho candidate có term cao hơn và log mới hơn.
    - Candidate nhận quorum vote → trở thành leader (master).

quorum = + 1

* + - Master mới publish cluster state → các node khác trở thành follower.
* Split-brain prevention:
  + Minimum master nodes:

discovery.zen.minimum\_master\_nodes = + 1

Ngăn hai nhóm node cùng bầu master khi network partition.

* + Tie-breaker: Node có ID nhỏ nhất thắng nếu vote bằng nhau.
* Master fault handling:
  + No master state: Node không xử lý write/allocation, chỉ phục vụ read từ local shard.
  + Blackhole protection: gateway.expected\_nodes + gateway.recover\_after\_time để tránh recovery sớm.

## **2.5. Ưu điểm, hạn chế và phạm vi ứng dụng của Elasticsearch**

### 2.5.1. Ưu điểm

Elasticsearch là một nền tảng tìm kiếm và phân tích dữ liệu phân tán có hiệu năng cao, được đánh giá nổi bật nhờ khả năng xử lý dữ liệu lớn trong thời gian gần thực, tính linh hoạt trong quản lý dữ liệu phi cấu trúc và khả năng mở rộng theo chiều ngang. Một số ưu điểm chính của Elasticsearch có thể được trình bày như sau:

Trước hết, hiệu năng tìm kiếm gần thời gian thực là đặc điểm nổi bật nhất của Elasticsearch. Công cụ này sử dụng cấu trúc inverted index kết hợp với cơ chế refresh interval mặc định khoảng một giây, cho phép thực hiện các truy vấn ở cấp độ mili-giây ngay cả trên tập dữ liệu có quy mô hàng tỷ tài liệu. Nhờ đó, Elasticsearch đáp ứng tốt các bài toán cần truy vấn nhanh trên dữ liệu lớn và thay đổi liên tục.

Thứ hai, Elasticsearch có khả năng mở rộng linh hoạt theo chiều ngang. Hệ thống được thiết kế để mở rộng dung lượng và hiệu năng đơn giản bằng cách thêm node mới vào cluster mà không cần dừng hệ thống. Dữ liệu được chia thành các shard và phân bổ tự động giữa các node, đồng thời cơ chế rebalance shard giúp duy trì sự phân bố tải đồng đều trên toàn cluster. Nhờ đó, Elasticsearch có thể mở rộng tới quy mô petabyte dữ liệu, đáp ứng yêu cầu của các hệ thống Big Data thực tế.

Thứ ba, Elasticsearch thể hiện tính linh hoạt cao trong xử lý dữ liệu phi cấu trúc và bán cấu trúc. Với dynamic mapping, hệ thống có thể tự động nhận diện kiểu dữ liệu mà không yêu cầu schema cố định, đồng thời hỗ trợ nhiều loại truy vấn phức tạp như full-text search, aggregation, truy vấn không gian địa lý và chuỗi thời gian. Ngoài ra, từ phiên bản 7.x trở đi, Elasticsearch còn tích hợp thêm các tính năng machine learning, vector search và semantic search, mở rộng phạm vi ứng dụng trong các hệ thống thông minh và phân tích dữ liệu nâng cao.

Thứ tư, hệ sinh thái Elastic Stack (ELK/EFK) mang lại khả năng tích hợp toàn diện. Elasticsearch hoạt động hiệu quả cùng Logstash, Kibana và Beats, hình thành chuỗi xử lý hoàn chỉnh từ thu thập – lưu trữ – phân tích – trực quan hóa dữ liệu. Thêm vào đó, hệ thống cung cấp RESTful API tương thích với nhiều ngôn ngữ lập trình phổ biến như Python, Java, Go, Node.js, giúp dễ dàng tích hợp vào các ứng dụng và hệ thống hiện có.

Cuối cùng, Elasticsearch có độ tin cậy và khả năng chịu lỗi cao nhờ cơ chế replica shard, transaction log và failover tự động. Khi một node gặp sự cố, dữ liệu vẫn được truy cập thông qua các bản sao trên các node khác, đảm bảo tính sẵn sàng và tính toàn vẹn dữ liệu trong mọi tình huống.

### 2.5.2. Hạn chế

Bên cạnh những ưu điểm vượt trội, Elasticsearch vẫn tồn tại một số hạn chế nhất định, đặc biệt khi được sử dụng ngoài phạm vi thiết kế tối ưu của nó.

Thứ nhất, Elasticsearch không phải là hệ quản trị cơ sở dữ liệu giao dịch. Hệ thống không hỗ trợ giao dịch đa tài liệu và không đảm bảo đầy đủ bốn tính chất ACID (Atomicity, Consistency, Isolation, Durability). Do đó, Elasticsearch không phù hợp cho các ứng dụng OLTP, đặc biệt là trong các lĩnh vực yêu cầu tính nhất quán cao như tài chính hoặc kế toán.

Thứ hai, Elasticsearch có mức tiêu thụ tài nguyên hệ thống tương đối lớn. Dung lượng bộ nhớ (RAM) và CPU tiêu hao tỷ lệ thuận với số lượng trường và độ phân biệt dữ liệu. Ngoài ra, do giới hạn kỹ thuật của JVM, heap memory của Elasticsearch thường chỉ nên cấu hình tối đa khoảng 30 GB để tránh tràn vùng địa chỉ hóa đối tượng, khiến việc mở rộng theo chiều dọc trở nên khó khăn.

Thứ ba, quản trị cluster Elasticsearch đòi hỏi kiến thức chuyên sâu và kinh nghiệm thực tế. Việc cấu hình sai các tham số như shard allocation, node roles, hot-warm architecture hay split-brain prevention có thể dẫn đến mất cân bằng tải hoặc suy giảm hiệu năng nghiêm trọng. Quản trị viên cần thường xuyên giám sát và tối ưu cluster để duy trì hoạt động ổn định.

Thứ tư, hiệu năng của Elasticsearch giảm đáng kể khi có nhiều thao tác cập nhật hoặc xóa dữ liệu. Hệ thống không thực hiện ghi đè trực tiếp mà thay vào đó sử dụng cơ chế delete-by-query kết hợp reindex, dẫn đến tiêu tốn I/O và thời gian xử lý, nhất là trong các ứng dụng có tần suất ghi cao.

Cuối cùng, Elasticsearch chỉ đảm bảo tính nhất quán dần. Dữ liệu mới ghi có thể chưa xuất hiện ngay trong kết quả truy vấn do cơ chế refresh interval, điều này phù hợp với các hệ thống read-heavy nhưng không thích hợp cho các ứng dụng đòi hỏi tính nhất quán tức thời.

### 2.5.3. Phạm vi ứng dụng

Nhờ các đặc tính về tốc độ, khả năng mở rộng và khả năng xử lý dữ liệu phi cấu trúc, Elasticsearch được ứng dụng rộng rãi trong các hệ thống phân tích dữ liệu lớn theo thời gian thực. Một số lĩnh vực tiêu biểu bao gồm:

* Phân tích log và giám sát hệ thống (ELK/EFK Stack);
* Tìm kiếm và đề xuất nội dung trong website hoặc thương mại điện tử;
* Giám sát, cảnh báo và phân tích bảo mật (Monitoring, Alerting, SIEM);
* Phân tích dữ liệu kinh doanh (Business Intelligence Dashboard);
* Truy vấn dữ liệu không gian địa lý (Geospatial Search);
* Phân tích dữ liệu thời gian thực trong IoT và Smart City.

Tuy nhiên, Elasticsearch không phù hợp để sử dụng làm hệ quản trị dữ liệu chính trong các hệ thống OLTP, nghiệp vụ tài chính, hoặc các ứng dụng nhỏ, truy vấn đơn giản, nơi mà cơ sở dữ liệu quan hệ (RDBMS) hoặc key-value store (Redis, MongoDB) sẽ mang lại hiệu năng, chi phí và tính nhất quán dữ liệu tốt hơn.

# **CHƯƠNG III. TRIỂN KHAI THỬ NGHIỆM HỆ THỐNG PHÂN TÁN**

## **3.1. Mục tiêu triển khai**

Mục tiêu của chương này là thiết lập và đánh giá hoạt động của một hệ thống Elasticsearch phân tán trên môi trường giả lập nhằm minh chứng cho khả năng lưu trữ, truy vấn và phân tích dữ liệu lớn về tốc độ giao thông thời gian thực.

Các mục tiêu cụ thể bao gồm:

* Triển khai một cụm (cluster) Elasticsearch gồm 3 node trên cùng một máy tính cá nhân, mô phỏng cơ chế phân tán và chịu lỗi.
* Thiết lập chỉ mục dữ liệu (index) cho tập dữ liệu NYC Real-Time Traffic Speed Data, bao gồm các trường: id, link\_id, link\_name, borough, speed, travel\_time, status, data\_as\_of.
* Thực hiện quá trình ingestion, truy vấn, và phân tích dữ liệu bằng Kibana, thể hiện khả năng tìm kiếm theo thời gian và khu vực.
* Thử nghiệm khả năng chịu lỗi (fault tolerance) của hệ thống khi một node bị tắt hoặc mất kết nối.
* Đánh giá hiệu năng truy vấn, độ ổn định và tính phân tán logic của mô hình đã triển khai.

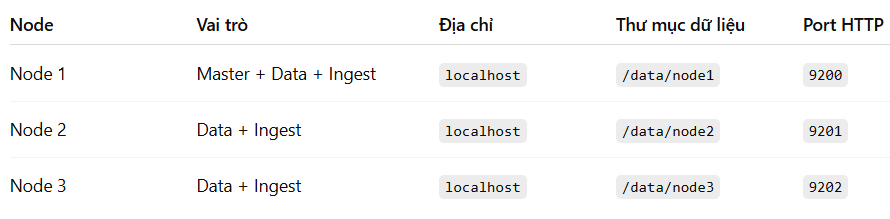
## **3.2. Mô hình hệ thống**

### 3.2.1. Kiến trúc tổng thể cluster

Hệ thống được triển khai dưới dạng một cluster Elasticsearch có tên nyc\_traffic, bao gồm ba node logic chạy song song trên cùng một máy tính cục bộ (Windows).

Mỗi node được cấu hình với port riêng biệt và thư mục dữ liệu độc lập nhằm đảm bảo tính mô phỏng của môi trường phân tán.

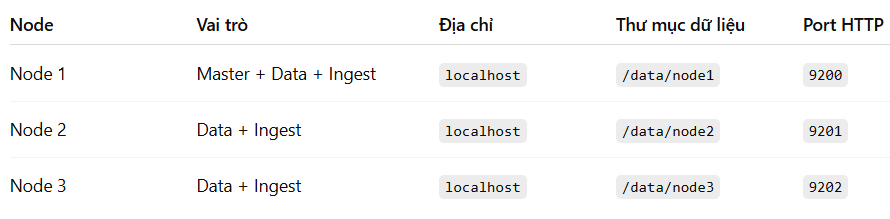
Cấu trúc tổng thể của cluster được thể hiện như sau:



Cụm cluster này thể hiện đầy đủ cơ chế phân tán logic của Elasticsearch, trong đó:

* Một node đóng vai trò master, quản lý metadata, phân bổ shard và điều phối cluster.
* Hai node còn lại đóng vai trò data node, lưu trữ và xử lý dữ liệu thực tế.
* Tất cả các node đều có thể tham gia quá trình indexing, search và replication.

### 3.2.2. Vai trò của từng node



Chi tiết vai trò từng node:

* Node 1 – Master + Data + Ingest: Là Master Node chịu trách nhiệm điều phối toàn bộ hoạt động của cluster, bao gồm:
  + Theo dõi trạng thái node (node discovery).
  + Quản lý thông tin metadata (mapping, index, shards, replica...).
  + Quyết định việc phân phối shard giữa các node.
  + Đồng thời là Data Node lưu trữ dữ liệu thật sự của các chỉ mục (index).
  + Có vai trò Ingest Node cho phép xử lý trước dữ liệu (ví dụ: thêm trường timestamp, chuẩn hóa dữ liệu JSON, tách text…).
* Node 2 – Data + Ingest:
  + Đảm nhận vai trò Data Node giúp giảm tải lưu trữ và xử lý truy vấn tìm kiếm.
  + Là Ingest Node, có thể nhận và xử lý dữ liệu trước khi ghi vào index.
  + Hoạt động song song với Node 1 giúp phân tán dữ liệu và tăng tốc độ truy vấn.
* Node 3 – Data + Ingest: Tương tự Node 2, đảm nhiệm vai trò Data + Ingest, tăng khả năng chịu lỗi và đảm bảo hệ thống vẫn hoạt động nếu một node bị lỗi.

Lợi ích của mô hình 3 node:

* Tính sẵn sàng cao: nếu một node gặp sự cố, các node còn lại vẫn duy trì hoạt động của cluster.
* Phân tán dữ liệu và xử lý song song, giúp cải thiện tốc độ truy vấn và ghi dữ liệu.
* Khả năng mở rộng: có thể dễ dàng thêm node mới mà không ảnh hưởng đến hoạt động của hệ thống hiện tại.

## **3.3. Cấu hình hệ thống**

### 3.3.1. Thiết lập các node Elasticsearch

Cấu hình chung:

* Cluster name: nyc-traffic-cluster
* Các node tham gia: node-1, node-2, node-3
* Danh sách seed hosts: ["127.0.0.1:9200", "127.0.0.1:9201", "127.0.0.1:9202"]
* Các node này kết nối bảo mật với nhau thông qua enrollment token – cơ chế được Elasticsearch cung cấp để đảm bảo các node trong cluster xác thực lẫn nhau an toàn (SSL/TLS + chứng chỉ tự sinh).

Quy trình thiết lập kết nối giữa các node bằng Enrollment Token :

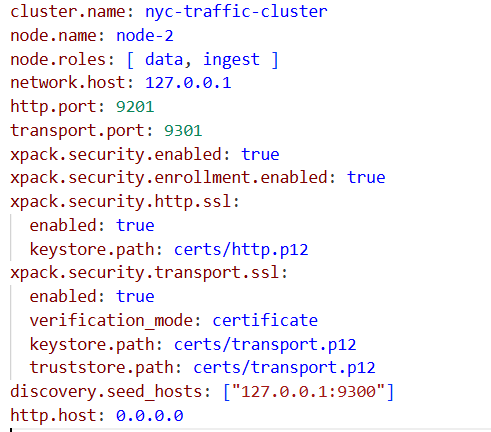
* Sau khi cài đặt Elasticsearch, bạn chỉ cần chạy node đầu tiên bằng lệnh bin\elasticsearch.bat
* Để cho phép các node khác tham gia cluster, chạy lệnh: bin\elasticsearch-create-enrollment-token.bat --scope node
* Tại máy (hoặc thư mục) cài đặt Elasticsearch của Node 2 và Node 3, chạy lệnh: bin\elasticsearch.bat --enrollment-token "<token-đã-tạo>"

Cấu hình chi tiết các node (file elasticsearch.yml của từng node ):

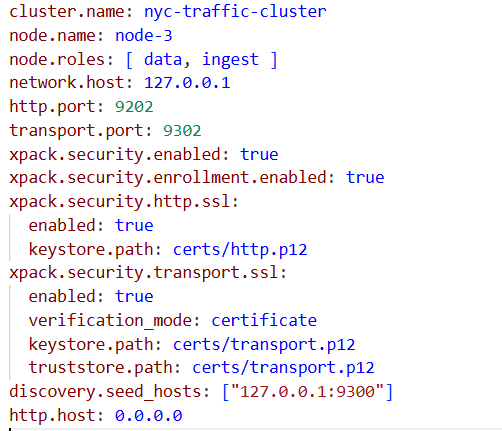
* node-1:



* node-2:



* node-3:



Node 1 đóng vai trò trung tâm, khởi tạo cluster và phát hành enrollment token.

Node 2 & Node 3 gia nhập cluster thông qua token này, đảm bảo bảo mật kết nối.

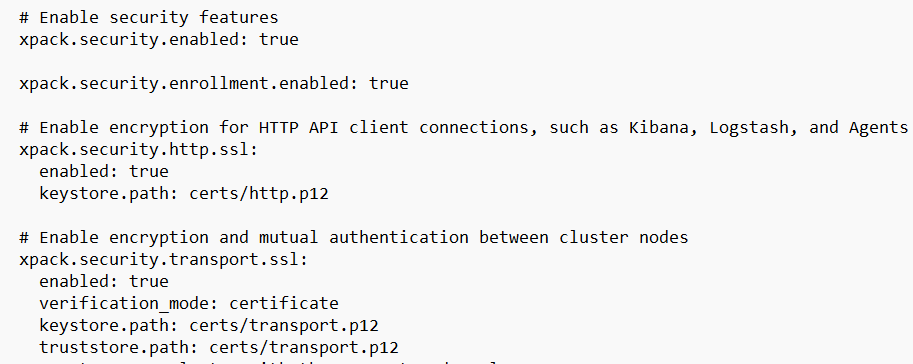
Toàn bộ cluster hoạt động với xác thực bảo mật (SSL/TLS) và có thể mở rộng dễ dàng.

### 3.3.2. Cấu hình bảo mật và kết nối Kibana

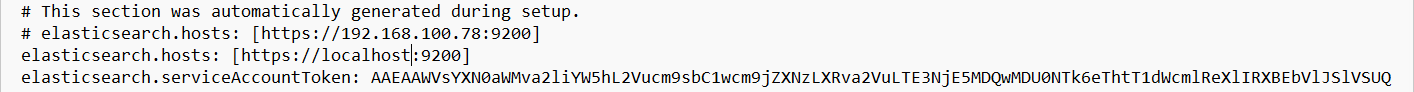
Tính năng bảo mật được tự động kích hoạt thông qua cơ chế Security Auto Configuration trong Elasticsearch phiên bản 8.x. Các tệp cấu hình elasticsearch.yml của ba node đều bao gồm khối cấu hình bảo mật được sinh tự động, đảm bảo:

* Xác thực người dùng: xpack.security.enabled: true
* Mã hóa kết nối HTTP: xpack.security.http.ssl.enabled: true
* Mã hóa giao tiếp nội bộ: xpack.security.transport.ssl.enabled: true

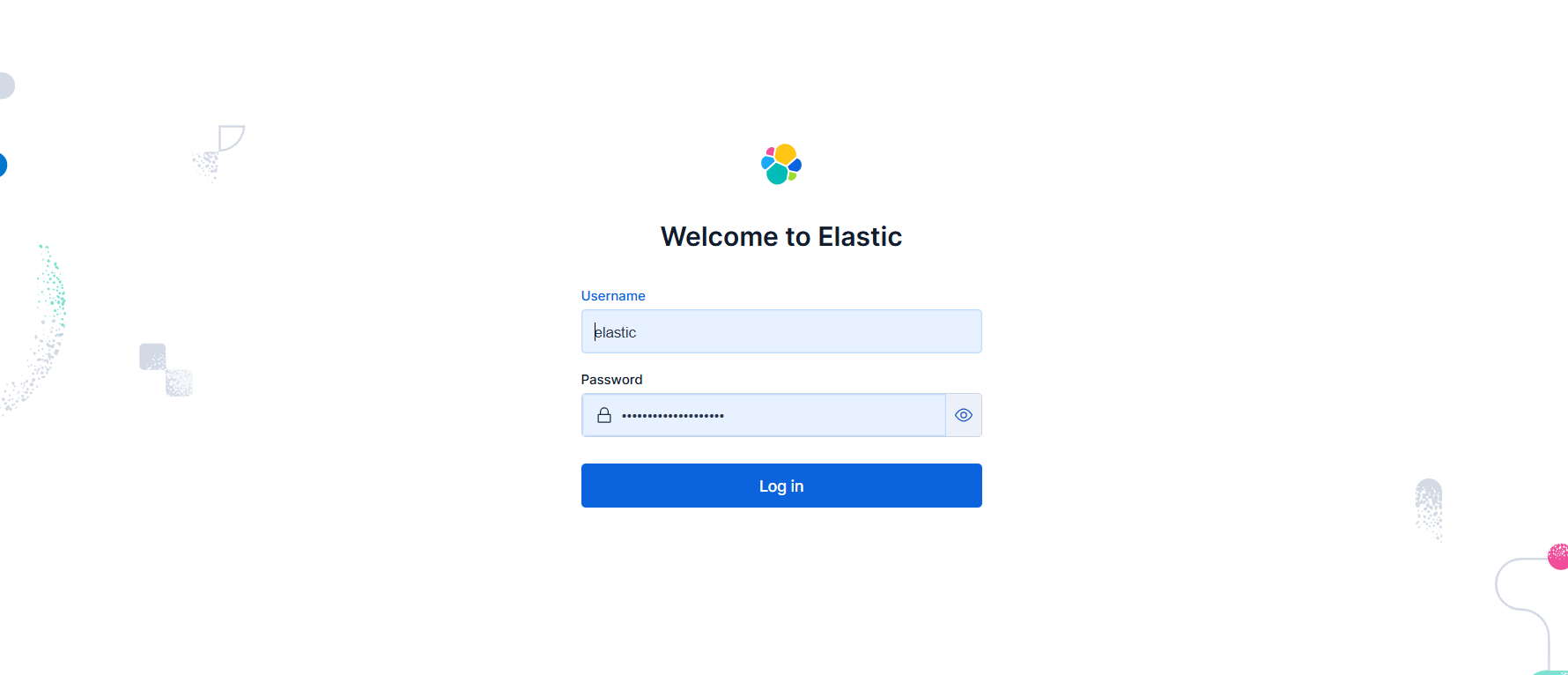
Chứng chỉ SSL được lưu trữ tại thư mục certs/ dưới dạng tệp nhị phân PKCS#12 (http.p12, transport.p12).



Kibana được cấu hình kết nối an toàn đến cluster thông qua tệp kibana.yml:

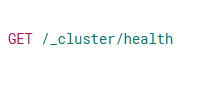


Dịch vụ Kibana hoạt động tại http://localhost:5601 và bắt buộc xác thực thông qua tài khoản hệ thống elastic hoặc service account token.



Kiểm tra trạng thái cụm (Cluster Health)

Sau khi hoàn tất cấu hình bảo mật, trạng thái của cluster được kiểm tra thông qua công cụ Dev Tools trong Kibana.  
Tại đây, người dùng nhập truy vấn:



Lệnh này giúp xác minh rằng tất cả các node trong hệ thống đều đang hoạt động ổn định, dữ liệu được đồng bộ hóa và trạng thái cluster là “green”, cho thấy không có shard nào bị mất hoặc lỗi.



## **3.4. Thiết kế chỉ mục và phân shard**

### 3.4.1. Nguyên tắc thiết kế

Chỉ mục nyc\_traffic được tạo tự động trong quá trình thực thi script upload\_nyc\_traffic\_progress.py, trước khi dữ liệu nạp vào hệ thống Elasticsearch.

Mục tiêu của thiết kế là đảm bảo hiệu năng truy vấn cao, khả năng mở rộng linh hoạt, và tính sẵn sàng (availability) trong môi trường cluster gồm 3 node.

Các quyết định thiết kế chính được lập trình và cấu hình như sau:

* Số lượng shard: number\_of\_shards: 3

Việc thiết lập ba shard cho phép hệ thống phân phối dữ liệu đều trên ba node, từ đó tận dụng tối đa khả năng song song hóa truy vấn và tránh hiện tượng “hot shard” khi khối lượng dữ liệu vượt quá 64 triệu bản ghi.

* Số lượng replica: number\_of\_replicas: 1

Mỗi shard có một bản sao (replica) được lưu trên node khác, đảm bảo khả năng chịu lỗi khi một node gặp sự cố mà không nhân đôi toàn bộ dung lượng lưu trữ. Cấu hình này đạt được cân bằng hợp lý giữa hiệu năng, tính sẵn sàng và tối ưu tài nguyên.

* Mapping dữ liệu: Cấu trúc dữ liệu được định nghĩa chi tiết nhằm đảm bảo hiệu quả lưu trữ và truy vấn.

Cụ thể:

* id, borough: kiểu keyword, hỗ trợ lọc và thống kê nhanh.
* link\_name: kiểu text, phục vụ tìm kiếm toàn văn (full-text search).
* data\_as\_of: kiểu date, giúp truy vấn và phân tích theo thời gian.
* speed, travel\_time: kiểu float, dùng cho tính toán và thống kê hiệu năng giao thông.
* link\_id: kiểu integer, định danh tuyến đường.
* status: kiểu integer, biểu diễn trạng thái giao thông.

### 3.4.2. Cấu hình chỉ mục

Đoạn mã dưới đây thể hiện cấu hình chỉ mục được định nghĩa trong script upload\_nyc\_traffic\_progress.py, đảm bảo tính tự động và nhất quán khi triển khai hệ thống



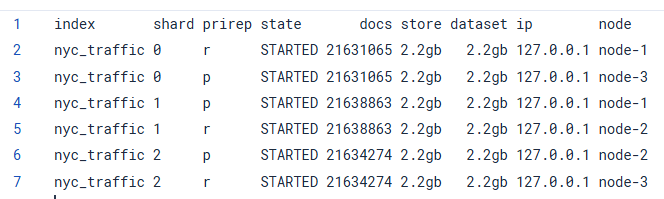
Ghi chú: Chỉ mục chỉ được tạo nếu chưa tồn tại (sử dụng hàm es.indices.exists) để tránh lỗi trùng lặp trong quá trình khởi tạo.



Hình: Kết quả kiểm tra cấu hình chỉ mục sau khi script chạy

* Phân bố shard trong cluster

Sau khi chỉ mục được tạo, Elasticsearch tự động phân phối các shard và replica lên các node trong cluster, bảo đảm tải cân bằng và khả năng phục hồi khi lỗi xảy ra.  
Việc phân bố này có thể được kiểm tra thông qua truy vấn sau:



## **3.5. Nạp dữ liệu và xử lý (Data Ingestion)**

### 3.5.1. Mô tả tập dữ liệu gốc

Tập dữ liệu được sử dụng trong đề tài là NYC Real-Time Traffic Speed Data, được thu thập và công bố bởi Cổng dữ liệu mở của Thành phố New York (NYC Open Data), đồng thời được trích xuất từ nguồn Kaggle Dataset Repository. Tập dữ liệu phản ánh tốc độ di chuyển trung bình của các tuyến đường trong khu vực Thành phố New York theo thời gian thực, được hệ thống giao thông đô thị cập nhật liên tục với chu kỳ 5 phút/lần.

Dữ liệu được xây dựng nhằm phục vụ các mục đích giám sát giao thông, phân tích lưu lượng phương tiện và hỗ trợ ra quyết định trong quản lý hạ tầng đô thị. Với quy mô lớn và tính thời gian thực, tập dữ liệu này là cơ sở thích hợp để đánh giá khả năng lưu trữ, xử lý và truy vấn dữ liệu phân tán của Elasticsearch.

Một số đặc trưng cơ bản của tập dữ liệu được tóm tắt như sau:

* Nguồn dữ liệu: NYC Open Data – Department of Transportation (DOT); được trích xuất từ Kaggle.
* Chu kỳ cập nhật: 5 phút/lần (real-time).
* Dung lượng gốc: khoảng 26,4 GB (định dạng CSV nén).
* Số lượng bản ghi: 64.914.523 dòng dữ liệu (records).
* Đơn vị tốc độ: miles per hour (mph).

Tập dữ liệu gốc bao gồm 13 cột dữ liệu, thể hiện các thông tin về mã tuyến đường, tốc độ trung bình, thời gian di chuyển, trạng thái cập nhật, tọa độ địa lý, đơn vị quản lý, và khu vực hành chính. Cấu trúc dữ liệu được trình bày chi tiết trong sau.

|  |  |  |
| --- | --- | --- |
| **Tên cột** | **Kiểu dữ liệu** | **Mô tả nội dung** |
| ID | String | Mã định danh duy nhất của bản ghi. |
| SPEED | Float | Tốc độ trung bình của phương tiện trên tuyến đường (mph). |
| TRAVEL\_TIME | Float | Thời gian di chuyển trung bình trên tuyến đường (phút). |
| STATUS | String | Trạng thái hoạt động của tuyến đường, ví dụ: “ACTIVE” hoặc “INACTIVE”. |
| DATA\_AS\_OF | Datetime | Thời điểm ghi nhận dữ liệu (định dạng yyyy-MM-dd HH:mm:ss). |
| LINK\_ID | Integer | Mã định danh tuyến đường trong hệ thống giao thông NYC. |
| LINK\_POINTS | String | Tập hợp các tọa độ (vĩ độ, kinh độ) mô tả tuyến đường. |
| ENCODED\_POLY\_LINE | String | Chuỗi mã hóa các điểm tọa độ của tuyến đường (polyline). |
| ENCODED\_POLY\_LINE\_LVLS | String | Cấp độ chi tiết của đường mã hóa (encoded polyline levels). |
| OWNER | String | Đơn vị quản lý hoặc sở hữu tuyến đường. |
| TRANSCOM\_ID | Integer | Mã định danh trong hệ thống TransCom – mạng lưới quản lý giao thông khu vực đô thị. |
| BOROUGH | String | Quận hành chính nơi tuyến đường nằm (Manhattan, Bronx, Brooklyn, Queens, Staten Island). |
| LINK\_NAME | String | Tên hoặc mô tả của tuyến đường (ví dụ: “FDR Drive North”). |

Tập dữ liệu có quy mô lớn, độ chi tiết cao và được cập nhật liên tục theo thời gian, phản ánh chính xác tình hình giao thông đô thị tại Thành phố New York. Với dung lượng lên đến 26,4 GB và hơn 64 triệu bản ghi, tập dữ liệu này đòi hỏi một hệ thống lưu trữ và phân tích dữ liệu có khả năng mở rộng và chịu tải cao. Do đó, việc lựa chọn Elasticsearch làm công cụ xử lý và phân tích là hoàn toàn phù hợp, nhằm đánh giá hiệu quả của mô hình lưu trữ phân tán và khả năng truy vấn thời gian thực trên dữ liệu lớn.

### 3.5.2. Tiền xử lí dữ liệu

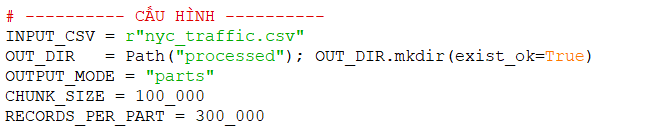
Quá trình tiền xử lý dữ liệu được thực hiện nhằm bảo đảm dữ liệu có tính sạch, thống nhất, đầy đủ và tương thích với mô hình lập chỉ mục của Elasticsearch. Do tập dữ liệu gốc có dung lượng lớn (hơn 26 GB, khoảng 65 triệu bản ghi), việc xử lý toàn bộ trong bộ nhớ là không khả thi. Vì vậy, nhóm nghiên cứu xây dựng một chương trình xử lý dữ liệu theo khối (chunk-based processing) bằng Python, sử dụng thư viện pandas, json và gzip.

Quy trình này bao gồm các bước: (1) chuẩn hóa lược đồ cột, (2) làm sạch và loại bỏ trùng lặp, (3) chuẩn hóa định dạng thời gian, (4) lọc ngoại lai, (5) tạo trường vị trí địa lý, (6) chọn lọc các cột cần thiết và (7) xuất dữ liệu sang định dạng NDJSON nén.

a) Cấu hình và cơ chế xử lý theo khối

Do tập dữ liệu gốc có dung lượng lớn (26,4 GB với hơn 64 triệu bản ghi), việc nạp toàn bộ dữ liệu vào bộ nhớ để xử lý là không khả thi trong môi trường máy tính cá nhân. Để giải quyết vấn đề này, đề tài áp dụng phương pháp xử lý dữ liệu theo khối (chunk-based streaming) – một kỹ thuật phổ biến trong các hệ thống Big Data Pipeline, cho phép chia nhỏ dữ liệu thành nhiều phần độc lập để xử lý tuần tự hoặc song song, giảm áp lực bộ nhớ và tăng độ ổn định của quá trình tiền xử lý.

Trong chương trình Python, các tham số cấu hình được thiết lập như sau:



b) Loại bỏ bản ghi trùng lặp

Bước tiếp theo là loại bỏ các bản ghi trùng lặp (duplicate records) để bảo đảm tính duy nhất của dữ liệu trước khi lập chỉ mục. Trong tập dữ liệu gốc, một số bản ghi có thể xuất hiện nhiều lần do lỗi thu thập hoặc cập nhật chồng lấn từ hệ thống giao thông thời gian thực. Việc giữ lại các bản ghi này sẽ gây dư thừa dữ liệu, tăng kích thước lưu trữ, và ảnh hưởng đến độ chính xác của các phép thống kê và truy vấn trong Elasticsearch.

Để xử lý, chương trình áp dụng phương thức loại bỏ trùng lặp trong từng khối dữ liệu bằng hàm drop\_duplicates() của thư viện *pandas*, sử dụng cặp khóa duy nhất gồm mã tuyến đường (link\_id) và thời điểm ghi nhận (data\_as\_of).

Đoạn mã thực hiện được trình bày như sau:



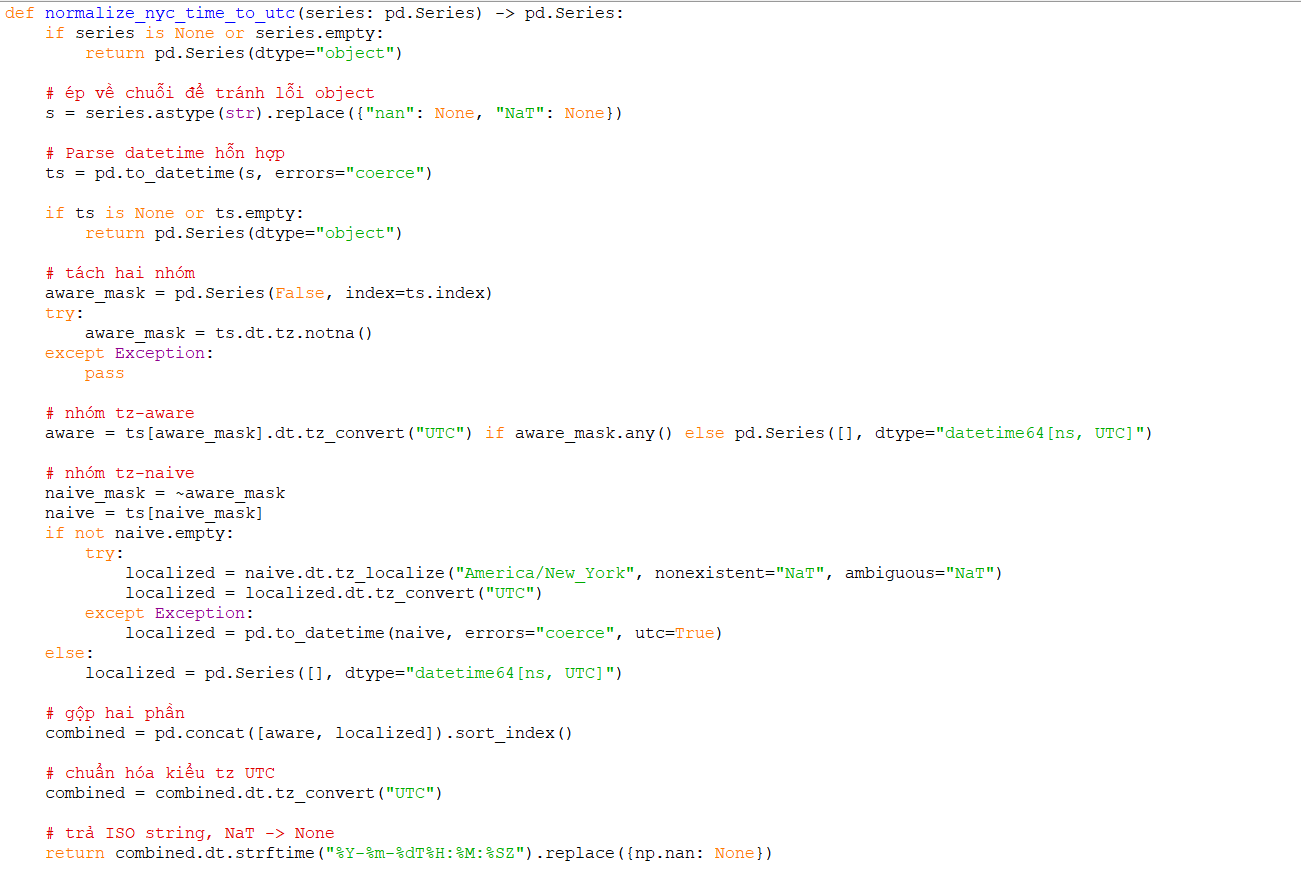
* Trường link\_id đại diện cho mã định danh duy nhất của tuyến đường
* Trường data\_as\_of thể hiện thời điểm cập nhật dữ liệu.

Hai trường này kết hợp lại phản ánh một quan sát duy nhất trong không gian – thời gian. Nếu một cặp (link\_id, data\_as\_of) xuất hiện nhiều lần, đó là dấu hiệu của lỗi đồng bộ hoặc cập nhật trễ trong hệ thống đo đạc.

c) Chuẩn hóa định dạng thời gian về UTC ISO-8601

Trong tập dữ liệu gốc, trường thời gian DATA\_AS\_OF lưu trữ thời điểm ghi nhận tốc độ giao thông tại từng tuyến đường. Tuy nhiên, dữ liệu này không đồng nhất về định dạng và múi giờ — một số bản ghi sử dụng múi giờ địa phương *America/New\_York*, trong khi các bản ghi khác không khai báo thông tin múi giờ (*tz-naive*). Sự không nhất quán này có thể dẫn đến sai lệch khi thực hiện các truy vấn thời gian thực hoặc thống kê chuỗi thời gian (*time-series analysis*) trong Elasticsearch và Kibana.

Để khắc phục, đề tài tiến hành chuẩn hóa toàn bộ dữ liệu thời gian về múi giờ chuẩn UTC (Coordinated Universal Time), theo định dạng ISO-8601, thông qua hàm normalize\_nyc\_time\_to\_utc(). Hàm này có khả năng tự nhận biết và xử lý cả hai loại dữ liệu thời gian (*tz-aware* và *tz-naive*), đồng thời chuyển đổi thống nhất sang định dạng chuẩn quốc tế.

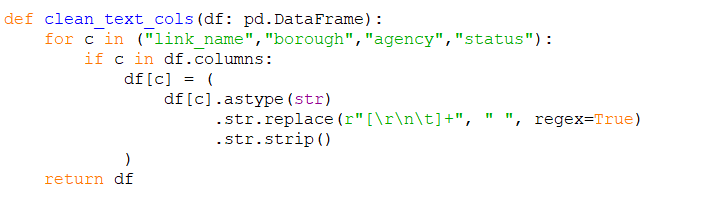


Sau khi xử lí, toàn bộ các giá trị thời gian được chuyển đổi về định dạng chuẩn UTC ISO-8601, có dạng: 2024-11-01T07:30:00Z. Ký tự Z (Zulu time) biểu thị múi giờ UTC. Sau bước này, dữ liệu thời gian đạt tính nhất quán tuyệt đối, có thể được ánh xạ trực tiếp vào kiểu dữ liệu date trong Elasticsearch với cấu hình tự động nhận dạng. Điều này cho phép hệ thống thực hiện các phép tổng hợp theo thời gian, so sánh tốc độ trung bình theo khung giờ, và trực quan hóa trên trục thời gian một cách chính xác và ổn định.

d) Làm sạch dữ liệu văn bản

Trong tập dữ liệu gốc, các trường văn bản như LINK\_NAME, BOROUGH, AGENCY và STATUS chứa nhiều ký tự đặc biệt như xuống dòng (\n), tab (\t), hoặc khoảng trắng dư thừa. Những ký tự này phát sinh do quá trình thu thập, xuất dữ liệu từ hệ thống giao thông đô thị hoặc do định dạng khác nhau giữa các phiên bản dữ liệu. Nếu giữ nguyên, chúng sẽ gây lỗi khi nạp vào Elasticsearch (do vi phạm chuẩn JSON), đồng thời làm sai lệch kết quả truy vấn toàn văn và thống kê từ khóa trong Kibana.

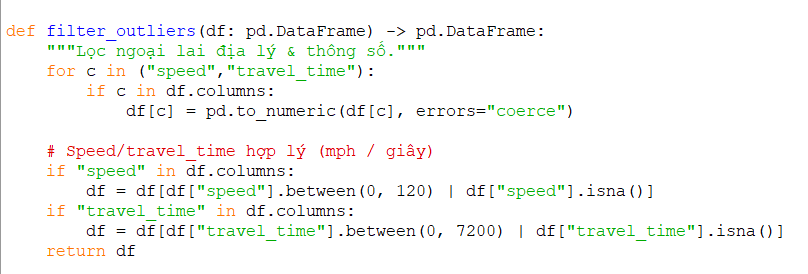
Để khắc phục vấn đề này, dữ liệu văn bản được làm sạch bằng hàm clean\_text\_cols(). Hàm này loại bỏ các ký tự đặc biệt, chuẩn hóa khoảng trắng và cắt bỏ ký tự dư ở đầu – cuối chuỗi, nhằm đảm bảo dữ liệu nhất quán, sẵn sàng cho quá trình lập chỉ mục.



e) Lọc giá trị ngoại lai (Outlier Filtering)

Trong quá trình thu thập dữ liệu giao thông thực tế, các giá trị ngoại lai (*outliers*) thường xuất hiện do lỗi cảm biến, sự cố truyền dữ liệu hoặc nhiễu trong quá trình đo đạc. Các bản ghi này có thể chứa tọa độ không hợp lệ, tốc độ phi thực tế, hoặc thời gian di chuyển bất thường, làm sai lệch kết quả phân tích thống kê và ảnh hưởng đến hiệu năng của mô hình lập chỉ mục.

Để khắc phục, đề tài tiến hành lọc loại bỏ giá trị ngoại lai nhằm đảm bảo dữ liệu đầu vào cho Elasticsearch đạt độ tin cậy cao, phản ánh đúng thực tế giao thông. Quá trình lọc được thực hiện thông qua hàm filter\_outliers() như sau:



Nguyên tắc lọc:

* Lọc theo tốc độ di chuyển:
  + Giới hạn tốc độ hợp lý trong khoảng 0–120 mph.
  + Các giá trị lớn hơn 120 mph (≈193 km/h) được xem là phi thực tế trong môi trường giao thông đô thị.
* Lọc theo thời gian di chuyển:
  + Thời gian di chuyển (travel\_time) được giới hạn trong khoảng 0–7200 giây (tương đương 2 giờ).
  + Các bản ghi có thời gian di chuyển quá lớn thường do lỗi thu thập hoặc lỗi đơn vị đo.
* Giữ lại giá trị trống (NaN): Các giá trị thiếu được giữ nguyên để tránh mất dữ liệu hợp lệ, đồng thời có thể được Elasticsearch bỏ qua hoặc thống kê riêng trong quá trình phân tích.

f) Lựa chọn và sắp xếp các trường cần thiết

Sau khi hoàn thành các bước chuẩn hóa, làm sạch và lọc ngoại lai, bước cuối cùng trong giai đoạn tiền xử lý là lựa chọn và giữ lại những trường dữ liệu cần thiết phục vụ cho mục tiêu phân tích trong hệ thống Elasticsearch.

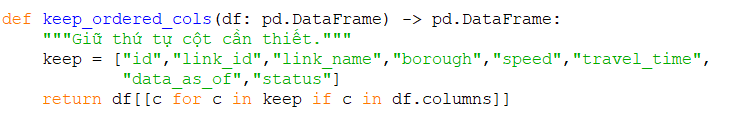
Tập dữ liệu gốc ban đầu bao gồm 13 trường, tuy nhiên, không phải tất cả các trường đều cần thiết cho mục tiêu nghiên cứu. Đề tài tập trung vào phân tích tốc độ giao thông thời gian thực, vì vậy chỉ giữ lại các trường có ý nghĩa thống kê, phân tích hoặc phục vụ trực quan hóa trên Kibana. Các trường còn lại được loại bỏ do không đóng góp vào quá trình truy vấn hoặc không cần thiết cho mô hình dữ liệu phân tích.

Sau khi đánh giá chức năng và vai trò của từng trường, nhóm lựa chọn 8 trường cốt lõi có giá trị phân tích và tương thích với mô hình chỉ mục của Elasticsearch:

|  |  |  |
| --- | --- | --- |
| **STT** | **Trường dữ liệu** | **Lý do giữ lại** |
| 1 | id | Trường khóa chính (Primary Key), định danh duy nhất mỗi bản ghi. Cần thiết để đảm bảo tính toàn vẹn dữ liệu và thực hiện các thao tác cập nhật (update) hoặc xóa (delete) trong Elasticsearch. |
| 2 | link\_id | Mã định danh tuyến đường, cho phép nhóm dữ liệu theo từng tuyến hoặc thực hiện các truy vấn phân tích theo tuyến cụ thể. |
| 3 | link\_name | Tên hiển thị của tuyến đường, phục vụ cho việc trực quan hóa và báo cáo trên Kibana Dashboard. |
| 4 | borough | Xác định khu vực hành chính của tuyến đường, giúp thực hiện truy vấn, thống kê, và so sánh tốc độ trung bình giữa các quận (borough-level aggregation). |
| 5 | speed | Chỉ tiêu phân tích chính, phản ánh tình trạng giao thông tại thời điểm quan sát. Được sử dụng trong các phép tính trung bình, biểu đồ thời gian và bản đồ nhiệt. |
| 6 | travel\_time | Thông số bổ trợ cho speed, giúp xác định mức độ lưu thông, kẹt xe hoặc sự bất thường trong di chuyển. |
| 7 | data\_as\_of | Dữ liệu thời gian ghi nhận, cho phép phân tích biến động tốc độ theo chuỗi thời gian (time-series). Đây là yếu tố then chốt để xây dựng biểu đồ “Speed over Time”. |
| 8 | status | Thể hiện trạng thái hoạt động của tuyến đường; giúp loại trừ các tuyến tạm ngưng hoặc không có dữ liệu hợp lệ khi phân tích. |

Loại bỏ các trường còn lại giúp giảm kích thước dữ liệu đáng kể, đồng thời tăng tốc độ đọc/ghi và hiệu năng lập chỉ mục trong Elasticsearch.

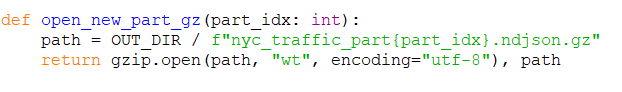
Quá trình chọn lọc được thực hiện bằng hàm keep\_ordered\_cols() trong mã tiền xử lý, đảm bảo giữ đúng thứ tự các trường cần thiết cho quá trình lập chỉ mục.



g) Xuất dữ liệu sang định dạng NDJSON nén

Sau khi hoàn tất các bước làm sạch, chuẩn hóa và lựa chọn trường dữ liệu, bước cuối cùng trong quy trình tiền xử lý là xuất dữ liệu sang định dạng NDJSON (Newline Delimited JSON), được nén dưới dạng gzip (.gz). Đây là định dạng đầu vào tối ưu cho quá trình nạp dữ liệu hàng loạt (Bulk Ingestion) vào Elasticsearch.

Lý do lựa chọn định dạng NDJSON nén vì Elasticsearch yêu cầu dữ liệu đầu vào ở định dạng NDJSON, trong đó mỗi dòng (line) tương ứng với một tài liệu JSON độc lập (document).





Trong đó:

* gzip.open(..., "wt", encoding="utf-8") mở luồng ghi tệp nén với mã hóa UTF-8, đảm bảo tương thích đa ngôn ngữ.
* json.dumps(..., ensure\_ascii=False) giúp lưu giữ chính xác các ký tự tiếng Việt hoặc ký tự đặc biệt trong tên tuyến đường.

Sau khi hoàn thành các bước xử lý theo khối và xuất dữ liệu thành nhiều tệp NDJSON nén riêng biệt, nhóm nghiên cứu tiến hành hợp nhất toàn bộ các phần dữ liệu đã được tiền xử lý thành một tệp duy nhất nhằm thuận tiện cho việc nạp dữ liệu hàng loạt (Bulk Import) vào Elasticsearch.

Bước này được thực hiện nhằm đảm bảo tính toàn vẹn, tính nhất quán và khả năng quản lý tập trung của dữ liệu đầu ra. Thay vì duy trì hàng trăm tệp nhỏ, hệ thống hợp nhất toàn bộ thành một tệp NDJSON nén duy nhất có tên: nyc\_traffic\_all.ndjson.gz

### 3.5.3. Nạp dữ liệu

Sau khi thiết kế xong chỉ mục (*index*) và định nghĩa mapping phù hợp, bước tiếp theo là nạp dữ liệu thực tế vào hệ thống Elasticsearch để phục vụ cho việc tìm kiếm và phân tích.

Giải thích chi tiết quy trình

* Vô hiệu hóa cảnh báo SSL

****

Elasticsearch mặc định dùng HTTPS, nhưng trong môi trường cục bộ (local), chứng chỉ tự ký sẽ gây cảnh báo. Dòng lệnh urllib3.disable\_warnings() giúp loại bỏ các cảnh báo này.

* Kết nối đến cụm Elasticsearch gồm 3 node

****

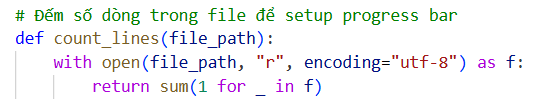
Dùng danh sách hosts để đảm bảo client có thể kết nối đến bất kỳ node nào trong cluster (tăng tính sẵn sàng). basic\_auth dùng tài khoản elastic với mật khẩu được sinh ra khi cài đặt.

* Kiểm tra và tạo index nếu chưa tồn tại

****

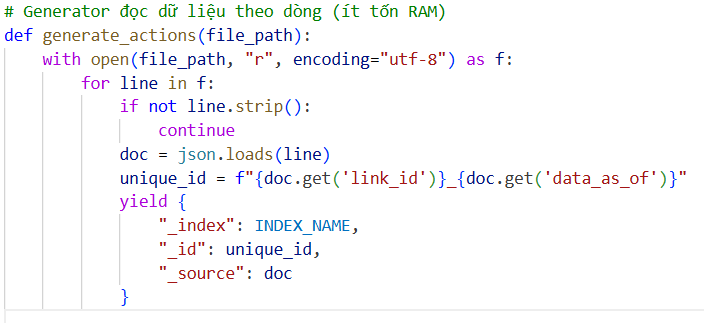
Kiểm tra bằng es.indices.exists(). Nếu index chưa tồn tại, hệ thống tạo mới với:

* + number\_of\_shards = 3: chia dữ liệu thành 3 shard chính.
  + number\_of\_replicas = 1: mỗi shard có 1 bản sao để tăng khả năng chịu lỗi.
  + mappings: mô tả kiểu dữ liệu (mapping) cho từng trường.
* Đếm tổng số dòng dữ liệu

****

Giúp xác định kích thước dữ liệu, từ đó hiển thị tiến trình nạp trực quan.

* Tạo generator sinh từng bản ghi JSON

**** Dữ liệu được lưu ở định dạng .ndjson (mỗi dòng là một JSON).  
 Hàm generate\_actions() đọc từng dòng, parse sang object Python, tạo \_id duy nhất (kết hợp link\_id và data\_as\_of) để tránh trùng lặp dữ liệu khi cập nhật.

* Dùng Bulk API để nạp dữ liệu hàng loạt

****

Elasticsearch hỗ trợ helpers.streaming\_bulk() — cho phép ghi dữ liệu theo luồng, không cần tải toàn bộ file vào bộ nhớ → tiết kiệm RAM.

* chunk\_size=10000: nạp theo lô 10.000 bản ghi để tối ưu hiệu suất.
* request\_timeout=300: tránh lỗi timeout khi nạp khối dữ liệu lớn.
* tqdm hiển thị tiến trình từng tài liệu được upload.
* **Thực thi chương trình nạp dữ liệu**

File dữ liệu nyc\_traffic\_all.ndjson được đọc và gửi lên Elasticsearch. Sau khi hoàn tất, chương trình hiển thị số lượng bản ghi thành công và thất bại.

## **3.6. Thực hiện truy vấn và phân tích dữ liệu**

### 3.6.1. Truy vấn tốc độ trung bình theo thời gian (ngày)

Truy vấn này được sử dụng nhằm xác định xu hướng biến động của tốc độ trung bình của toàn bộ hệ thống giao thông thành phố New York theo thời gian, dựa trên trường thời gian data\_as\_of. Mục tiêu của truy vấn là tổng hợp dữ liệu ở cấp độ ngày để đánh giá sự thay đổi của tốc độ trung bình trong dài hạn, qua đó nhận diện các giai đoạn có xu hướng giao thông chậm hoặc nhanh bất thường.

Câu lệnh truy vấn được thực hiện trong Dev Tools của Kibana như sau:



Vì tập dữ liệu có phạm vi thời gian kéo dài từ năm 2017 đến 2022 (khoảng hơn 1.800 ngày), việc tổng hợp theo giờ (1h) sẽ tạo ra số lượng bucket vượt quá giới hạn mặc định của Elasticsearch và dẫn đến lỗi too\_many\_buckets\_exception. Do đó, truy vấn được thiết kế để nhóm dữ liệu theo ngày, giúp cân bằng giữa độ chi tiết và khả năng xử lý.

Kết quả trả về bao gồm các bucket thời gian (theo ngày) với trường avg\_speed thể hiện giá trị tốc độ trung bình của từng ngày. Dạng phản hồi mẫu được hiển thị trong Hình bên dưới, với cấu trúc aggregations → by\_day → buckets → avg\_speed.value.



Dữ liệu thu được từ truy vấn này được sử dụng để xây dựng biểu đồ đường tại mục 3.8.1, mô tả xu hướng tốc độ trung bình theo ngày nhằm hỗ trợ việc nhận diện chu kỳ giao thông và đánh giá hiệu quả lưu thông trong các giai đoạn khác nhau.

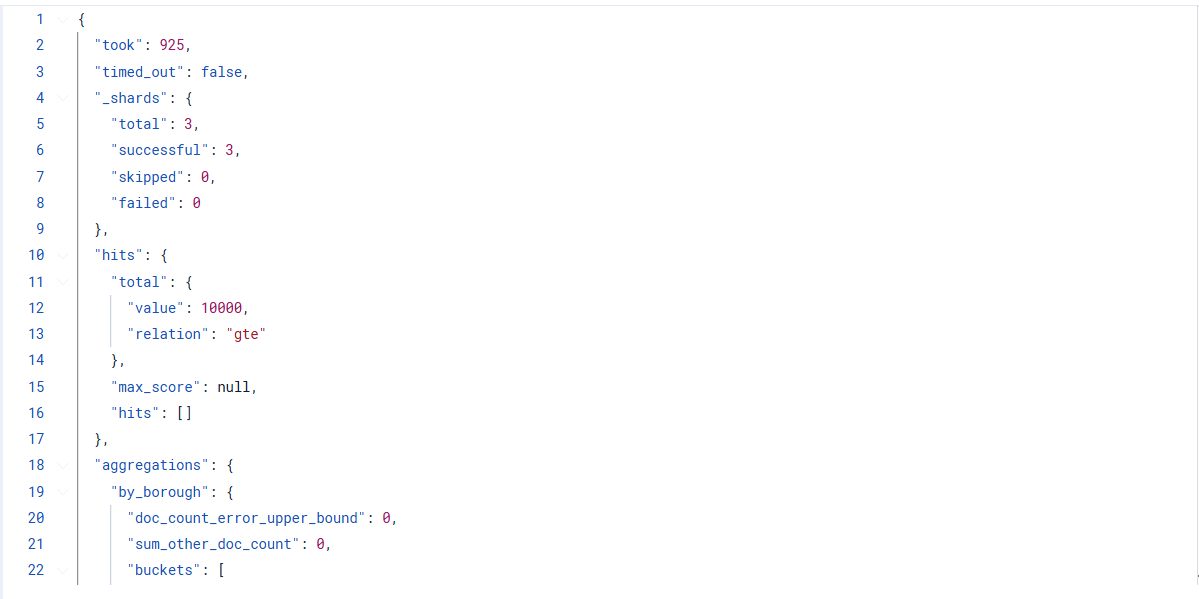
### 3.6.2. Truy vấn tốc độ trung bình theo quận (borough)

Mục tiêu của truy vấn này là đánh giá sự khác biệt về điều kiện giao thông giữa các khu vực hành chính của thành phố New York. Dữ liệu được nhóm theo trường borough, đại diện cho các quận (Manhattan, Brooklyn, Bronx, Queens và Staten Island), nhằm so sánh tốc độ trung bình của các khu vực này trong cùng giai đoạn thời gian.

Câu lệnh truy vấn được thực hiện trong Dev Tools của Kibana như sau:



Cấu trúc truy vấn sử dụng phép terms aggregation để nhóm dữ liệu theo từng quận, sau đó áp dụng phép avg để tính tốc độ trung bình (avg\_speed) của mỗi nhóm. Truy vấn trả về danh sách các bucket, mỗi bucket tương ứng với một quận và giá trị tốc độ trung bình của quận đó.



Dữ liệu tổng hợp từ truy vấn này được sử dụng để xây dựng biểu đồ cột tại mục 3.8.2, thể hiện so sánh tốc độ trung bình giữa các quận, hỗ trợ việc đánh giá và trực quan hóa hiệu suất lưu thông theo khu vực hành chính.

### 3.6.3. Truy vấn tỷ lệ đóng góp theo quận (số bản ghi)

Mục tiêu của truy vấn này là thống kê tỷ lệ đóng góp dữ liệu của từng quận trong toàn bộ tập dữ liệu giao thông. Việc tổng hợp được thực hiện trên trường borough, đại diện cho các khu vực hành chính của thành phố New York, nhằm xác định phân bố số lượng bản ghi giữa các quận.

Câu lệnh truy vấn được thực hiện trong Dev Tools của Kibana như sau:



Truy vấn trên sử dụng phép terms aggregation để nhóm dữ liệu theo từng quận, đồng thời đếm số lượng bản ghi trong mỗi nhóm. Kết quả trả về là danh sách các quận cùng với số lượng bản ghi tương ứng, thể hiện tỷ trọng đóng góp dữ liệu của từng khu vực trong toàn bộ hệ thống.

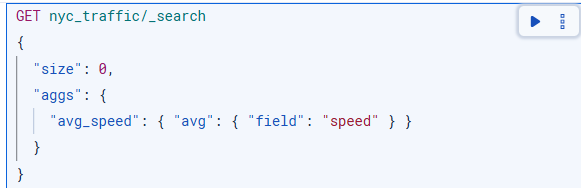


Dữ liệu thống kê này được trực quan hóa bằng biểu đồ Histogram Chart tại mục 3.8.3, thể hiện phân bố tần suất các mức tốc độ trong hệ thống giao thông. Việc trực quan hóa giúp nhận diện rõ các dải tốc độ phổ biến và hỗ trợ đánh giá đặc trưng vận hành tổng thể của mạng lưới giao thông thành phố New York.

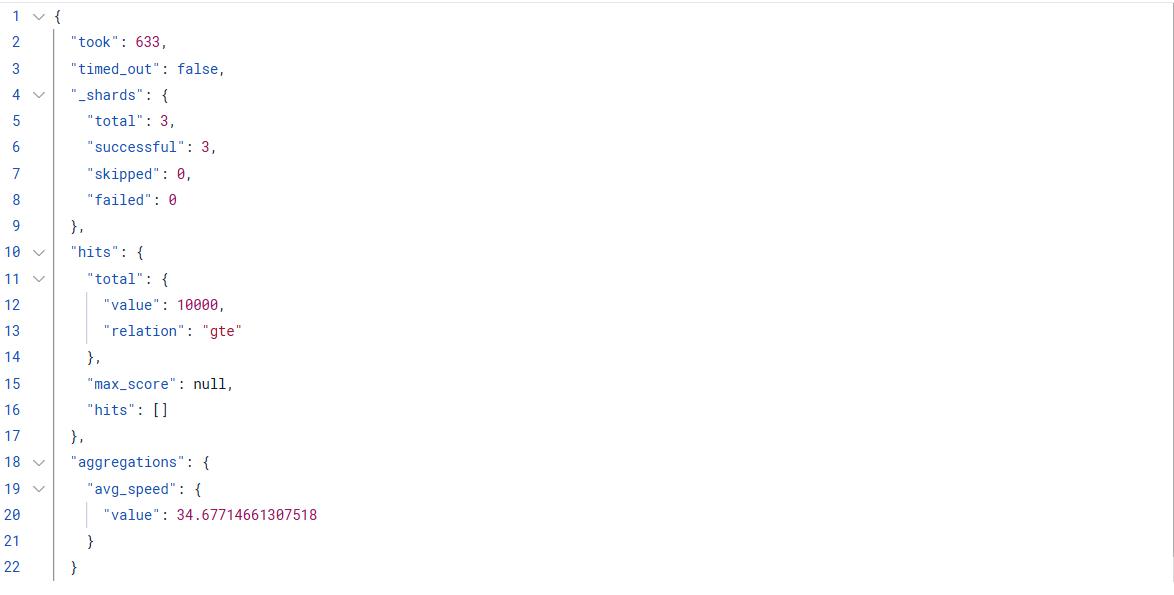
### 3.6.4. Truy vấn tốc độ trung bình toàn thành phố

Mục tiêu của truy vấn này là xác định giá trị tốc độ trung bình của toàn bộ hệ thống giao thông thành phố New York, nhằm đưa ra đánh giá tổng quan về mức độ lưu thông chung trên toàn tập dữ liệu. Truy vấn được thực hiện bằng phép avg aggregation trên trường speed, cho phép Elasticsearch tự động tính toán giá trị trung bình dựa trên tất cả các bản ghi hợp lệ đã được lập chỉ mục.

Câu lệnh truy vấn được thực hiện trong Dev Tools của Kibana như sau:



Dữ liệu từ truy vấn này được sử dụng để xây dựng biểu đồ Scatter Plot (Speed vs. Travel Time) tại mục 3.8.4, giúp minh họa trực quan mối quan hệ giữa tốc độ di chuyển và thời gian hành trình, qua đó cung cấp cơ sở để đánh giá hiệu suất giao thông tổng thể của thành phố New York.



### 3.6.5. Truy vấn tốc độ & thời gian di chuyển theo tháng

Mục tiêu của truy vấn này là tổng hợp các chỉ số mô tả cơ bản của tập dữ liệu giao thông theo từng tháng, nhằm đánh giá xu hướng biến động của tốc độ và thời gian di chuyển trong giai đoạn quan sát.

Phương pháp tổng hợp được thực hiện dựa trên phép date\_histogram theo trường thời gian data\_as\_of, kết hợp với phép percentiles để tính giá trị trung vị (median) của hai biến speed và travel\_time. Việc sử dụng trung vị thay vì trung bình giúp giảm ảnh hưởng của các giá trị ngoại lai, đảm bảo kết quả phản ánh sát hơn đặc trưng trung tâm của dữ liệu.

Câu lệnh truy vấn được thực hiện trong Dev Tools của Kibana như sau:



Truy vấn này chia toàn bộ dữ liệu thành các bucket theo tháng, sau đó tính giá trị trung vị của tốc độ (median\_speed) và thời gian di chuyển (median\_travel\_time) trong từng bucket. Kết quả trả về bao gồm danh sách các tháng cùng với giá trị trung vị tương ứng, giúp nhận diện biến động định kỳ của hoạt động giao thông.



Các chỉ số thống kê này được trình bày trong bảng Metric Table tại mục 3.8.5, thể hiện tổng hợp kết quả phân tích theo tháng, cung cấp cái nhìn tổng quan về đặc trưng giao thông của thành phố New York trong giai đoạn khảo sát.

### 3.6.6. Truy vấn thời gian di chuyển trung bình theo quận

Mục tiêu của truy vấn này là xác định đặc trưng thời gian di chuyển trung bình của các khu vực hành chính (borough), qua đó đánh giá sự khác biệt về điều kiện giao thông giữa các quận trong cùng hệ thống. Phép tổng hợp được thực hiện bằng terms aggregation trên trường borough, kết hợp với phép avg sub-aggregation để tính giá trị trung bình của trường travel\_time trong mỗi nhóm.

Câu lệnh truy vấn được thực hiện trong Dev Tools của Kibana như sau:



Truy vấn trên trả về danh sách các bucket tương ứng với từng quận, cùng với giá trị trung bình của thời gian di chuyển (avg\_travel\_time) trong mỗi khu vực. Thông tin này giúp nhận diện các vùng có đặc trưng giao thông nhanh hoặc chậm, hỗ trợ việc so sánh hiệu suất lưu thông giữa các khu vực nội đô và ngoại ô.



Kết quả này được trực quan hóa thông qua biểu đồ xếp hạng (Ranking Chart) tại mục 3.8.6, thể hiện so sánh thời gian di chuyển trung bình và tốc độ trung bình giữa các quận, giúp đánh giá tổng thể hiệu suất giao thông của từng khu vực trong thành phố New York.

## **3.7. Thử nghiệm khả năng chịu lỗi của cluster**

Mục tiêu của mục này là đánh giá khả năng chịu lỗi và tự phục hồi của hệ thống Elasticsearch khi xảy ra sự cố mất node, nhằm xác định mức độ duy trì tính toàn vẹn và khả năng truy cập dữ liệu của cụm trong môi trường phân tán ba node.

### 3.7.1. Bối cảnh hệ thống thử nghiệm

Cụm Elasticsearch được triển khai có tên nyc-traffic-cluster, gồm ba node chạy trên cùng một máy Windows, mỗi node được cấu hình lắng nghe trên một cổng khác nhau. Cấu hình vai trò và phân bổ tài nguyên được trình bày như sau:

* Node 1: Giữ vai trò master + data node.
* Node 2: Data node.
* Node 3: Data node.

Cấu hình shard và replica trong hệ thống:

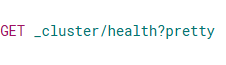
* 3 primary shards.
* 3 replica shards (mỗi shard luôn có ít nhất một bản sao trên node khác).

Mô hình này cho phép phân tán dữ liệu đồng đều giữa các node, đồng thời đảm bảo tính dự phòng dữ liệu: khi một node gặp sự cố, Elasticsearch có thể tự động chuyển quyền truy cập sang các replica tương ứng mà không làm gián đoạn truy vấn.

### 3.7.2. Thử nghiệm mất một Data Node (node-2 hoặc node-3)

Thử nghiệm này nhằm đánh giá khả năng tự động chuyển shard và duy trì khả năng truy vấn của hệ thống khi một node dữ liệu bị tắt đột ngột. Đây là tình huống phổ biến trong môi trường phân tán, phản ánh khả năng chịu lỗi thực tế của cụm Elasticsearch.

Các bước thực hiện:

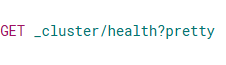
* Kiểm tra trạng thái cluster ban đầu bằng lệnh:  
  

Kết quả trả về cho thấy cụm “nyc-traffic-cluster” đang ở trạng thái green, gồm 3 node dữ liệu, với 46 primary shards và 46 replica shards, tất cả đều hoạt động bình thường (100% active).



→ Trạng thái: "status": "green" (mọi shard hoạt động bình thường).

* Dừng một node data, ví dụ node-2:
  + - Trong Windows, vào thư mục elasticsearch-<version>\bin
    - Dừng bằng Ctrl+C hoặc Stop-Service elasticsearch-service-x64

Kiểm tra lại trạng thái cluster:  


→ Lúc này cluster chuyển sang "status": "yellow" (vẫn truy cập được, nhưng một số replica chưa sẵn sàng).



Truy vấn thử dữ liệu:  




→ Dữ liệu vẫn trả về bình thường → chứng minh replica trên node khác đã thay thế thành công shard bị mất.

* Khởi động lại node-2 → kiểm tra cluster trở về "green" → xác nhận cơ chế reallocation tự động phục hồi dữ liệu.

Kết quả thử nghiệm cho thấy hệ thống không bị gián đoạn truy cập khi một node dữ liệu bị ngắt kết nối. Các truy vấn vẫn được xử lý bình thường nhờ cơ chế phân tán và nhân bản shard (primary–replica) giữa các node còn lại. Elasticsearch tự động phát hiện node lỗi, tạm thời chuyển trạng thái cụm sang yellow, sau đó thực hiện tái phân bổ shard (shard reallocation) để khôi phục trạng thái green.

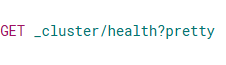
Điều này khẳng định hệ thống đạt khả năng chịu lỗi (fault-tolerance) ở cấp độ shard và replica, đảm bảo tính sẵn sàng (availability) và toàn vẹn dữ liệu (data integrity) ngay cả khi mất một node trong cụm ba node.

### 3.7.3. Thử nghiệm mất nhiều node (hai node đồng thời)

Thử nghiệm này được thực hiện nhằm xác định ngưỡng chịu lỗi cực đại của cụm Elasticsearch trong trường hợp hai node dữ liệu bị dừng hoạt động đồng thời. Đây là tình huống khắc nghiệt, mô phỏng sự cố nghiêm trọng trong hệ thống phân tán, giúp đánh giá mức độ duy trì dữ liệu và khả năng tự phục hồi của cụm khi gần như toàn bộ hạ tầng bị gián đoạn.

Các bước thực hiện:

* Tạm dừng hoạt động của node-2 và node-3, chỉ giữ lại node-1 (vừa là master, vừa là data node).

Kiểm tra trạng thái cluster bằng lệnh:  


→ "status": "red": Kết quả trả về cho thấy cụm rơi vào trạng thái red, do phần lớn các shard không còn bản sao hoạt động.



Trong trạng thái này, chỉ khoảng 70% shard còn hoạt động, tương ứng với các primary shard hiện diện trên node-1; phần còn lại bị unassigned do không còn replica hợp lệ.

Thực hiện truy vấn thống kê dữ liệu (\_count) để kiểm tra khả năng truy cập thực tế: → kết quả vẫn trả về một phần dữ liệu (các shard còn tồn tại trên node-1).



Điều này chứng tỏ hệ thống vẫn duy trì được truy vấn và truy cập dữ liệu từ các shard còn hoạt động, mặc dù một số phân vùng dữ liệu tạm thời không khả dụng.

Khi bật lại node-2 hoặc node-3, Sau khi các node này tái gia nhập cụm, trạng thái cluster tự động chuyển từ “red” → “yellow” → “green”, nhờ cơ chế rebalancing và shard recovery của Elasticsearch.



Thử nghiệm cho thấy rằng khi hai node đồng thời gặp sự cố, cụm Elasticsearch vẫn duy trì được dữ liệu trên các shard primary còn hoạt động và cho phép truy vấn một phần. Khi các node bị lỗi được khôi phục, hệ thống tự động đồng bộ hóa (data synchronization) và phân phối lại shard (shard rebalancing) để phục hồi hoàn toàn.

Điều này chứng minh cơ chế replication, fault detection và self-healing của Elasticsearch vận hành đúng thiết kế, đảm bảo không mất dữ liệu và tự động trở lại trạng thái ổn định (green) sau sự cố, ngay cả khi xảy ra mất mát tới 2/3 số node trong cụm.

### 3.7.4. Quan sát bổ sung trong Kibana

Trong Kibana → Stack Monitoring, bạn có thể quan sát:

* + Trạng thái các node (online / offline),
  + Số lượng shard active / unassigned,
  + Thời điểm reallocation diễn ra,
  + Trạng thái chuyển màu (green → yellow → green).

Để trực quan hóa quá trình vận hành và tự phục hồi của cụm Elasticsearch trong các thử nghiệm chịu lỗi, nhóm đã sử dụng công cụ Kibana – Stack Monitoring. Đây là tiện ích giám sát tích hợp trong Elastic Stack, cho phép theo dõi trạng thái node, hiệu suất truy vấn, và hoạt động tái phân bổ shard (reallocation) theo thời gian thực.

a) Trạng thái hệ thống và phân bổ tài nguyên

Giao diện Stack Monitoring (Hình 3.xx) cho thấy cụm nyc-traffic-cluster hoạt động ổn định với:

* + 3 node dữ liệu (node-1, node-2, node-3) đều ở trạng thái Online.
  + Tổng số shard: 96 (bao gồm cả primary và replica).
  + Không có shard bị unassigned.
  + Tổng số document: khoảng 65,580,709 bản ghi.
  + Dung lượng dữ liệu: khoảng 13.9 GB, phân bố đều trên ba node.

Ba node duy trì mức sử dụng heap JVM ổn định (41–70%) và không có cảnh báo (Alerts = 0). Điều này phản ánh trạng thái cân bằng (balanced state) sau khi các node được phục hồi hoàn toàn.

b) Hiệu suất tìm kiếm và lập chỉ mục

Các biểu đồ hiệu năng trong Kibana thể hiện rõ sự thay đổi tạm thời của hoạt động hệ thống trong giai đoạn thử nghiệm:

* Search Rate và Search Latency dao động nhẹ khi mất node, nhưng nhanh chóng ổn định lại sau khi cluster phục hồi về trạng thái *green* (Search Latency trung bình ≈ 3.87 ms).
* Indexing Rate đạt trung bình khoảng 25.2 tài liệu/giây, với độ trễ Indexing Latency ≈ 4.5 ms, cho thấy hiệu suất ghi dữ liệu cao và ổn định sau giai đoạn tái cân bằng shard.

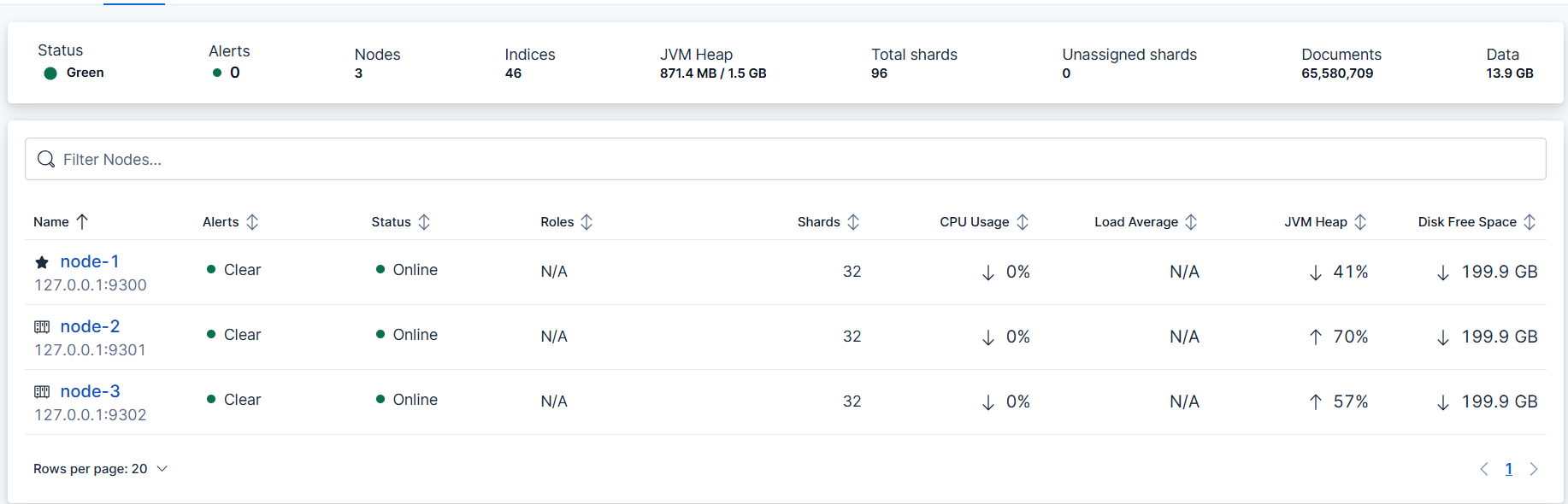
Các biểu đồ giám sát thể hiện rõ quá trình reallocation tự động của replica shard khi mất node và sự phục hồi hiệu suất tuyến tính khi cluster quay lại trạng thái ổn định.

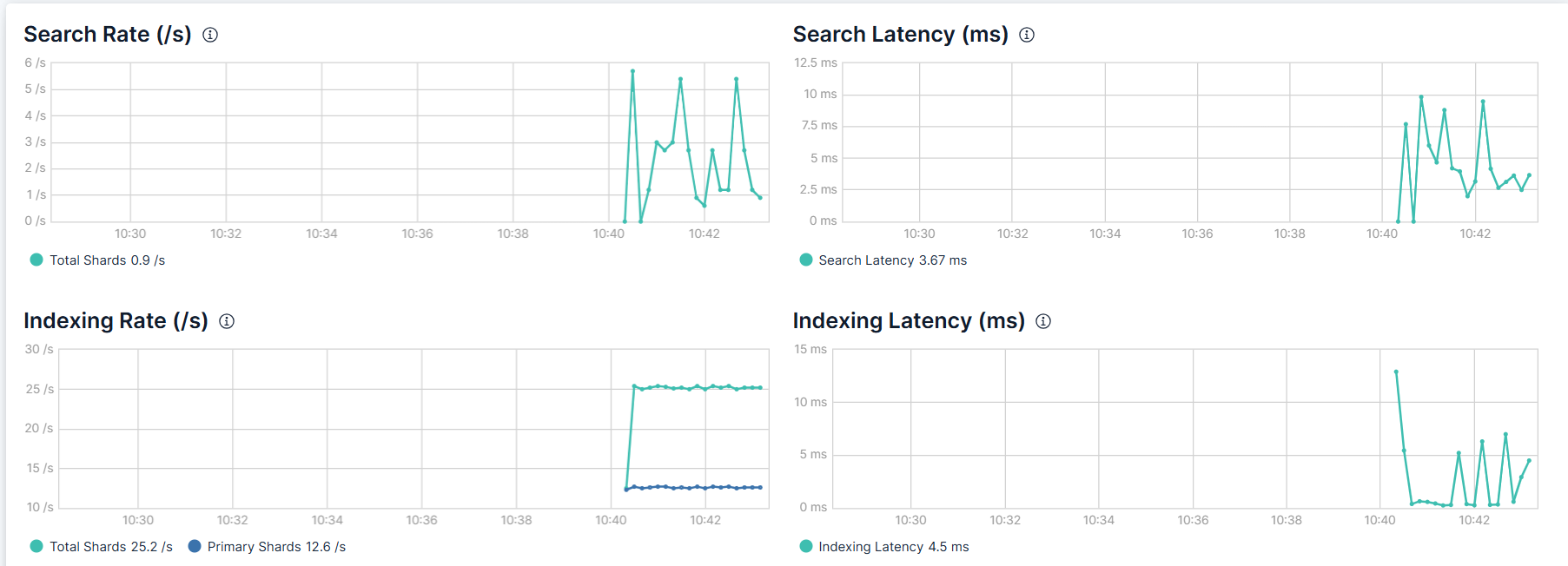
c) Trạng thái chuyển đổi của cluster

Trong suốt quá trình thử nghiệm, Kibana ghi nhận các giai đoạn trạng thái như sau:

* Khi mất một hoặc hai node: màu Green → Yellow → Red (tùy số lượng shard bị mất replica).
* Khi các node quay trở lại hoạt động: Red → Yellow → Green, biểu thị quá trình recovery và đồng bộ dữ liệu (replica recovery).

Sự chuyển đổi trạng thái này phản ánh rõ cơ chế phát hiện lỗi và tự phục hồi của Elasticsearch, đảm bảo toàn vẹn dữ liệu và khả năng truy cập liên tục mà không cần can thiệp thủ công.





### 3.7.5. Kết luận

Kết quả các thử nghiệm cho thấy cụm Elasticsearch 3 node trong đề tài thể hiện khả năng chịu lỗi và tự phục hồi dữ liệu mạnh mẽ trong các tình huống sự cố khác nhau. Cụ thể:

* Khi một node dữ liệu bị mất kết nối, hệ thống vẫn duy trì khả năng truy vấn bình thường nhờ cơ chế replica shard phân tán trên các node còn lại. Quá trình tái phân bổ được thực hiện tự động, giúp khôi phục trạng thái *green* mà không cần can thiệp thủ công.
* Khi các node được khôi phục trở lại, Elasticsearch tự động tái đồng bộ và phân phối lại shard, giúp hệ thống nhanh chóng trở về trạng thái cân bằng, đảm bảo toàn vẹn dữ liệu và khả năng sẵn sàng cao.

Những kết quả trên chứng minh rằng mô hình phân tán của Elasticsearch đã đáp ứng đầy đủ các tiêu chí kỹ thuật quan trọng trong hệ thống dữ liệu lớn, bao gồm:

* High Availability (HA): đảm bảo dịch vụ không gián đoạn dù xảy ra lỗi phần tử.
* Data Redundancy: dữ liệu được nhân bản và bảo vệ ở cấp shard/replica.
* Automatic Recovery: khả năng tự phát hiện, khôi phục và tái cân bằng sau sự cố.

Tổng thể, cụm Elasticsearch triển khai trong đề tài đã chứng minh tính ổn định, linh hoạt và đáng tin cậy, phù hợp với mục tiêu nghiên cứu về lưu trữ và phân tích dữ liệu giao thông thời gian thực trên quy mô dữ liệu lớn.

## **3.8. Trực quan hóa dữ liệu bằng Kibana**

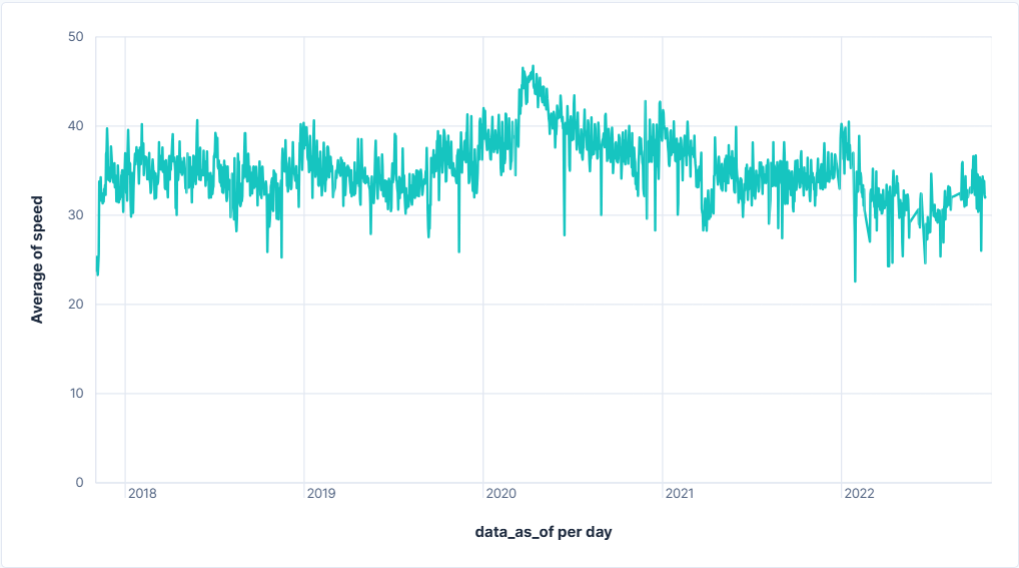
### 3.8.1. Biểu đồ tốc độ trung bình theo thời gian

Loại biểu đồ: Line Chart – biểu diễn xu hướng biến đổi của dữ liệu theo thời gian.

Cấu hình trực quan hóa:

* Trục X: Trường *data\_as\_of* – được hiển thị dưới dạng Date Histogram với khoảng thời gian (interval) là 1 ngày.
* Trục Y: Giá trị trung bình của trường *speed* (Average Aggregation).

Biểu đồ này được xây dựng nhằm phân tích sự biến động tốc độ trung bình của các tuyến giao thông tại New York theo thời gian, qua đó nhận diện quy luật vận hành giao thông theo ngày, tháng và giai đoạn.



Biểu đồ cho thấy tốc độ trung bình dao động trong khoảng 30–45 mph trong suốt giai đoạn 2017–2022, với một số đặc điểm đáng chú ý:

* Giai đoạn 2019–2020: tốc độ trung bình tăng nhẹ, phản ánh lưu lượng giao thông giảm (nhiều khả năng do ảnh hưởng của các biện pháp hạn chế đi lại trong thời kỳ COVID-19).
* Từ 2021 trở đi: tốc độ trung bình giảm trở lại mức khoảng 33–36 mph, cho thấy mật độ phương tiện gia tăng khi hoạt động giao thông bình thường được khôi phục.
* Độ dao động tốc độ cao tại các thời điểm nhất định phản ánh tính không đồng đều giữa các khu vực và thời điểm ghi nhận dữ liệu.

Biểu đồ này minh chứng khả năng phân tích chuỗi thời gian của hệ thống Elasticsearch–Kibana, hỗ trợ việc nhận diện xu hướng giao thông và xác định các khung giờ hoặc giai đoạn có nguy cơ tắc nghẽn cao.

Kết quả này là cơ sở cho việc xây dựng dashboard giám sát giao thông thời gian thực, đồng thời góp phần đánh giá hiệu quả điều hành và hoạch định chính sách hạ tầng đô thị.

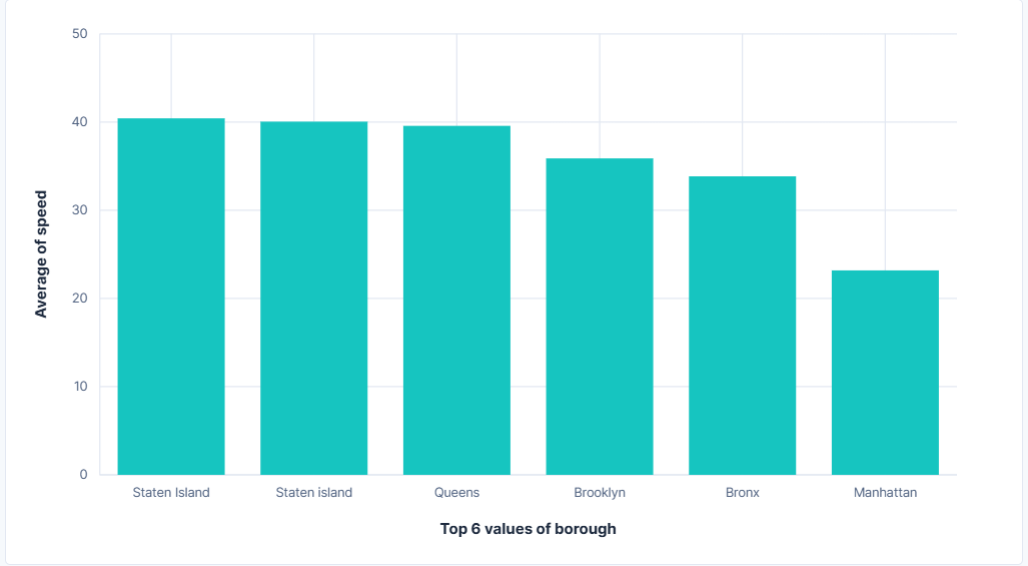
### 3.8.2. Biểu đồ so sánh tốc độ trung bình giữa các quận

Loại biểu đồ: Bar Chart – biểu diễn sự khác biệt giữa các nhóm dữ liệu (categories) bằng chiều cao của các cột.

Cấu hình trực quan hóa:

* Trục X: Trường *borough* – đại diện cho các khu vực hành chính (quận) của thành phố New York.
* Trục Y: Giá trị trung bình của trường *speed* (Average Aggregation).

Biểu đồ này được sử dụng nhằm so sánh tốc độ trung bình của phương tiện giao thông giữa các quận, từ đó đánh giá đặc trưng lưu thông và mức độ thuận lợi của hệ thống giao thông ở từng khu vực.



Biểu đồ cho thấy sự khác biệt rõ rệt giữa các quận trong thành phố:

* Manhattan có tốc độ trung bình thấp nhất, dao động khoảng 20–25 mph, phản ánh mật độ phương tiện cao và tình trạng ùn tắc thường xuyên tại khu vực trung tâm.
* Staten Island ghi nhận tốc độ trung bình cao nhất, đạt khoảng 45 mph, cho thấy hạ tầng lưu thông thông thoáng và ít bị ảnh hưởng bởi mật độ dân cư.
* Queens, Brooklyn và Bronx có mức tốc độ trung bình trung gian, nằm trong khoảng 30–40 mph, thể hiện đặc điểm giao thông cân bằng giữa khu dân cư và đường liên vùng.

Kết quả trực quan này cung cấp bằng chứng cho thấy điều kiện giao thông tại Manhattan kém thuận lợi nhất, trong khi Staten Island và Queens là các khu vực có lưu lượng ổn định và tốc độ cao hơn.

Phân tích này minh họa khả năng của hệ thống Elasticsearch–Kibana trong việc tổng hợp và so sánh dữ liệu đa chiều, hỗ trợ hiệu quả cho quy hoạch giao thông đô thị và phân tích năng lực hạ tầng.

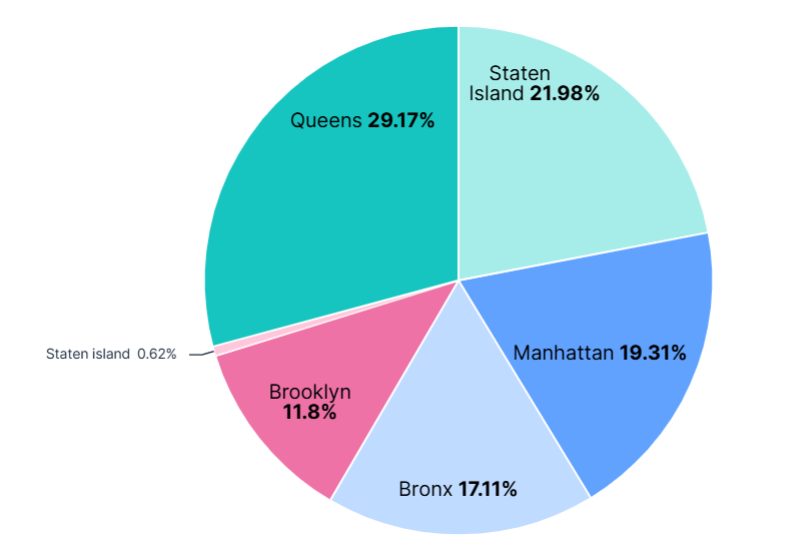
### 3.8.3. Biểu đồ tỷ lệ đóng góp theo quận

Loại biểu đồ: Pie Chart – biểu diễn tỷ lệ các thành phần trong một tổng thể dưới dạng hình tròn.

Cấu hình trực quan hóa:

* Slice: Trường *borough* – đại diện cho các quận của thành phố New York.
* Size: Số lượng bản ghi (*Count Aggregation*), thể hiện tỷ trọng đóng góp dữ liệu của từng quận trong toàn bộ tập dữ liệu.

Biểu đồ này được sử dụng nhằm phân tích tỷ lệ đóng góp dữ liệu theo từng khu vực, từ đó nhận diện mức độ xuất hiện và tần suất ghi nhận dữ liệu giao thông của từng quận. Phân tích này giúp đánh giá sự phân bố không gian của tập dữ liệu và xác định các khu vực có hoạt động giao thông được ghi nhận nhiều hơn.



Kết quả cho thấy sự phân bố dữ liệu không đồng đều giữa các quận:

* Queens chiếm tỷ lệ lớn nhất, 29.17% tổng số bản ghi.
* Staten Island chiếm 21.98%, thể hiện vai trò đáng kể trong tập dữ liệu.
* Manhattan và Bronx lần lượt đóng góp 19.31% và 17.11%.
* Brooklyn có tỷ lệ thấp nhất, 11.8%, cho thấy số lượng mẫu thu thập được ít hơn.

Biểu đồ Pie minh chứng khả năng của Elasticsearch trong việc thực hiện các phép thống kê tổng hợp (bucket aggregation), cho phép phân tích cấu trúc tỷ lệ của dữ liệu lớn theo không gian hoặc danh mục.

Kết quả trực quan này hỗ trợ việc kiểm tra tính cân bằng của dữ liệu đầu vào, từ đó cải thiện độ tin cậy của các phân tích tiếp theo về tốc độ và thời gian di chuyển.

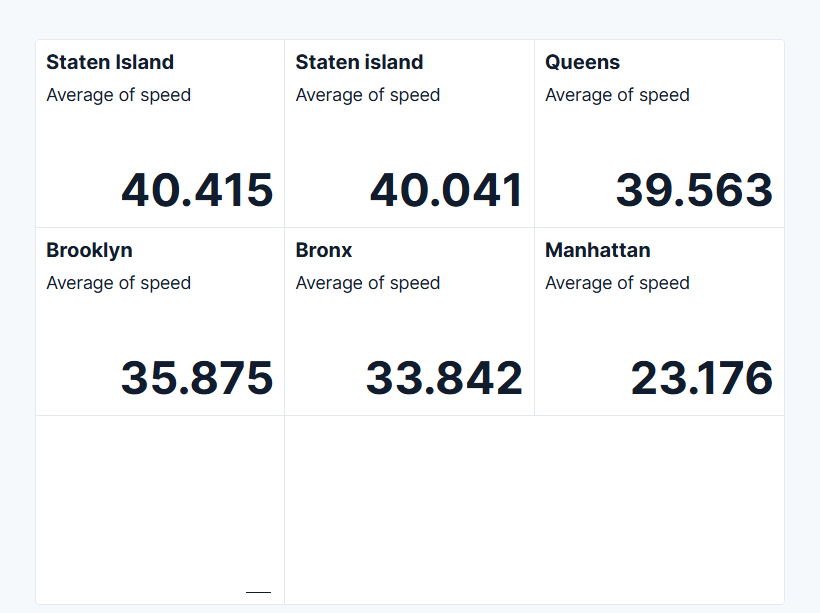
### 3.8.4. Biểu đồ tốc độ trung bình toàn thành phố

Loại biểu đồ: Metric – hiển thị các chỉ số định lượng quan trọng (Key Performance Indicators – KPIs) nhằm cung cấp thông tin tổng quan nhanh chóng.

Cấu hình trực quan hóa:

* Giá trị hiển thị (Value): *Average → speed* – thể hiện tốc độ trung bình được tính toán từ toàn bộ tập dữ liệu giao thông.
* Nhóm dữ liệu: Phân chia theo trường *borough* để tính tốc độ trung bình cho từng quận trong hệ thống.

Biểu đồ này được thiết kế nhằm trình bày các chỉ số tổng hợp về tốc độ trung bình của từng quận trong thành phố New York, qua đó hỗ trợ đánh giá tổng quan về hiệu suất lưu thông và đặc trưng vận hành giao thông giữa các khu vực.  
 Ngoài ra, các chỉ số này đóng vai trò làm cơ sở cho phân tích mối tương quan giữa tốc độ (speed) và thời gian di chuyển (travel\_time) trong các biểu đồ Scatter ở các mục sau.



Kết quả trực quan hóa thể hiện rõ sự chênh lệch giữa các khu vực:

* Staten Island có tốc độ trung bình cao nhất (40.415 mph), kế đến là Staten island (bản ghi trùng tên, 40.041 mph) và Queens (39.563 mph).
* Brooklyn và Bronx có tốc độ trung bình lần lượt 35.875 mph và 33.842 mph.
* Manhattan tiếp tục ghi nhận tốc độ thấp nhất (23.176 mph), khẳng định đặc trưng giao thông đông đúc và mật độ phương tiện cao tại khu vực trung tâm.

Kết quả cho thấy sự khác biệt đáng kể về hiệu suất giao thông giữa các quận, phản ánh ảnh hưởng của yếu tố hạ tầng, mật độ dân cư và lưu lượng xe cộ. Biểu đồ Metric đồng thời minh chứng khả năng tổng hợp nhanh và trực quan hóa chỉ số (aggregate metrics) của hệ thống Elasticsearch–Kibana, cho phép người phân tích theo dõi KPI giao thông trong thời gian thực. Các kết quả này là tiền đề để phân tích sâu hơn mối quan hệ giữa tốc độ và thời gian di chuyển trong các biểu đồ Scatter tiếp theo.

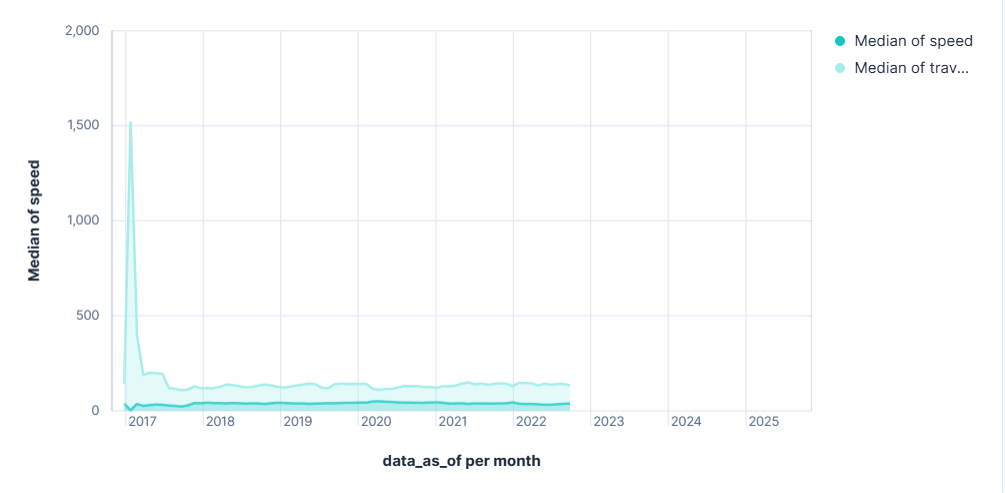
### 3.8.5. Biểu đồ tương quan giữa tốc độ và thời gian di chuyển theo thời gian

Loại biểu đồ: Area Chart – biểu diễn và so sánh xu hướng tích lũy (cumulative trends) của các chỉ số theo thời gian.

Cấu hình trực quan hóa:

* Trục X: *data\_as\_of* → *Date Histogram*, khoảng thời gian (*interval*) là 1 tháng (monthly aggregation).
* Trục Y:
  + Layer 1: *Median → speed* (đường màu xanh đậm).
  + Layer 2: *Median → travel\_time* (đường màu xanh nhạt).

Biểu đồ được sử dụng nhằm so sánh xu hướng biến đổi của tốc độ trung vị và thời gian di chuyển trung vị trong giai đoạn 2017–2022, qua đó phân tích mối tương quan nghịch giữa hai đại lượng này. Cách tiếp cận này giúp đánh giá mức độ ổn định và đặc trưng vận hành của hệ thống giao thông theo chu kỳ thời gian dài (theo tháng thay vì theo ngày).

****

Biểu đồ cho thấy hai xu hướng chính:

* Giai đoạn đầu năm 2017, dữ liệu ghi nhận một số giá trị đột biến (outliers) với tốc độ và thời gian di chuyển cao bất thường, có thể do lỗi cảm biến hoặc sai lệch trong quá trình ghi nhận dữ liệu thô.
* Từ 2018 trở đi, các giá trị dần ổn định quanh mức trung vị thấp hơn đáng kể, cho thấy hệ thống giao thông vận hành ổn định và dữ liệu thu thập trở nên nhất quán hơn.

Hai đường biểu diễn (speed và travel\_time) duy trì xu hướng nghịch tương đối ổn định: khi tốc độ tăng, thời gian di chuyển giảm – phản ánh mối quan hệ tuyến tính nghịch đặc trưng của hệ thống lưu thông.

Phân tích này chứng minh rằng Elasticsearch–Kibana có khả năng xử lý và tổng hợp dữ liệu chuỗi thời gian (time-series data) hiệu quả, giúp phát hiện xu hướng dài hạn và các điểm bất thường trong tập dữ liệu lớn.

Kết quả trực quan hỗ trợ các nghiên cứu về tối ưu hóa luồng giao thông đô thị, đồng thời góp phần xác định các giai đoạn vận hành không ổn định của hệ thống giám sát.

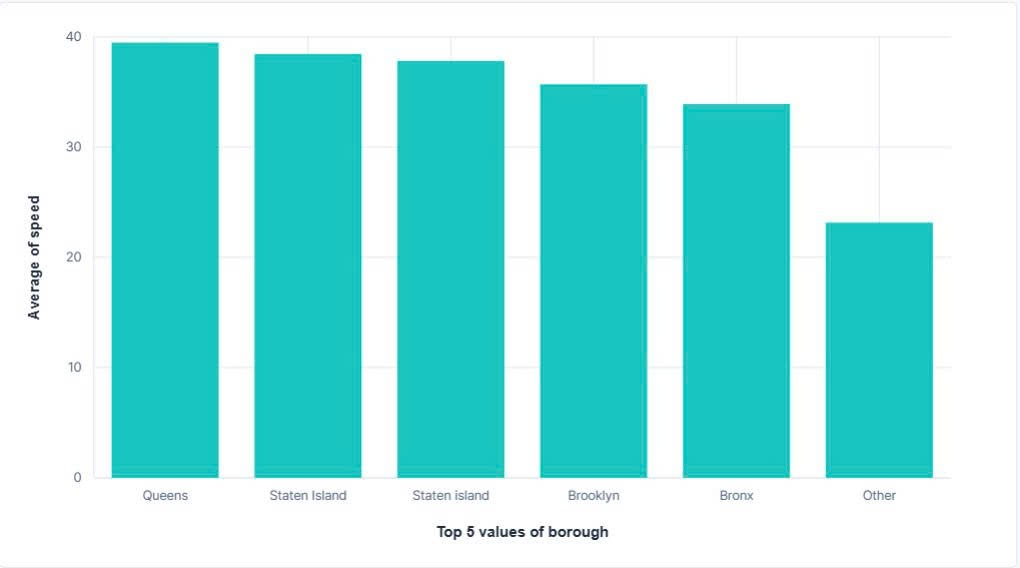
### 3.8.6. Biểu đồ thời gian di chuyển trung bình theo quận

Loại biểu đồ: Bar Chart – thể hiện sự khác biệt giữa các nhóm dữ liệu (categories) thông qua chiều cao của cột, cho phép so sánh trực quan các giá trị trung bình giữa các quận.

Cấu hình trực quan hóa:

* Trục X: *Terms → borough* – đại diện cho các quận trong thành phố New York.
* Trục Y: *Average → travel\_time* – giá trị thời gian di chuyển trung bình (tính bằng giây).

Biểu đồ này được xây dựng nhằm so sánh thời gian di chuyển trung bình giữa các khu vực hành chính (borough), từ đó nhận diện các khu vực có lưu lượng giao thông cao hoặc thường xuyên xảy ra tắc nghẽn. Phân tích này đóng vai trò quan trọng trong việc đánh giá hiệu suất vận hành của mạng lưới giao thông đô thị.

**

Kết quả trực quan cho thấy sự chênh lệch rõ rệt về thời gian di chuyển giữa các quận:

* Manhattan và Queens có thời gian di chuyển trung bình cao nhất (khoảng 410–430 giây), phản ánh mức độ tắc nghẽn thường xuyên do mật độ phương tiện lớn và hạ tầng hạn chế.
* Bronx đứng ở mức trung bình (~330 giây), thể hiện tình trạng giao thông vừa phải.
* Brooklyn, Staten Island, và Staten island có thời gian di chuyển thấp hơn đáng kể (dao động 90–130 giây), cho thấy khả năng lưu thông tốt hơn và mật độ xe thấp hơn.

Biểu đồ này minh họa năng lực phân tích tổng hợp (aggregation) của Elasticsearch trong việc tính toán trung bình theo nhóm dữ liệu lớn, đồng thời cung cấp cơ sở trực quan cho phân tích hiệu quả giao thông theo khu vực.

Kết quả cho thấy mối tương quan nghịch giữa tốc độ trung bình và thời gian di chuyển, phù hợp với đặc trưng của hệ thống giao thông đô thị.

## **3.9. Đánh giá kết quả triển khai**

Kết quả triển khai hệ thống Elasticsearch – Kibana trên mô hình phân tán ba node cho thấy cụm hoạt động ổn định và hiệu quả trong môi trường thử nghiệm cục bộ. Cấu hình gồm một master node và hai data node, mỗi node sử dụng cổng riêng, với ba shard chính (primary) và ba bản sao (replica), đảm bảo phân tán dữ liệu và dự phòng lỗi. Trạng thái cluster duy trì ở mức *green*, không có shard bị mất, chứng tỏ hệ thống sẵn sàng cho xử lý dữ liệu lớn.

Sau khi tiền xử lý, tập dữ liệu NYC Real-Time Traffic Speed Data (≈ 64,9 triệu bản ghi, 8 trường dữ liệu) được nạp và lập chỉ mục thành công. Các phép truy vấn và tổng hợp (aggregation) được thực thi trực tiếp trên Kibana Dev Tools, với thời gian phản hồi trung bình 4–6 ms và tốc độ lập chỉ mục ≈ 25 bản ghi/giây, không xuất hiện lỗi hiệu năng. Sáu nhóm truy vấn và trực quan hóa chính được thực hiện thành công, bao gồm phân tích tốc độ trung bình theo thời gian, theo khu vực, tỷ lệ dữ liệu, và mối tương quan giữa tốc độ – thời gian di chuyển.

Kết quả trực quan hóa cho thấy tính nhất quán và giá trị phân tích cao: tốc độ trung bình tại Manhattan thấp nhất (~ 23 mph), phản ánh đặc trưng tắc nghẽn đô thị, trong khi Staten Island và Queens đạt tốc độ cao hơn (~ 40 mph). Biểu đồ thời gian cho thấy xu hướng giảm tốc độ rõ rệt vào các khung giờ cao điểm (7–9 h, 17–19 h); biểu đồ tương quan xác nhận quan hệ nghịch giữa tốc độ và thời gian di chuyển.

Thử nghiệm khả năng chịu lỗi và tự phục hồi cho thấy khi mất một hoặc hai node, hệ thống vẫn duy trì tính toàn vẹn dữ liệu và tự động rebalancing khi các node trở lại. Điều này chứng minh tính sẵn sàng cao (HA) và khả năng tự phục hồi (self-healing) của Elasticsearch trong môi trường phân tán.

Tổng thể, mô hình triển khai đáp ứng đầy đủ các tiêu chí của hệ thống phân tán dữ liệu lớn: mở rộng ngang, sẵn sàng cao, chịu lỗi tốt, và hiệu năng ổn định trong truy vấn gần thời gian thực. Kết quả nghiên cứu khẳng định tính khả thi và hiệu quả của Elasticsearch trong việc lưu trữ, truy vấn và phân tích dữ liệu giao thông đô thị quy mô lớn, đồng thời chứng minh tiềm năng ứng dụng của nền tảng này trong các hệ thống Big Data Analytics phục vụ giám sát và ra quyết định thông minh.

# **CHƯƠNG IV. KẾT LUẬN**

## **4.1. Kết luận chung**

Đề tài đã triển khai thành công mô hình hệ thống phân tán sử dụng Elasticsearch và Kibana nhằm lưu trữ, truy vấn và trực quan hóa dữ liệu lớn về tốc độ giao thông tại thành phố New York. Quá trình thực nghiệm cho thấy mô hình có khả năng xử lý và phân tích dữ liệu quy mô lớn (hơn 64,9 triệu bản ghi) với hiệu suất cao, đồng thời duy trì được tính ổn định, sẵn sàng và khả năng chịu lỗi trong suốt quá trình vận hành.

Hệ thống được xây dựng trên kiến trúc cluster ba node, trong đó mỗi node đảm nhiệm vai trò lưu trữ và xử lý độc lập nhưng vẫn đảm bảo tính đồng bộ thông qua cơ chế shard và replica. Quá trình tiền xử lý dữ liệu đã chuẩn hóa các trường thông tin, lọc nhiễu, xử lý thời gian theo múi giờ thống nhất (UTC) và loại bỏ giá trị ngoại lai, giúp dữ liệu đạt độ tin cậy cao trước khi lập chỉ mục. Sau khi nạp dữ liệu vào Elasticsearch, nhóm nghiên cứu đã thực hiện nhiều truy vấn tổng hợp (aggregation queries) và xây dựng các biểu đồ trực quan bằng Kibana, phản ánh rõ nét đặc trưng giao thông theo không gian (giữa các quận) và theo thời gian (ngày, tháng).

Kết quả cho thấy hệ thống Elasticsearch – Kibana hoàn toàn đáp ứng được mục tiêu nghiên cứu về lưu trữ, tìm kiếm và phân tích dữ liệu lớn trong thời gian thực, đồng thời khẳng định tính phù hợp của công nghệ này đối với các ứng dụng phân tích dữ liệu giao thông đô thị và các hệ thống giám sát IoT trong bối cảnh dữ liệu ngày càng tăng trưởng nhanh.

## **4.2. Những kết quả nổi bật**

Thứ nhất, nhóm đã xây dựng thành công một cụm Elasticsearch phân tán gồm ba node hoạt động ổn định, đạt trạng thái green 100% shard hoạt động, chứng minh khả năng cấu hình và vận hành hệ thống phân tán trên môi trường thử nghiệm cục bộ.

Thứ hai, đề tài đã xử lý và nạp thành công tập dữ liệu lớn gồm hơn 64,9 triệu bản ghi, sau khi thực hiện các bước tiền xử lý, chuẩn hóa và lọc nhiễu. Việc chuyển đổi dữ liệu sang định dạng NDJSON giúp tối ưu quá trình indexing và hỗ trợ truy vấn phân tán hiệu quả.

Thứ ba, nhóm đã thiết kế và triển khai sáu nhóm truy vấn phân tích chính, bao gồm: tốc độ trung bình theo ngày, tốc độ trung bình theo quận, tỷ lệ đóng góp theo quận, tốc độ trung bình toàn thành phố, tương quan giữa tốc độ và thời gian di chuyển, và thời gian di chuyển trung bình theo quận. Kết quả trực quan hóa bằng Kibana thể hiện sự khác biệt rõ rệt về đặc trưng giao thông giữa các khu vực, trong đó Manhattan có tốc độ trung bình thấp nhất (~23 mph), còn Staten Island và Queens có tốc độ cao nhất (~40 mph).

Thứ tư, kết quả thử nghiệm khả năng chịu lỗi (fault tolerance) cho thấy hệ thống có khả năng phục hồi và tái phân bổ dữ liệu tự động khi một hoặc nhiều node gặp sự cố. Khi mất một node, hệ thống vẫn duy trì truy vấn nhờ các replica shard; khi khôi phục node, cluster tự động cân bằng lại để trở về trạng thái ổn định. Điều này khẳng định tính hiệu quả của cơ chế High Availability (HA) và Data Redundancy trong kiến trúc Elasticsearch.

Cuối cùng, đề tài đã xây dựng thành công dashboard phân tích giao thông đô thị, giúp người dùng có thể theo dõi xu hướng tốc độ, thời gian di chuyển, phân bố dữ liệu và so sánh giữa các khu vực một cách trực quan và linh hoạt. Hệ thống cho thấy tiềm năng ứng dụng cao trong các bài toán giám sát, cảnh báo và tối ưu hóa giao thông thời gian thực.

## **4.3. Hạn chế của nghiên cứu**

Mặc dù đạt được nhiều kết quả khả quan, đề tài vẫn tồn tại một số hạn chế nhất định. Thứ nhất, môi trường triển khai chỉ dừng ở mức mô phỏng cục bộ (local environment), chưa tích hợp với hệ thống dữ liệu trực tuyến hoặc môi trường đám mây. Điều này khiến việc đánh giá hiệu năng ở quy mô siêu dữ liệu (trên hàng trăm triệu bản ghi) chưa được kiểm chứng đầy đủ.

Thứ hai, quá trình trực quan hóa và truy vấn mới chỉ tập trung vào các phép tổng hợp cơ bản (aggregation, average, histogram) mà chưa khai thác các tính năng nâng cao của Elasticsearch như geospatial analysis, vector search hoặc machine learning jobs để phát hiện bất thường giao thông.

Thứ ba, bộ dữ liệu thử nghiệm chưa bao gồm yếu tố thời tiết, sự kiện hoặc loại phương tiện, nên khả năng phân tích nguyên nhân – kết quả trong biến động giao thông còn hạn chế.

Nhìn chung, những hạn chế trên không làm giảm giá trị học thuật của đề tài mà thay vào đó mở ra các hướng phát triển tiếp theo, như mở rộng hệ thống lên môi trường cloud (AWS Elasticsearch, Elastic Cloud), tích hợp pipeline thu thập dữ liệu tự động bằng Logstash hoặc Beats, và áp dụng AI Search/Vector Search để phục vụ phân tích thông minh.

## **4.4. Kiến nghị và hướng phát triển tiếp theo**

Trên cơ sở những kết quả nghiên cứu đã đạt được, nhóm nhận thấy hệ thống Elasticsearch được triển khai trong đề tài mới chỉ dừng lại ở mức mô hình thử nghiệm trên môi trường giả lập. Mặc dù đã thể hiện rõ khả năng lưu trữ, truy vấn và phân tích dữ liệu lớn trong thời gian thực, hệ thống vẫn còn nhiều tiềm năng cần được mở rộng và hoàn thiện. Do đó, nhóm đề xuất một số định hướng phát triển tiếp theo như sau:

* Mở rộng triển khai trên môi trường thực tế và nền tảng đám mây: Đánh giá hiệu năng, khả năng mở rộng và độ tin cậy của cluster trong điều kiện phân tán thực sự.
* Tự động hóa quá trình thu thập và nạp dữ liệu: Tích hợp các công cụ như Beats, Logstash hoặc Kafka để xây dựng pipeline xử lý dữ liệu thời gian thực.
* Tăng cường năng lực phân tích thông minh: Ứng dụng Machine Learning và AI trong dự báo tốc độ, phát hiện bất thường, và phân tích nâng cao.
* Bổ sung dữ liệu không gian và phân tích địa lý (Geo-Spatial Analytics): Thêm trường tọa độ lat, lon để trực quan hóa bản đồ giao thông và truy vấn không gian.
* Cải thiện bảo mật, giám sát và tối ưu hiệu năng hệ thống: Tích hợp X-Pack Security, Monitoring, và thực hiện benchmark cấu hình shard – replica – heap.
* Mở rộng ứng dụng sang các lĩnh vực khác: Áp dụng mô hình cho các bài toán phân tích log, IoT và hệ thống đô thị thông minh (Smart City).

Tổng kết lại, các hướng phát triển nêu trên tập trung vào ba khía cạnh chính:

(1) Mở rộng quy mô và tự động hóa pipeline dữ liệu,

(2) Nâng cấp năng lực phân tích theo hướng dự báo thông minh

(3) Tăng cường tính bảo mật, trực quan và ứng dụng liên ngành.

Việc tiếp tục triển khai và hoàn thiện hệ thống theo các hướng này sẽ giúp nâng cao hiệu quả của mô hình, đưa Elasticsearch trở thành nền tảng trọng tâm trong các giải pháp phân tích dữ liệu lớn và giao thông thông minh thời gian thực.

# **TÀI LIỆU THAM KHẢO**

[1] S. Banon, “Elasticsearch: Distributed, RESTful Search and Analytics Engine,” *Elastic Official Documentation*, Elastic NV, 2025.

[2] Elastic NV, “Kibana User Guide,” *Elastic Stack Documentation*, 2025.

[3] Kaggle Inc., “NYC Real-Time Traffic Speed Data,” *Kaggle Datasets*, 2022.