****

**TÌM HIỂU VÀ CÀI ĐẶT ELK ELASTICSEARCH LOGSTASH KIBANA**

**MỤC LỤC**

[I. GIỚI THIỆU VỀ ELK: 2](#_Toc168392412)

[**1. Tổng quan về ELK:** 2](#_Toc168392413)

[**2. Các khái niệm cơ bản:** 3](#_Toc168392414)

[**3. Tại sao nên sử dụng ELK:** 4](#_Toc168392415)

[**4. Luồng hoạt động:** 5](#_Toc168392416)

[II. HƯỚNG DẪN CÀI ĐẶT ELK: 5](#_Toc168392417)

[**1. Các bước chuẩn bị:** 5](#_Toc168392418)

[**2. Chi tiết các bước thực hiện:** 6](#_Toc168392419)

[2.1. Tạo file Docker-Compose và các file cấu hình liên quan: 6](#_Toc168392420)

[2.2. Nhập lệnh Docker-Compose up để cài đặt và khởi chạy 3 container tương ứng với 3 ứng dụng: **Elasticsearch – Kibana – Logstash**: 9](#_Toc168392421)

[2.3. Cấu hình và chạy ứng dụng Java Spring: 11](#_Toc168392422)

[2.4. Vào Kibana, mở giao diện Index Management và kiểm tra dữ liệu đã được Logstash đẩy lên Elasticsearch hay chưa. 13](#_Toc168392423)

[III. GIỚI THIỆU CÁC CHỨC NĂNG CƠ BẢN: 16](#_Toc168392424)

[**1. Hiển thị dữ liệu trên Kibana:** 16](#_Toc168392425)

[1.1. Lọc dữ liệu theo thời gian: 16](#_Toc168392426)

[1.2. Chọn ra các trường cụ thể để trực quan hoá việc quan sát, theo dõi: 17](#_Toc168392427)

[1.3. Lọc dữ liệu theo trường, thuộc tính cụ thể: 17](#_Toc168392428)

[1.4. Tạo biểu đồ thống kê log trong hệ thống: 18](#_Toc168392429)

[**2. Truy vấn Elastichsearch cơ bản:** 21](#_Toc168392430)

[2.1. Kiểm tra toàn bộ các index: 21](#_Toc168392431)

[2.2. Tìm tất cả các Document trong Index: 21](#_Toc168392432)

[2.3. Thêm tham số size để lấy số kết quả trả về mặc định: 22](#_Toc168392433)

[2.4. Sắp xếp kết quả trả về: 22](#_Toc168392434)

[2.5. Tuỳ chọn trường dữ liệu trả về: 23](#_Toc168392435)

[2.6. Tìm kiếm dữ liệu phù hợp: 23](#_Toc168392436)

[2.7. Sử dụng điều kiện logic bool trong truy vấn tìm kiếm: 23](#_Toc168392437)

[**3. Cấu hình Index Policy Management:** 23](#_Toc168392438)

[3.1. Giải thích ý nghĩa: 23](#_Toc168392439)

[3.2. Cách thực hiện: 25](#_Toc168392440)

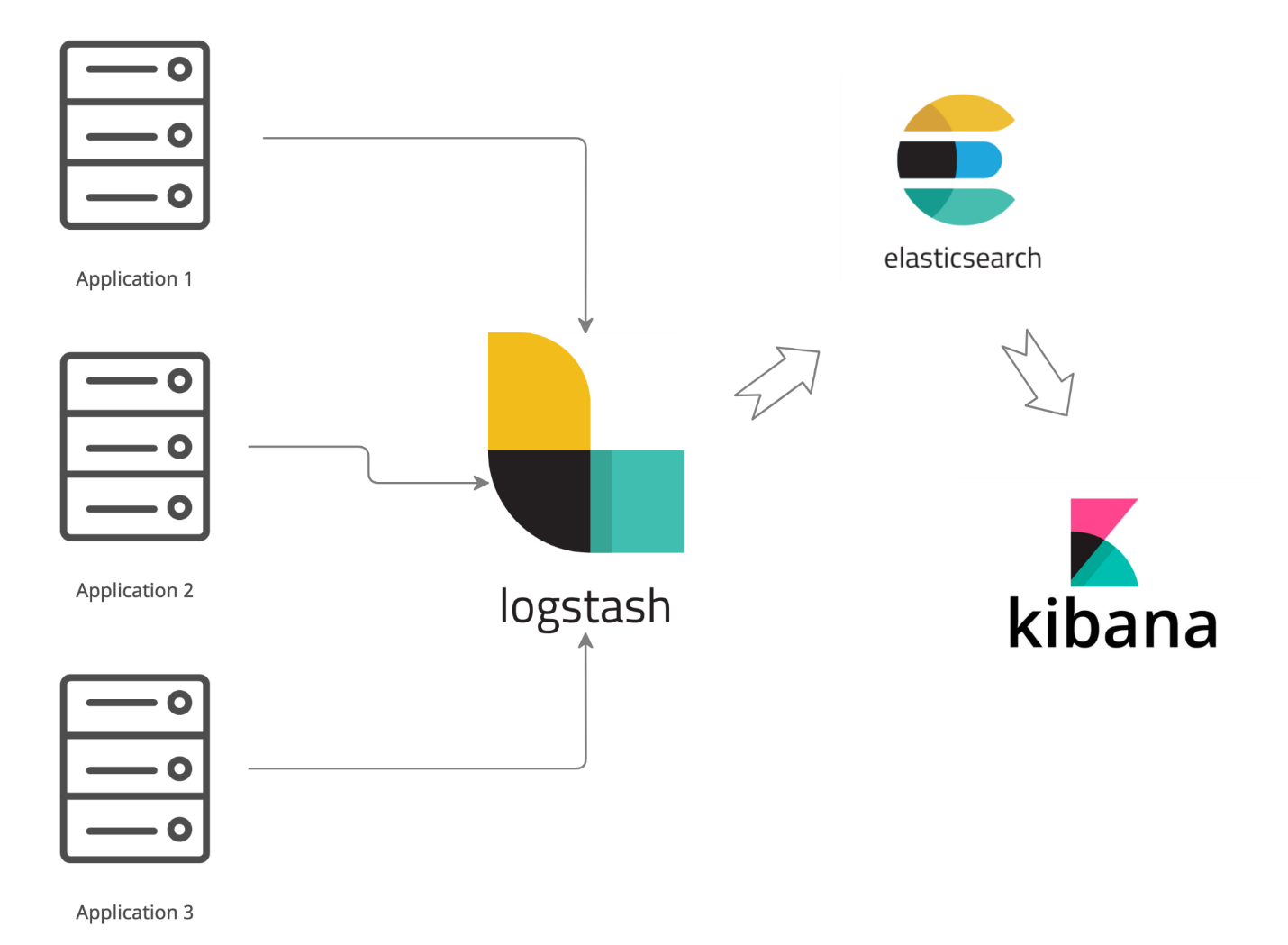
# I. GIỚI THIỆU VỀ ELK:

## **1. Tổng quan về ELK:**

**- Elasticsearch** (trang chủ [Elasticsearch](https://www.elastic.co/)) phát hành năm 2010, là phần mềm miễn phí, nguồn mở viết bằng ngôn ngữ Java, nó có chức năng như là công cụ tìm kiếm và phân tích phân tán (máy chủ ở nhiều nơi). Elasticsearch được xây dựng dựa trên [Apache Lucene](https://lucene.apache.org/), Elasticsearch trở thành công cụ tìm kiếm (search engine) thông dụng nhất và được sử dụng rộng rãi cho các công việc liên quan đến chỉ mục và tìm kiếm tài liệu, phân tích dữ liệu ...

- ELK là viết tắt của tập hợp 3 phần mềm cốt lõi đi kèm với nhau, phục vụ cho công việc giám sát hệ thống. Ba phần mềm này lần lượt là **Elasticsearch**, **Logstash** và **Kibana**.

* **Elaticsearch**: là một search engine được rất nhiều anh em sử dụng, trong bộ ba này thì nó đóng vai trò là một store để chứa logs kiêm vai trò tìm kiếm và phân tích mạnh mẽ vốn có.
* **Logstash**: đây là một công cụ sử dụng để thu thập, xử lý log được viết bằng Java. Nhiệm vụ chính của logstash là thu thập log sau đó chuyển vào Elastichsearch. Mỗi dòng log của logstash được lưu trữ đưới dạng json.
* **Kibana**: là công cụ cho phép trực quan hoá dữ liệu từ Elasticsearch, ở đây chính là đống logs của chúng ta.



- Với các ưu điểm như: khả năng phân tích, tìm kiếm dữ liệu nhanh chóng, hỗ trợ tìm kiếm mờ, dễ dàng tạo ra các truy vấn phức tạp, cung cấp sẵn thư viện client cho nhiều ngôn ngữ (Elasticsearch Clients) như Java, Php ... Elasticsearch được sử dụng bởi nhiều công ty lớn như Facebook, GitHub, Wikimedia ...

## **2. Các khái niệm cơ bản:**

**- Cluster:** Là các server (nodes) - Elasticsearch kết nối với nhau để lưu giữ dữ liệu và cung cấp chức năng đánh chỉ mục, tìm kiếm dữ liệu trên các server đó. Một cluster xác định bởi tên duy nhất *(mặc định tên là elasticsearch)*. Bạn cũng lưu ý, dù bạn chỉ tạo ra hệ thống với 1 server (node) thì vẫn có một cluster, sau này có thể nối nhiều server phân tán vào cluster để mở rộng khả năng của hệ thống mà ở đó một node (server) có thể có một chức năng riêng như (master node, data, client...).

**- Node:** Là một server tham ra tạo thành cluster, nó có vai trò lưu dữ liệu, đánh chỉ mục và cung cấp khả năng tìm kiếm. Một node cũng cần xác định một cái tên, mặc định một tên duy nhất (UUID) được sinh ra cho nó khi nó chạy trừ khi bạn gán cho nó một cái tên cụ thể. Mặc định mỗi node cần thiết lập nối vào một cluster đã đặt tên (mặc định là elasticsearch), có nghĩa là khi node hoạt động nó sẽ khám phá mạng tìm đến cluster với tên chỉ định để nối vào. Nếu như bạn chạy một server (node) mà mạng của bạn chưa có một cluster nào với tên chỉ ra, thì node này sẽ tự động là một cluster với tên mặc định elasticsearch

**- Index:** Là một tập hợp các tài liệu (document), những tài liệu này có một số tên thuộc tính tính chất chung. Thường mỗi index là một loại dữ liệu nào đó của bạn ví dụ như index chữa các sản phẩm, index chứa các đơn hàng, index chứa các bài viết ... Mỗi index được đặt một cái tên (nhớ là chữ thường), tên này dùng để thi hành các chức năng như đánh chỉ mục, tìm kiếm, cập nhật ... cho các dữ liệu (document) trong nó. Mỗi node có thể tạo nhiều index

**- Document:** Là đơn vị thông tin lưu trong index (giống như dòng lưu trong bảng), đơn vị thông tin này biểu diễn bằng JSON (JavaScript Object Notation), qua đó nó lưu loại dữ liệu của bạn như sản phẩm, bài viết, đơn hàng ... Một index lưu bao nhiêu Document cũng được.

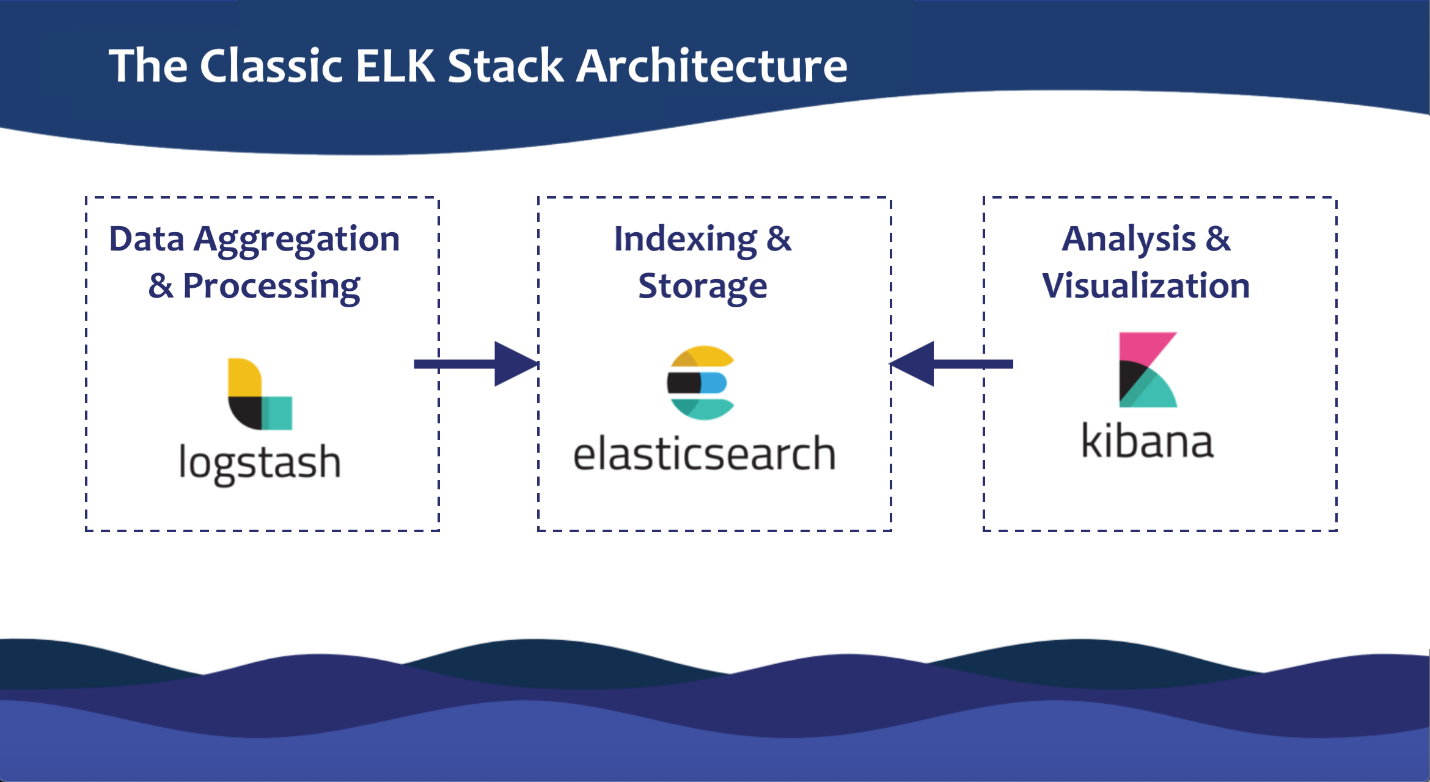
**- Shard:** Để giải quyết các vấn đề hiệu năng khi lưu trữ dữ liệu lớn bị giới hạn bởi phần cứng ES cung cấp khả năng chia một index ra thành từng mảnh nhỏ hơn - mỗi mảnh nhỏ đó gọi là shard. Khi tạo ra index, bạn có thể chia nó ra thành bao nhiêu shard tùy bạn. Một shard đó vẫn có đầy đủ chức năng như index nhưng độc lập với index, vào có thể lưu ở các node khác nhau. Shard nó giúp giải quết vấn đề hiệu năng tốc độ, tìm kiếm song song trên nhiều node phân tán. Ngoài ra bạn cũng có khái niệm replica đó là một bản backup, copy của shard để ES có thể phục hồi nếu một shard nào đó bị chết.

## **3. Tại sao nên sử dụng ELK:**

Vì đâu mà mà ELK Stack lại vô cùng mạnh mẽ, tiện dùng, được nhiều công ty sử dụng như vậy? Vì các lý do sau:

* **Đọc log từ nhiều nguồn**: Logstash có thể đọc được log từ rất nhiều nguồn, từ log file cho đến log database cho đến UDP hay REST request.
* **Dễ tích hợp**: Dù bạn có dùng Nginx hay Apache, dùng [MSSQL, MongoDB](https://toidicodedao.com/2015/09/24/nosql-co-gi-hay-ho-tong-quan-ve-nosql-phan-1/) hay Redis, Logstash đều có thể đọc hiểu và xử lý log của bạn nên việc tích hợp rất dễ dàng.
* **Hoàn toàn free**: Chỉ cần tải về, setup và dùng, không tốn một đồng nào cả. Công ty tạo ra ELK Stack kiếm tiền bằng các dịch vụ cloud hoặc các sản phẩm premium phụ thêm.
* **Khả năng scale tốt**: Logstash và Elasticsearch chạy trên nhiều node nên hệ thống ELK [cực kì dễ scale](https://toidicodedao.com/2017/11/28/review-sach-release-it-xay-dung-va-thiet-ke-he-thong-trau-bo-cho-hang-trieu-nguoi-dung/). Khi có thêm service, thêm người dùng, muốn log nhiều hơn, bạn chỉ việc thêm node cho Logstash và Elasticsearch là xong.
* **Search và filter mạnh mẽ**: Elasticsearch cho phép lưu trữ thông tin kiểu NoSQL, hỗ trợ luôn Full-Text Search nên việc query rất dễ dàng và mạnh mẽ.
* **Cộng đồng mạnh, tutorial nhiều**: Nhiều công ty dùng và có nhiều tutorial để học hỏi, cộng đồng hỗ trợ nhiệt tình.

## **4. Luồng hoạt động:**



- Đầu tiên, log sẽ được đưa đến Logstash. (Thông qua nhiều con đường, ví dụ như server gửi UDP request chứa log tới URL của Logstash, hoặc Beat đọc file log và gửi lên Logstash)

- Tiếp theo, Logstash sẽ đọc những log này, thêm những thông tin như thời gian, IP, parse dữ liệu từ log (server nào, độ nghiêm trọng, nội dung log) ra, sau đó ghi xuống database là Elasticsearch.

- Sau đó, khi muốn xem log, người dùng vào URL của Kibana. Kibana sẽ đọc thông tin log trong Elasticsearch, hiển thị lên giao diện cho người dùng query và xử lý.

# II. HƯỚNG DẪN CÀI ĐẶT ELK:

## **1. Các bước chuẩn bị:**

- Cài đặt OpenJDK, Docker, cụ thể link cài đặt bên dưới đây:

+ Java SE Development Kit: <https://www.oracle.com/java/technologies/downloads/>

+ Docker:

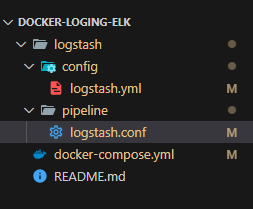
<https://www.docker.com/products/docker-desktop/>

- Ứng dụng Java Spring Boot.

## **2. Chi tiết các bước thực hiện:**

### 2.1. Tạo file Docker-Compose và các file cấu hình liên quan:

- Tạo một folder chứa các file cấu hình như dưới đây:



2.1.1. Chi tiết file docker-compose.yml như sau:

version: "3"

services:

    elasticsearch:

      image: docker.elastic.co/elasticsearch/elasticsearch:8.3.3

      container\_name: elasticsearch\_springboot

      environment:

          - node.name=node-01

          - cluster.name=es-docker-cluster-01

          - bootstrap.memory\_lock=true

          - "ES\_JAVA\_OPTS=-Xms512m -Xmx512m"

          - "discovery.type=single-node"

          - xpack.security.enabled=false

      ports:

          - "9200:9200"

      volumes:

          - elasticsearch\_data:/usr/share/elasticsearch/data

      networks:

          - elastic

    kibana:

      image: docker.elastic.co/kibana/kibana:8.3.3

      container\_name: kibana\_springboot

      ports:

          - "5601:5601"

      environment:

          ELASTICSEARCH\_URL: http://elasticsearch:9200

          ELASTICSEARCH\_HOSTS: '["http://elasticsearch:9200"]'

      depends\_on:

          - elasticsearch

      networks:

          - elastic

    logstash:

      image: docker.elastic.co/logstash/logstash:8.3.3

      container\_name: logstash\_springboot

      volumes:

        - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml:ro

        - ./logstash/pipeline:/usr/share/logstash/pipeline:ro

      ports:

        - "5044:5044"

        - "5000:5000/tcp"

        - "5000:5000/udp"

        - "9600:9600"

      environment:

        LS\_JAVA\_OPTS: "-Xmx1024m -Xms1024m"

      networks:

        - elastic

      depends\_on:

        - elasticsearch

networks:

  elastic:

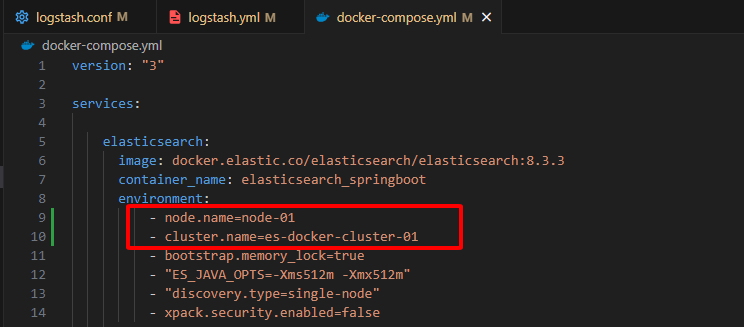
    driver: bridge

volumes:

    elasticsearch\_data:

        driver: local

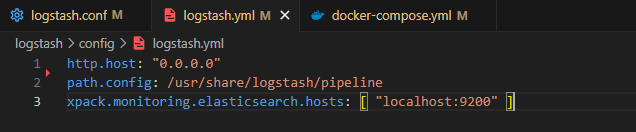
\* Chú thích:



- Node.name: Tên node chúng ta đặt là **node-01**

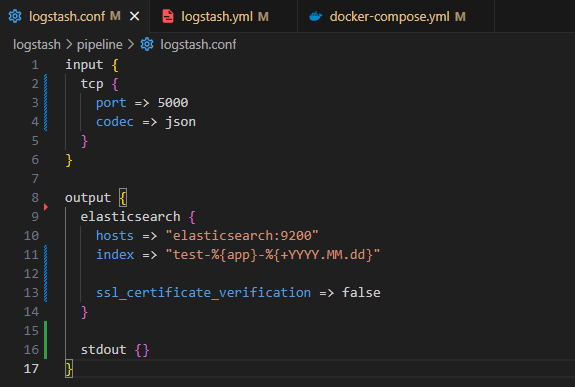
- Cluster.name: Tên cluster chúng ta đặt là **es-docker-cluster-01**

2.2.2. Chi tiết file cấu hình Logstash.yml như sau:



- Đây là file cấu hình chính cho Logstash, chúng ta sẽ cấu hình các thiết lập toàn cục như cổng lắng nghe, đường dẫn lưu trữ, cài đặt plugin, và các cấu hình khác mà ảnh hưởng đến toàn bộ hoạt động của Logstash.

2.2.2. Chi tiết file cấu hình Logstash.conf như sau:



- Đây là file cấu hình cho các luồng dữ liệu cụ thể (pipeline) trong Logstash. Trong file này, chúng ta sẽ định nghĩa các input, filter và output để xác định cách Logstash sẽ xử lý và chuyển đổi dữ liệu từ nguồn đến đích. Mỗi file logstash.conf thường đại diện cho một pipeline cụ thể.

- Giải thích cụ thể:

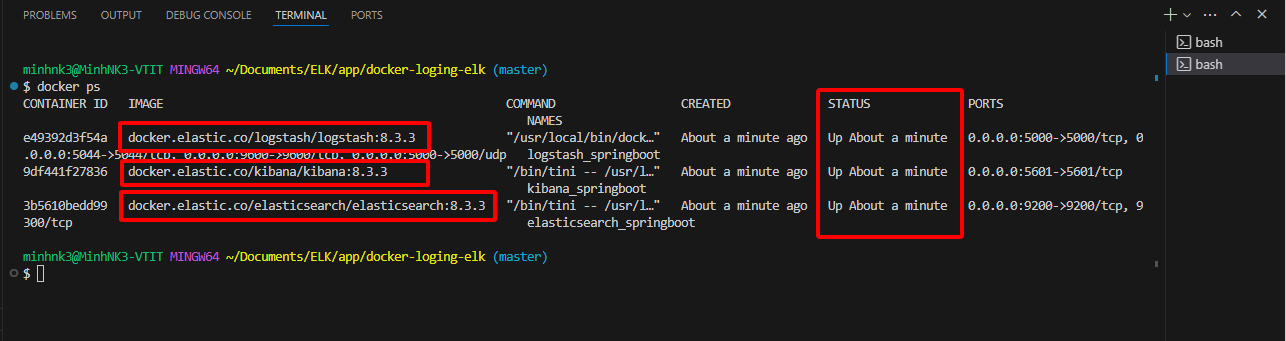
+ Input: Đầu vào là logstash ở cổng localhost:5000 với nhiệm vụ đẩy log lên Elastiseach.

+ Output: Đầu ra là Elasticsearch đang ở cổng 9200. Khi đã đẩy dữ liệu lên thành công thì một Index theo format như trong hình sẽ được tạo. Với {app} là tên một ứng dụng Web cụ thể, stdout cho phép chúng ta xem được log ở màn hình console.

### 2.2. Nhập lệnh Docker-Compose up để cài đặt và khởi chạy 3 container tương ứng với 3 ứng dụng: **Elasticsearch – Kibana – Logstash**:

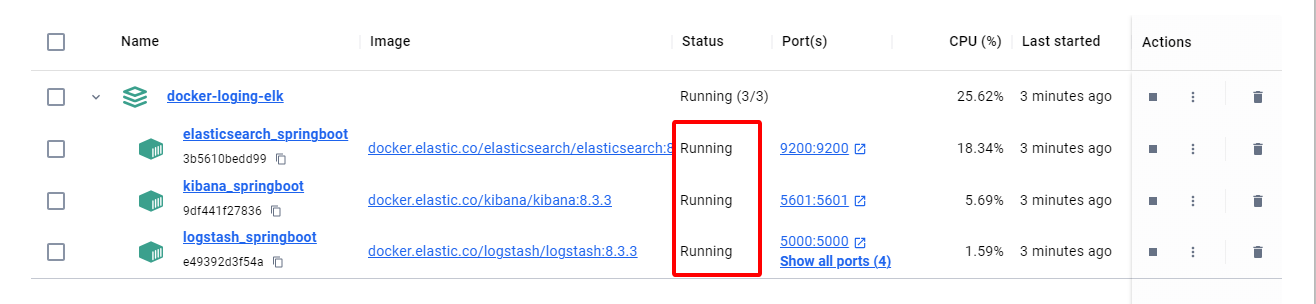
2.2.1. Sau khi nhập lênh Docker-compose up xong, đợi một lúc cho việc cài đặt hoàn tất. Và dùng lệnh sau để kiểm tra xem 3 container đã được cài đặt và khởi chạy thành công hay chưa.

*Docker ps*

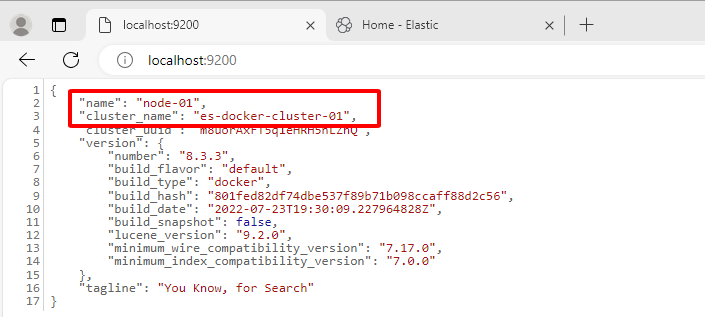


- Nếu STATUS hiển thị là Up thì tức là 3 container tương ứng với Elasticsearch – Kibana – Logstash đã khởi chạy lên thành công.

- Hoặc, chúng ta có thể kiểm tra ở Docker Desktop với giao diện trực quan hơn, Status hiển thị là Running tức là cả 3 container cũng đã chạy lên thành công.

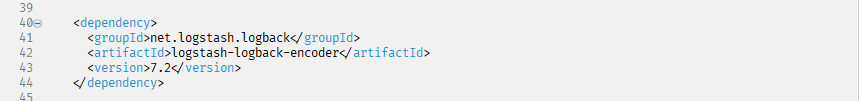


2.2.2. Truy cập địa chỉ localhost:9200, chúng ta có thể thấy các thông tin bao gồm: Tên node, tên cluster đã được tạo thành công.



### 2.3. Cấu hình và chạy ứng dụng Java Spring:

2.3.1 Tạo một project Java Spring, trong file pom.xml chúng ta thêm Dependency Logstash như dưới đây:



- Tạo một file logback-spring.xml và khai báo như bên dưới:



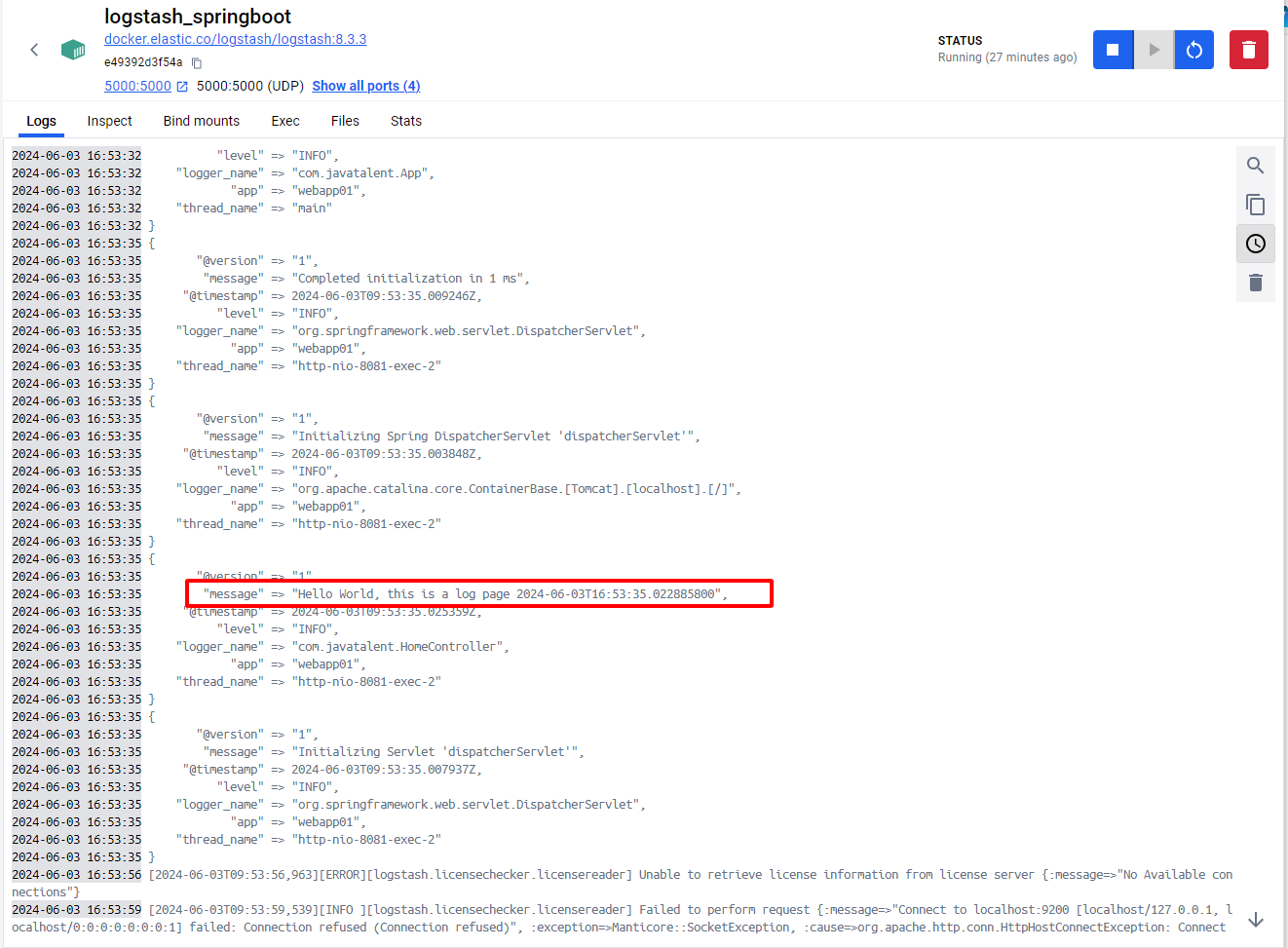
- Trong đó, chúng ta tạo một pattern là “app”: “webapp01” để khi trên elasticsearch sẽ tạo một Index có tên đính kèm bao gồm {app}. Mục đích là để phân biệt với các Index được tạo từ các ứng dụng khác.

2.3.2. Tạo một Rest Controller, trong đó có khai báo một số API bất kỳ và thêm một đoạn Log làm ví dụ, cụ thể như sau:



2.3.3. Gọi một API bất kỳ và kiểm tra log ở Logstash:

- Vào trong Docker Desktop, ở mục Logs chúng ta thấy Logstash đã lấy Log thành công.



- Không chỉ Log message mà chúng ta thấy được các properties khác như “app” (Đã khai báo ở bước 2.3.1), @timestamp (Thời gian ghi lại log) hay thread\_name (tên luồng) …

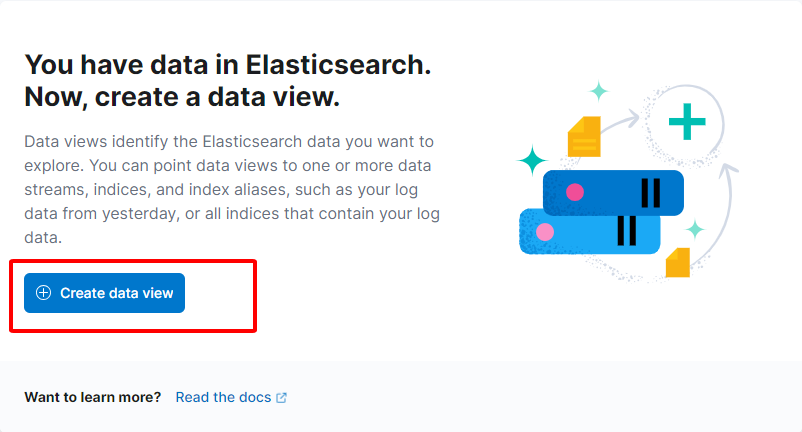
### 2.4. Vào Kibana, mở giao diện Index Management và kiểm tra dữ liệu đã được Logstash đẩy lên Elasticsearch hay chưa.

2.4.1. Truy cập địa chỉ localhost:5601 để mở Dashboard của Kibana.

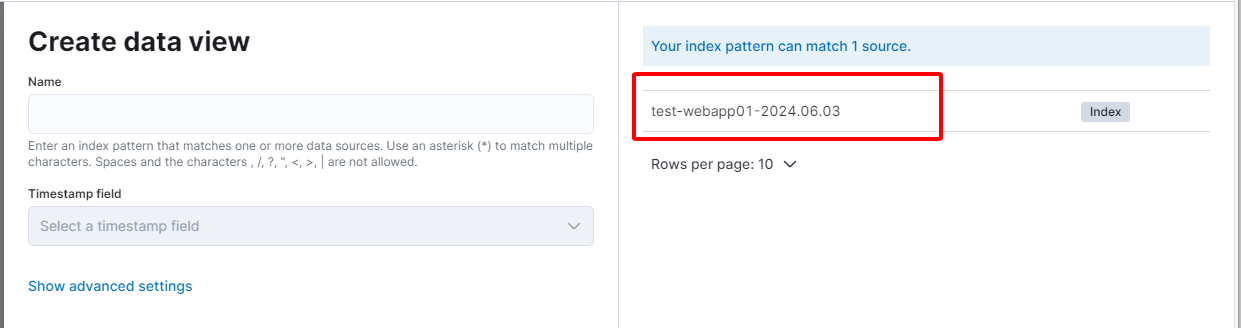
- Chọn thanh Tab, ở mục Analytics chọn Discover.



2.4.2. Chọn “Create data view”

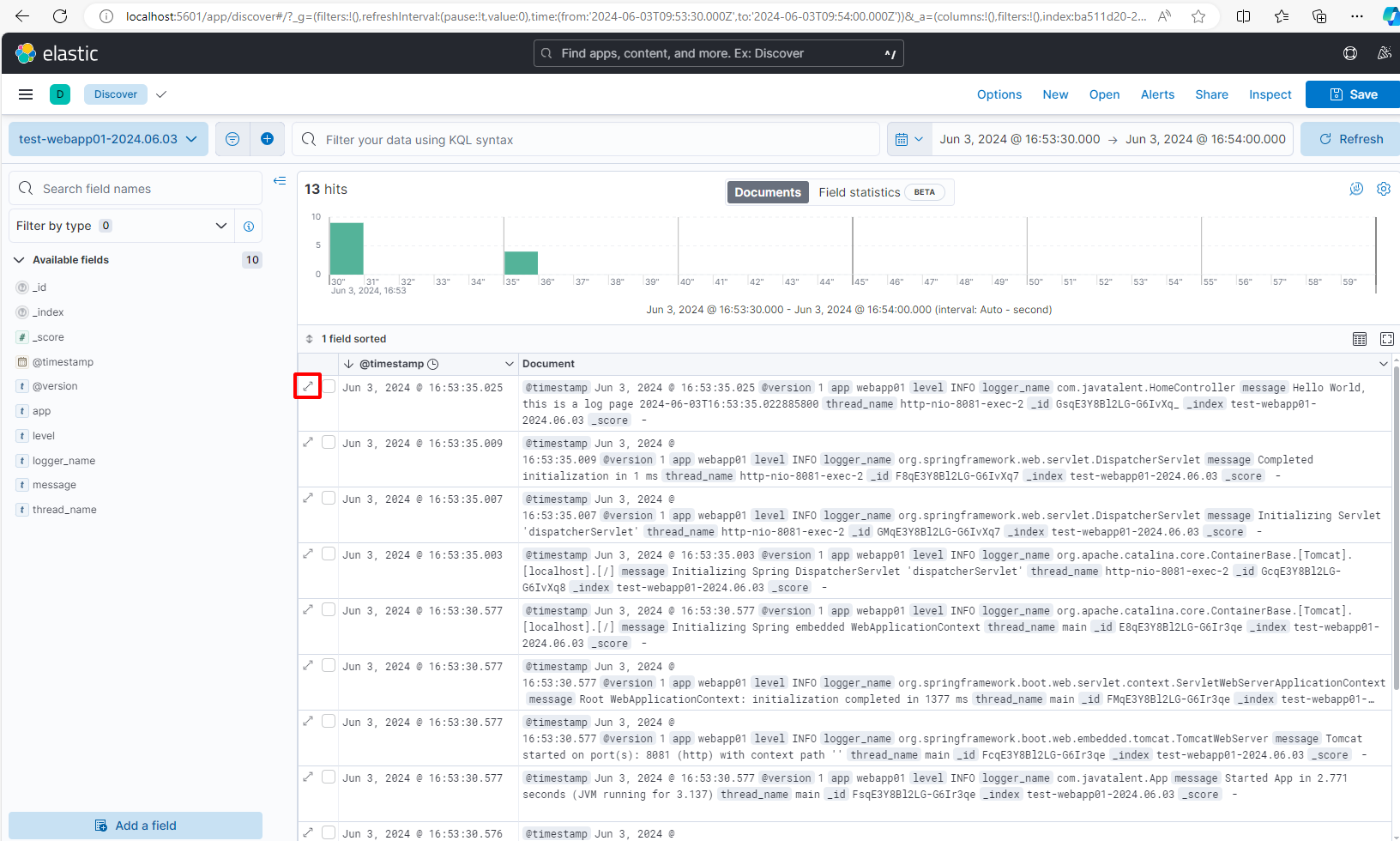


2.4.3. Chọn index như trong hình, như ta thấy index được tạo có tên bao gồm webapp01 nên ta có thể dễ dàng phân biệt được Index nào của ứng dụng nào trong trường hợp có rất nhiều ứng dụng chạy đồng thời.

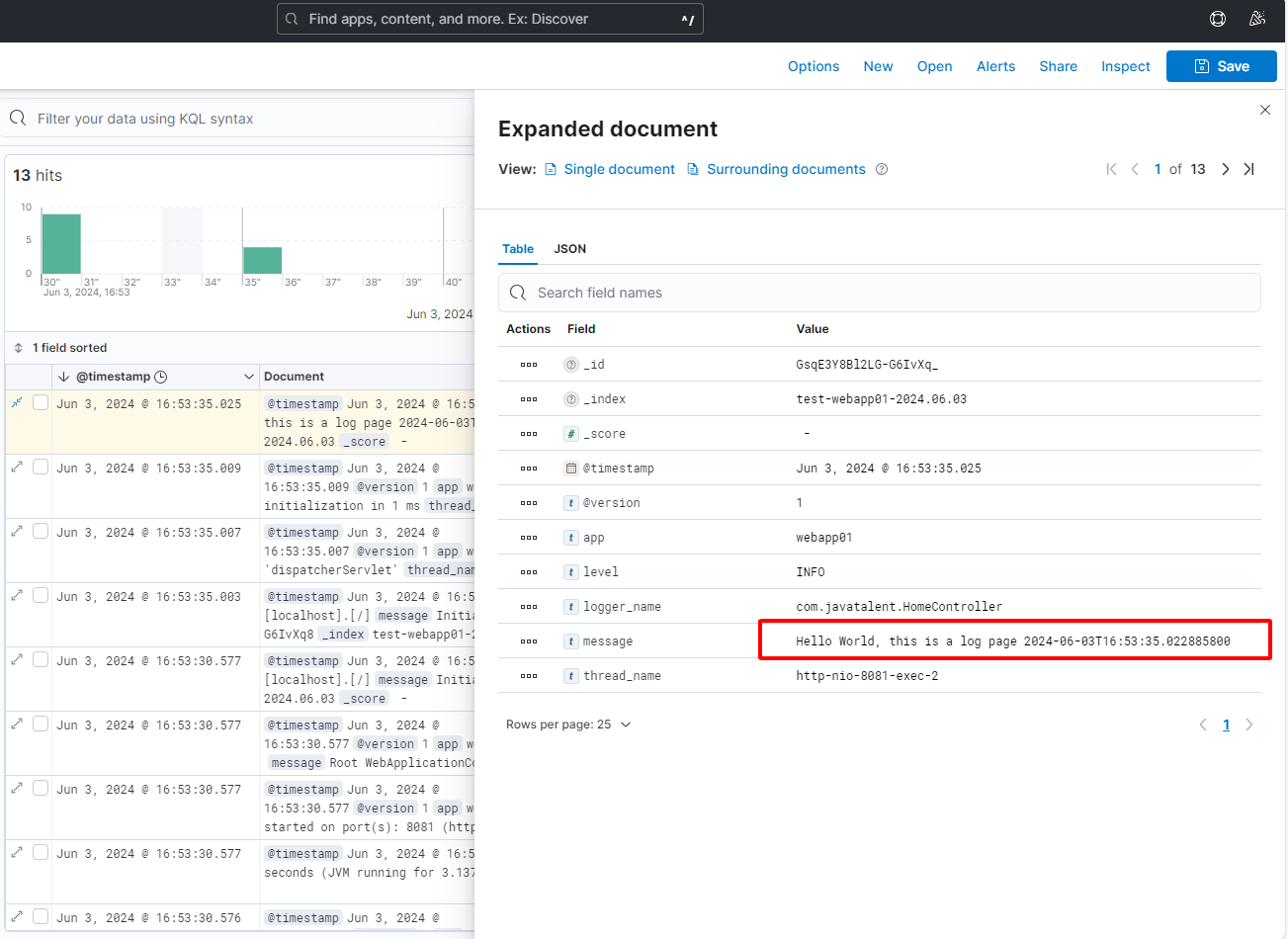


- Chọn Create data view.

2.4.4. Đoạn log đã được Logstash lưu lên Elasticsearch thành công:



- Chọn nút mũi tên mở rộng (khoanh đỏ) để xem chi tiết đoạn log đó.

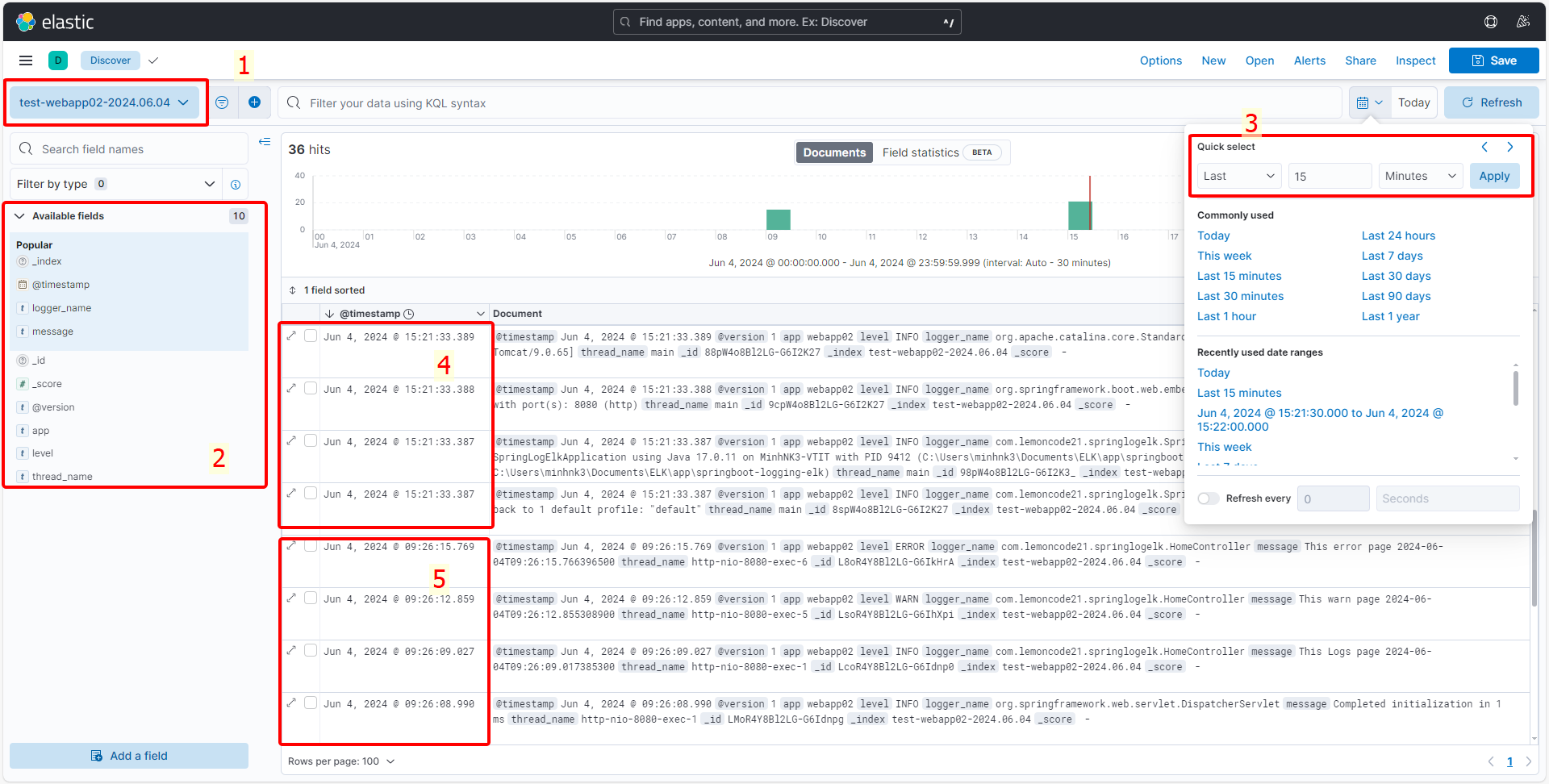


- Trong hình, đoạn message khoanh đỏ là đoạn log lấy được từ ứng dụng Java Web.

# III. GIỚI THIỆU CÁC CHỨC NĂNG CƠ BẢN:

## **1. Hiển thị dữ liệu trên Kibana:**

### 1.1. Lọc dữ liệu theo thời gian:



- Chú thích:

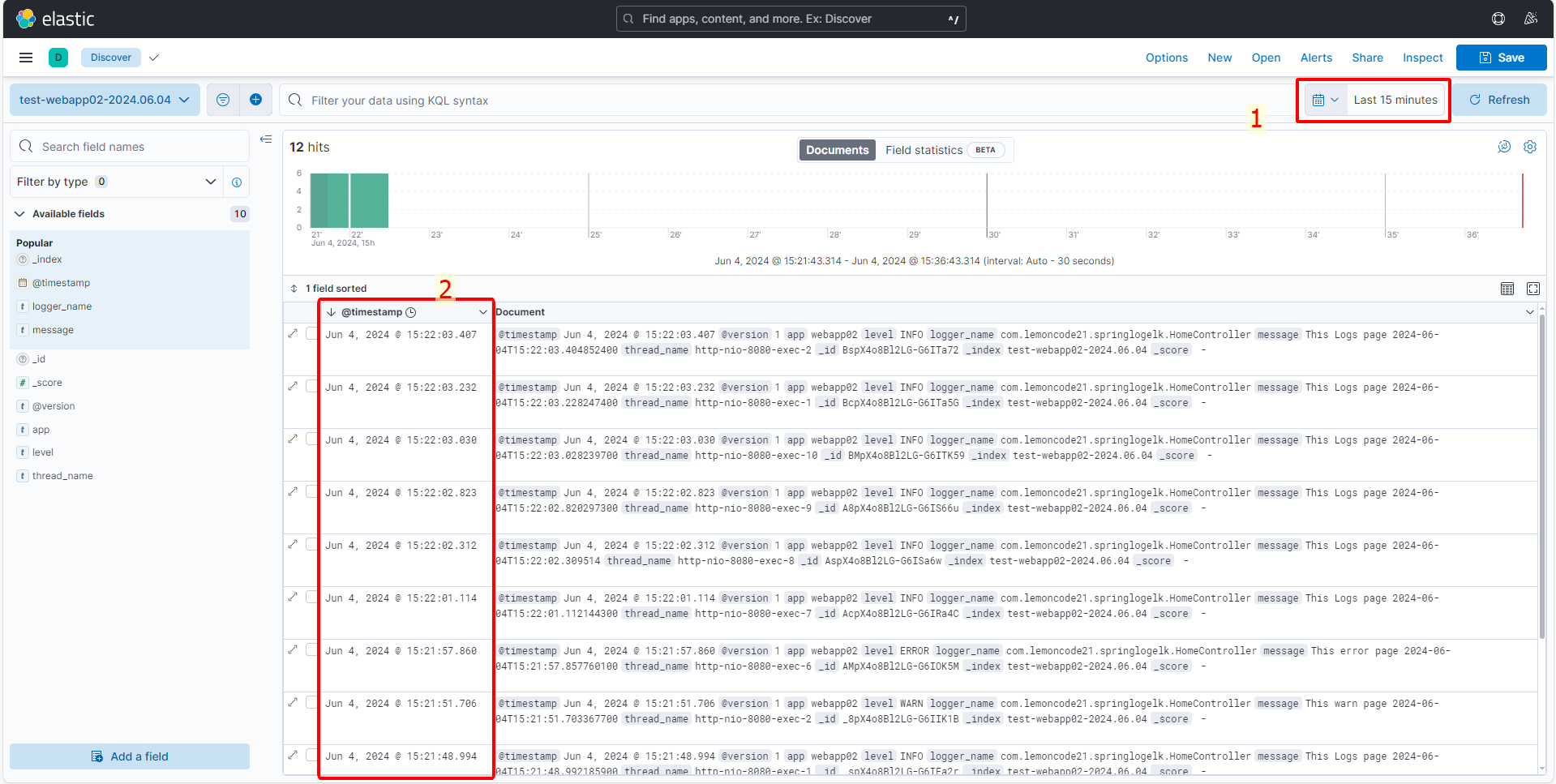
+ Ô thứ (1) cho ta biết tên của Index Pattern.

+ Ô thứ (2) hiển thị các trường, thuộc tính đang hiển thị dữ liệu ra ngoài Dashboard.

+ Ô thứ (3) là phần lọc dữ liệu theo thời gian. Ở ví dụ này ta sẽ hiển thị dữ liệu được tạo từ 15 phút trước.

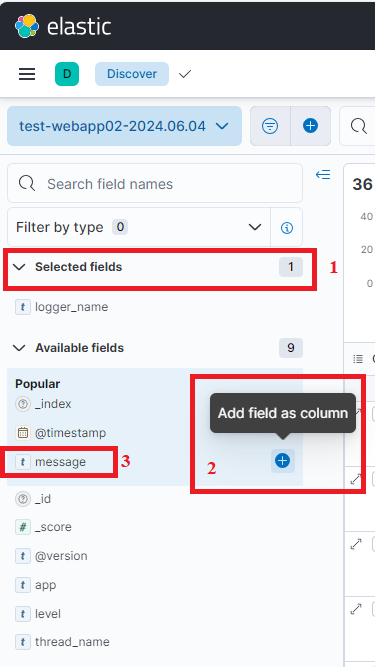
+ Ô thứ (4) và ô thứ (5) đang hiển thị dữ liệu đã tạo từ 9h26’ sáng cho đến 15h21’ chiều

- Ở ô thứ (3), khi chọn Apply để lọc dữ liệu, kết quả được trả về như bên dưới.



- Dữ liệu trả về ra Dashboard có phần @timestamp ở ô thứ (2) đều có thời gian tạo từ 15h21’. Tức là trong khoảng 15 phút cho đến hiện tại.

### 1.2. Chọn ra các trường cụ thể để trực quan hoá việc quan sát, theo dõi:



- Chú thích:

+ Ở ô (1), đây là phần **Selected fields,** phần này cho ra danh sách các thuộc tính đã được chọn. Ở ví dụ này, field **logger\_name** đã được thêm vào danh sách này.

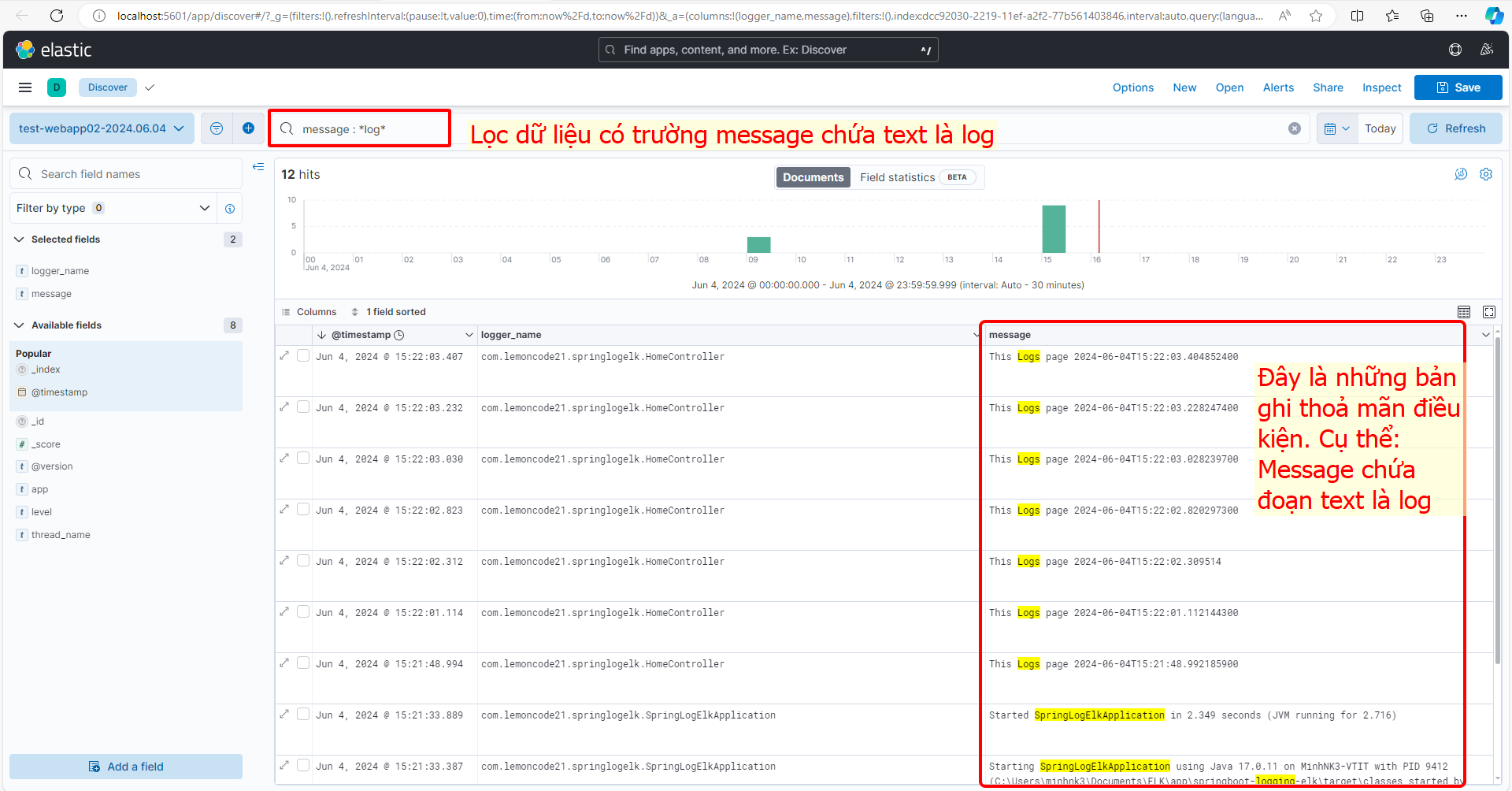
+ Tiếp với field **message** ở ô (3), ở bên phải của field đó có icon dấu cộng và hiện nay tooltip có tên là “Add field as column” (ô thứ 2). Chọn nút đó thì field message sẽ được thêm vào phần **Selected fields** (ô thứ 1)

- Khi đã thêm xong, ở Dashboard sẽ trả về 2 trường là **message** và **logger\_name**. Cộng với **@timestamp** tương ứng.

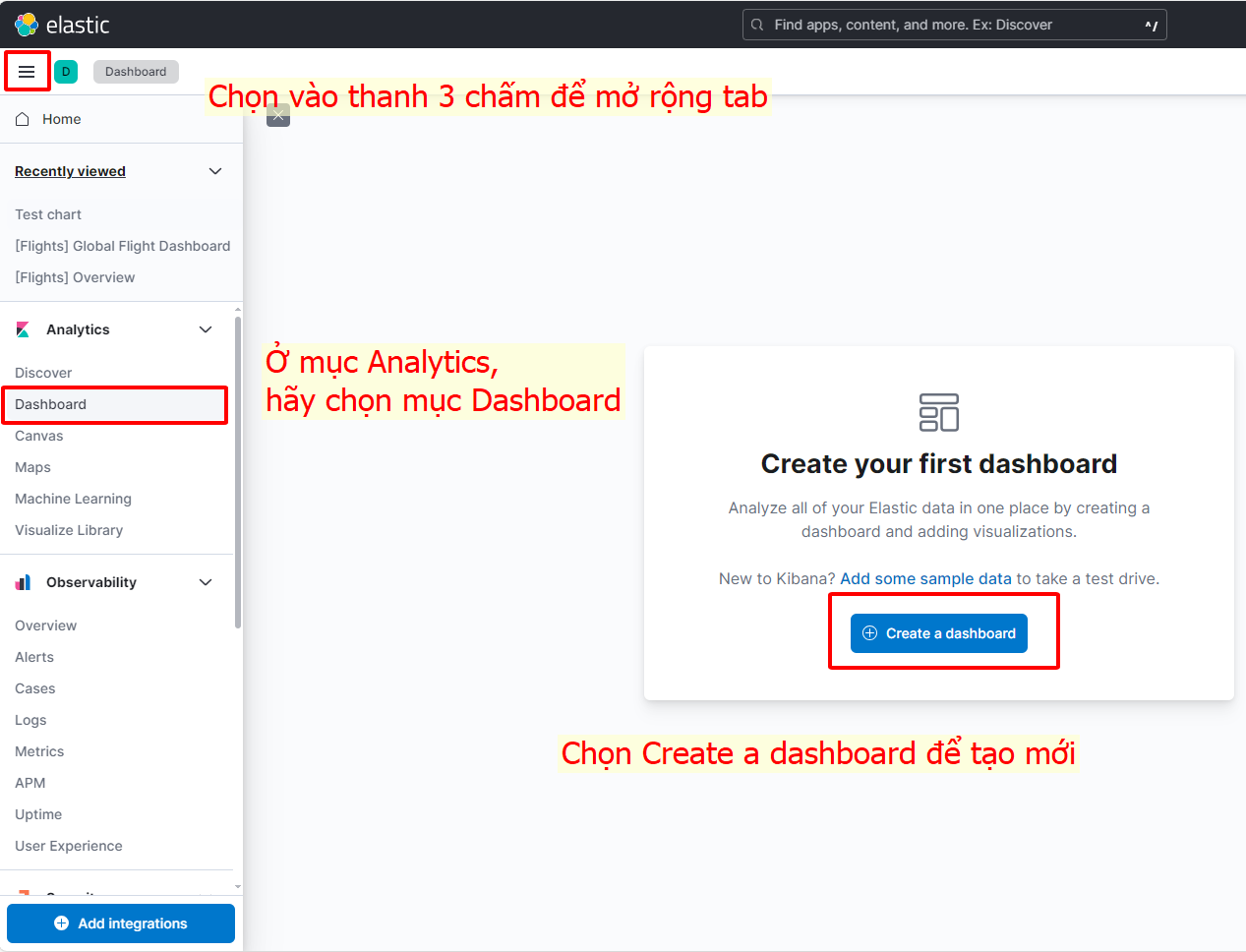


### 1.3. Lọc dữ liệu theo trường, thuộc tính cụ thể:

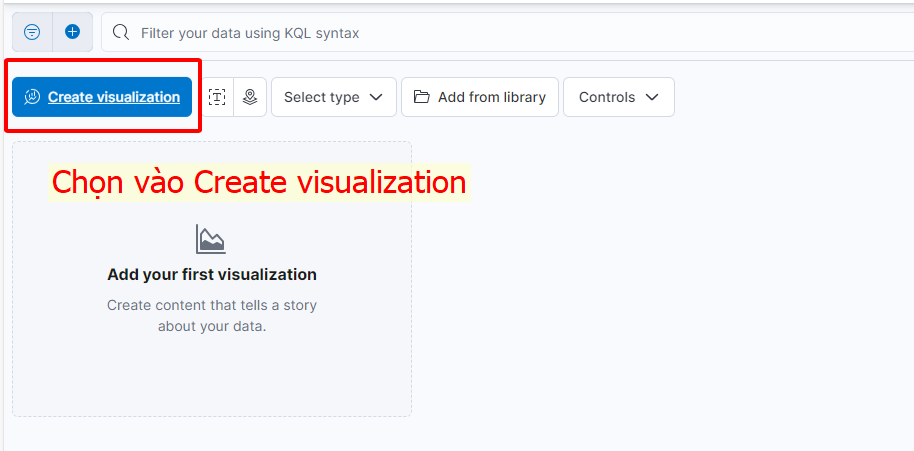
- Ở ví dụ này, ta sẽ viết điều kiện để hiển thị những bản ghi có trường **message** chứa đoạn text là log. Kết quả cho ra ngoài Dashboard như bên dưới:



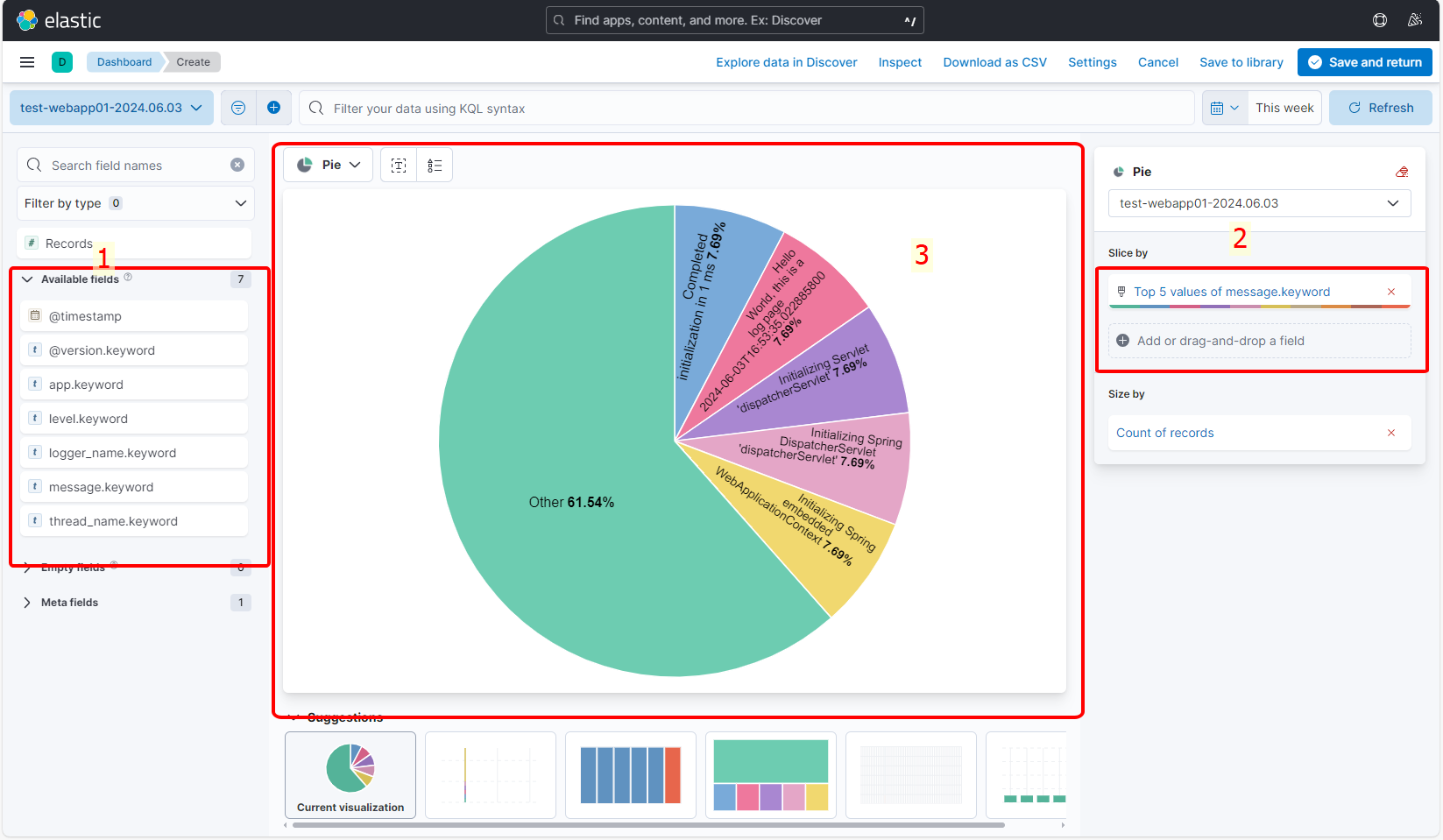
### 1.4. Tạo biểu đồ thống kê log trong hệ thống:



- Ở mục Analytics, hãy chọn Dashboard. Sau đó chọn Create a dashboard.



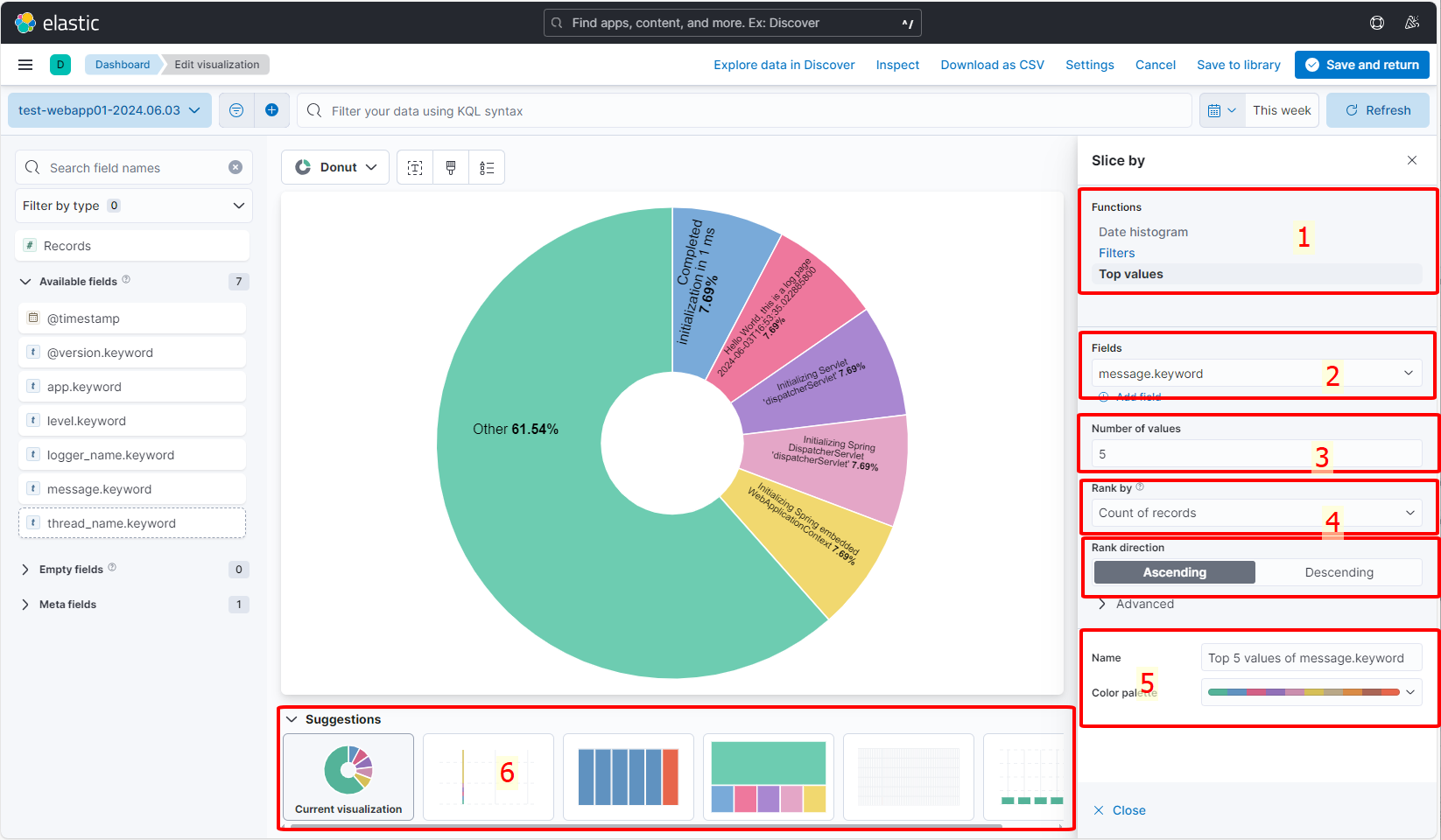
- Sau đó chọn Create visualization.



\* Chú thích:

- Ở mục 1, chúng ta kéo thả một hoặc nhiều trường, thuộc tính vào mục 3 để hệ thống có dữ liệu để lập nên biểu đồ thống kê. Ví dụ ở đây, chúng ta kéo mục message.keyword vào và hệ thống tự động thống kê top 5 message.keyword (mục 2) và hiển thị dạng Pie chart ở mục 3.

- Chọn mục 2 để mở ra các tuỳ chọn mở rộng như dưới đây:



\* Chú thích:

(1): Các hàm xử lý dữ liệu

- **Date histogram**: hiển thị dữ liệu theo thời gian. Biểu đồ này chia dữ liệu thành các khoảng thời gian nhất định.

- **Filters**: Lọc dữ liệu theo truy vấn cụ thể.

- **Top values**: Hiển thị các giá trị phổ biến nhất hoặc có tần suất cao nhất của một trường dữ liệu cụ thể.

(2): Các trường, thuộc tính cần đem ra truy vấn và đo lường, thống kê.

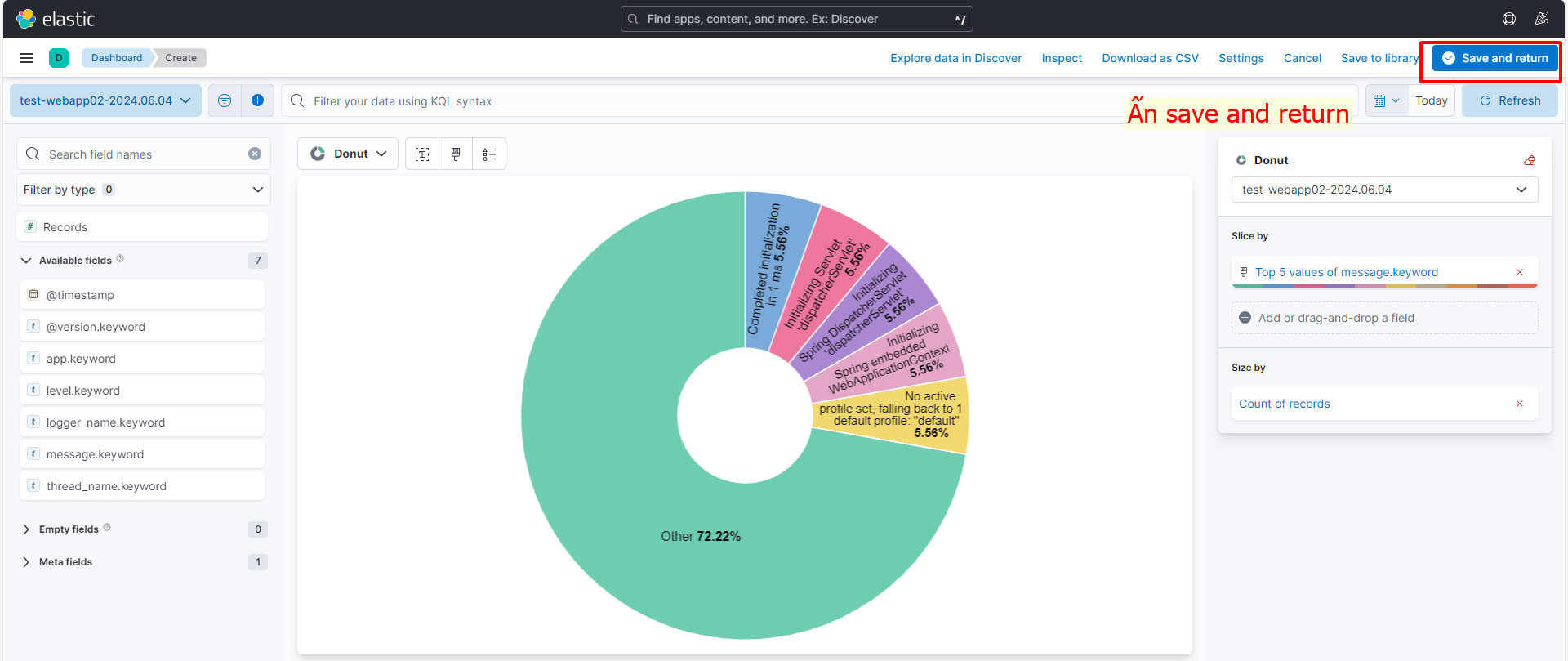
(3): Số giá trị lấy ra trong trường đã lấy ở bước (2)

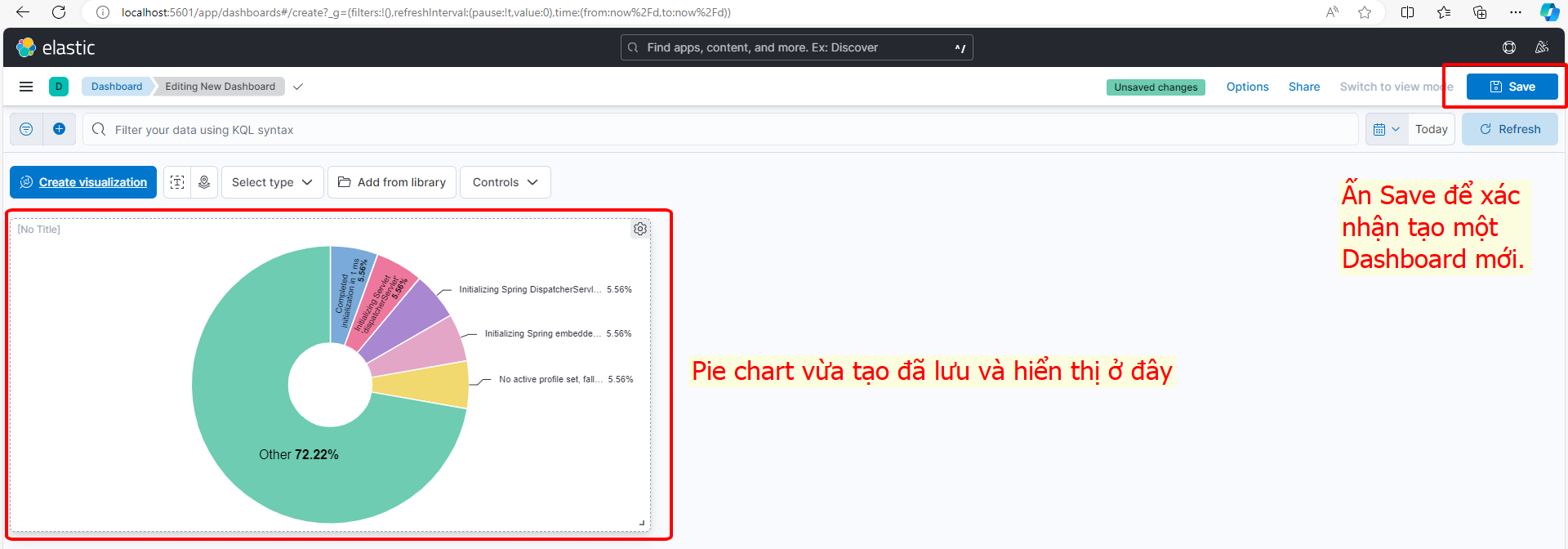
(4): Lọc dữ liệu từ giá trị thấp đến cao, và ngược lại.

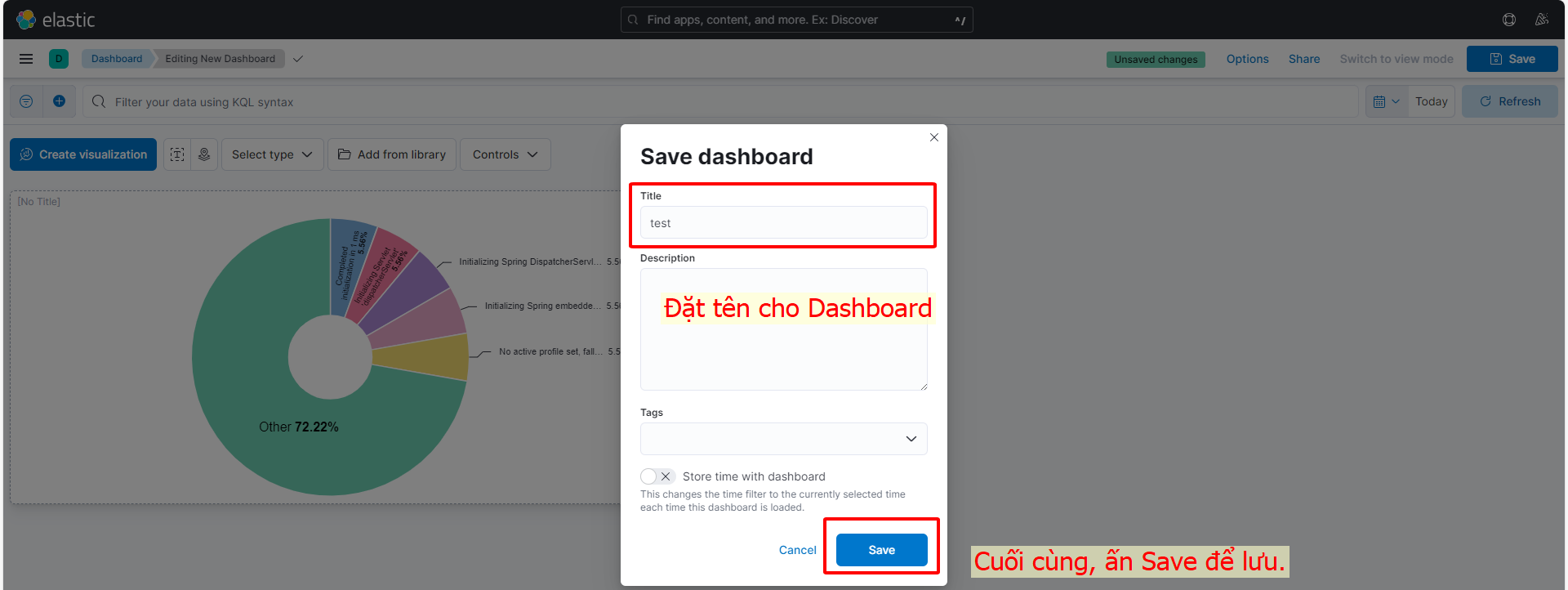
(5): Đặt tên biểu đồ thống kê, chọn màu.

(6): Lựa chọn biểu đồ.

- Sau đó ấn **Save and return** để lưu lại:

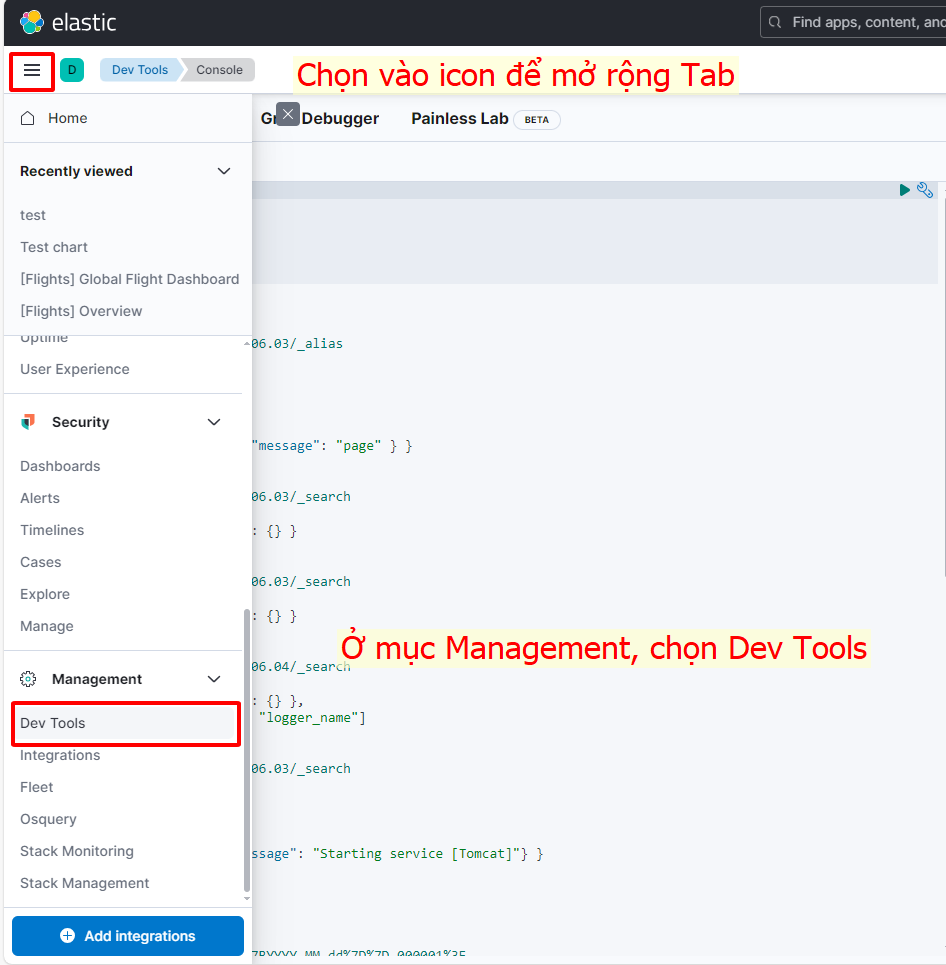


- Ấn Save để tạo một Dashboard mới.

-Đặt tên cho Dashboard mới và Ấn Save để xác nhận.

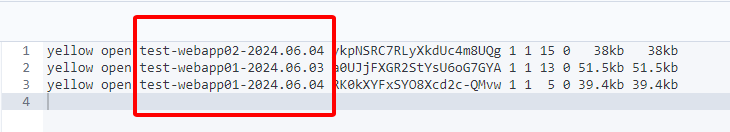
## **2. Truy vấn Elastichsearch cơ bản:**

- Để bắt đầu, hãy vào Tab, ở mục Management chọn Dev Tools.



### 2.1. Kiểm tra toàn bộ các index:

- Cú pháp câu lệnh: GET /\_cat/indices



- Kết quả trả về trong ví dụ này là 3 index bao gồm: 2 index của webapp01 trong 2 ngày từ 03/06 và 04/06 và index còn lại của webapp02.

### 2.2. Tìm tất cả các Document trong Index:

- Cú pháp câu lệnh:

GET /test-webapp01-2024.06.03/\_search

{

"query": {"match\_all": {}}

}



- Tìm tất cả document trong index test-webapp01-2024.06.03. Ở hình trên, các bản ghi trả về đều có trường \_index là test-webapp01-2024.06.03. Như vậy, dữ liệu trả về đúng như mong đợi.

### 2.3. Thêm tham số size để lấy số kết quả trả về mặc định:

- Ta cũng có thể chỉ ra dữ liệu trả về bắt đầu từ phần tử nào với tham số from với cú pháp lệnh như sau:

GET /test-webapp02-2024.06.04/\_search

{

"query": {"match\_all": {}},

"from": 1,

"size": 10

}

- Trong ví dụ này, ta sẽ lấy document trong index test-webapp02-2024.06.04 từ bản ghi đầu tiên đến bản ghi thứ 10.

### 2.4. Sắp xếp kết quả trả về:

- Trong ví dụ, ta sắp xếp document theo timestamp giảm dần (DESC) với cú pháp câu lệnh như sau:

GET /test-webapp02-2024.06.04/\_search

{

"query": {"match\_all": {}},

"sort": {"@timestamp": {"order": "desc"}}

}

### 2.5. Tuỳ chọn trường dữ liệu trả về:

- Cú pháp câu lệnh:

GET /test-webapp02-2024.06.04/\_search

{

"query": {"match\_all": {}},

"\_source": ["message", "logger\_name"]

}

- Ví dụ này kết quả trả về bao gồm 2 trường message và logger\_name

### 2.6. Tìm kiếm dữ liệu phù hợp:

- Cú pháp câu lệnh:

GET /test-webapp01-2024.06.03/\_search

{

"query": {"match": {"logger\_name": "com.javatalent.App"}}

}

- Kết quả trả về bản ghi có logger\_name là com.javatalent.App

### 2.7. Sử dụng điều kiện logic bool trong truy vấn tìm kiếm:

- Sử đụng điều kiện must kết quả trả về khi tất cả các truy vấn là đúng. Ở ví dụ này, truy vấn đúng khi trường message có giá trị là: Starting service [Tomcat]

- Cú pháp câu lệnh:

GET /te st-webapp01-2024.06.03/\_search

{

"query": {

"bool": {

"must": [

{"match": {"message": "Starting service [Tomcat]"}}

]

}

}

}

## **3. Cấu hình Index Policy Management:**

### 3.1. Giải thích ý nghĩa:

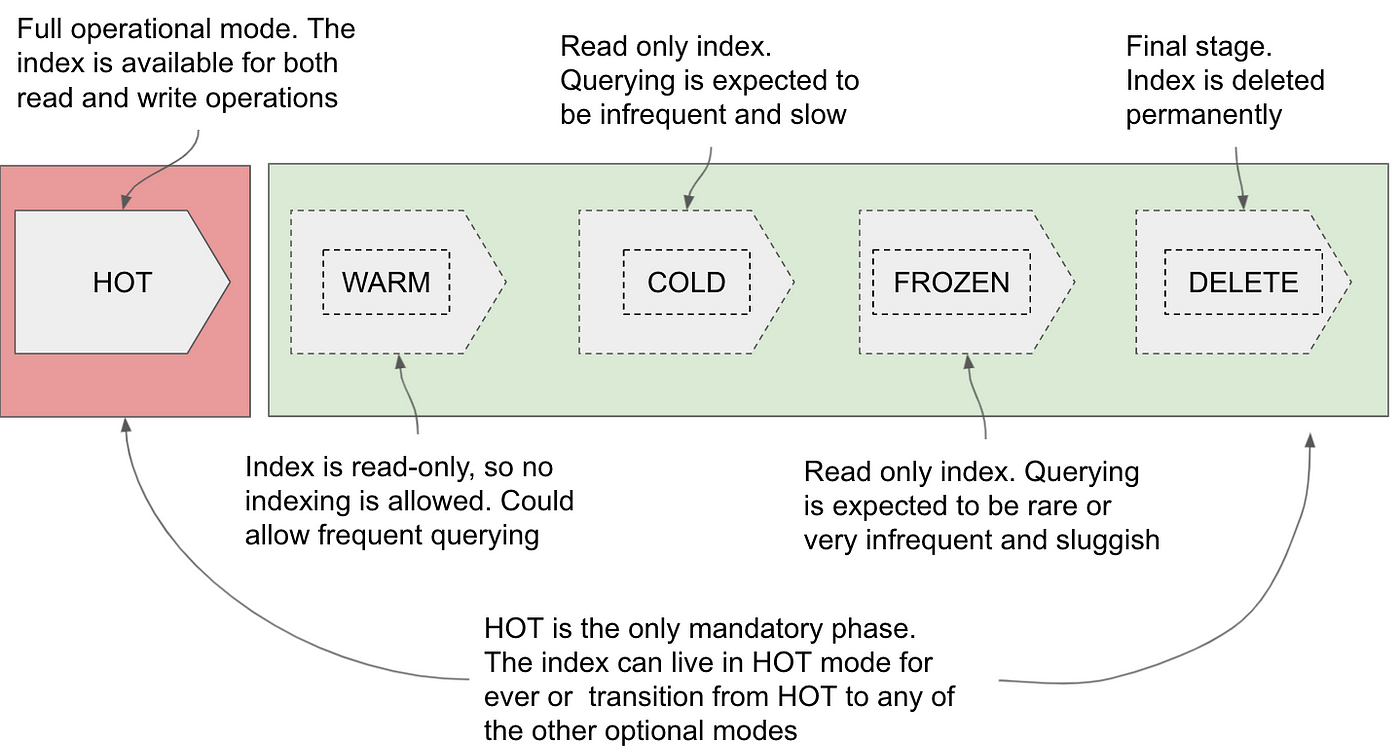
- Nếu chúng ta đang sử dụng ElasticSearch để lưu trữ log của ứng dụng, thì cụm ES có thể trở nên khổng lồ rất nhanh do tích trữ dữ liệu một thời gian.

- May mắn thay, ElasticSearch cung cấp chức năng automatic rollover (tự động luân chuyển) có thể xóa các index cũ để giải quyết vấn đề trên.

**- Có 3 bước chính cần thực hiện:**

* Tạo lifecycle policy chứa các giai đoạn và hành động cho từng giai đoạn.
* Tạo index template và thêm vào lifecycle policy.
* Kiểm ra các index đang đi qua các lifecycle phases như mong muốn.

- Các giai đoạn trong quản lý vòng đời Index:



- Một index trải qua 4 giai đoạn trong: Hot -> Warm -> Cold -> Frozen trong suốt vòng đời dữ liệu sẽ đi qua thứ tự này

* **Giai đoạn Hot:** Chỉ mục vẫn đang được cập nhật và truy vấn tích cực tài nguyên sẽ được ưu tiên phân bổ vào với thời gian phản hồi nhanh nhất.
* **Giai đoạn Warm:** Chỉ mục không còn được cập nhật nhưng vẫn đang được truy vấn, tài nguyên ít được phân bổ.
* **Giai đoạn Cold:** Chỉ mục không còn được cập nhật và hiếm khi được truy vấn. Thông tin vẫn cần có thể tìm kiếm được, những truy vấn đó sẽ chậm hơn.
* **Giai đoạn Frozen (Delete):** Chỉ mục không còn cần thiết nữa và có thể xóa an toàn.

### 3.2. Cách thực hiện:

- Để bắt đầu, ở Tab mục Management, chọn Stack Management.

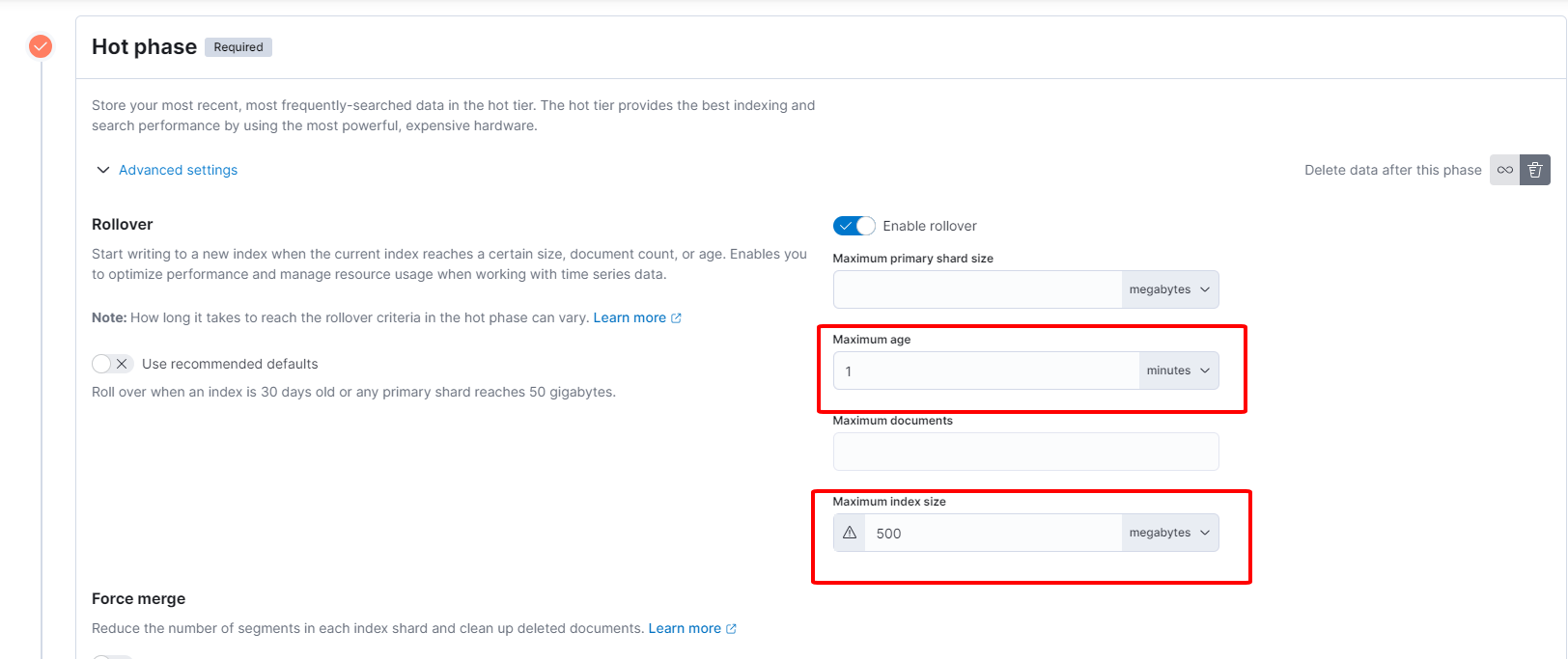
3.2.1. Tạo lifecycle policy:

- Trong ví dụ bên dưới, chúng ta đang tạo một lifecycle policy có tên "test". Giai đoạn "**hot**", khi mà index đang được sử dụng, kết thúc khi index lớn hơn 500MB hoặc có thời gian sống trên 1 phút. Giai đoạn "**delete**" (khi index bị xóa hoàn toàn) bắt đầu 1 phút sau khi luân chuyển (hoặc 2 phút sau khi tạo).

- Chúng ta có thể tạo Index Policy bằng câu lệnh như dưới đây:



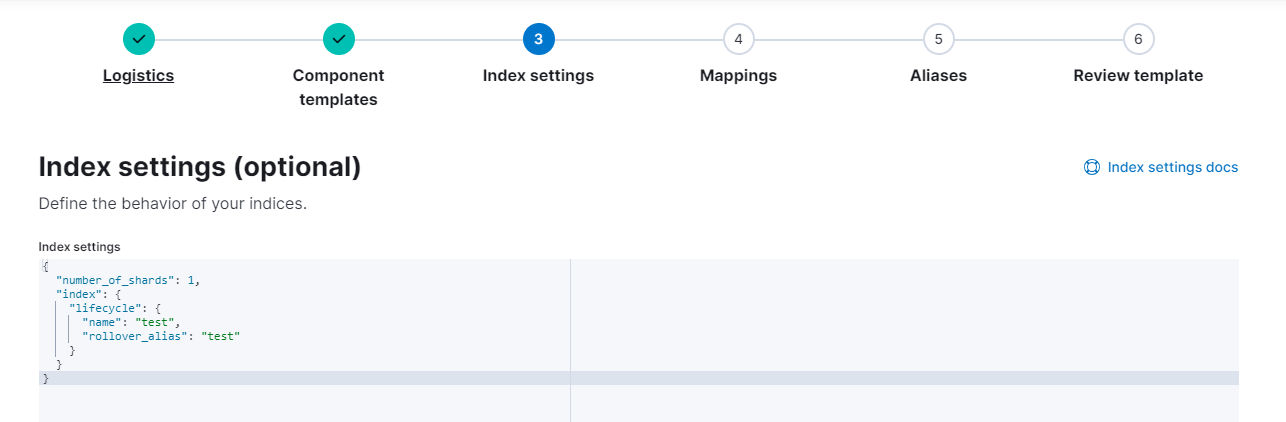
- Hoặc ở mục Index LifeCycle Policies, chọn Create Policy:



3.2.2. Tạo index template:

Index template sẽ đảm bảo rằng các index mới được tạo mới có áp dụng các lifecycle policy mong muốn.

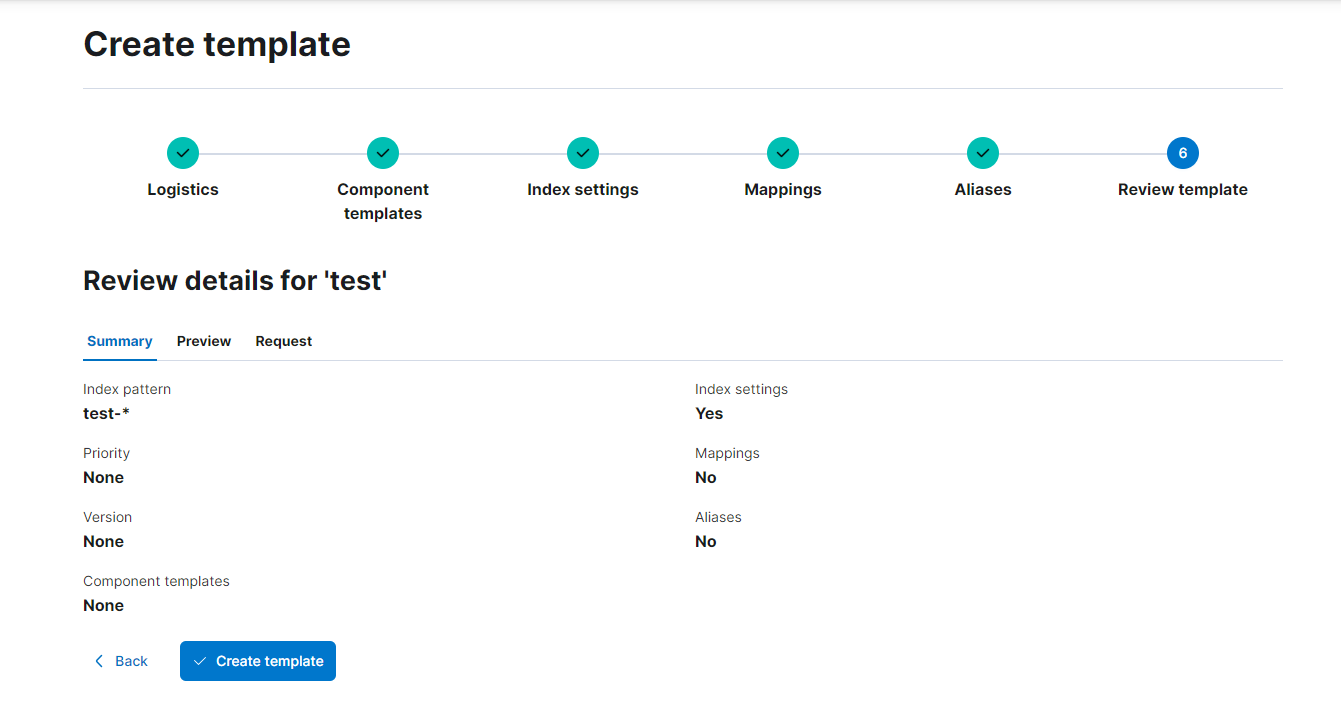
Template bên dưới sẽ được áp dụng cho bất kỳ index nào sẽ được tạo mới có tên bắt đầu với từ "**test**".



- Trong đó:

index.lifecycle.name: Mapping đến tên index policy đã tạo ở bước 3.2.1

index.lifecycle.rollover\_alias: Tất cả các index được tạo từ template này có thể được tham chiếu bằng một tên duy nhất trong các truy vấn.



- Cuối cùng, chọn Create template để hoàn tất việc tạo Index template.

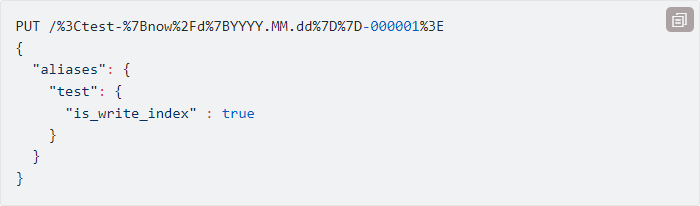
3.2.3. Tạo index có alias:

- Alias là tên thay thế cho một hoặc nhiều index. Có 2 cách để sử dụng alias - có hoặc không có thuộc tính "is\_write\_index".

- **TH1**: Nếu sử dụng thuộc tính "is\_write\_index" (được khuyên dùng và sử dụng trong ví dụ bên dưới), alias index sẽ trỏ tới tất cả các index. Index mới nhất sẽ là index được "write". Cấu hình này cho phép bạn lấy dữ liệu từ tất cả các index (kể cả index cũ) khi tìm kiếm, trong khi các tài liệu mới sẽ được ghi vào index mới nhất.

- **TH2**: Nếu không sử dụng thuộc tính "is\_write\_index", alias sẽ luôn trỏ đến index mới nhất. Ngay cả khi những cái khác vẫn còn, chúng ta sẽ không lấy được dữ liệu cũ thông qua alias (bạn sẽ phải tìm kiếm trực tiếp trong chúng).

- Trong ví dụ bên dưới, chúng ta đang tạo một index "test-2024.04.06-000001" với alias là "test".

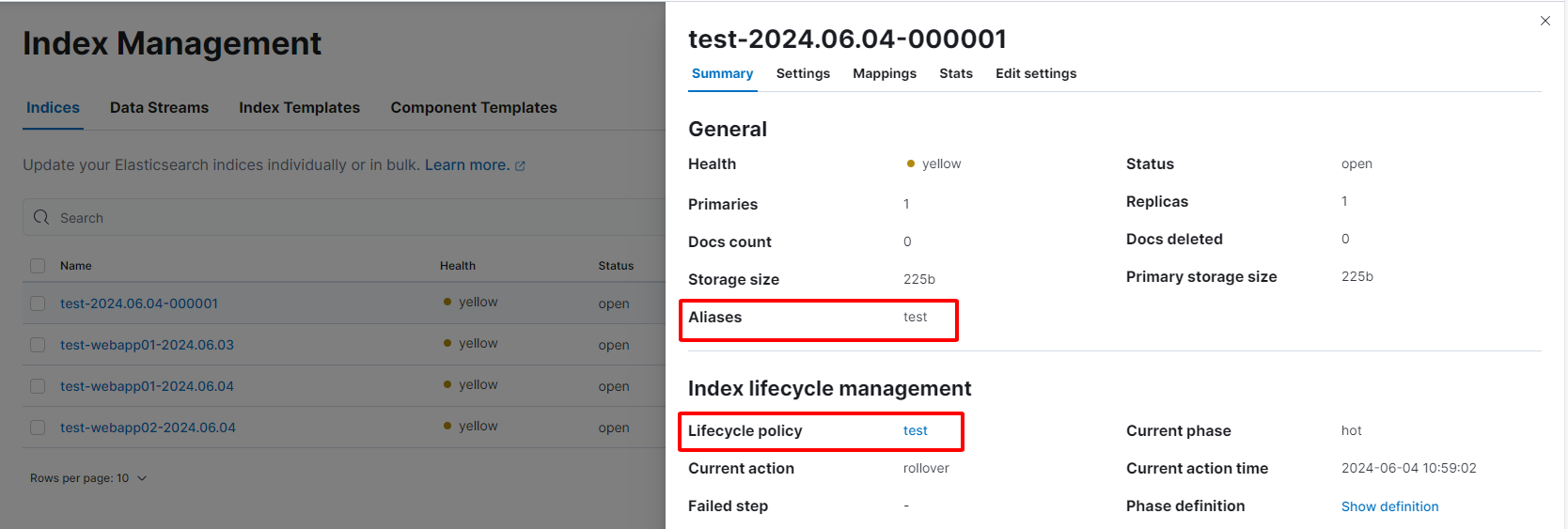


3.2.4. Kiểm tra alias:

- Giờ đây, alias "test" trỏ đến index duy nhất có sẵn - "test-2021.02.05-000001". Hãy nhập câu lệnh sau: GET /test (trong đó: test là tên của alias vừa tạo)

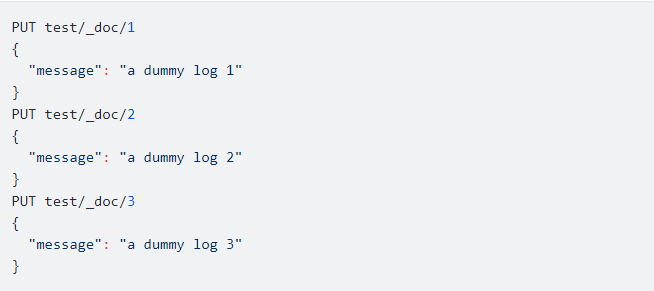


- Ở mục Index Management, chọn Index vừa tạo chúng ta thấy được Index đã được gán Index Policy và kèm theo alias.



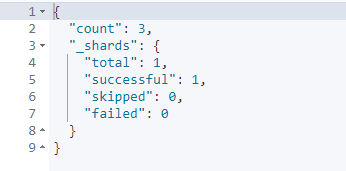
3.2.5. Thêm dữ liệu vào index:

- Giờ ta hãy thử insert 3 documents vào trong index test.



3.2.6. Kiểm tra số lượng documents trong index bằng alias:

- Số lượng documents giờ đã là 3.



3.2.7. Kiểm tra index bằng alias (một lần nữa):

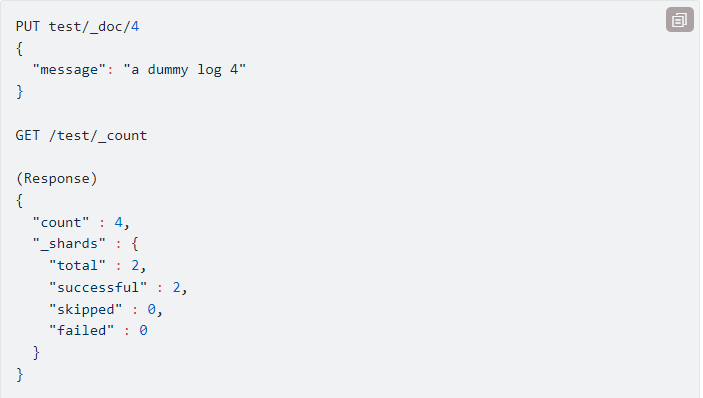
Giờ đây, alias "test" trỏ đến cả hai index "test-2024.06.04-000002" và "test-2024.06.04-000001". Index mới (000002) có thuộc tính "is\_write\_index" được đặt thành true, index cũ (000001) - thành false. Tất cả các bản ghi mới, được chuyển đến alias, được lưu trong index mới, nhưng tìm kiếm bằng alias sẽ tìm thấy tài liệu từ cả hai index.

- Kiểm tra bằng cách nhập lệnh: GET /test

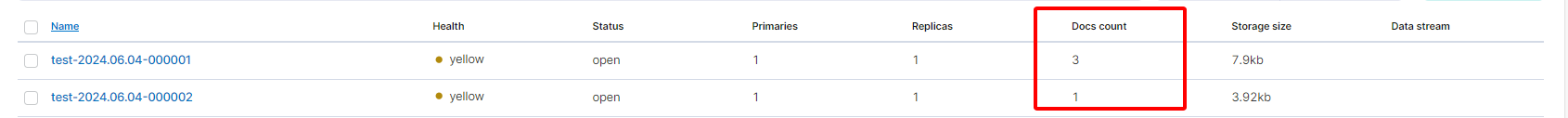


3.2.8. Cho thêm một document và kiểm tra lại số lượng:

- Nếu ta thêm một document, sô lượng sẽ là 4: 3 doc từ index đầu và 1 doc từ index mới.



- Kiểm tra ở phần Index Management, ta thấy index mới tạo là test-2024.06.04-000002 có số docs count là 1.



- Đồng thời, ở index test-2024.06.04-000001 đã chuyển sang phase Delete.

