**[Truy vấn Elastic search]**

Đầu tiên, trước khi đi vào cách tìm kiếm dữ liệu trong ES thì mình sẽ giới thiệu qua về các thao tác create, read, update và delete dữ liệu trong ES. Có 2 cách để thao tác với ES:

* Qua native API: hỗ trợ phía client với nhiều ngôn ngữ lập trình như java, php, ruby.., cách này có ưu điểm về performance khi sử dụng trực tiếp socket để kết nối client và server qua port 9300 (default)
* Qua Restful API: base trên giao thức HTTP và được build trên native API để connect tới ES server thông qua cổng 9200, ưu diểm của cách này alf không phụ thuộc vào nền tảng, ngôn ngữ lập trình, tuy nhiên hiệu năng sẽ không bằng native API

Trong bài viết này mình sẽ chỉ đề cập cách thao tác sử dụng Restful API.

* Đầu tiên, để tạo mới một document thì ta có thể dùng phương thức POST hoặc PUT và truyền vào các tham số bao gồm index\_name, type\_name và id của document ta muốn tạo.

**POST /{index\_name}/{type\_name}/{document\_id}**

**{**

**“field”: “value”,**

**“field2”:”value”,**

**…**

**}**

Hoặc

**PUT /{index\_name}/{type\_name}/{document\_id}**

**{**

**“field”:”value”,**

**…**

**}**

Trong đó index\_name và type\_name là tên của index và tên của Type mà ta muốn lưa document tại đó, nếu Index hoặc Type chưa được tạo thì ES sẽ tự động tạo ra Index và Type tương ứng. Id sẽ là id của document ta tạo, nếu ta không truyền vào id thì nó sẽ tự tạo một id cho document (đối với POST).

Lưu ý: khi sử dụng PUT: nếu document tương ứng với các tham số truyền vào đã tồn tại thì ES sẽ xóa document đó đi vào thêm mới document với dữ liệu mới được truyền vào trong hàm PUT.

* **Delete:**

Có 2 cách xóa document trong Elastich search:

* + Sử dụng delete API:

DELETE /{index\_name}/{type\_name}/id

* + Sử dụng query API:

POST /{index\_name}/{type\_name}/\_delete\_by\_query

{

“query”:{

“match”:{

“message”:”message to delete”

}

}

}

Ví dụ nếu muốn xóa các document có name = “abc” ở trong type customer của index company:

POST /company/customer/\_delete\_by\_query

{

"query": {

"term": {

"name": "abc"

}

}

}

* **Update:**
  + Cách một: reindex document đó

PUT /{index\_name}/{type\_name}/{id}

{

"field": "data"

}

Như đã nói ở trên thì document cũ sẽ được thay thế bởi một document mới

* + Sử dụng POST

POST /{index\_name}/{type\_name}/{id}

{

"field\_name": "data"

}

Với cách này thì ES sẽ update trường thông tin có tên “field\_name” và update data của trường đó.

* **Search document**

Có 2 cách tìm kiếm dữ liệu trong ES là sử dụng Search Lite và search với query DSL.

* **Với Search Lite:**

Đây là cách tiếp cận đơn giản nhất để có thể build các câu truy vấn cơ bản bằng cách truyền các tham số trong query string.

Để lấy ra một document ta sẽ truyền vào id của document đó cùng với index name, type name:

**GET /{index\_name}/{type\_name}/{id}**

Để lấy ra tất cả các document:

**GET /\_search** hoặc **GET /\_all/\_search**

Tìm tất cả document trong một index:

**GET /{index\_name}/\_ search**

Để lấy ra các document trong cùng một type:

**GET /{index\_name}/{type\_name}/\_ search**

Search dữ liệu theo các tham số truyền vào ta sẽ dùng biến **q=**{tham số}

*Ví dụ: để tìm một employee trong index google có last\_name = “Sam” và age = 21*

*GET /google/employee/\_search?q=last\_name:Smith AND age:21*

* **q**  chỉ là một tham số phổ biến nhất cho phép trong URI của truy vấn dùng để truyền vào một query string, ngoài ra còn có một số tham số khác để xác định analyzer sẽ được sử dụng khi truy vấn hay số kết quả trả về cho mỗi truy vấn. Xem tại [link sau](https://www.elastic.co/guide/en/elasticsearch/reference/current/search-uri-request.html).
* **Với query DSL:**

Với những câu query phức tạp hơn ví dụ: aggregation thì việc sử dụng URI search không thể đáp ứng được, hơn nữa khi build các câu lệnh dài và các truy vấn lồng nhau thì rất dễ mắc lỗi, khi đó chúng ta sẽ sử dụng query DSL (domain-specific language) là một ngôn ngữ đặc tử sử dụng cú pháp của JSON trong request body để diễn đạt các câu query.

Cú pháp các câu query:

{

"query" : {

"query\_name" : {

"field\_to\_search" : "value",

…..

}

}

}

Dưới đây mình sẽ giới thiệu qua các câu truy vấn phổ biến trong Elastic search mà người dùng hay sử dụng nhất:

**match**: Tìm kiếm các document theo giá trị của các trường, các document sẽ match nếu một term trong query đó match, càng nhiều term được match thì score của document càng lớn. Query search sẽ được apply analyzer config cho field search đó hoặc để mặc định (nếu không set). Tức là, khi các term được truyền vào thì nó sẽ được là mịn trước khi tìm kiếm, ví dụ: “CAT” hay “cat” thì đều tìm là “cat”.

(term: khi các value được đưa vào để truy vấn thì nó sẽ được cắt ra thành các term dựa trên một số quy định như là chia cắt các khoảng trắng. Ví dụ: “one two” sẽ là “one” và “two”. Việc chia các giá trị tìm kiếm thành các term là để có thể tìm kiếm với Inverted index)

*Ví dụ:*

*GET /infore/employee/\_search*

*{*

*“query”:{*

*“match”:{*

*“name”:”kiên phúc”*

*}}}*

* *Kết quả trả về là Kiên Nguyễn và Phúc Hoàng*

**match\_phrase:** document chỉ match nếu các tẻm match có cùng thứ tự và liên tiếp nhau.

Tức là với ví dụ trên thì kết quả là *Kiên Nguyễn* và *Phúc Hoàng* đều không thỏa mãn khi thay thế **match** bằng **match\_phrase** mà chỉ những người có tên là một string trong đó chứa chuỗi “Kiên Phúc” mới là kết quả hợp lý.

**match\_phrase\_prefix**: tương tự như match\_phrase nhưng có thêm điều kiện khớp tiền tố của từ trong văn bản.

*Ví dụ khi tìm kiếm với “Phong hen” thì cả “Phong hen” và “Phong henry” đều match*

**term:** query trên câu truy vấn truyền vào nhưng analyzer sẽ không được apply, tức là các term phải được match một cách chính xác hoàn toàn, ví dụ: “Cat” sẽ khác “cat”.

**multi\_match:** aply match query trên nhiều field khác nhau

*Ví dụ:*

*{*

*"query": {*

*"multi\_match" : {*

*"query": "this is a test",*

*"fields": [ "subject", "message" ]*

*}}}*

**bool:** cho phép kết hợp các câu truy vấn khác để tạo ra một logic hợp lý. Có các loại truy vấn bool:

* + - Must: phải phù hợp với tất cả các điều kiện và đóng góp vào score search của document
    - Filter: giống với must nhưng bỏ qua đóng góp điểm số
    - Should: chỉ cần document match một trong các điều kiện
    - Must\_not: ngược lại với must, tức là phải không match với tất cả các term.

**must** và **filter** sẽ tương tự với **and**, **should** tương đương với **or** còn **must\_not** tương đương với **nor** trong toán học logic, chỉ cần hiểu như vậy là ta có thể xây dựng các truy vấn lồng nhau sử dụng truy vấn **bool**.

*Ví dụ:*

*{*

*"query" : {*

*"constant\_score" : {*

*"filter" : {*

*"bool" : {*

*"should" : [*

*{ "term" : {"productID" : "KDKE-B-9947-#kL5"}},*

*{ "bool" : {*

*"must" : [*

*{ "term" : {"productID" : "JODL-X-1937-#pV7"}},*

*{ "term" : {"price" : 30}}*

*]*

*}}*

*]*

*}*

*}*

*}*

*}*

*}*

*Câu truy vấn trên sẽ tương đương với câu truy vấn sau trong mysql:*

*SELECT document*

*FROM products*

*WHERE productID = "KDKE-B-9947-#kL5"*

*OR ( productID = "JODL-X-1937-#pV7"*

*AND price = 30 )*