ELASTICSEARCH

The Basic of Elasticsearch

Table of Contents

Contents

Apa itu Elasticsearch?

Elasticsearch adalah mesin pencari dan analisis data terdistribusi yang dibangun di atas Apache Lucene. Ia menyediakan mesin pencari teks lengkap yang terdistribusi dengan antarmuka web HTTP dan dokumen JSON bebas skema.

Kelebihan Elasticsearch:

- **Skalabilitas:** Elasticsearch dapat di-skala secara horizontal dengan menambahkan lebih banyak node ke cluster. Hal ini memungkinkan Anda untuk menangani volume data yang besar dan permintaan pencarian yang kompleks.
- **Performa:** Elasticsearch sangat cepat dan efisien dalam melakukan pencarian teks. Ia menggunakan berbagai teknik, seperti sharding dan replikasi, untuk mengoptimalkan kinerja.
- **Fleksibilitas:** Elasticsearch mendukung berbagai jenis data, termasuk teks, JSON, dan struktur data yang kompleks. Ia juga menyediakan API yang kuat untuk melakukan berbagai jenis analisis data, seperti agregasi dan analisis geospasial.
- **Kemudahan Penggunaan:** Elasticsearch relatif mudah digunakan dan dipelajari. Ia menyediakan antarmuka web yang intuitif dan dokumentasi yang lengkap.
- **Open Source:** Elasticsearch adalah perangkat lunak open source, yang berarti gratis untuk digunakan dan dimodifikasi.

Kekurangan Elasticsearch:

- **Kompleksitas:** Elasticsearch dapat menjadi kompleks untuk dikonfigurasi dan dikelola, terutama untuk cluster besar.
- **Kebutuhan Perangkat Keras:** Elasticsearch membutuhkan perangkat keras yang kuat untuk berjalan dengan lancar.
- **Ketergantungan pada Lucene:** Elasticsearch bergantung pada Apache Lucene, yang dapat berubah seiring waktu. Hal ini dapat menyebabkan masalah kompatibilitas.

Mengapa Harus Menggunakan Elasticsearch? Elasticsearch adalah pilihan yang tepat untuk berbagai kasus penggunaan, termasuk:

- **Pencarian Teks Lengkap:** Elasticsearch sangat ideal untuk aplikasi pencarian teks lengkap, seperti situs web e-commerce dan mesin pencari internal.
- **Analisis Log:** Elasticsearch dapat digunakan untuk menganalisis log aplikasi dan sistem, yang dapat membantu Anda mengidentifikasi masalah dan meningkatkan kinerja.
- **Analisis Geospasial:** Elasticsearch mendukung analisis geospasial, yang memungkinkan Anda untuk melakukan analisis data yang terkait dengan lokasi geografis.
- **Monitoring Infrastruktur:** Elasticsearch dapat digunakan untuk memantau infrastruktur TI Anda, seperti server, jaringan, dan aplikasi.
- **Analisis Bisnis:** Elasticsearch dapat digunakan untuk melakukan analisis bisnis, seperti analisis pelanggan dan analisis pasar.

Download & Install Elasticsearch

Elasticsearch mendukung banyak Sistem Operasi, mulai dari Windows, Linux, MacOS, deb, rpm. Dalam pembahasan kali ini, saya akan menjelaskan cara instalasi elasricsearch pada Windows. berikut cara-caranya:

1. Silahkan kunjungi link download berikut: https://www.elastic.co/downloads/elasticsearch.
2. Pilih versi elasticsearch, disini saya pilih versi Windows.

3. Klik tombol Download.
4. Extract file zip yang sudah didownload.
5. Lalu buka folder hasil extact dan jalankan perintah berikut pada command:
   ```
   bin/elasticsearch
   ```
6. Selesai.

Configuration Elasticsearch

Pada tahap instalasi sebelumnya kita berhasil menjalankan elasticsearch tanpa melakukan perubahan config, dengan kata lain kita menggunakan default configuration yang ada. Kita dapat mengubah config tersebut sesuai dengan kebutuhan kita dengan membuka file config **/config/elasticsearch.yml** dan ubah isi dari config tersebut. Berikut contoh perubahan config:

```
cluster.name: my-application
node.name: node-1
xpack.security.enabled: false
http.port: 9200
path.data: /path/to/data
path.logs: /path/to/logs
path.repo: backup
```

Penjelasan:

1. cluster.name: my-application
   Pengaturan ini menentukan nama cluster Elasticsearch Anda. Ini berfungsi sebagai pengenal unik untuk kumpulan node yang saling terhubung dan bekerja sama. Dalam kasus ini, cluster diberi nama "my-application".
2. node.name: node-1
   Pengaturan ini memberikan nama pada node Elasticsearch khusus ini. Ini membantu dalam membedakan node individual di dalam cluster. Di sini, node diberi nama "node-1".
3. xpack.security.enabled: false
   Pengaturan ini mengontrol apakah fitur keamanan X-Pack diaktifkan. X-Pack merupakan fitur berbayar dari elasticsaerch, saya non-aktifkan karena pada pembahasan kali ini kita tidak menggunakan fitur ini.
4. http.port: 9200
   Pengaturan ini menentukan nomor port tempat Elasticsearch mendengarkan permintaan HTTP yang masuk. Port default 9200 umumnya digunakan untuk komunikasi dengan klien (aplikasi, alat) yang berinteraksi dengan cluster.
5. path.data: /path/to/data
   Pengaturan ini menentukan direktori tempat Elasticsearch menyimpan shard (partisi) data yang berisi dokumen Anda yang telah diindeks. Di sini, path data diatur ke /path/to/data. Sangat penting untuk memilih lokasi dengan kapasitas penyimpanan dan performa yang memadai.
6. path.logs: /path/to/logs
   Pengaturan ini menentukan direktori tempat Elasticsearch menyimpan file log-nya. Log ini berisi informasi berharga tentang kesehatan cluster, operasi, dan potensi masalah. Path log diatur ke /path/to/logs dalam konfigurasi ini.
7. path.repo: backup
   Pengaturan ini mungkin terkait dengan path repository yang digunakan untuk operasi snapshot (membuat cadangan dan mengembalikan data cluster). Tujuan spesifik tergantung pada

bagaimana Anda mengonfigurasi snapshotting di lingkungan Anda. Dalam beberapa kasus, ini mungkin tidak didefinisikan secara eksplisit jika snapshot tidak digunakan.

Elasticsearch Index

Elasticsearch Indices tidak sama dengan istilah index pada Relational Database. Index pada elasricsearch sama dengan Table pada Relational Database. Isi dari index adalah document yang didalamnya berupa JSON, mirip seperti document di MongoDB.

- RDBMS => Databases => Tables => Columns/Rows
- Elasticsearch => Clusters => Indices => Shards => Documents with key-value pairs

Elasticsearch menggunakan antarmuka REST API, dengan kata lain Ketika ingin berinteraksi dengan Elasticsearch kita perlu melakukan request pada host yang digunakan Elasticsearch. Anda bisa menggunakan Tools apapun untuk melakukan request http, disini saya menggunakan Extensions dari VS Code yaitu REST Client.

- API untuk membuat Index baru (customers):

```
PUT http://localhost:9200/customers
```

- API untuk menghapus index baru (customers):

```
DELETE  http://localhost:9200/customers
```

- API untuk melihat semua index yang ada:

```
GET http://localhost:9200/_cat/indices?v
```

Mapping

Mapping adalah definisi struktur data yang disimpan dalam Elasticsearch. Ini menentukan bagaimana dokumen diindeks, disimpan, dan dicari dalam cluster. Mapping memungkinkan Anda untuk mendefinisikan tipe data, format, dan properti untuk setiap bidang dalam dokumen Anda. Untuk melihat Mapping pada index, anda dapat menjalankan berikut (Index customers):

```
GET http://localhost:9200/customers/_mapping
```

Dynamic Mapping

Secara default, Ketika membuat Index maka akan menggunakan Dynamic Mapping. Elasticsearch secara otomatis membuat mapping berdasarkan struktur data dokumen yang Anda masukkan untuk pertama kalinya. Cara ini cocok untuk situasi di mana struktur data Anda tidak terdefinisi dengan baik atau dapat berubah seiring waktu.

Explicit Mapping

Explicit Mapping memungkinkan kita membuat tipe data secara langsung pada index. Cara ini memberikan control penuh atas struktur dan tipe pada index. Untuk menggunakan explicit mapping kita harus memberitahu property dan tipe data apa yang ingin disimpan pada document. Contoh Explicit Mapping pada index customers:

```
PUT http://localhost:9200/customers/_mapping
Content-Type: application/json

{
    "properties": {
        "username": {
            "type": "keyword"
```

```
        },
        "first_name": {
            "type": "text"
        },
        "last_name": {
            "type": "text"
        },
        "email": {
            "type": "keyword"
        },
        "gender": {
            "type": "keyword"
        },
        "birth_date": {
            "type": "date",
            "format": "yyyy-MM-dd"
        }
    }
}
```

Data Type

Elasticsearch menyediakan berbagai tipe data untuk menyimpan dan mengelola data secara efisien. Berikut adalah beberapa tipe data utama yang tersedia:

- **integer:** Bilangan bulat 32-bit dengan rentang -2^31 hingga 2^31-1. Cocok untuk data numerik yang lebih kecil dan hemat ruang.
- **double:** Bilangan desimal 64-bit dengan presisi ganda. Cocok untuk data desimal dengan presisi tinggi, seperti nilai ilmiah dan keuangan.
- **float:** Bilangan desimal 32-bit dengan presisi tunggal. Cocok untuk data desimal dengan presisi sedang, seperti nilai sensor dan data statistik.
- **keyword:** String yang dioptimalkan untuk pencarian. Cocok untuk data teks yang akan digunakan sebagai kunci pencarian, seperti nama produk, ID pengguna, dan alamat email.
- **text:** String yang dioptimalkan untuk analisis teks. Cocok untuk data teks yang akan dianalisis, seperti deskripsi produk, ulasan pelanggan, dan artikel berita.
- **date:** Tanggal dalam format ISO 8601. Cocok untuk menyimpan tanggal dan waktu kejadian, seperti tanggal lahir, tanggal pendaftaran, dan tanggal publikasi.
- **datetime:** Tanggal dan waktu dalam format ISO 8601 dengan presisi milidetik. Cocok untuk menyimpan waktu kejadian yang presisi, seperti waktu login, waktu transaksi, dan waktu penyelesaian.
- **boolean:** True or False. Cocok untuk menyimpan nilai yang bersifat biner, seperti status aktif/tidak aktif, pengaturan preferensi, dan hasil tes.
- **geo_point:** Koordinat geografis dalam format lat/lon. Cocok untuk menyimpan lokasi geografis, seperti alamat, titik peta, dan koordinat GPS.
- **nested:** Struktur data bersarang yang memungkinkan Anda untuk menyimpan data kompleks dalam satu bidang. Cocok untuk data yang memiliki hierarki, seperti komentar pada posting blog, tag pada produk, dan bagian dalam dokumen.
- Dan tipe-tipe data lainnya.

Elasticsearch menyediakan berbagai tipe data yang memungkinkan anda untuk menyimpan, mengelola, dan menganalisis berbagai jenis data dengan cara yang efisien dan fleksibel. Untuk melihat tipe data lainnya yang tidak bisa saya tuliskan disini, kalian lihat pada link dokumentasi berikut https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html.

Doc API

Doc API memungkinkan kita melakukan operation CRUD (Create Read Update Delete) data atau document pada index. Bentuk umum dari Doc API sebagai berikut:

```
<METHOD> <Host>/<Index>/_doc/<_id>
```

Penjelasan:

- <METHOD>: metode request yang akan menentukan apa document akan di GET, POST, DELETE.
- <Host>: host pada elasticsearch.
- <Index>: nama index pada elasticsearch.
- <_id>: data unik yang membedakan pada setiap document.

Create Or Replace Doc API

API ini akan membuat document baru jika <_id> belum ada pada index dan akan mengganti document jika <_id> sudah ada dengan data yang baru. Contoh penggunaan pada index products dengan <_id> 1:

```
POST http://localhost:9200/products/_doc/1
Content-Type: application/json

{
    "name": "Pop Mie Sedap",
    "price": 2500
}
```

Get Doc API

API yang akan menampilkan data berdasarkan <_id> yang dikirimkan. Contoh menampikan products dengan <_id> 1:

```
GET http://localhost:9200/products/_doc/1
```

Delete Doc API

API ini akan menghapus document berdasarkan <_id> yang dikirimkan. Contoh menghapus products dengan <_id> 1.

```
DELETE http://localhost:9200/products/_doc/1
```

Create & Update API

Secara umum Create dan Update API sama dengan Create or Replace pada Doc API yaitu menambah atau mengupdate data, akan tetapi fungsinya lebih spesifik lagi. Pada Create API Ketika <_id> dalam document sudah ada maka akan terjadi error saat menambah data atau document baru, sebaliknya pada Update API jika <_id> tidak ada maka akan terjadi error saat mengubah data. Ketika menggunakan Update API anda hanya perlu megirimkan property dan value bagi data yang ingin diubah saja.

- Contoh Create API menambah products dengan <_id> 2:

```
POST http://localhost:9200/products/_create/2
Content-Type: application/json

{
    "name": "Indo Mie Goreng",
    "price": 2000
}
```

- Contoh Update API mengubah price pada products dengan <_id> 2

```
POST http://localhost:9200/products/_update/2
Content-Type: application/json

{
    "doc": {
        "price": 4000
    }
}
```

Bulk API

Saat melakukan request pada elasticsearch kita selalu menerima response untuk setiap hit pada API dan tentu ini akan memakan banyak proses jika kita melakukan request dalam jumlah besar. Lalu bagaimana jika ingin melakukan request dalam jumlah yang besar dengan sekali request dan response saja? Jawabannya dengan menggunakan Bulk API, API ini memungkinkan kita untuk melakukan request sekaligus dengan sekali response.

Berikut Endpoint Untuk melakukan Bulk API pada elasticserach:

```
POST http://localhost:9200/_bulk
Content-Type: application/json

< bulk.json
```

Isi dari bulk.json sendiri yaitu:

```
{"create": {"_index": "customers", "_id": "joko"}}
{"name": "Joko Anwar", "register_at": "2024-01-01 00:00:00"}
{"index": {"_index": "customers", "_id": "budi"}}
{"name": "Budi Nugroho", "register_at": "2024-01-01 00:00:00"}
{"update": {"_index": "products", "_id": "1"}}
{"doc": {"price": 3000}}
{"delete": {"_index": "customers", "_id": "joko"}}
```

Didalam bulk.json sendiri setiap data dalam **{}** harus dipisah dengan **\n** atau **Enter**. Baris pertama adalah jenis perintahnya dan baris kedua merupakan body.

- Baris untuk melakukan Create API:

```
{"create": {"_index": "customers", "_id": "joko"}}
{"name": "Joko Anwar", "register_at": "2024-01-01 00:00:00"}
```

- Baris untuk melakukan Doc API:

```
{"index": {"_index": "customers", "_id": "budi"}}
{"name": "Budi Nugroho", "register_at": "2024-01-01 00:00:00"}
```

- Baris untuk melakukan Update API:

```
{"update": {"_index": "products", "_id": "1"}}
```

```
{"doc": {"price": 3000}}
```

- Baris untuk melakukan Delete API:

```
{"delete": {"_index": "customers", "_id": "joko"}}
```

Aliases API

Dalam Elasticsearch, aliases adalah sebutan untuk nama alternatif yang diberikan kepada index atau kumpulan data stream. Aliases menawarkan fleksibilitas dan kemudahan pengelolaan dalam hal cara Anda berinteraksi dengan data Anda.

Contoh penggunaan dalam menambah alias **customer** pada index **customers** sebagai berikut:

```
POST http://localhost:9200/_aliases
Content-Type: application/json

{
    "actions": [
        {
            "add": {
                "alias": "customer",
                "index": "customers"
            }
        }
    ]
}
```

Untuk melihat list alias apa saja yang ada pada index kita dapat melihat dengan API berikut:

```
GET http://localhost:9200/_aliases
```

Kita juga menghapus alias jika tidak dipakai lagi dengan cara berikut ini:

```
POST http://localhost:9200/_aliases
Content-Type: application/json

{
    "actions": [
        {
            "remove": {
                "alias": "customer",
                "index": "customers"
            }
        }
    ]
}
```

Reindex API

Reindex API adalah alat yang ampuh di Elasticsearch untuk menyalin data dari satu index ke index lainnya. Contoh saya membuat index **orders_v2** dan ingin memindahkan data lama dari index **orders** ke index **orders_v2** yang baru saya buat. Kita bisa melakukannya dengan menggunakan API pada redindex berikut:

```
POST http://localhost:9200/_reindex
```

```
Content-Type: application/json

{
    "source": {
        "index": "orders"
    },
    "dest": {
        "index": "orders_v2"
    }
}
```

Search API

Search API adalah salah satu fitur inti dari Elasticsearch dan berfungsi sebagai mesin penelusuran yang kuat untuk data terstruktur Anda. Melalui Search API, Anda dapat mencari dokumen secara efisien berdasarkan berbagai kriteria dan mendapatkan hasil yang relevan. Search API bisa menggunakan dua Method yaitu GET dan POST. Data hasil search akan selalu paginate dengan limit 10 data dan offset dari 0.

Methods GET:

- Mengambil semua data pada Index products:
  ```
  GET http://localhost:9200/products/_search
  ```
- Melakukan paginate data pada index products:
  ```
  GET http://localhost:9200/products/_search?size=1&from=0
  ```
- Melakukan sorting data berdasarkan value di index products (data dengan tipe text tidak bisa disorting, ubah data dengan tipe keyword jika ingin melakukan sorting pada string):
  ```
  GET http://localhost:9200/products/_search?sort=price:asc
  ```

Methods POST, dengan menggunakan Method POST kita bisa meletakkan params pada body dan tentu mempermudah dalam membuat request Search disbanding harus menambah query pada URL. Contoh penggunaan Search API dengan POST Method pada index categories:

```
POST http://localhost:9200/categories/_search
Content-Type: application/json

{
    "query": {
        "match_all": {}
    },
    "size": 10,
    "from":0,
    "sort": [
        {
            "id": {
                "order": "asc"
            }
        }
    ]
}
```

Search Query

Dalam Search API Elasticsearch, query adalah komponen inti yang menentukan kriteria untuk memilih dokumen yang relevan dengan pencarian Anda. Dengan query, Anda dapat menentukan kata kunci, filter, dan kriteria lainnya untuk mempersempit hasil dan menemukan informasi yang Anda butuhkan.

Match All Query

Match All adalah query yang akan mengambil semua data pada index. Contoh mengambil semua data pada index customers:

```
POST http://localhost:9200/customers/_search
Content-Type: application/json

{
    "query": {
        "match_all": {}
    }
}
```

Match Query

Match Query adalah salah satu query paling dasar dan umum digunakan dalam Search API Elasticsearch. Fungsinya untuk mencari dokumen yang mengandung istilah tertentu di dalam bidang tertentu dan query ini sangat cocok untuk mencari data dengan tipe data text. Berikut contoh mencari data menggunakan match query pada index customers yang memiliki bank “BCA”:

```
POST http://localhost:9200/customers/_search
Content-Type: application/json

{
    "query": {
        "match": {
            "banks.name": "BCA"
        }
    }
}
```

Term Query

Term Query adalah jenis query lain yang tersedia dalam Search API Elasticsearch. Berbeda dengan Match Query yang mencari kecocokan secara umum, Term Query digunakan untuk mencari dokumen yang memiliki nilai persis pada bidang tertentu. Berikut contoh pencarian menggunakan Term Query pada index customers dengan denger “Female”.

```
POST http://localhost:9200/customers/_search
Content-Type: application/json

{
    "query": {
        "term": {
            "gender": "Female"
        }
    }
```

```
}
```

Terms Query

Terms Query adalah query yang digunakan untuk mencari dokumen yang mengandung satu atau lebih istilah dari daftar istilah yang ditentukan. Berbeda dengan Match Query yang mencari kecocokan secara umum, Terms Query memberikan kontrol lebih besar atas istilah pencarian yang spesifik. Berikut penggunaan Terms query untuk mencari pada index customers dengan username "username1", "username2", "username3":

```
POST http://localhost:9200/customers/_search
Content-Type: application/json

{
    "query": {
        "terms": {
            "username": [
                "username1",
                "username2",
                "username3"
            ]
        }
    }
}
```

Range Query

Range Query adalah salah satu query yang ampuh dalam Search API Elasticsearch untuk memfilter dokumen berdasarkan rentang nilai pada suatu bidang. Berikut contoh penggunaan Range Query untuk mencari data index customers yang lahir direntang tanggal 1995-01-01 sampai 2000-01-01:

```
POST http://localhost:9200/customer/_search
Content-Type: application/json

{
  "query": {
    "range": {
      "birth_date": {
        "gte": "1995-01-01",
        "lte": "2000-01-01"
      }
    }
  }
}
```

Bool Query

Bool Query memungkinkan Anda untuk menggabungkan beberapa query lain menggunakan operator logika seperti AND, OR, NOT, dan FILTER. Hal ini memberikan fleksibilitas tinggi untuk membangun pencarian yang kompleks dan presisi. Beberapa klausul utama dalam Bool Query yaitu must, filter, should, dan must_not.

Must

Sub-query yang ada di dalam must wajib dipenuhi oleh dokumen agar dokumen tersebut bisa masuk ke dalam hasil pencarian. Sub-query ini juga akan memberikan kontribusi pada skor relevansi pencarian. Contoh pencarian dokumen pada index customers untuk mendapatkan customer yang memilki hobi gaming dan bank bca digital:

```
POST http://localhost:9200/customers/_search
Content-Type: application/json

{
    "query": {
        "bool": {
            "must": [
                {
                    "match": {
                        "hobbies": "gaming"
                    }
                },
                {
                    "match": {
                        "banks.name": "bca digital"
                    }
                }
            ]
        }
    }
}
```

Filter

Mirip dengan must, sub-query yang ada di dalam filter juga harus dipenuhi oleh dokumen. Namun, tidak mempengaruhi skor relevansi dokumen.

```
POST http://localhost:9200/customers/_search
Content-Type: application/json

{
    "query": {
        "bool": {
            "filter": [
                {
                    "match": {
                        "hobbies": "gaming"
                    }
                },
                {
                    "match": {
                        "banks.name": "bca digital"
                    }
                }
            ]
```

```
        }
    }
}
```

Should

Sub-query yang ada di dalam should bersifat opsional. Kehadiran sub-query ini tidak wajib namun dapat meningkatkan skor relevansi dokumen jika dipenuhi. Semakin banyak sub-query should yang cocok, semakin tinggi skor relevansinya. Contoh pencarian pada index customers yang memiliki bank bca dan bni:

```
POST http://localhost:9200/customers/_search
Content-Type: application/json

{
    "query": {
        "bool": {
            "should": [
                {
                    "match": {
                        "banks.name": "bca"
                    }
                },
                {
                    "match": {
                        "banks.name": "bni"
                    }
                }
            ]
        }
    }
}
```

Must Not

Sub-query yang ada di dalam must_not tidak boleh dipenuhi oleh dokumen agar dokumen tersebut bisa masuk ke dalam hasil pencarian. Dokumen yang cocok dengan salah satu sub-query must_not akan dikeluarkan dari hasil pencarian. Anda dapat menggunakan must_not untuk mengecualikan dokumen yang mengandung kata "error" dari hasil pencarian. Contoh pencarian pada index customers dengan gender selain Female:

```
POST http://localhost:9200/customers/_search
Content-Type: application/json

{
    "query": {
        "bool": {
            "must_not": [
                {
                    "term": {
                        "gender": "Female"
                    }
```

```
                }
            ]
        }
    }
}
```

Score Boost

Dalam Elasticsearch, score adalah nilai numerik yang menunjukkan relevansi dokumen terhadap query pencarian Anda. Dokumen dengan skor lebih tinggi dianggap lebih relevan dengan query Anda dan akan diberi peringkat lebih tinggi dalam hasil pencarian.

Score boost adalah teknik dalam Elasticsearch untuk menyesuaikan skor relevansi dokumen secara manual saat pencarian. Dengan menerapkan faktor peningkatan (boost), Anda dapat menambah atau mengurangi skor dokumen terlepas dari faktor-faktor yang biasanya memengaruhi relevansinya.

Berikut contoh penggunaan Boost dalam pencarian pada index customers untuk customer yang memiliki bank bni (skor relavansi meningkat 2) atau memiliki bank bca (skor relevansi menjadi 0) serta memiliki hobi gaming (skor relevansi 0):

```
POST http://localhost:9200/customers/_search
Content-Type: application/json

{
    "query": {
        "bool": {
            "must": [
                {
                    "term": {
                        "hobbies": {
                            "value": "gaming",
                            "boost": 0
                        }
                    }
                }
            ],
            "should": [
                {
                    "term": {
                        "banks.name": {
                            "value": "bca",
                            "boost": 0
                        }
                    }
                },
                {
                    "term": {
                        "banks.name": {
                            "value": "bni",
```

```
                            "boost": 2
                        }
                    }
                }
            ]
        }
    }
}
```

Update By Query

Ketika kita mengupdate Mapping Index pada elasticsearch maka dokumen baru yang dibuat akan menyesuaikan dengan struktur Mapping terbaru. Lalu bagaimana dengan data lama? Data lama akan tetap menggunakan struktur lama sampai kita melakukan update document tersebut. Hal ini akan menimbulkan masalah baru dimana sebelumnya kita mengetahui jika ingin mengupdate data pada elasticsearch kita perlu mengetahui <_id> unik dari setiap data pada dokumen lalu menggunakan Doc API atau Update API. Untuk menyelesaikan masalah tersebut elasticsearch menyediakan fitur Update by Query, dimana kita dapat mengupdate data berdasarkan query yang kita berikan. Contoh memperbarui semua data pada index products:

```
POST http://localhost:9200/products/_update_by_query
Content-Type: application/json

{
    "query": {
        "match_all": {}
    }
}
```

Delete By Query

Delete by Query adalah fitur di Elasticsearch yang memungkinkan Anda untuk menghapus dokumen secara efisien berdasarkan kriteria tertentu yang ditentukan oleh query. Ini berguna ketika Anda ingin menghapus banyak dokumen yang cocok dengan pola tertentu, alih-alih menghapusnya satu per satu. Contoh menghapus semua data pada index categories dengan hobi gaming:

```
POST http://localhost:9200/categories/_delete_by_query
Content-Type: application/json

{
    "query": {
        "match": {
            "hobbies": "gaming"
        }
    }
}
```

Search After

Search After adalah teknik paging yang efisien di Elasticsearch. Ini memungkinkan Anda untuk mengambil halaman hasil pencarian berikutnya tanpa perlu mengulang seluruh query.

Cara Kerja:

1. Lakukan Pencarian Awal: Jalankan query awal dengan parameter sort untuk menentukan urutan hasil. Responnya akan berisi dokumen halaman pertama dan sort key.
2. Gunakan Sort Key: Pada permintaan selanjutnya, gunakan search_after parameter dengan nilai sort key dari respon sebelumnya. Ini memberi tahu Elasticsearch untuk mengambil halaman berikutnya berdasarkan urutan yang sama.
3. Ulangi: Terus ulangi langkah 2 untuk mengambil halaman berikutnya.

Keuntungan:

- Efisien: Menghemat sumber daya dengan menghindari pengulangan query.
- Skalabilitas: Cocok untuk kumpulan data besar.
- Fleksibel: Kontrol ukuran halaman dengan size parameter.

Contoh:

Misalkan Anda ingin mengambil 10 dokumen pertama pada index categories yang akan diurutkan berdasarkan id secara ascending.

1. Pencarian Pertama

```
GET http://localhost:9200/categories/_search
Content-Type: application/json

{
    "query": {
        "match_all": {}
    },
    "size": 10,
    "sort": [
        {
            "id": {
                "order": "asc"
            }
        }
    ]
}
```

2. Halaman Selanjutnya

```
GET http://localhost:9200/categories/_search
Content-Type: application/json

{
    "query": {
        "match_all": {}
    },
    "size": 10,
    "sort": [
        {
            "id": {
                "order": "asc"
            }
```

```
        }
    ],
    "search_after": [
        10
    ]
}
```

Snapshot

Snapshot adalah fitur penting di elasticsearch yang memungkinkan anda untuk mencadangkan dan memulihkan data cluster anda. Elasticsearch memungkinkan kita untuk melakukan backup pada cloud storage seperti AWS, Google Cloud, S3, dan lainnya, akan tetapi pada pembahasan kita kali ini kita akan melakukan backup pada local atau file system, kalian bisa mempelajari backup lainnya pada link berikut ini https://www.elastic.co/guide/en/elasticsearch/reference/current/snapshot-restore.html. Untuk menggunakan snapshot kita harus mengatur configurasi path repository pada **config/elasticsearch.yml** dan tambahkan **path.repo: <folder-path >** lalu jalankan Kembali elasticsearch.

Create Repository & Snapshot

Sebelum menyimpan snapshot kita harus membuat repository dahulu lalu melakukan snapshot cluster pada folder yang sudah kita atur pada config elasticsearch. Berikut langkah-langkah membuat repository dan snapshot:

1. Membuat repository **first_backup**:

```
PUT http://localhost:9200/_snapshot/first_backup
Content-Type: application/json

{
    "type": "fs",
    "settings": {
        "location": "first_backup"
    }
}
```

Penjelasan:

- type: tipe penyimpanan yang kita gunakan, karena kita menyimpan pada file system maka disini kita menggunakan “fs”.
- location: nama repository yang akan menyimpan snaphot.

2. Cek repository:

```
GET http://localhost:9200/_snapshot
```

3. Membuat snapshot **snapshot1**:

```
PUT http://localhost:9200/_snapshot/first_backup/snapshot1
Content-Type: application/json

{
    "indices": [],
    "metadata": {
        "taken_by": "yusril",
        "taken_because": "backup before update"
    }
```

```
}
```

Penjelasan:

- indices: index apa yang ingin di backup. Jika kosong maka akan meng-backup semuanya.
- taken_by: siapa yang melakukan backup.
- taken_because: alasan melakukan backup.

4. Cek snapshot dalam respository:

```
GET http://localhost:9200/_snapshot/first_backup/snapshot1
```

5. Cek semua repository:

```
GET http://localhost:9200/_cat/snapshots?v
```

6. Backup berhasil.

Restore Repository & Snapshot

Setelah kita melakukan backup repository dan snapshot kita bisa melakukan restore untuk snapshot yang sudah kita simpan. Silahkan lakukan request pada repository dan snapshot lalu tambahkan endpoint _restore seperti berikut ini:

```
POST http://localhost:9200/_snapshot/first_backup/snapshot1/_restore
Content-Type: application/json

{
    "indices": [
        "categories"
    ]
}
```

Penjelasan: indices adalah index apa saja yang ingin di restore pada elasticsearch.

Delete Repository & Snapshot

Ketika kita membuat repository dan snapshot kita tidak bisa menimpa snapshot lama dengan snapshot baru yang berarti kita harus membuat snapshot yang berbeda jika ingin melakukan backup secara rutin, hal ini akan membuat snapshot semakin banyak dan tentu memakan penyimpanan. Untuk itu elasticsearch memungkinkan kita menghapus snapshot dan repository yang mungkin tidak dipakai lagi. Berikut langkah-langkah menghapus snapshot dan repository:

1. Hapus snapshot **snapshot1** pada repository **first_backup**:

```
DELETE http://localhost:9200/_snapshot/first_backup/snapshot1
```

2. Hapus juga repository **first_backup**:

```
DELETE http://localhost:9200/_snapshot/first_backup
```

3. Lalu cek repository, jika tidak ada maka delete berhasil:

```
GET http://localhost:9200/_cat/snapshots?v
```

Reference

Programmer Zaman Now: https://www.youtube.com/watch?v=JfW7tg0yWsc&list=PL-CtdCApEFH8s_TV6u0qJqFQH-szZd9Cq&index=1

Elasticsearch Docs: https://www.elastic.co/guide/index.html