02

Carregando dados no ElasticSearch

Juntando tudo o que aprendemos

Depois de todos os testes que fizemos, finalmente podemos criar a versão final do nosso índice e carregar dados de verdade. Iremos utilizar basicamente o mesmo índice que estudamos no capítulo anterior, porém com uma pequena mas importante adição.

Como vimos anteriormente, quando declaramos um atributo como **analyzed**, gera-se tokens no índice invertido. Sabemos também que os termos durante as buscas são analisados e os tokens gerados são utilizados para encontrar os documentos. Sabemos também que o documento original nos é retornado quando encontrado. Há um pequeno detalhe que veremos em breve, porém, por ora basta sabermos que precisamos também do valor original no índice invertido além do valor analisado. Para tal, usaremos uma técnica/funcionalidade conhecida como **multi-campos**.

Multi-campos

Como parte da definição de um atributo no mapeamento de um índice, basta adicionar os campos extras a serem gerados para um atributo e sua configuração. Veja o exemplo a seguir para o atributo `nome`:

```
"nome": {
    "type": "string",
    "fields": {
        "original": {
            "type": "string",
            "index": "not_analyzed"
        }
    },
    "index": "analyzed",
    "analyzer": "portuguese"
}
```

O ElasticSearch irá criar um campo chamado **`nome.original`** e manterá uma cópia do valor para o atributo `nome`, sem analisá-lo. Vale notar que **original** poderia ser qualquer nome válido de campo. É comum o uso nomes como *raw* ou *plain*.

Criando nosso índice

Para não confundir com os índices anteriores (fique à vontade para removê-los), utilizaremos um nome diferente. Chamaremos nosso índice de **`pessoas`** e nosso tipo de **registros** e faremos uso de tudo o que aprendemos até o momento, inclusive multi-campos. Segue a configuração do nosso índice:

```
PUT /pessoas
{
  "settings": {
    "index": {
      "number_of_shards": 3,
      "number_of_replicas": 0
    },
```

```
    "analysis": {
      "filter": {
         "portuguese_stop": {
          "type":       "stop",
          "stopwords":  "_portuguese_"
        },
        "portuguese_stemmer": {
          "type": "stemmer",
          "language": "light_portuguese"
        },
        "filtro_de_sinonimos": {
          "type": "synonym",
          "synonyms": [
            "futebol => futebol,society",
            "society => society,futebol",
            "volei,voleibol,volleyball",
            "esport => esport,futebol,society,volei,basquet",
            "exat => exat,matematic,fisic,computaca",
            "arte => arte,pintur,teatr,music,cinem"
          ]
        }
      },
      "analyzer": {
        "sinonimos": {
          "tokenizer": "standard",
          "filter": [
            "lowercase",
            "portuguese_stop",
            "portuguese_stemmer",
            "filtro_de_sinonimos"
          ]
        }
      }
    },
    "mappings": {
      "registros": {
        "_all": {
          "type": "string",
          "index": "analyzed",
          "analyzer": "portuguese"
        },
        "properties": {
          "cidade": {
            "type": "string",
            "fields": {
              "original": {
                "type": "string",
                "index": "not_analyzed"
              }
            },
            "index": "analyzed",
            "analyzer": "portuguese"
          },
          "estado": {
            "type": "string",
            "index": "not_analyzed"
          },
```

```
          "formação": {
            "type": "string",
            "fields": {
              "original": {
                "type": "string",
                "index": "not_analyzed"
              }
            },
            "index": "analyzed",
            "analyzer": "portuguese"
          },
          "interesses": {
            "type": "string",
            "index": "analyzed",
            "analyzer": "portuguese",
            "search_analyzer": "sinonimos"
          },
          "nome": {
            "type": "string",
            "fields": {
              "original": {
                "type": "string",
                "index": "not_analyzed"
              }
            },
            "index": "analyzed",
            "analyzer": "portuguese"
          },
          "país": {
            "type": "string",
            "fields": {
              "original": {
                "type": "string",
                "index": "not_analyzed"
              }
            },
            "index": "analyzed",
            "analyzer": "portuguese"
          }
        }
      }
    }
  }
```

Usando a API _bulk

ElasticSearch possui a API **_bulk** para permite carregamento de dados em massa. A API é tão flexível que acaba se tornando um pouco confusa, pois ela permite que um mesmo *request* possua comandos para criar, atualizar e remover documentos em diferentes índices.

Por questão de simplicidade, vamos focar no que precisamos para atingir nossos objetivos. A _bulk API é suportada no método POST e a usaremos da seguinte maneira:

```
POST /indice/tipo/_bulk (1)
{"create": {}} (2)
```

```
{"campo1": "valor1", "campo_n": "valor_n"} (3)
{"create": {}} (4)
{"campo1": "valor2", "campo_n": "valor_n2"} (5)
```

Onde:

- (1) Índice e tipo onde API _bulk será executada
- (2) Ação e metadado. Neste caso, estamos interessados apenas na criação de documentos, logo usamos `create`. Junto à operação, são informados valores como índice, tipo e identificador do registro que vem a seguir. Como informamos índice e tipo, e utilizaremos a geração automática de identificadores, podemos ocultar os demais valores.
- (3) O documento a ser inserido.
- (4) A ação e metadado para o próximo documento da requisição.
- (5) O próximo documento da requisição. Note que (2) e (3) são repetidos para cada documento parte da requisição.

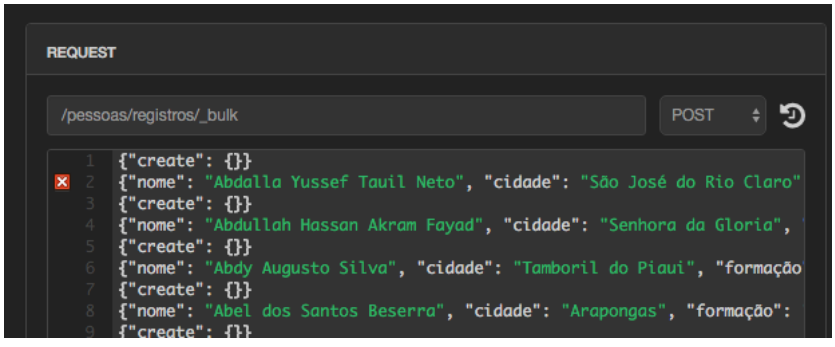
Inserindo muitos dados

Em respeito aos computadores de todos os alunos, iremos inserir em nosso recém criado índice pouco mais de 1100 documentos. Considerações sobre uso de ElasticSearch com volumes maiores são feitos ao final do curso. Continuaremos utilizando o plugin Kopf. Vamos baixar o zip com dois arquivos aqui (https://s3.amazonaws.com/caelum-online-public/elasticsearch/scripts/documentos.zip):

Vamos agora enviar o seguinte *request* para o ElasticSearch:

```
POST /pessoas/registros/_bulk
… conteudo do arquivo 1 …
```

Exemplo:



Importante: Existe uma validação de JSON na interface do Kopf que força que o conteúdo seja um único JSON. Fique tranquilo pois esta validação não afeta em nada a requisição.

Vamos verificar a quantidade de registros do índice:

```
GET /pessoas/registros/_count
{}
```

E confirmar que temos 558 documentos.

Agora, vamos envie o seguinte *request* para o ElasticSearch:

```
POST /pessoas/registros/_bulk
… conteudo do arquivo 2 …
```

E confirmar que temos todos os 1116 documentos no índice.

```
GET /pessoas/registros/_count
{}
```

Podemos verificar a quantidade de pessoas cujo estado é SP:

```
GET /pessoas/registros/_search?q=estado:SP
{}
```

O resultado esperado é de 108 documentos.

Usos mais avançados para a _bulk API

Para mais detalhes sobre a `_bulk` API, acesse o link:

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-bulk.html (https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-bulk.html)

O que aprendemos?

- Multi-campos e como eles nos ajudam a preservar os valores originais de atributos que são indexados.
- `_bulk` API para fazer ingestão de registros em grandes volumes.