04

Introdução ao ElasticSearch

A complexidade da busca

Procurar por registros em meio a grandes volumes de dados é uma tarefa complicada e há muitos anos recebe atenção de diversas áreas de pesquisa. Por décadas, bancos de dados relacionais foram os principais motores por trás de qualquer solução de busca. Diversos bancos de dados oferecem a capacidade de busca cheia em texto (*full-text search*). Habilitar este recurso necessita, em geral, de tarefas administrativas como instalação de *add-ons* trocas de configurações do banco e execuções de *procedures*. Seria ideal uma solução mais simples, flexível e portável, afinal de contas, não queremos contratar um DBA ou nos tornarmos especialistas em um banco de

dados para poder fazer buscas que vão além do famoso e nada eficiente `campo like %valor%`.

Uma outra dificuldade em lidar com buscas está ligada a maneira com que a busca em si é orientada. Tanto em bancos de dados relacionais quanto em bancos de dados não relacionais, conhecidos como *NoSQL*, buscas são orientadas a colunas ou atributos chaves que devem ser indexados previamente. Não queremos também estar atrelados a campos específicos e exigir que nosso usuário precise saber detalhes do nosso sistema para poder extrair a informação que ele necessita. Por exemplo, imagine que queremos encontrar em uma tabela todas as vendas de computadores que ocorreram no estado de São Paulo na última semana. Para termos uma busca eficiente precisamos que boa parte destes campos estejam previamente indexados, caso contrario a busca pode acabar por varrendo a tabela toda. Além disso, caso disponibilizemos

esta busca para usuários de um sistema, somos praticamente obrigados a colocar um formulário de busca. Imagine o 'sucesso' que o Google teria caso oferecesse um formulário de busca, ao invés de um campo livre de busca.

Engine de busca

Felizmente, existem bibliotecas que nos ajudam a ter bastante flexibilidade em nossas buscas. Um exemplo é o Apache Lucene (https://lucene.apache.org/core), que é uma biblioteca em Java que oferece um *search engine* poderoso. Lucene é orientado a documentos e possui funcionalidades como busca exata, busca por similaridade, *highlight* do termo procurado no resultado, busca por termos em documento e busca por frase, além de também suportar analisadores de texto em diversos idiomas. Ainda que o Lucene seja excelente do ponto de vista de *search engine*,

ele possui alguns aspectos que limitam sua adoção. Por exemplo, como o Lucene é escrito em Java, é difícil utilizá-lo diretamente com linguagens de programação como Ruby ou C#. O Lucene foi desenhado para ser executado em uma única JVM. Isso nos limita tanto no volume de dados, afinal toda informação fica armazenada em um único nó, não temos tolerância a falhas e alta disponibilidade, afinal, caso este nó tenha problemas, nosso *search-engine* estaria indisponível.

Dado os volumes de dados e as necessidades de disponibilidade atuais, ter um sistema que depende de uma única máquina (seja virtual ou física) é de longe um grande problema. Estamos interessados no poder de busca oferecido pelo Lucene, porém em um ambiente na nuvem, com alta disponibilidade e escalabilidade horizontal, afinal queremos poder adicionar mais máquinas (baratas) quando for necessário.

ElasticSearch: Lucene e análise de dados na Nuvem

ElasticSearch (https://www.elastic.co/products/elasticsearch) é uma implementação de código aberto disponibilizada sob a licença Apache 2, suportada pela empresa Elastic.co que traz o poder do Lucene para o ambiente da nuvem. ElasticSearch endereça os problemas descritos acima como distribuição de dados e alta disponibilidade, tomando conta de detalhes como replicação de dados e tolerância a falhas de hardware. Com o aumento do poder de processamento que a nuvem nos dá, o ElasticSearch suporta também operações como agregações e buscas geo-espaciais que escalam horizontalmente ao nível de terabytes de dados.

Vale destacar que o ElasticSearch tem sido muito utilizado como ferramenta de análise de dados (*data analytics*), já que ele nos oferece a

combinação de um poderoso *full-text search engine* com *data analytics* em larga escala. Como um exemplo real, imagine que queremos saber os 100 produtos mais vendidos no último ano e os seus valores médios de venda. Este problema é relativamente fácil de resolver em um banco de dados comum. Porém, queremos também limitar nossos resultados a produtos que tiveram bons comentários de clientes. Existem algumas frases (e não palavras) chaves com palavras que devemos utilizar na busca e nos ajuda a definir o que é um comentário positivo ou negativo.

A interação com o ElasticSearch, diferentemente da interação com o Lucene, acontece com o uso do formato JSON através de Restful APIs. Tal abordagem que nos permite ter um *search engine* em Java que interopere com diversas linguagens de programação. ElasticSearch oferece também como opção bibliotecas oficiais em diversas

linguagens, como Javascript, Python, Java, Groovy, Perl, PHP, Ruby e .Net que abstraem os detalhes da interação via protocolo HTTP.

Instalando o ElasticSearch

Antes de apresentarmos nosso projeto, vamos instalar o ElasticSearch localmente para começar a esquentar os motores e ter uma idéia mais prática de como será nossa interação com ele.

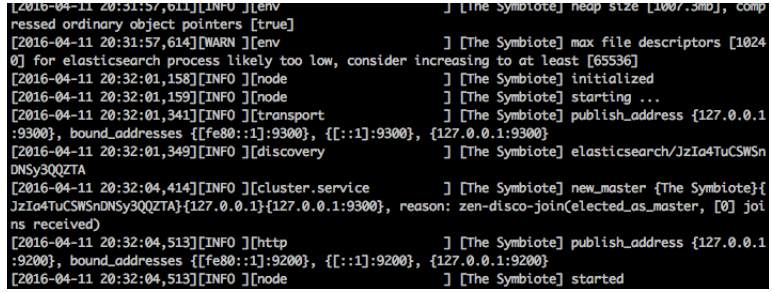
A versão 2.3.1 está disponível aqui (https://s3.amazonaws.com/caelum-online-public/elasticsearch/downloads/elasticsearch-2.3.1.zip). Outras versões podem ser encontradas direto no site (https://www.elastic.co/downloads/elasticsearch) da Elastic.co. A instalação do ElasticSearch é bem simples. Basta descompactar o arquivo compactado em

qualquer pasta que você tenha acesso de escrita.

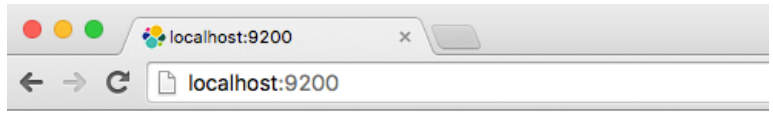
Para iniciar o processo do ElasticSearch com as configurações padrão (não se preocupe, falaremos delas mais a frente) basta executar o comando elasticsearch para sua plataforma que está disponível dentro da pasta bin.

Atenção: A execução do ElasticSearch depende da JVM estar previamente instalada no seu ambiente. Tenha certeza que você tem a versão mais atual da JVM instalada e que o comando *java* está no *path* do seu ambiente. Note também que o ElasticSearch, por padrão, utiliza as portas 9200 e 9300.

A inicialização é bem rápida e devemos ter o seguinte resultado:



Um teste bem rápido que podemos fazer é acessar a URL [http://localhost:9200 (http://localhost:9200)](http://localhost:9200) no nosso browser favorito. Veja o resultado esperado:




```
{
  "name" : "The Symbiote",
  "cluster_name" : "elasticsearch",
  "version" : {
    "number" : "2.3.1",
    "build_hash" : "bd980929010aef404e7cb0843e61d0665269fc39",
    "build_timestamp" : "2016-04-04T12:25:05Z",
    "build_snapshot" : false,
    "lucene_version" : "5.5.0"
  },
  "tagline" : "You Know, for Search"
}
```


Para encerrar o processo, basta utilizar Ctrl+C. Note que o valor para o campo *name* varia. Neste momento podemos ignorar esta

variação.

O plugin Kopf

Como comentado acima, a interação com o ElasticSeach acontece via RestAPI com base nos métodos do protocolo HTTP. Para facilitar nossa interação durante o curso, vamos instalar um plugin chamado Kopf que, dentre várias funcionalidades, nos proporciona também um REST client bem prático.

Atenção: Caso você possua comandos como *curl* ou algum cliente REST favorito, fique a vontade para utilizá-los durante o curso. Contudo, para este capítulo é fortemente recomendado que seja usado o kopf por motivos que ficarão claros a seguir.

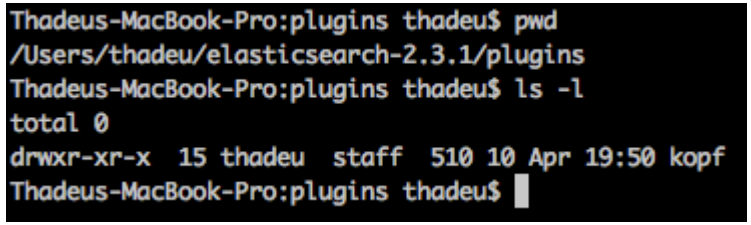
A instalção de plugins no ElasticSearch é bem simples. Basta, na pasta bin, utilizar o comando *plugin* para sua plataforma, seguido

do nome do plugin e versão. O example abaixo mostra como instalar o plugin *kopf* na sua versão 2.1.2 direto do repositório git lmenezes/elasticsearch-kopf.

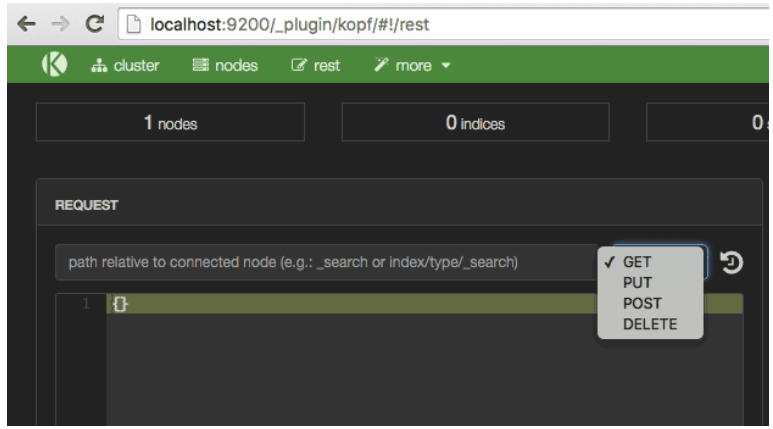
```
./plugin install lmenezes/elastics
```

COPIAR CÓDIGO

O arquivo vai ser baixado direto de um dos repositórios disponíveis e descompactado na pasta plugins. Alternativamente, podemos baixar o kopf aqui (https://s3.amazonaws.com/caelum-online-public/elasticsearch/downloads/elasticsearch-kopf-2.1.2.zip) e descompactá-lo diretamente na pasta plugins como mostrado na figura abaixo:



Após reiniciamos o ElasticSearch, podemos acessar a URL [http://localhost:9200/_plugin/kopf/#!/rest (http://localhost:9200/_plugin/kopf/#!/rest)](http://localhost:9200/_plugin/kopf/#!/rest) e teremos o seguinte resultado:



Repare que podemos alterar qual verbo do protocolo HTTP uma determinada requisição deve utilizar.

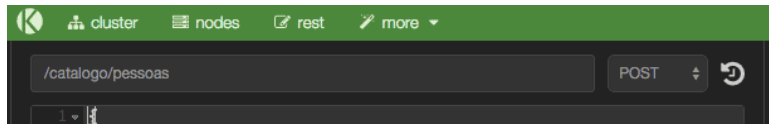
Primeiras interações com ElasticSearch

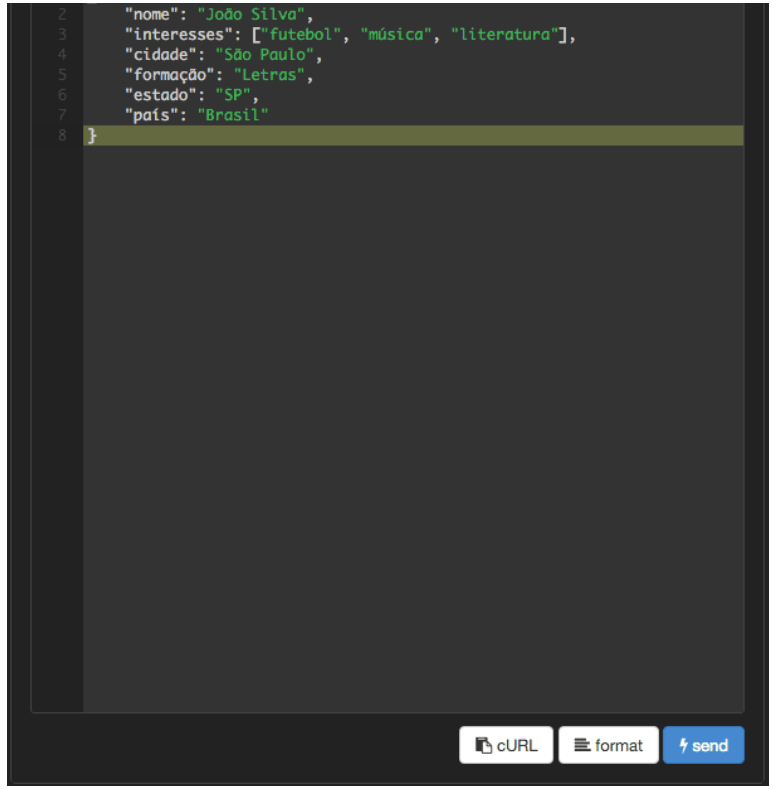
Vamos criar alguns documentos no ElasticSearch e fazer algumas buscas bem

básicas para entender os padrões básicos de interação com o ElasticSearch. Neste exemplo utilizaremos o índice *catalogo* e o tipo *pessoas*. Por hora, basta acreditarmos que índice é o equivalente a base de dados e tipo é o equivalente a uma tabela. Vamos executar inicialmente o seguinte comando via kopf como mostrado na imagem a seguir. Note que o botão de *send* fica na parte de baixo da página:

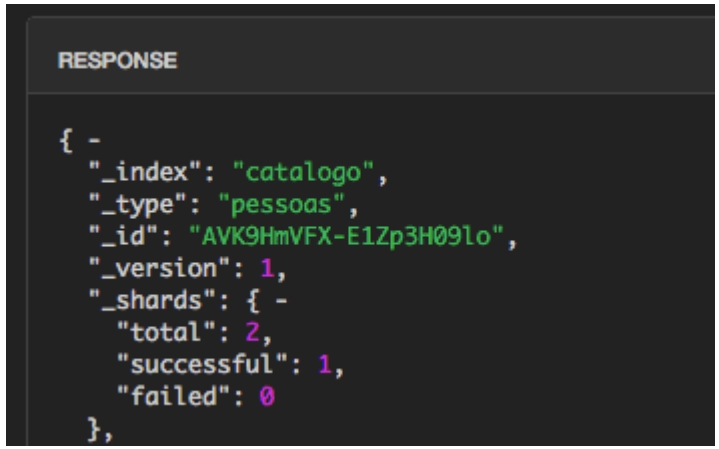
```
POST /catalogo/pessoas
{
    "nome" : "João Silva",
    "interesses" : ["futebol", "mú
    "cidade" : "São Paulo",
    "formação" : "Letras",
    "estado" : "SP",
    "país" : "Brasil"
}
```

COPIAR CÓDIGO





O resultado esperado é mostrado a seguir:



A resposta nos diz, dentre outras informações, que o documento foi criado com sucesso. Veja que o identificador (atributo _id) deste documento foi gerado para nós. Caso tivéssemos utilizado a URI /catalogo/pessoas/1 e o verbo PUT, o campo *_id* possuiria o valor 1.

**Atenção: Conforme começamos a adicionar documentos, o cabeçalho do kopf deve mudar de verde para a cor bege. Isso acontece devido ao fato de termos apenas um host para o ElasticSearch e um índice que requer uma réplica. Discutiremos este problema durante o curso. Por hora, basta mudarmos o numero de replica para zero conforme mostrado a seguir:*

```
PUT /catalogo/_settings
{
```

```
    "index" : {
        "number_of_replicas" : 0
    }
}
```

COPIAR CÓDIGO

Para verificar quanto documentos existem em um tipo dentro de um índice, utilizamos o seguinte comando:

```
GET /<indice>/<tipo>/_count
```

COPIAR CÓDIGO

Para o índice que acabamos de criar, utilizamos o comando a seguir:

```
GET /catalogo/pessoas/_count
{}
```

COPIAR CÓDIGO

Como esperado, o resultado recebido é 1:

```
{
    "count": 1,
    "_shards": {
        "total": 5,
        "successful": 5,
        "failed": 0
    }
}
```

COPIAR CÓDIGO

Para localizar um documento pelo seu identificador, basta utilizar o seguinte comando:

```
GET /<indice>/<tipo>/<identificat
{}
```

COPIAR CÓDIGO

Por exemplo:

```
GET /catalogo/pessoas/1
```

```
{}
```

COPIAR CÓDIGO

E o resultado, como esperado:

```
{
    "_index": "catalogo",
    "_type": "pessoas",
    "_id": "1",
    "_version": 1,
    "found": true,
    "_source": {
        "nome": "João Silva",
        "interesses": [
            "futebol",
            "música",
            "literatura"
        ],
        "cidade": "São Paulo",
        "formação": "Letras",
        "estado": "SP",
        "país": "Brasil"
    }
```

```
}
```

COPIAR CÓDIGO

Por fim, para retornar todos os documentos em de um tipo sem aplicar filtro algum, basta utilizar o comando:

```
GET /<indice>/<tipo>/_search
```

COPIAR CÓDIGO

Por exemplo:

```
GET /catalogo/pessoas/_search
{}
```

COPIAR CÓDIGO

E o resultado:

```
{
```

```
"took": 11,
"timed_out": false,
"_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
},
"hits": {
    "total": 2,
    "max_score": 1,
    "hits": [
        {
            "_index": "catalog
            "_type": "pessoas"
            "_id": "AVK9Zg4VX-
            "_score": 1,
            "_source": {
                "nome": "João
                "interesses":
                    "futebol",
                    "música",
                    "literatu
                ],
                "cidade": "São
                "formação": "L
```

```
                "estado": "SP"
                "país": "Brasi
            }
        },
        {
            "_index": "catalog
            "_type": "pessoas"
            "_id": "1",
            "_score": 1,
            "_source": {
                "nome": "João
                "interesses":
                    "futebol",
                    "música",
                    "literatu
                ],
                "cidade": "São
                "formação": "l
                "estado": "SP"
                "país": "Brasi
            }
        }
    ]
  }
}
```

Podemos ainda utilizar o parametro *q* para passar uma valor de filtro para o ElasticSearch:

COPIAR CÓDIGO

```
GET /catalogo/pessoas/_search?q=fu
{}
```

COPIAR CÓDIGO

O que aprendemos?

- O que é e para que serve o ElasticSearch.
- Como fazer uma instalação básica do ElasticSeach a ser usada em uma única máquina.
- Como instalar plugins no ElasticSearch.
- Como utilizar um cliente REST para criar e localizar documentos por identificador, listar todos os documentos ou aplicar um valor simples para consulta.
- Como alterar o número de réplicas de um