2 - Getting Started

Sunday, 2 February, 2025 11:29 Manhã

Interação com REST direto ou Kibana (mais amigável)

```
# Create an index
PUT books/_doc/1
{
  "title":"Effective Java",
  "author":"Joshua Bloch",
  "release_date":"2001-06-01",
  "amazon_rating":4.7,
  "best_seller":true,
  "prices": {
    "usd":9.95,
    "gbp":7.95,
    "eur":8.95
  }
}
```

Exemplo: livros

Formato cURL:

Versões anteriores incluíam campo de "type" depois do index_name,
mas não mais desde 7.x. Agora só um único tipo por index.

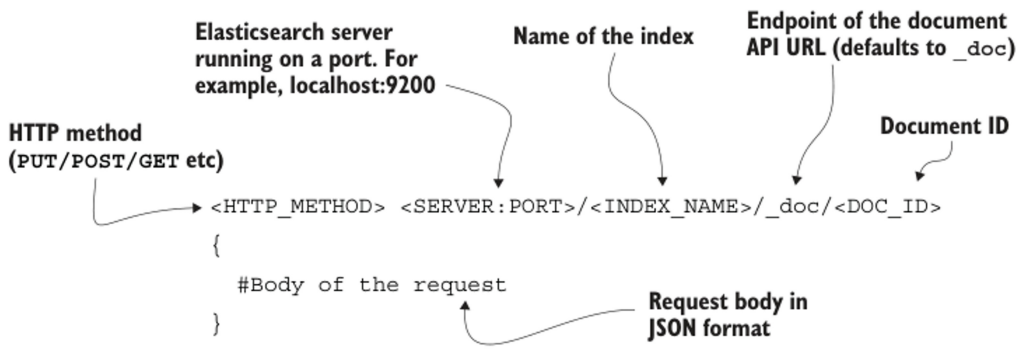


Figure 2.2 Elasticsearch URL invocation endpoint using an HTTP method

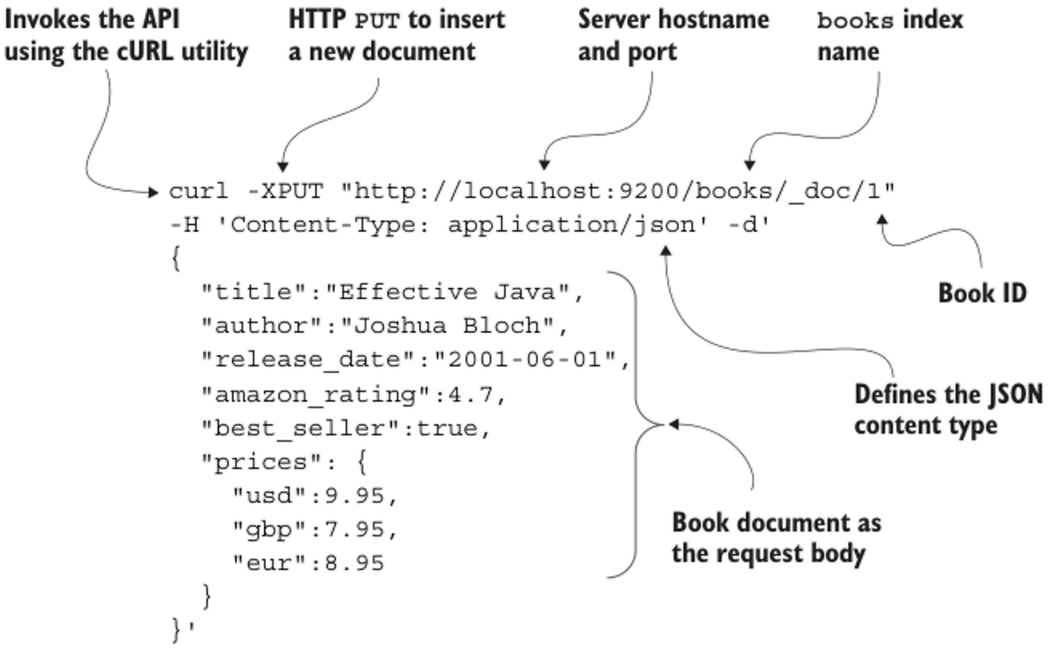


Figure 2.3 Elasticsearch URL invocation endpoint using cURL

Kibana permite pular o boilerplate da requisição, usar só o corpo.
Devtools no console permite testasr comandos.
HTTP code da resposta diz se indexação funcionou certo (200 = OK)

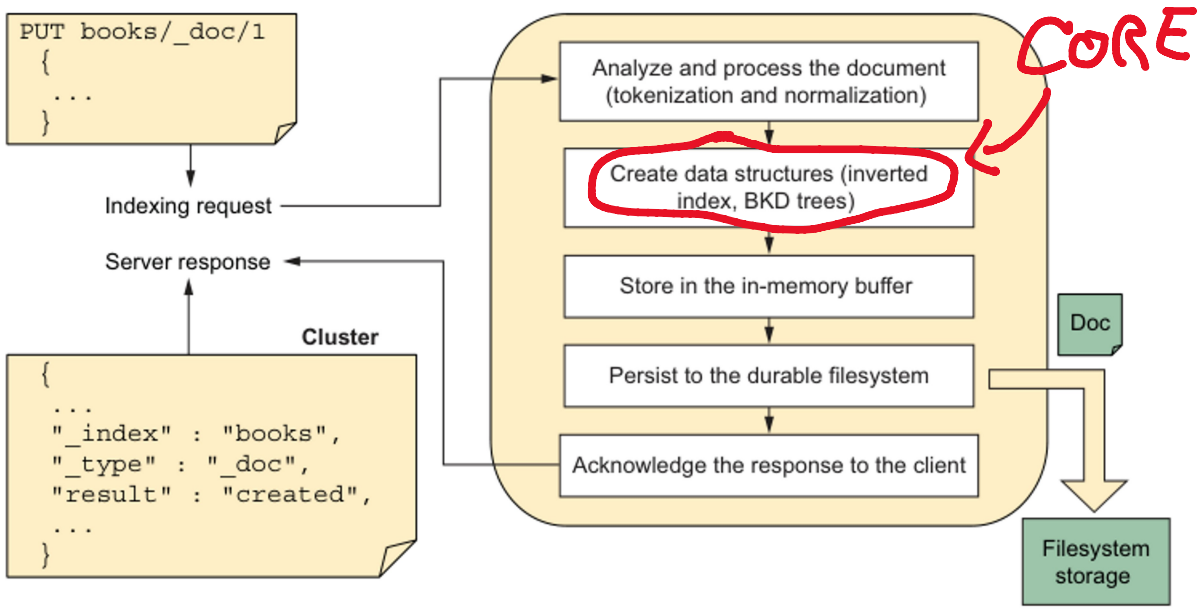


Figure 2.8 Elasticsearch's request and response flow at a high level

Indexar um documento é equivalente a inserir uma entrada num banco de dados relacional.

Não há definição de schema a priori. Schema é derivado a partir da primeira entrada.

Contagem:

```
GET books/_count
GET books1,books2/_count   (múltiplos índices de uma vez)
GET _count (todos)
```

Get:

Também pode ser um, vários ou todos

```
  GET books/_doc/1
```

```
GET books/_search
{
"query": {
"ids": {
"values": [1,2,3,4]
         }
    }
}
```

```
GET books/_search (todos)
```

(é igual a:)

GET books/_search
{
"query": {
"match_all": { }
}
}

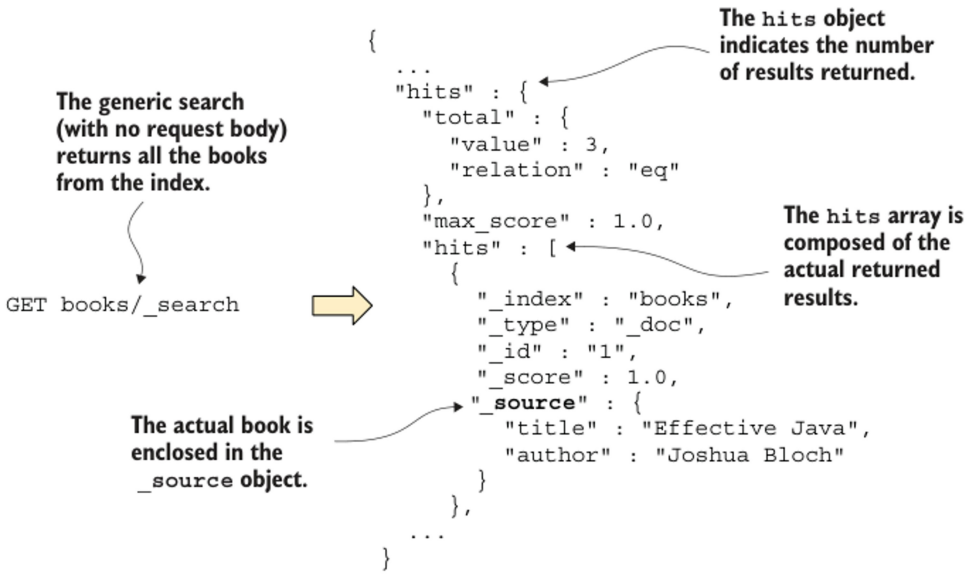


Figure 2.12 Retrieving all documents using the search API

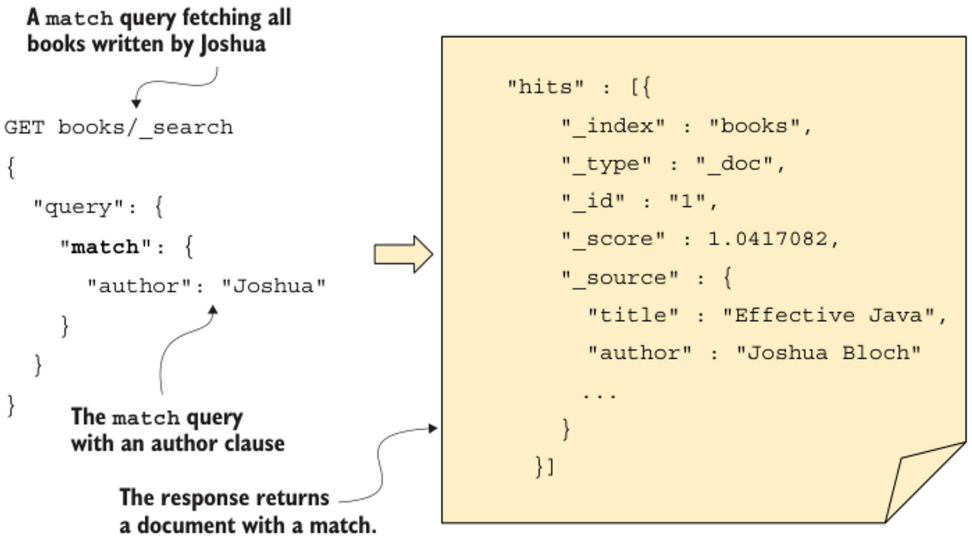


Figure 2.13 Fetching books authored by Joshua

```
"author":"JoShUa"
"author":"joshua" (or "JOSHUA")
"author":"Bloch"
```

All these queries will succeed, but searching for a shortened name will fail:

```
"author":"josh"
```

"prefix" no lugar do "match" funciona também, pega pelo início.

Se tiver mais de uma palavra, presume-se "OR"
Posso definir como "AND" manualmente na query para match exato.

"_bulk" serve para indexar vários documentos de uma vez.
Sobreescreve.

Texto:

"multi_match" para procurar em vários campos ao mesmo tempo.
Posso usar "boost" para priorizar matches em um campo em vez do outro. Sintaxe "^3" ou "^2"

```
GET books/_search
{
  "query": {
    "multi_match": {
      "query": "Java",
      "fields": ["title^3","synopsis"]
    }
  }
}
```

Searches through multiple fields

A caret followed by the boost number indicates the field being boosted.

"match_phrase" procura por frase inteira, exatamente. "slop" pode tornar busca mais flexível, número inteiro de palavras faltantes / desordenadas.
Parâmetro **"highlight"** pode destacar match encontrado com tags .
"fuzziness" é um número inteiro que permite matches a esta distância de Levenshtein (e.g. trocas de caracteres)

Termo:

Busca em dados estruturados.
Data types são inferidos pela primeira indexação. Há campos de não-texto.
Resultados são binários, há match ou não, sem fuzziness ou operações textuais, logo não há score de relevância (1.0 ou 0).

Listing 2.16 Fetching third edition books

```
GET books/_search
{
  "_source": ["title","edition"],
  "query": {
    "term": {
      "edition": {
        "value": 3
      }
    }
  }
}
```

We return just two fields in the response document.

Declares the query as a term-level query

Provides the field and value as search criteria

Range:

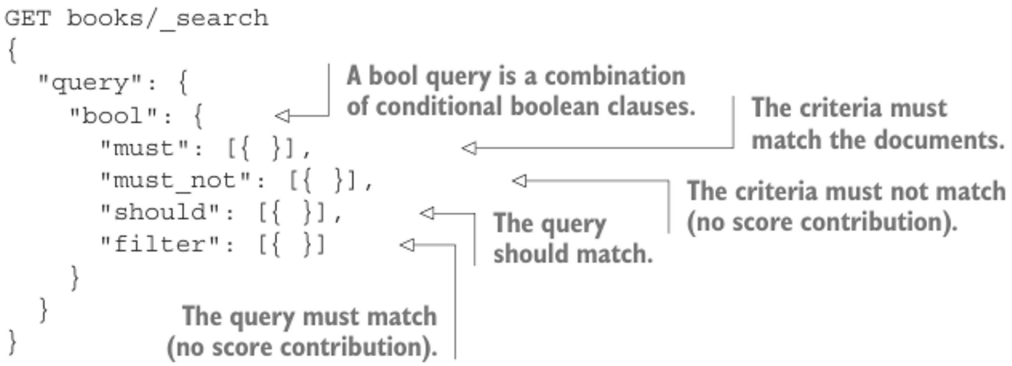
Igual term query mas permite ranges de valores, greater than, lower than, etc.

Compostas:

Queries com várias requisições de match simultaneamente (cada uma é uma "leaf query").
Vários tipos, uma delas e a mais comum é a *bool:*

- Boolean (`bool`)
- Constant score (`constant_score`)
- Function score (`function_score`)
- Boosting (`boosting`)
- Disjunction max (`dis_max`)

Listing 2.18 Format of a `bool` query with the expected clauses



MUST e **MUST_NOT** são obrigatórios para o match ser retornado.
Desses dois apenas **MUST** altera o score.
SHOULD apenas melhora o score, ajusta relevância sem alterar o conjunto retornado.
FILTER remove os não-matchs, não altera scores (é o que difere do MUST).

```
GET books/_search
{
  "query": {
    "bool": {
      "must": [{"match": {"author": "Joshua"}}],
      "must_not":[{"range":{"amazon_rating":{"lt":4.7}}}],
      "should": [{"match": {"tags": "Software"}}],
      "filter":[
        {"range":{"release_date":{"gte": "2015-01-01"}}},
        {"term": {"edition": 3}}
      ]}
  }
}
```

term query in the filter clause

Agregações:

Estatísticas, recortes, resultados de outras agregações.

- *max*
- *min*
- *sum*
- *avg*
- *all_stats*
- *extended_stats, etc...*

Objeto *"aggs"*

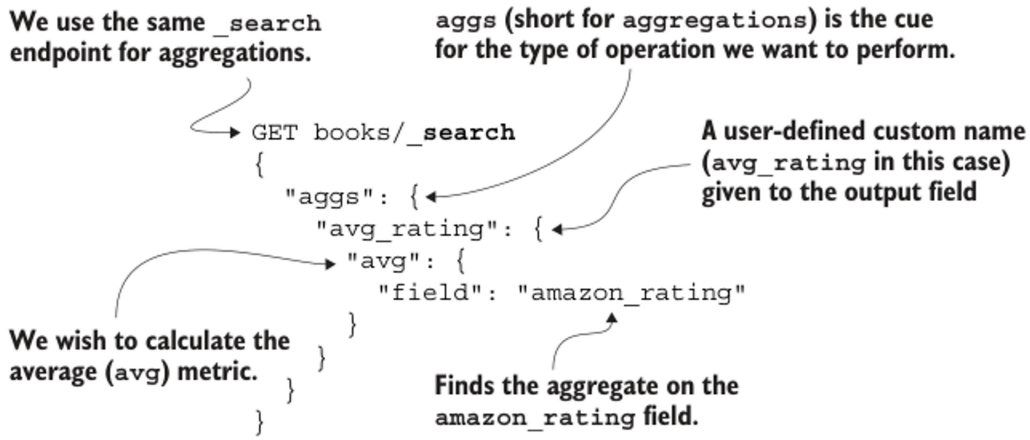


Figure 2.16 Query DSL syntax for finding an average rating aggregation

Operação retorna todos os documentos analisados, mas pode ser suprimido com size=0.

Bucket_aggregation gera estatísticas para grupos (e.g histograma, i.e. contagem por bin)
Para definir bucket por bucket em vez de apenas um intervalo fixo, é possível realizar "*range bucketing"*.

" * " para definir intervalo iniciando/finalizando em infinito/-infinito

Summary

- Elasticsearch provides a set of document APIs for indexing data, and Kibana's Dev Tools console helps with writing indexing queries for persistence.
- To retrieve a document using a single document API, we issue a `GET` command on the index with an ID (`GET <index_name>/_doc/<ID>`).

- To retrieve multiple documents, if the document identifiers are available, we can use an `ids` query.
- Elasticsearch exposes a wide set of search APIs, including basic and advanced queries.
- Full-text queries search through unstructured data to find relevant documents.
- Term queries search through structured data like numbers and dates to find documents that match.
- Compound queries allow us to compile leaf queries and create a more advanced set of queries. The `bool` query, one of the compound queries, provides a mechanism to create an advanced query with multiple clauses (for example, `must`, `must_not`, `should`, and `filter`). These clauses help us create sophisticated queries.
- Whereas a search looks for matching documents based on given criteria, analytics lets us aggregate data by providing statistical functions.
- Metric aggregations fetch common aggregations like `max`, `min`, `sum`, and `avg`.
- Bucketing aggregations segregate documents into various groups (buckets) based on certain criteria.