5 - Working with Documents

Sunday, 2 February, 2025 11:29 Manhã

- Operações single document e multi document tipo CRUD

INDEXAÇÃO

- Indexação pode ser feita com ou sem Ids
  - HTTP PUT para doc com ID provido pelo cliente na indexação
  - HTTP POST para doc ter ID autogerado pelo sistema

Listing 5.1 Indexing a new document into the `movies` index

```
PUT movies/_doc/1
{
  "title":"The Godfather",
  "synopsis":"The aging patriarch of an organized crime dynasty transfers
  ➥ control of his clandestine empire to his reluctant son"
}
```

Body of the request

Document indexing URL

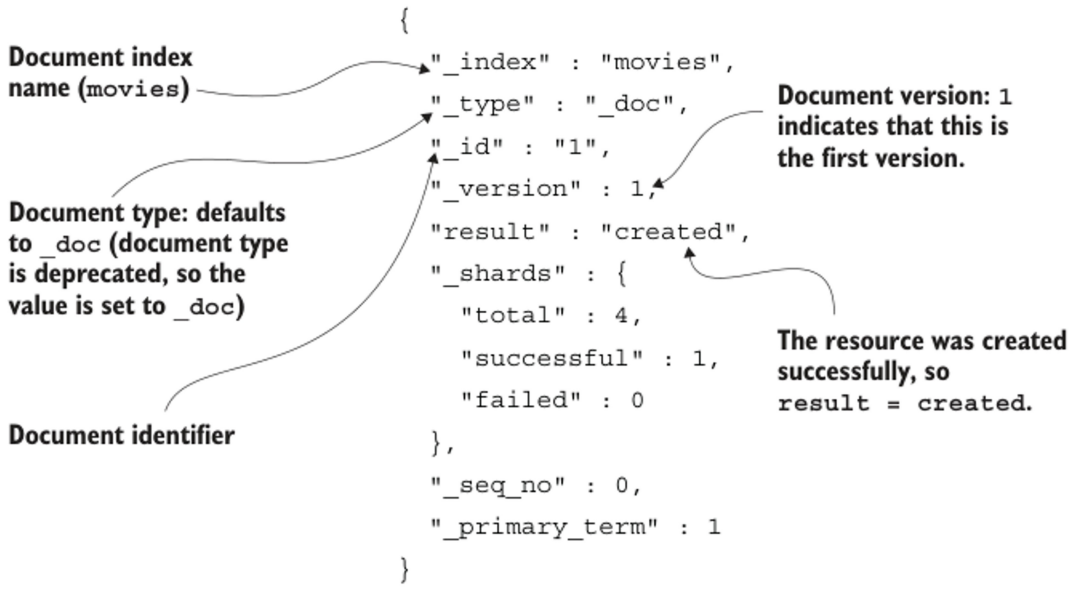


Figure 5.2 Response from the server when a document is indexed successfully

Listing 5.2 Indexing a document with no ID using `POST`

```
POST myindex/_doc
{
  "title":"Elasticsearch in Action"
}
```

The URL has no ID attached to it.

The request body is a JSON document.

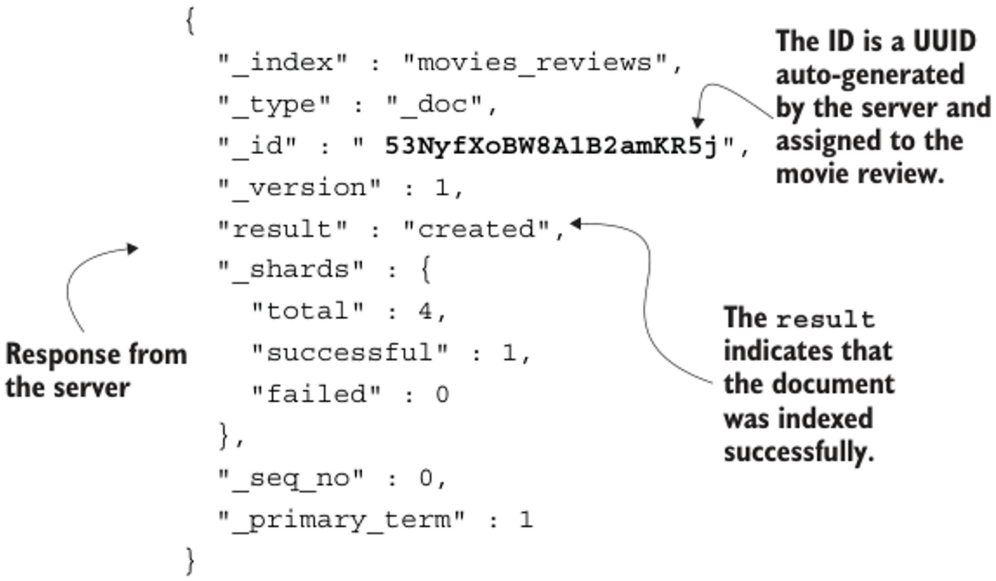


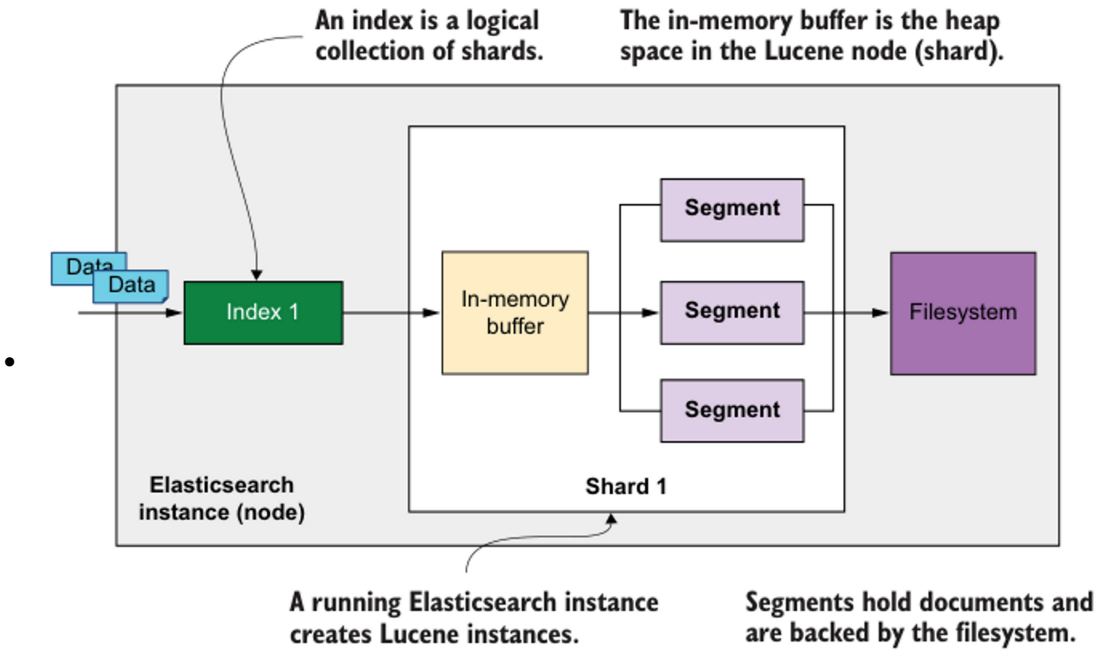
Figure 5.3 The server creates and assigns a randomly auto-generated ID to the document.

- Por default, operação de indexação ocorre igualmente caso o documento já exista na base ou não, sobreescreve.
  - API *_create* evita isso, não sobreescreve.
    - Dá erro caso id já exista na base

Listing 5.5 Indexing a new movie using the `_create` endpoint

```
PUT movies/_create/100
{
  "title":"Mission: Impossible",
  "director":"Brian De Palma"
}
```

- Criação dinâmica de index não pré-declarado pode ser desativada nas configurações do cluster
  - dá erro caso não exista mapping de index.



A running Elasticsearch instance creates Lucene instances.

Segments hold documents and are backed by the filesystem.

Figure 5.4 Mechanics of indexing documents

- Documento antes vai pra heap do shard, e a cada segundo tem um refresh para criar os segmentos com os documentos e respectivos índices invertidos.
    - Depois vai para o disco físico
    - Nos segmentos documentos já são buscáveis
    - Cada três segmentos são mergeados em um único maior, recursivamente.
        - Até aos poucos salvar tudo
        - Caso algo seja deletado, o segmento é apenas marcado como deletado, e depois aos poucos deletado de fato.
        - It's like writing notes on sticky pads, grouping three pads into a notebook, then three notebooks into a binder—so searching through them stays organized and quick!

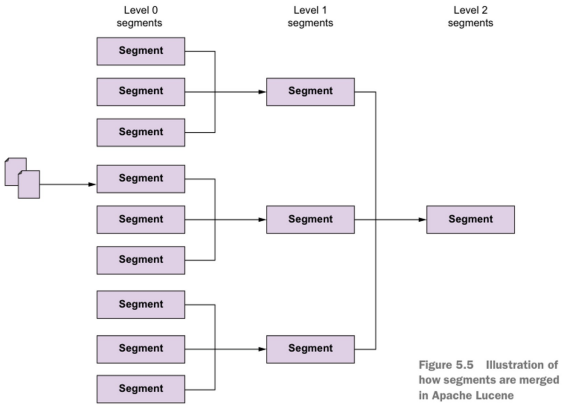


Figure 5.5 Illustration of how segments are merged in Apache Lucene

- Falha no servidor no intervalo entre refreshes pode causar perda de dados.
    - Mas refreshes mais frequentes são mais computacionalmente caros
        - I/O é lento.
- Elastic Search é *eventually consistent:*
    - Documentos são escritos no data store em algum momento.
- Intervalo de refresh é configurável por index - default é 1 segundo.
    - Refresh também pode ser feito manualmente com API *_refresh*.
    - Podemos pausar refresh completamente para por exemplo esperar migrar vários documentos de fora e torná-los buscáveis todos de uma vez, quando reativar refresh.
    - Tempo de refresh também pode ser alterado na própria operação de indexar um documento
        - "PUT movies/_doc/1?refresh=true"
            - □ TRUE - docs imediatamente buscáveis
            - □ FALSE(default)
            - □ WAIT_FOR - indexação só ocorre no próximo ciclo de refresh

BUSCA

- Busca pode ser de um documento ou vários de uma vez.

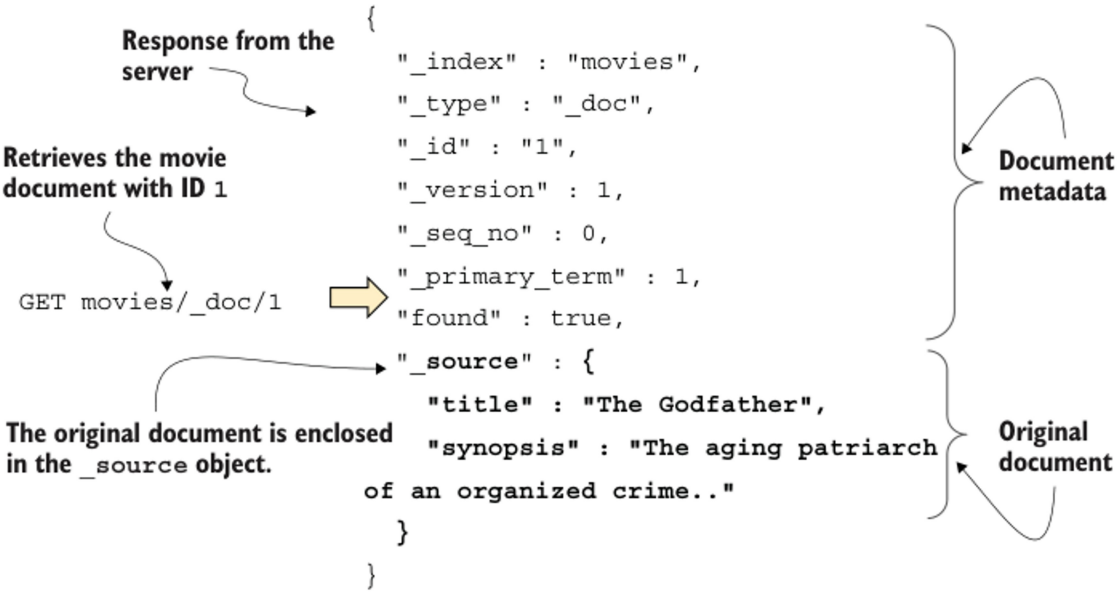


Figure 5.6 Retrieving a document using the GET API call

- Podemos apenas checar a existência de um documento:
  - `HEAD movies/_doc/1`

- Para buscar múltiplos documentos, podemos pegar:
  - De um único índice
  - De vários índices

Listing 5.9 Fetching multiple documents at once

```
GET movies/_mget
{
  "ids" : ["1", "12", "19", "34"]
}
```

Listing 5.10 Fetching documents from three different indexes

```
GET _mget                          <- _mget call with no index mentioned in the URL
{
  "docs":[
    {
      "_index":"classic_movies",   <- The first index is provided here.
      "_id":11
    },

    {
      "_index":"international_movies",   <- Index 2
      "_id":22
    },
    {
      "_index":"top100_movies",          <- Index 3
      "_id":33
    }
  ]
}
```

- Neste último, não tem como passar um array de Ids

- Também podemos buscar vários documentos usando a api *_search*

Listing 5.11 Using an `ids` query to fetch multiple documents

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

MANIPULATING RESPONSES

- Podemos querer omitir algumas das informações na resposta
  - Segurança
  - Eficiência
- Por exemplo, buscamos direto o atributo *_source* em vez de *_doc*.
  - Não vem metadados
- Para fazer o oposto, suprimir o documento e pegar só o metadado:
  - `GET movies/_doc/1?_source=false`
- Para customizar ainda mais, podemos incluir ou excluir campos específicos com os atributos *_source_includes* e *_source_excludes* na chamada
  - `GET movies/_doc/3?_source_includes=title,rating,genre`
  - `GET movies/_source/3?_source_excludes=actors,synopsis`
  - Também podemos incluir e excluir ao mesmo tempo:
    - `GET movies/_source/13?_source_includes=rating*&_source_excludes=rating_amazon`

UPDATING DOCUMENTS

- API _update atualiza um único documento
- API *_update_by_query* permite atualizar vários ao mesmo tempo
- Por trás, esta operação é uma substituição, consiste das três etapas:
  - Busca
  - Modificação
  - Reindexação
- *_update* pode tanto modificar o que já existe quanto adicionar campos novos
- Podemos usar scripts para facilitar e granularizar a atualização
  - E.g. um script que apenas adiciona um item a um array, sem precisar reescrever o array inteiro.
  - **Listing 5.23 Adding an actor to the `actors` list via a script**

```
POST movies/_update/1
{
  "script": {
    "source": "ctx._source.actors.add('Diane Keaton')"    <-- Additional actor
  }
}
```

- Exemplos de script, que podem coexistir na mesma requisição:
  - Adicionar um ator a um array

- Remove um ator de um array
- Adicionar um novo campo a um documento
- Remover um campo de um documento
  - Remover um campo inexistente não retorna erro
- Adicionar múltiplos campos

- Atualização pode ser condicional:

Listing 5.28 Conditionally updating the document using an if/else block

```
POST movies/_update/1
{
  "script": {
    "source": """
    if(ctx._source.boxoffice_gross_in_millions > 125)
      {ctx._source.blockbuster = true}
     else
      {ctx._source.blockbuster = false}
    """
  }
}
```

- Um script tem três partes:

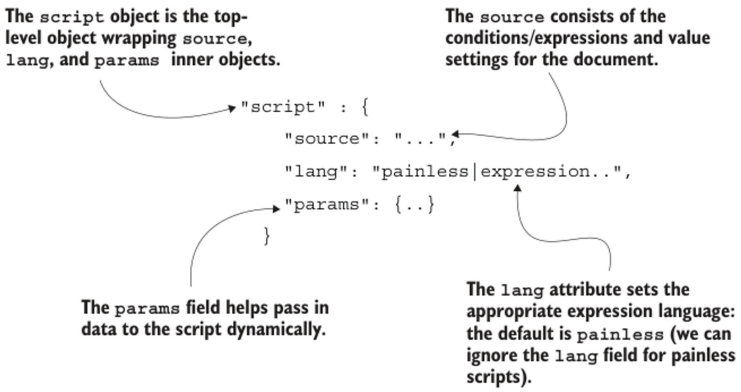


Figure 5.13 The anatomy of a script

  - Source
    - Contém a lógica
  - Params
  - Language
    - Linguagem do próprio script
      - Painelss(default), expression, moustache, ou java

- Podemos passar variáveis ao script sem deixá-la hardcoded

Listing 5.29 Dynamically passing a parameter to the script

```
POST movies/_update/1
{
  "script": {
    "source": """
 if(ctx._source.boxoffice_gross_in_millions >
    params.gross_earnings_threshold)
   {ctx._source.blockbuster = true}
 else
   {ctx._source.blockbuster = false}
 """,
    "params": {
      "gross_earnings_threshold":150
    }
  }
}
```

Business logic goes here.

Checks the value against params

Provides the parameter values

- Scripts são compilados quando rodam pela primeira vez
  - Computacionalmente caro
  - Com a parametrização, evitamos compilar várias vezes para cada valor hard-coded

- Para substituir um documento, bastar usar um PUT no ID do documento que eu quero substituir, como se ele nem existisse antes. É sobrescrito.
  -
```
PUT movies/_doc/1
{
  "title":"Avatar"
}
```

- Para modificar ou inserir um documento, usamos a operação *Upsert*.
  - Insere se não existe, atualiza (com um script, por exemplo) se existe.
  -
Listing 5.31 Upsert block example

```
POST movies/_update/5
{
  "script": {
    "source": "ctx._source.gross_earnings = '357.1m'"
  },
  "upsert": {
    "title":"Top Gun",
    "gross_earnings":"357.5m"
  }
}
```

  - Script só será executado se documento já existir, no exemplo acima.
    - Primeiro entraria como 357.7m, e na execução seguinte iria para 357.1m

- Para fazer upsert num campo só, usamos flag *doc_as_upsert*:
  -
Listing 5.33 Updating the document if the field doesn't exist

```
POST movies/_update/11
{
  "doc": {
    "runtime_in_minutes":110
  },
  "doc_as_upsert":true
}
```

  - Com a flag, se não houver doc com ID 11, não há erro.

- Também podemos atualizar os documentos a partir de uma query, sem saber os Ids previamente:
  - API *_update_by_query*
  -
Listing 5.34 Updating a document using a `query` method

```
POST movies/_update_by_query          <- Searches for documents and updates the set
{
  "query": {                          <- Search query: searches for all movies with Al Pacino
    "match": {
      "actors": "Al Pacino"
    }
  },
  "script": {                         <- Applies the following script logic to the matching documents
    "source": """
    ctx._source.actors.add('Oscar Winner Al Pacino');
    ctx._source.actors.remove(ctx._source.actors.indexOf('Al Pacino'))
    """,
    "lang": "painless"
  }
}
```

  - Essa função tem várias etapas por trás dos panos:
    - Busca os documentos
    - Recheca se a versão do documento é a mesma na busca acima e quando entra a operação de update.

        - Se não for, faz retry
    - Quando há erro, só o documento gerador de erro não é atualizado, os outros são.

DELETING DOCUMENTS

- Podemos fazer deleção por ID (um por vez) ou por query
- **DELETAR É IRREVERSÍVEL!**
    - `DELETE movies/_doc/1`
    - Dá erro se documento não existir
- Por query, usamos a API *_delete_by_query*
    - **Listing 5.36 Deleting all movies based on a criterion**

```
POST movies/_delete_by_query
{
  "query":{
    "match":{
      "director":"James Cameron"
    }
  }
}
```

    - Tão flexível quanto uma busca em si
        - Por ranges de algum campo
        - Com vários must e must not
        - Etc.
- Para deletar todos os documentos:
    - **Listing 5.39 Deleting all documents at once**

```
POST movies/_delete_by_query
{
  "query": {
    "match_all": {}
  }
}
```

    - Funciona igualmente para vários index:
        - **POST <index_1>,<index_2>,<index_3>/_delete_by_query**
    - Para listar todos os índex:
        - **GET _cat/indices**

WORKING WITH DOCUMENTS IN BULK

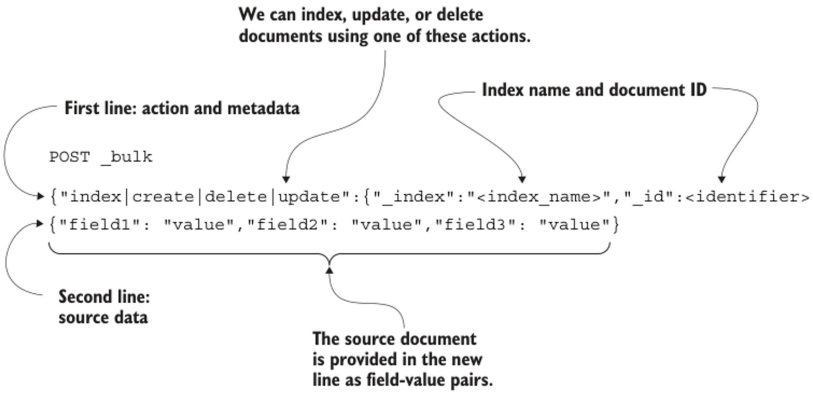
- API _bulk usado para indexar documentos em escala.
    - Podemos manipular e deletar com bulk também
    - Poupa bandwith do que fazer documento a documento
    - 

Figure 5.15 The _bulk API's generic format

    - Cada documento é definido numa linha.
        - NDJSON.
    - Podemos pré-definir o ID ou deixar o sistema autogerar.

    - Index pode estar incluído no URL em vez do corpo da requisição.
- Não precisam ser documentos do mesmo tipo
- Podemos usar create em vez de index para não substituir nada.
- Update e delete também funcionam analogamente
- Para requisição curl, passamos o json com os documentos como binário usando @

REINDEXING DOCUMENTS

- API *_reindex*
- Mover documentos para outro índex
    - Por exemplo, para alterar mapping schema

Listing 5.50 Migrating data between indexes using the `_reindex` API

- ```
  POST _reindex
  {
    "source": {"index": "movies"},
    "dest": {"index": "movies_new"}
  }
  ```
- Unido a aliases, é uma ferramenta importante para fazer migrações com zero downtime

Summary

- Elasticsearch provides a set of APIs that work on documents. We can use these APIs to execute CRUD actions (create, read, update, and delete) on individual documents.
- Our documents are held in the shard's in-memory buffer and pushed into a segment during the refresh process. Lucene employs a strategy of creating a new segment with the documents during the refresh. It then cumulatively merges three segments to form a new segment, and the process repeats.
  ```

- Documents with identifiers (IDs) use the HTTP `PUT` action when indexing (for example, `PUT <index>/_doc/<ID>`), whereas documents with no IDs invoke the `POST` method.
- Elasticsearch generates random unique identifiers (UUIDs) and assigns them to documents during the indexing process.
- To avoid overriding a document, we can issue the following commands:
  - `_create`—This API throws an error if the document already exists.
  - `_mget`—This API lets us retrieve multiple documents at once, given their IDs.
  - `_bulk`—This API performs document operations such as indexing, deleting, and updating multiple documents in one invocation call.
- We can tweak the source and the metadata retrieved as a result of query invocations and then customize the returned document source to include and/or exclude fields by setting the `_source_includes` and `_source_excludes` properties, respectively.
- The `_update` API lets us modify an existing document by updating and adding fields. The expected updates are wrapped in a `doc` object and passed as the request body.
- Multiple documents can be modified by constructing an `_update_by_query` query.
- Scripted updates allow us to modify documents based on conditions. If the conditional clause mentioned in the request body is evaluated to `true`, the script is put into action.
- Documents can be deleted using HTTP `DELETE` for a single document or by running a `_delete_by_query` method on multiple documents.
- Migrating data between indexes is performed by the reindexing API. The `_reindex` API call expects to be given the source and destination indexes to transfer data.