7 - Text Analysis

Monday, 17 March, 2025 22:09 Tarde

- Quebrar texto em tokens e comparar com tokens existentes
- Ideal é ser mais inteligente do que só fazer um match exato.

- Query de dado não-estruturado é mais complicado do que retornar sim ou não pra um documento.
- Tanto query quanto documentos indexados são analisados pelos mesmos analyzers.
- Ideal é que engine entenda abreviações, erros de ortografia, sinônimos, emojis, etc.
    - ElasticSearch faz isso

ANALYZER MODULES

- Analyzer tem dois processos:
    - Tokenização
        - Fatiar o texto em tokens, usando alguma regra
            - Por exemplo, separar tokens por espaço em branco
                - Mas pode ter outras separações
    - Normalização
        - Transformação dos tokens
            - Por exemplo, stemming i.e. transformar texto em seu radical
            - Ignorar stopwords - palavras irrelevantes muito frequentes
- Input é texto e o output é uma lista de tokens normalizados.
- Analyzer têm **três** componentes:
    - Character filters
        - Aplicado aos caracteres
        - Remove caracteres indesejados, e.g. tags HTML
        - Transforma caracteres estrangeiros em latinos.
        - São opcionais
    - Tokenizer
        - Deve existir e só pode haver apenas um único tokenizer
    - Token Filter
        - Transformações, como ignorar maiúsculas, stemming, sinônimos, etc.
        - São opcionais.

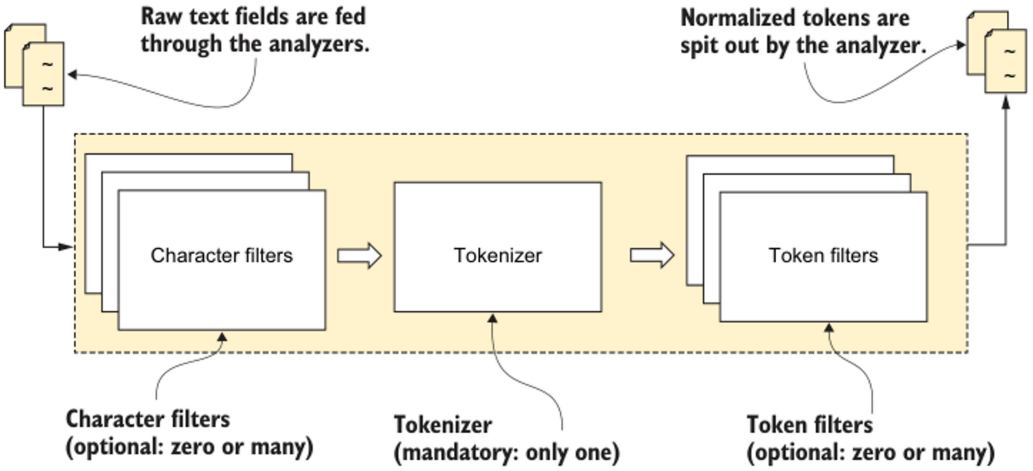


Figure 7.1 Anatomy of an analyzer module

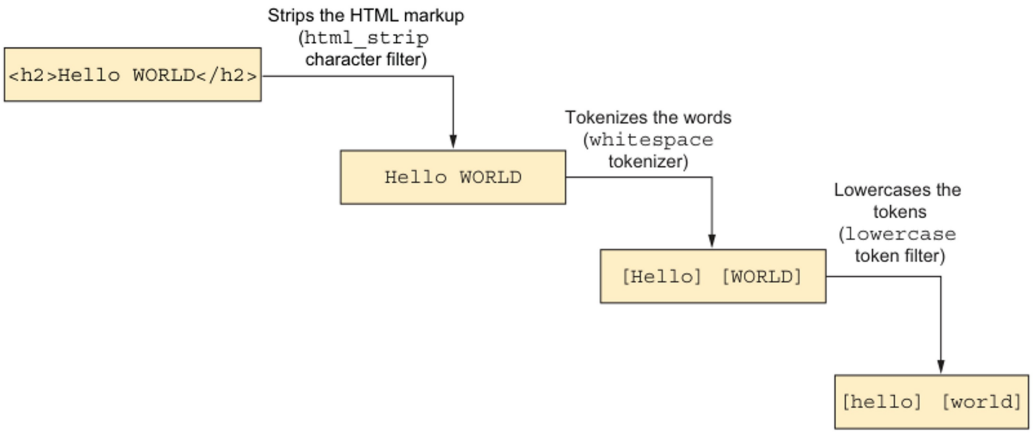


Figure 7.2 An example of text analysis in action

- Endpoint *_analyze* serve para testar analyzer.
  - Podemos passar analyzers não-default também, especificando "analyzer"
  - E também o token filter
  - Não muda nada nos índex ou no cluster, só transforma um texto ad hoc mesmo.

Listing 7.3 Creating a custom analyzer

```
GET _analyze
{
  "tokenizer": "path_hierarchy",
  "filter": ["uppercase"],
  "text": "/Volumes/FILES/Dev"
}
```

- Cada token gerado tem um tipo e uma posição:

```
{
  "tokens" : [
    {
      "token" : "james",
      "start_offset" : 0,
      "end_offset" : 5,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "bond",
      "start_offset" : 6,
      "end_offset" : 10,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "007",
      "start_offset" : 11,
      "end_offset" : 14,
      "type" : "<NUM>",
      "position" : 2
    }
  ]
}
```

- **The `standard` analyzer is used by default as it was not specified in the query.**
- **The input text is broken into a set of tokens. There are three tokens in this case: "james", "bond", and "007".**
- **Elasticsearch deduces the type associated with each token: `ALPHANUM` for alphanumerics and `NUM` for numbers.**
- **`start_offset` and `end_offset` indicate the start and end character offset of the word.**
- **All tokens are lowercased.**

Figure 7.3 Tokens produced by invoking the `_analyze` endpoint

BUILT-IN ANALYZERS

- ES já vem com vários analyzers que podem ser usados, para diversas situações.
- Também poderíamos fazer analyzers customizados.

Table 7.3 Built-in analyzers

Analyzer	Description
`standard`	The default analyzer that tokenizes input text based on grammar, punctuation, and whitespace. The output tokens are lowercased.
`simple`	Splits input text on any non-letters, such as whitespace, dashes, and numbers. Unlike the `standard` analyzer, the `simple` analyzer also lowercases the output tokens.
`stop`	A `simple` analyzer with English stop words enabled by default
`whitespace`	Tokenizes input text based on whitespace delimiters
`keyword`	Doesn't mutate the input text. The field's value is stored as is.
`language`	Helps work with human languages, as the name suggests. Elasticsearch provides dozens of analyzers for different language such as English, Spanish, French, Russian, Hindi, and so on.
`pattern`	Splits tokens based on a regular expression (regex). By default, all non-word characters help to split the sentence into tokens.
`fingerprint`	Sorts the tokens and removes duplicates to produce a single concatenated token

- **Standard**
  - É o mais comum:
  - Tokeniza baseado em espaço, gramática e pontuação.
  - Remove pontuações.
  - Inclui um token filter de stopwords mas por default está desativado

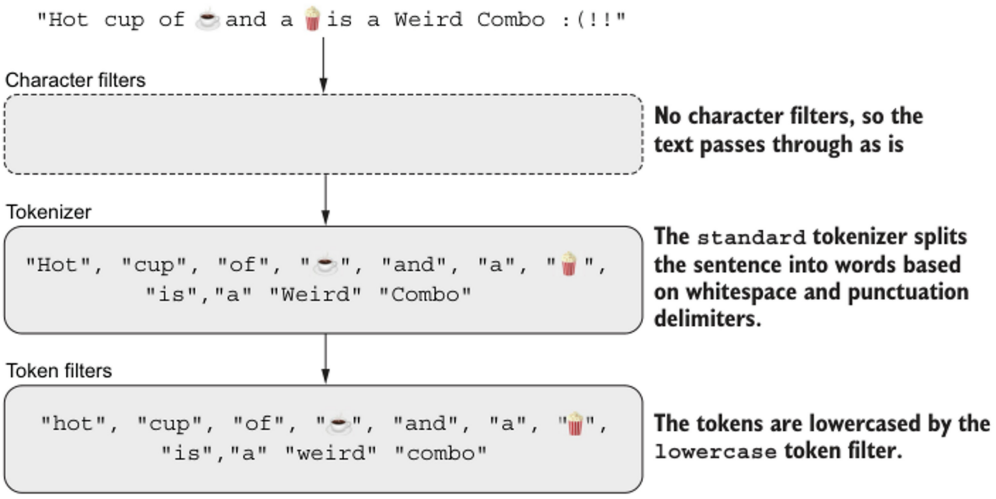
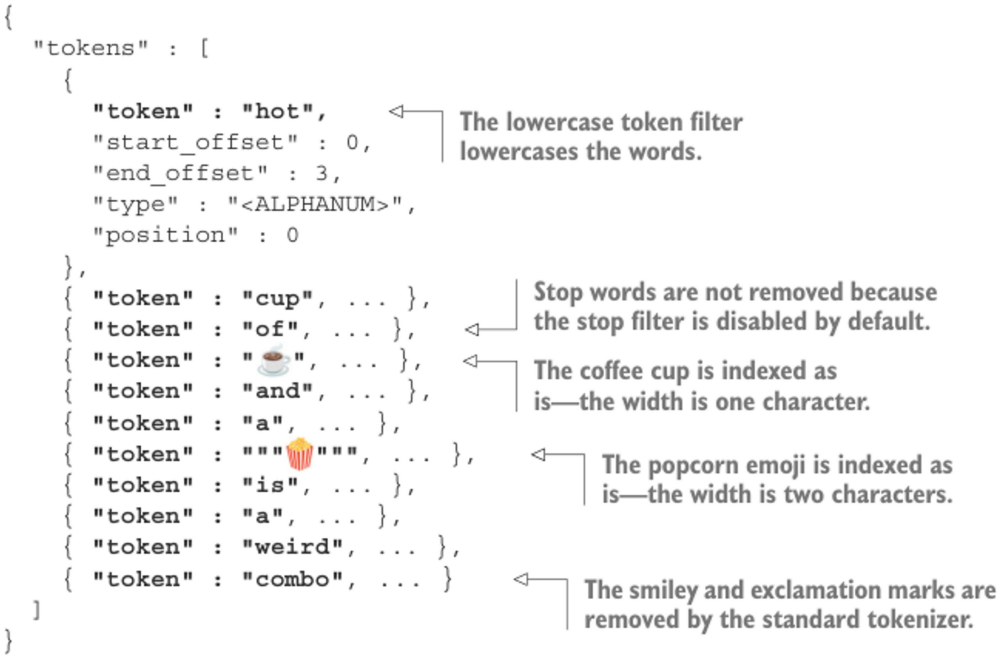


Figure 7.4 The `standard` (default) analyzer in action



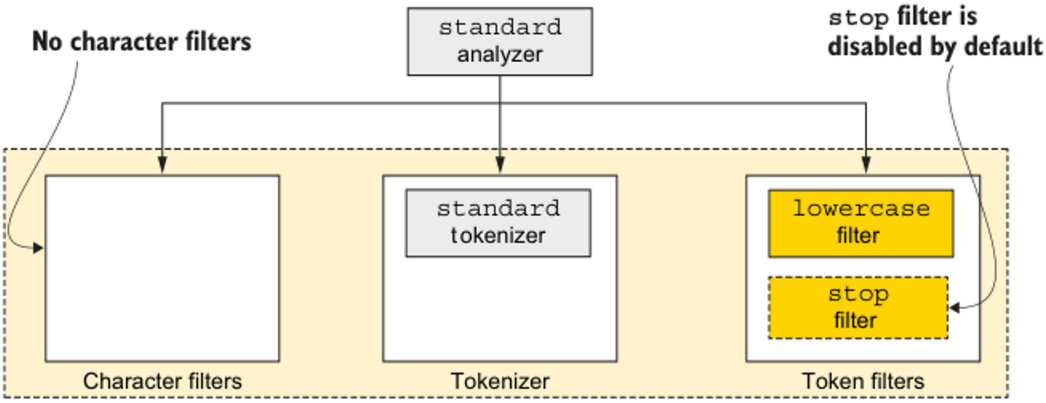


Figure 7.6 Anatomy of a `standard` analyzer

- Para alterar o analyzer, alteramos nos *settings* do index.

- A partir de então, todo texto passa pelo analyzer modificado.

Listing 7.5 The `standard` analyzer with a stop words filter enabled

```
PUT my_index_with_stopwords
{
  "settings": {
    "analysis": {
      "analyzer": {
        "standard_with_stopwords":{
          "type":"standard",
          "stopwords":"_english_"
        }
      }
    }
  }
}
```

Sets the analyzer for the index

Names the analyzer

The standard analyzer type

Enables the English stop words filter

- Podemos definir stopwords customizados a partir de um arquivo externo txt (um termo por linha).
- Podemos settar o token length máximo também (Default é 7 caracteres).

- **Simple**
  - Apenas um lowercase tokenizer, sem filters.
    - Separa baseado em não-letras

Listing 7.12 Analyzing text using a `simple` analyzer

```
POST _analyze
{
  "text": ["Lukša's K8s in Action"],
  "analyzer": "simple"
}
```

This results in the following:

```
["lukša","s","k","s","in","action"]
```

- **Whitespace**
  - Apenas um whitespace tokenizer.
    - Tokeniza baseado em espaço em branco

- **Keyword**
  - Apenas um "noop" (no-operation) tokenizer.
    - Tokenizer não faz nada, guarda o texto exatamente como ele vem, sem cortes.
  - Sem filtros.
  - Tokens resultantes são tipo "keyword"
  - Um token por texto.
  - Requer match exato para retornar resultado.

- **Fingerprint**
  - Remove palavras duplicadas, ordena palavras e gera um único token
  - Útil para detecção de itens duplicados.

- **Pattern**

- Tokeniza baseado em patterns (Regex padrão Java)

Listing 7.17 Testing the `pattern` analyzer

```
POST index_with_dash_pattern_analyzer/_analyze
{
  "text": "1234-5678-9000-0000",
  "analyzer": "pattern_analyzer"
}
```

The output of this command contains four tokens: `["1234","5678","9000","0000"]`.

- Inclui token filters lowercase e stopwords.

- **Language**
  - Analyzers específicos para cada idioma.
  - Já inclui as stopwords do idioma
  - Stemming também é específico de idioma.
    - Às vezes stemming age em excesso, é preciso regular isso com *stem_exclusions*.
      - Por exemplo, stemming de authorization e authority recaem em author.
        - Logo, busca por "authorization" não retornaria resultados relevantes. (indistinguível de author)

CUSTOM ANALYZER

- Podemos compor um analyzer não-default com os componentes que desejamos.
- Por exemplo, adicionar um filtro de tags HTML (e outras possibilidades)

Listing 7.24 Custom analyzer with filters and a tokenizer

```
PUT index_with_custom_analyzer
{
  "settings": {
    "analysis": {
      "analyzer": {
        "custom_analyzer":{
          "type":"custom",
          "char_filter":["html_strip"],
          "tokenizer":"standard",
          "filter":["uppercase"]
        }
      }
    }
  }
}
```

Specifies the custom analyzer

The type must be custom to let Elasticsearch know about our custom analyzer.

Array of character filters

Declares a single tokenizer (standard, in this case)

Token filter to uppercase incoming tokens

- Podemos definir char filters em si customizados também.
  - E.g. transformar & em "and".

SPECIFYING ANALYZERS

- Podemos especificar analyzers diferentes para indexing e para searching.
- Analyzer pode ser especificado em vários níveis:
- Para Indexing, dois níveis:
  - **Index**
    - Analyzer se aplica a todos os text fields do index

Listing 7.28 Creating an index with a default analyzer

```
PUT authors_with_default_analyzer
{
  "settings": {
    "analysis": {
      "analyzer": {
        "default":{
          "type":"keyword"
        }
      }
    }
  }
}
```

Setting this property sets the index's default analyzer.

- **Field**
  - Analyzer diferente para cada text field

Listing 7.27 Setting field-level analyzers during index creation

```
PUT authors_with_field_level_analyzers
{
  "mappings": {
    "properties": {
      "name":{
        "type": "text"
      },
      "about":{
        "type": "text",
        "analyzer": "english"
      },
      "description":{
        "type": "text",
        "fields": {
          "my":{
            "type": "text",
            "analyzer": "fingerprint"
          }
        }
      }
    }
  }
}
```

Uses the standard analyzer

Explicitly sets the english analyzer

Fingerprint analyzer on a multi-field

- Para Searching:
  - Em **Query**:
    - Definimos analyzer no momento da query, no corpo da requisição.
    - Pode ser diferente do analyzer usado durante o indexing.
  - Em **Field**:
    - Definimos nos mappings

Listing 7.31 Setting a search analyzer at the field level

```
PUT authors_index_with_both_analyzers_field_level
{
  "mappings": {
    "properties": {
      "author_name":{
        "type": "text",
        "analyzer": "stop",
        "search_analyzer": "simple"
      }
    }
  }
}
```

- Podemos definir um search_analyzer padrão nos settings do index.
- Análogo ao analyzer da indexação.

- Engine usa um analyzer apenas, e seleciona baseado nas seguintes prioridades, prioridade descrescente:
  i. Analyzer da Query
  ii. Search_analyzer dos fields, nos mapping
  iii. Analyzer no Index
  iv. Indexing Analyzer no field ou index

CHARACTER FILTERS

- Limpa texto antes da tokenização
  - Remove HTML, pontuação.
  - Substitui caracteres
- Três tipos:
  - **HTML strip**
    - Remove tags HTML
    - Pode ser customizado para só remover alguns tipos de tags
  - **Mapping**
    - Transforma caracteres em algum texto.
      - E.g. UK --> United Kingdom
    - Também customizável
    - Mapping pode ser importado a partir de um arquivo txt.
  - **Pattern Replace**
    - Substitui por regex

TOKENIZERS

- Separa o texto em tokens segundo algum critério
- Existem vários tipos de tokenizers que já vêm com o ElasticSearch
- Alguns deles:
  - **Standard:**
    - Separa baseado em separação entre palavras (e.g. espaço) e pontuação.
    - Única customização é max_token_length (default é 255)
  - **Tokenizers Ngram e edge_ngram**:
    - N-grama são sequencias se caracteres a partir de uma palavra:
      - Bigramas de *Coffee* --> co, of, ff, fe

        - Trigrama de *Coffee* --> coo, oof, off, ffe, fee.
    - Edge n-gramas são n-gramas ancorados no início da palavra:
        - Edge n-gramas de *Coffee*:
            - C, co, coo, coff, coffe, coffee.
    - **N-gram tokenizer:**
        - Transforma texto em n-gramas, n pode ser um range de valores (min e max)
            - Devolve todos os 1-grama e todos os 2-grama por default.
    - **Edge-gram tokenizer:**
        - Transforma texto em edge-grams.
        - Também podemos definir tamanho mínimo e máximo dos edge-gram.
- **Outros:**

Table 7.4 Out-of-the-box tokenizers

Tokenizer	Description
`pattern`	Splits a field into tokens on a regex match. The default pattern is to split words when a non-word character is encountered.
`uax_url_email`	Parses fields and preserves URLs and emails. The URLs and emails are returned as they are, without any tokenization.
`whitespace`	Splits text into tokens when whitespace is encountered
`keyword`	Doesn't touch the tokens. This tokenizer returns the text as is.
`lowercase`	Splits text into tokens when a non-letter is encountered and lowercases the tokens
`path_hierarchy`	Splits hierarchical text such as filesystem folders into tokens based on path separators

TOKEN FILTERS

- Transformação dos tokens após o tokenizador
    - E.g. fazer mínusculas, reverter, stemming, sinônimos, etc.
    - ES já inclui uns 50 filters possíveis
    - Stemmer:
        - Reduz palavras para sua raíz
    - Shingle:
        - N-gramas de palavras, não de caracteres.
            - E.g. James Bond --> ["James", "James Bond"]
        - Default é n-gramas de 1 e 2 palavras.
            - Isso pode ser customizado.
    - Synonym:
        - Transforma palavras em seus sinônimos.
        - Requer uma lista de sinônimos.
            - Pode ser passado por um arquivo txt.

Summary

- Elasticsearch analyzes `text` fields via the text analysis process. Text analysis is carried out using either built-in or custom analyzers. Non-text fields are not analyzed.
- Text analysis consists of two phases: tokenization and normalization. Tokenization splits the input field into individual words or tokens, and normalization enhances the word (for example, changing it to a synonym, stemming it, or removing it).
- Elasticsearch employs a software module called an analyzer to carry out text analysis. An analyzer is a package made of character filters, token filters, and a tokenizer.
- Elasticsearch uses a `standard` analyzer by default if no analyzer is given explicitly during indexing and searching. A `standard` analyzer employs no character filter, a `standard` tokenizer, and two token filters (`lowercase` and `stop`), although the `stop` filter is off by default.
- Every analyzer must have one (and only one) tokenizer, but it can have zero or more character or token filters.
- Character filters help strip unwanted characters from input fields. Tokenizers act on the fields processed by character filters (or on raw fields, since character filters are optional). Token filters work on the tokens emitted by the tokenizer.
- Elasticsearch provides several out-of-the-box analyzers. We can mix and match existing tokenizers with character or token filters to make custom analyzers that suit our requirements.