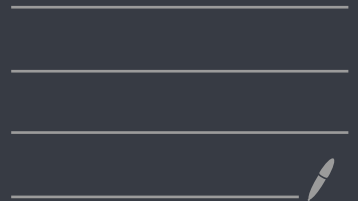


Elasticsearch.cpp



Creating Elasticsearch from Scratch

→ Inverted Index (all data in ES is stored in Apache Lucene as an inverted index)

hashing done on these

Metadata

Dictionary	Freq.	Document Id
ElasticSearch	1	Doc-1
Elastic	1	Doc-2
Search	1	Doc-2
Big	2	Doc-1, Doc-3

tokens

→ Data stored as JSON

DOCUMENT
(metadata
+ json)

← "index": ("student"/"teacher"/.....)
"type":
"id":
"source": this JSON

```
{  
  "name": "Ravi",  
  "id": 1  
}
```

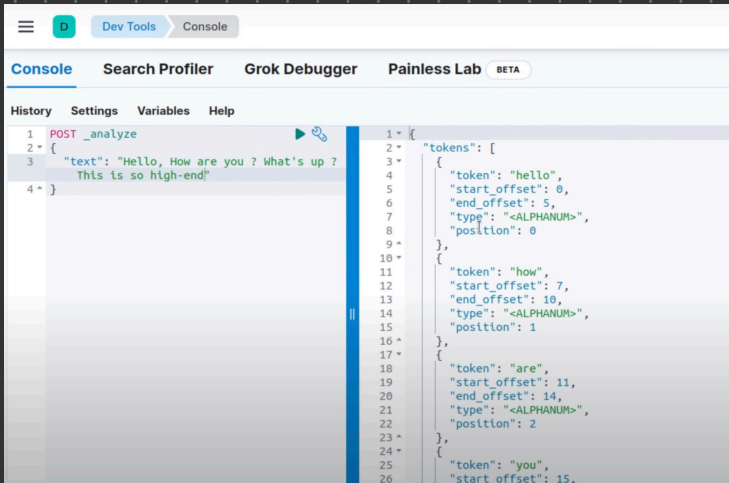
Analysis

Text → Terms (tokens) → store efficiently (build inverted index)
data structure

can skip

- character Filter : Basic Filtering (like removing HTML tags)
 - Tokenization : Standard Tokenisation (splits text into words based on whitespace and punctuation)
 - Token Filtering : converts all tokens to lowercase
- important

⇒ default found after running `-analyze`



```
1 POST _analyze
2 {
3   "text": "Hello, How are you ? What's up ? This is so high-end!"
4 }
5 {
6   "tokens": [
7     {
8       "token": "hello",
9       "start_offset": 0,
10      "end_offset": 5,
11      "type": "<ALPHANUM>",
12      "position": 0
13     },
14     {
15       "token": "how",
16       "start_offset": 7,
17       "end_offset": 10,
18       "type": "<ALPHANUM>",
19       "position": 1
20     },
21     {
22       "token": "are",
23       "start_offset": 11,
24       "end_offset": 14,
25       "type": "<ALPHANUM>",
26       "position": 2
27     },
28     {
29       "token": "you",
30       "start_offset": 15,
```

this is
STANDARD
analyser

I will implement WHITESPACE analyser

matchQuery: return all documents where we find any of the tokens of the query string



This is the only thing that I implemented.
Soo... basically, an inverted index.

↓
with frequency
and location
details