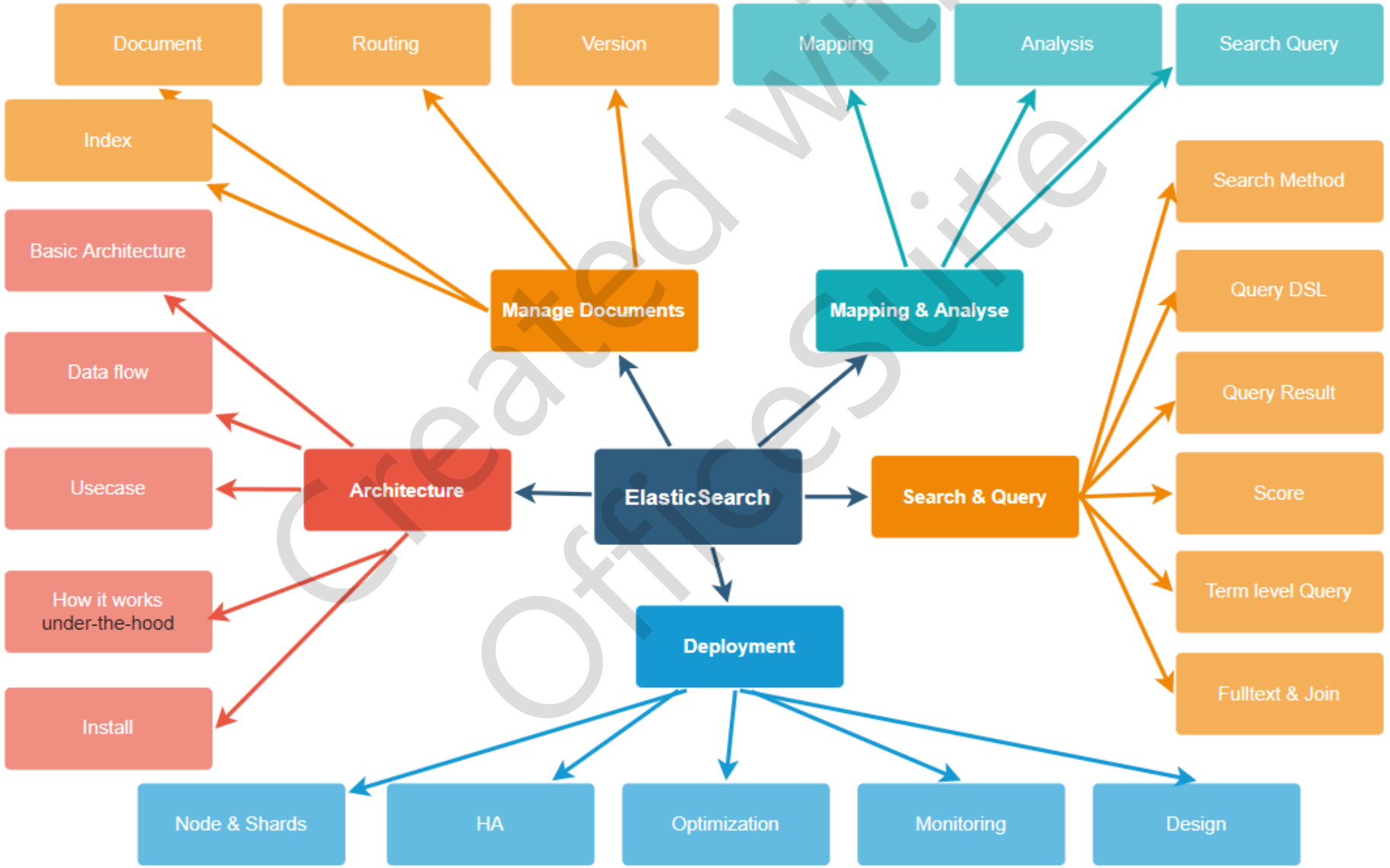
ElasticSearch

Techmaster

Nội dung



Search method

- .....

```
GET /product/default/_search
{
  "query": {
    "match": {
      "description": {
        "value": "red wine"
      }
    }
  }
}
```

```
GET /product/default/_search
{
  "query": {
    "match": {
      "description": "red wine"
    }
  }
}
```

```
GET /product/default/_search
{
    "query": {
        "query_string" : {
            "query": "name:pasta"
        }
    }
}
```

Searching with request URI

GET /products/_doc/100

GET /products/_search?

GET /products/_search?q=name:Maker

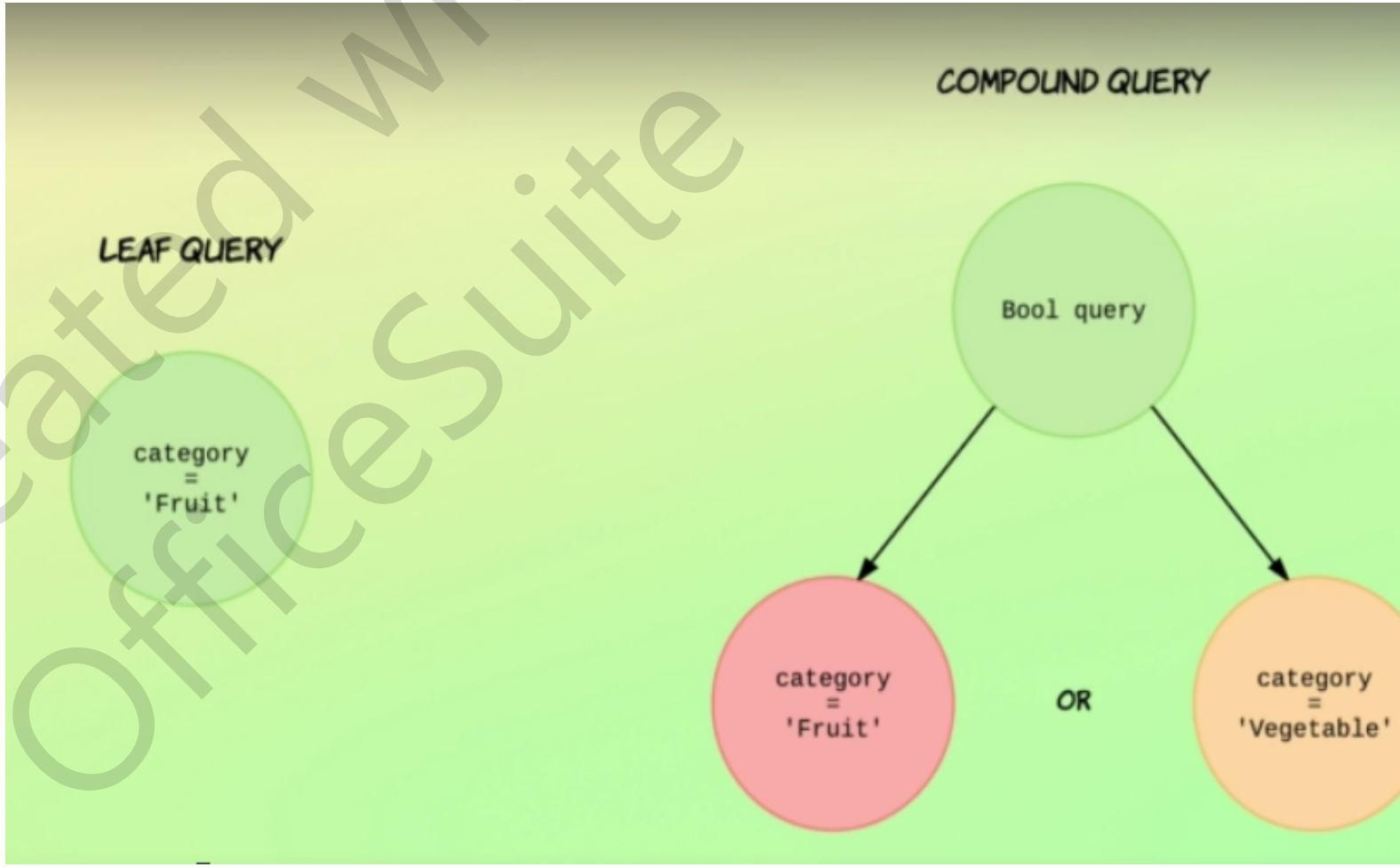
GET /products/_search?q=name:tenet

GET /products/_search?q=name:Maker
AND price:65

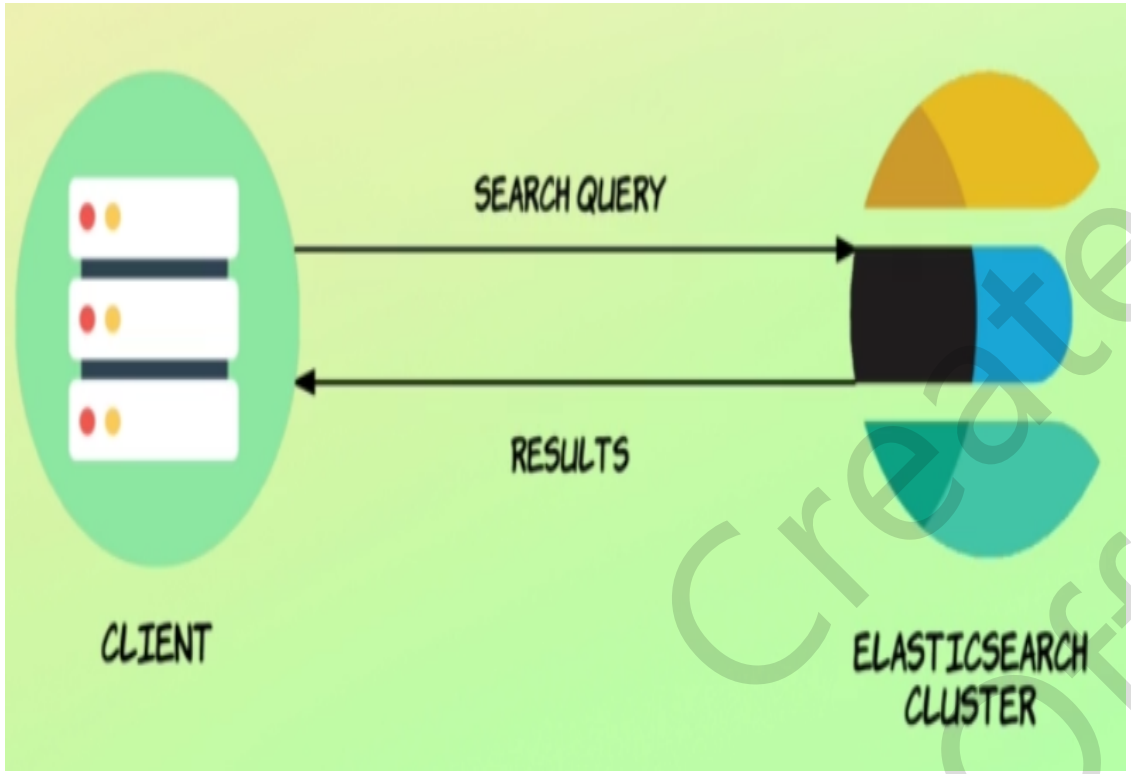

```
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 0.10536051,
    "hits" : [
      {
        "_index" : "products",
        "_type" : "_doc",
        "_id" : "q_OhHX0BHhtkMm9JUVTl",
        "_score" : 0.10536051,
        "_source" : {
          "name" : "Coffee Maker",
```

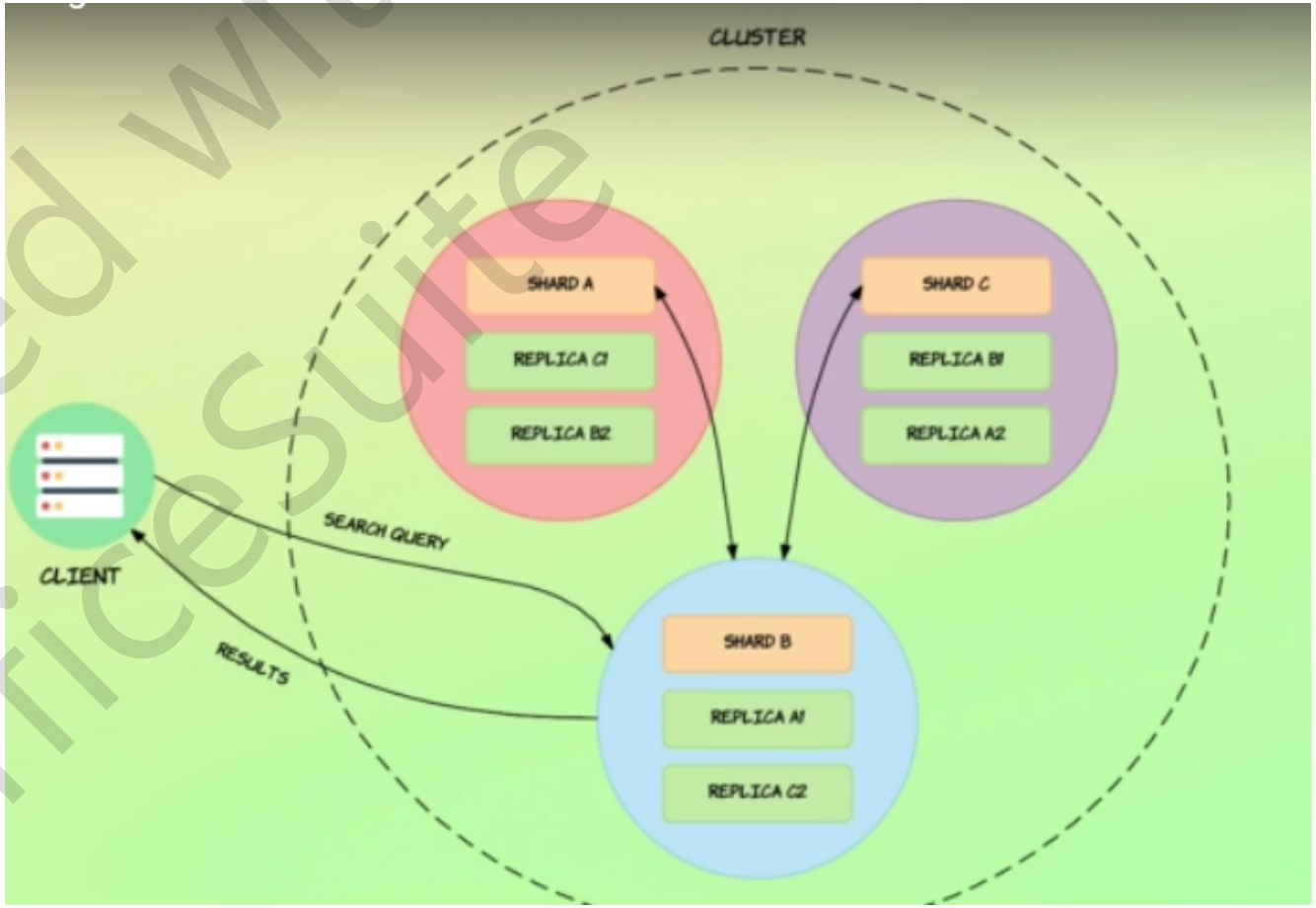
Introduce the Query DSL

```
GET /products/_search
{
  "query": {
    "match": {"in_stock":10}
  }
}
GET /products/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": {"in_stock":10}},
        { "match": {"name":"Maker"}}
      ]
    }
  }
}
```



How searching work



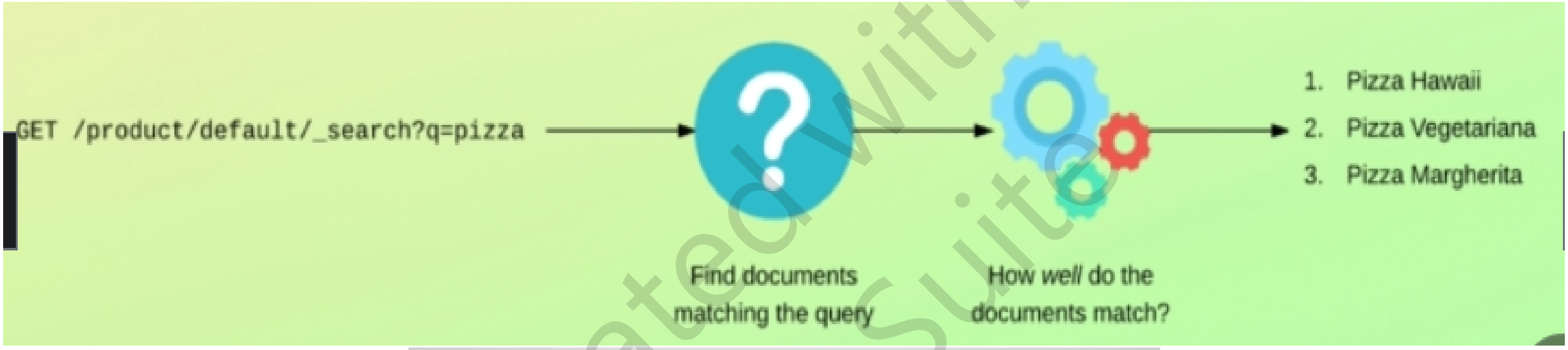


Understanding query results

```
GET /products/_search
{
  "query": {
    "match_all": {}
  }
}
```


```
{
  "took" : 4,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 8,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "products",
        "_type" : "_doc",
        "_id" : "q_OhHX0BHhtkMm9JUVTl",
        "_score" : 1.0,
        "_source" : {
          "name" : "Coffee Maker",
```

Understanding relevance score



TF/IDF

Term Frequency / Inverse Document Frequency

Okapi BM25

The relevance scoring algorithm currently used by Elasticsearch.

Understanding relevance score

Term Frequency (TF)

How many times does the term appear in the field for a given document?

Inverse Document Frequency (IDF)

How often does the term appear within the index (i.e. across all documents)?

Field-length norm

How long is the field?

score(q,d) =
queryNorm(q)
* coord(q,d)
* SUM (
tf(t in d),
idf(t)^2,
t.getBoost(),
norm(t,d)
) (t in q)

TF/IDF & BM25

IDF score * TF score * fieldNorms

TF/IDF

= (1+ log(numDocs / (docFreq + 1))) * √frequency * (1 / √numTerms)

BM25

= log(1 + (docCount - docFreq + 0.5) / (docFreq + 0.5))
*((k + 1) * freq) / (k + freq) * (1 / √numTerms)

Understanding relevance score

Comparison with the BM25 algorithm

Better at handling stop words

Nonlinear Term Frequency Saturation

Understanding relevance score

Comparison with the BM25 algorithm

Better at handling stop words

Improves the field-length norm factor

Can be configured with parameters

```
GET /products/_search?explaine=true
{
  "query": {
    "bool": {
      "must": [
        { "match": {"in_stock":10}},
        { "match": {"name":"Maker"}}
      ]
    }
  }
}
```

Term query

- Returns documents that contain an exact term in a provided field.
- You can use the term query to find documents based on a precise value such as a price, a product ID, or a username.
- Avoid using the term query for text fields.
- By default, Elasticsearch changes the values of text fields as part of analysis. This can make finding exact matches for text field values difficult.
- To search text field values, use the match query instead.

```
GET /products/_search
{
  "query": {
    “term": {
          “name” : “Maker”}
  }
}
```

Query and Filter context

Relevance score

- By default, Elasticsearch sorts matching search results by relevance score, which measures how well each document matches a query.
- The relevance score is a positive floating point number, returned in the _score metadata field of the search API. The higher the _score, the more relevant the document. While each query type can calculate relevance scores differently, score calculation also depends on whether the query clause is run in a query or filter context.

Query context

- In the query context, a query clause answers the question “How well does this document match this query clause?” Besides deciding whether or not the document matches, the query clause also calculates a relevance score in the _score metadata field.
- Query context is in effect whenever a query clause is passed to a query parameter, such as the query parameter in the search API.

Filter context

- In a filter context, a query clause answers the question “Does this document match this query clause?” The answer is a simple Yes or No — no scores are calculated. Filter context is mostly used for filtering structured data, e.g.
  - Does this timestamp fall into the range 2015 to 2016?
  - Is the status field set to "published"?
- Frequently used filters will be cached automatically by Elasticsearch, to speed up performance.
- Filter context is in effect whenever a query clause is passed to a filter parameter, such as the filter or must_not parameters in the bool query, the filter parameter in the constant_score query, or the filter aggregation.

Example

- Below is an example of query clauses being used in query and filter context in the search API. This query will match documents where all of the following conditions are met:
  - The title field contains the word search.
  - The content field contains the word elasticsearch.
  - The status field contains the exact word published.
  - The publish_date field contains a date from 1 Jan 2015 onwards.

```
GET /_search
{
 "query": {
   "bool": {
     "must": [
       { "match": { "title":   "Search"        }},
       { "match": { "content": "Elasticsearch" }}
     ],
     "filter": [
       { "term":  { "status": "published" }},
       { "range": { "publish_date": { "gte": "2015-01-01" }}}
     ]
   }
 }
}
```