Elasticsearch

Jakub Podeszwik

Yameo

28.03.2018

Agenda

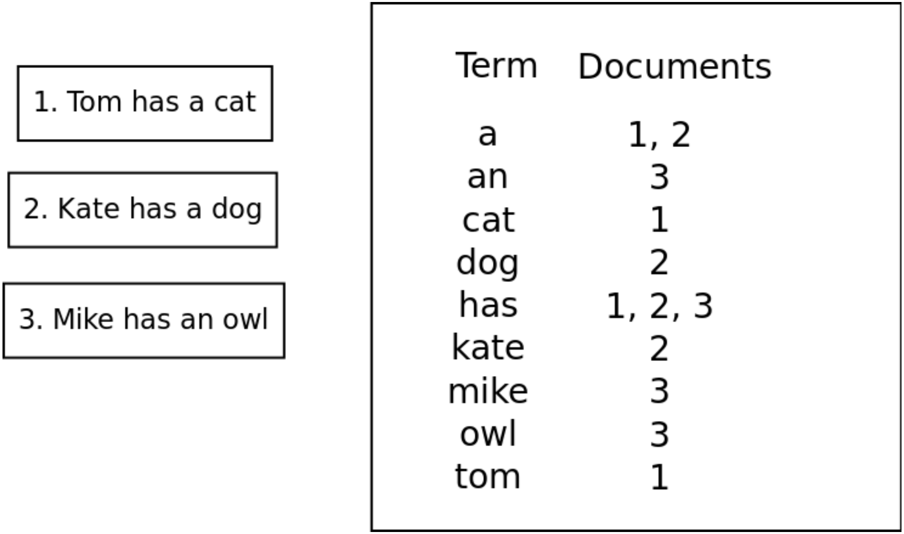
Apache Lucene

1. Full text search library written in Java
2. initlally released in 1999
3. in 2010 Apache Solr joined Lucene as subproject

Elasticsearch

1. Open source Full text search Engine writen on top of lucene
2. 02.2010 - version 0.4 released
3. 02.2014 - version 1.0 released
4. scalable, near realtime, highly available, restful

Inverted index



1. Tom has a cat

2. Kate has a dog

3. Mike has an owl

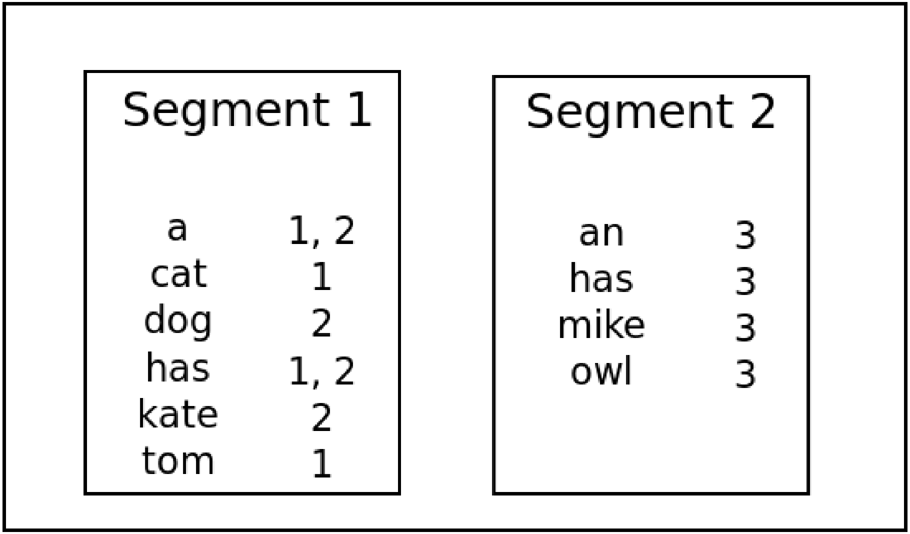
Term	Documents
a	1, 2
an	3
cat	1
dog	2
has	1, 2, 3
kate	2
mike	3
owl	3
tom	1

Lucene Segment

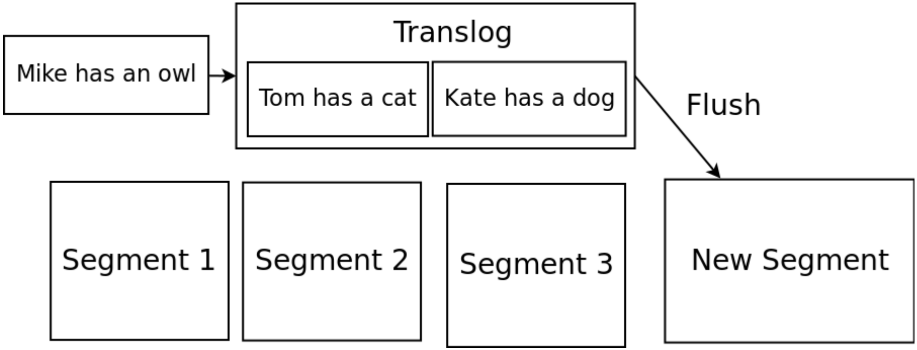
Immutable, stored on disk data structure consisting of:

1. inverted index
2. fielddata cache / doc_values
3. _source
4. live documents bitset (this one is not immutable)
5. ...

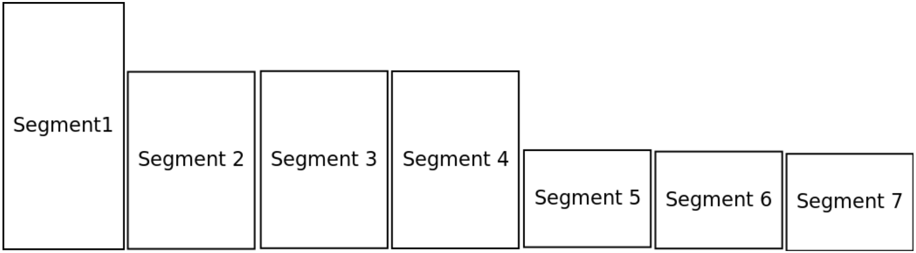
Lucene index



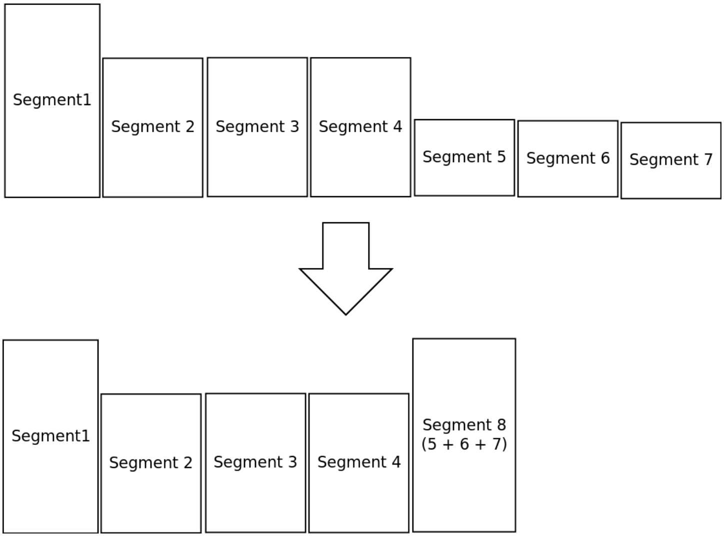
Translog


Mike has an owl
Translog
Tom has a cat
Kate has a dog
Flush
Segment 1
Segment 2
Segment 3
New Segment

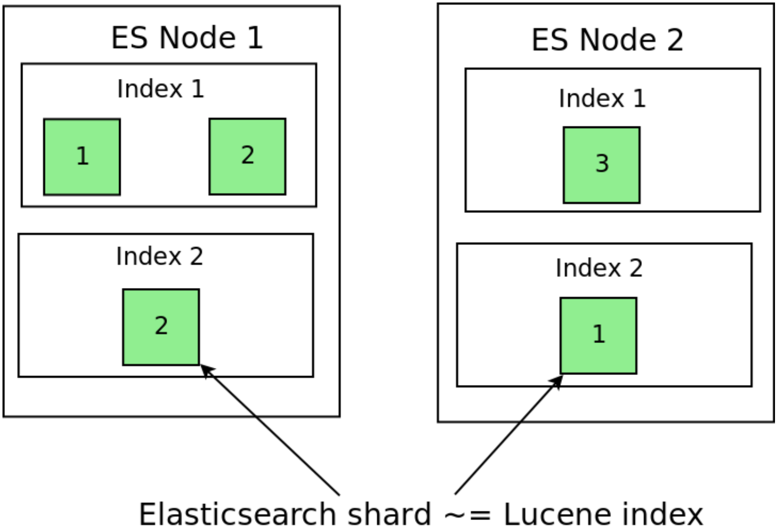
Segments merging



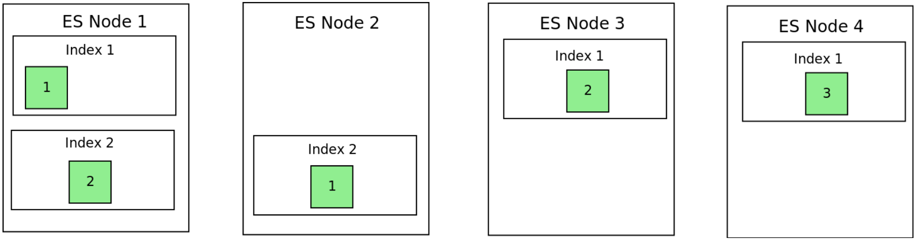
Segments merging



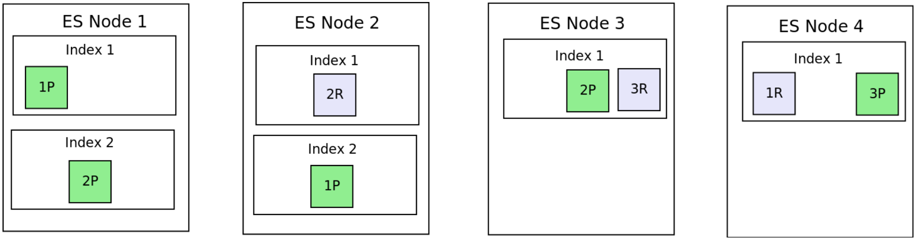
Elasticsearch index



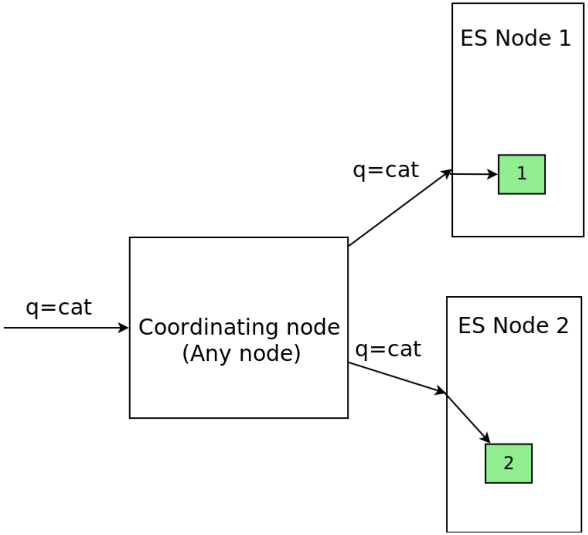
Elasticsearch index



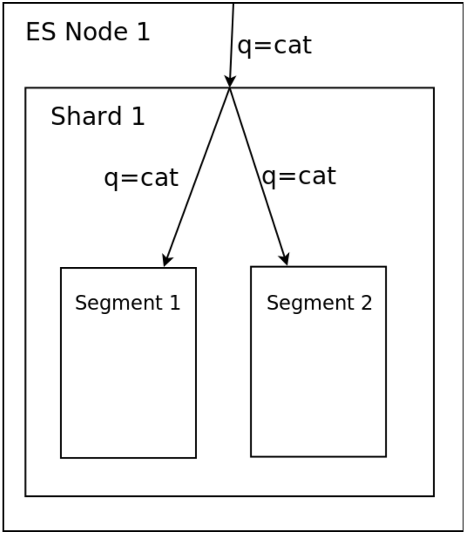
Elasticsearch shard replicas



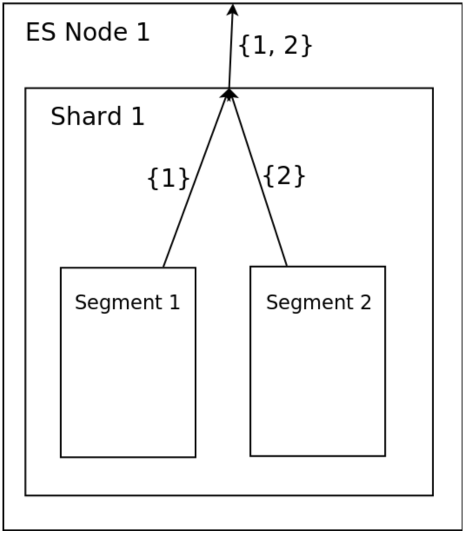
Search


ES Node 1
q=cat
1
q=cat
Coordinating node
(Any node)
ES Node 2
q=cat
2

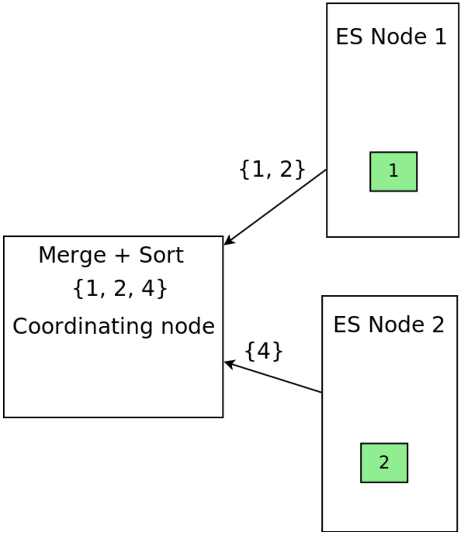
Search



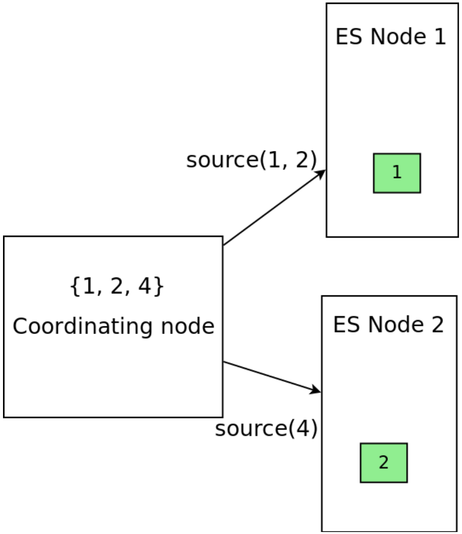
Search



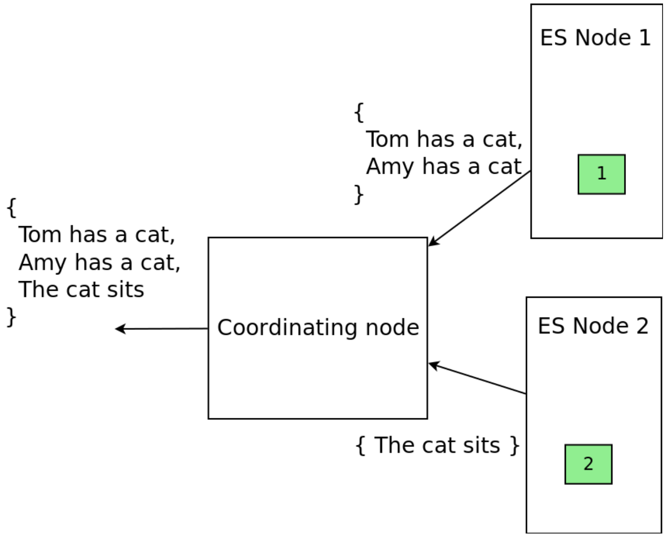
Search



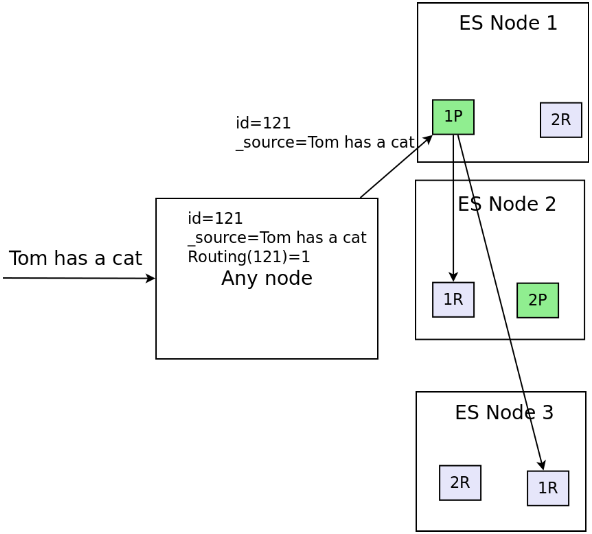
Search


ES Node 1
source(1, 2)
1
{1, 2, 4}
Coordinating node
ES Node 2
source(4)
2

Search


ES Node 1
{
Tom has a cat,
Amy has a cat
}
1
{
Tom has a cat,
Amy has a cat,
The cat sits
}
Coordinating node
ES Node 2
{ The cat sits }
2

Indexing


ES Node 1
1P
2R
id=121
_source=Tom has a cat
ES Node 2
1R
2P
Tom has a cat
id=121
_source=Tom has a cat
Routing(121)=1
Any node
ES Node 3
2R
1R

_cat api

1. indices
2. shards
3. nodes
4. master
5. ?v - legend

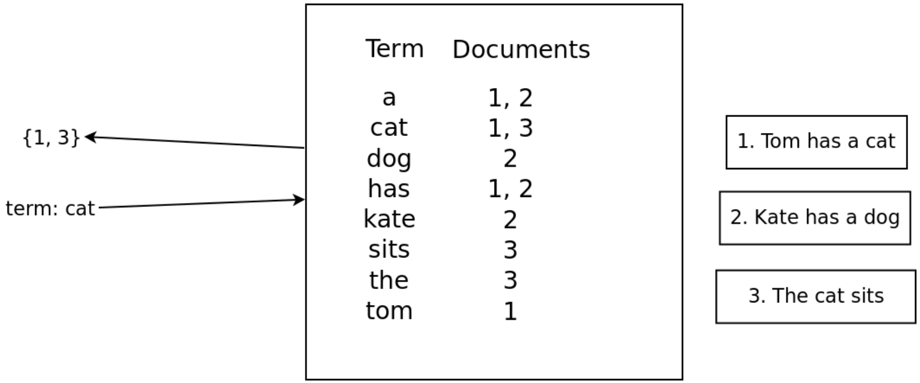
Mapping

Definition of how documents in elasticsearch should be stored / indexed / searched.
Elasticsearch will try to guess mapping from incomming documents if some of the fields are not specified.

Queries

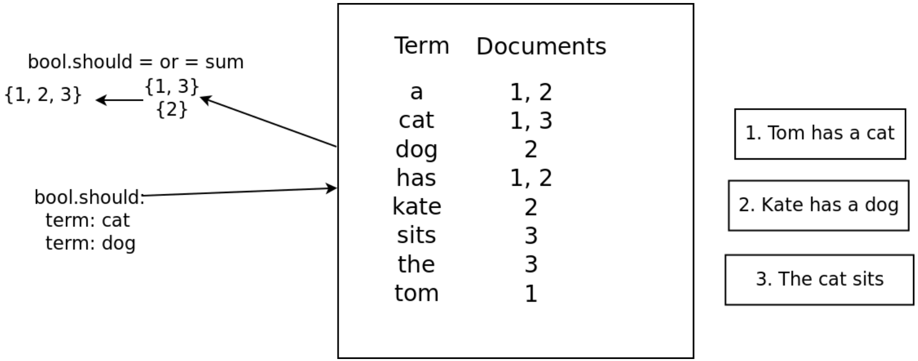
1. term
2. bool
3. prefix
4. wildcard
5. fuzzy
6. match, match_all
7. query_string
8. filter vs query

Term



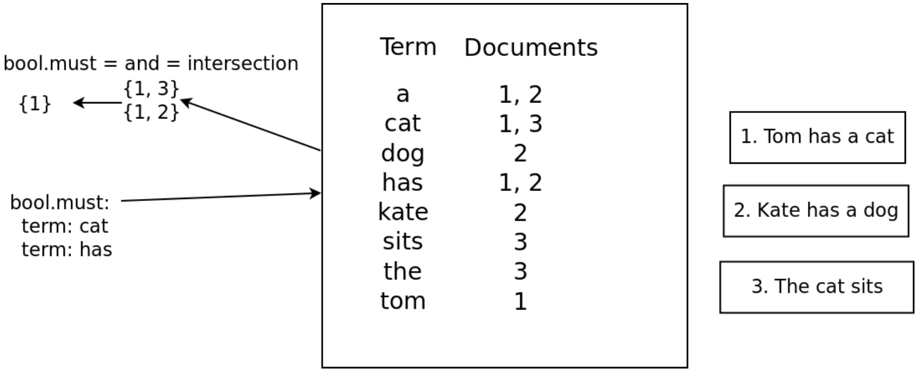
Term	Documents
a	1, 2
cat	1, 3
dog	2
has	1, 2
kate	2
sits	3
the	3
tom	1

Bool



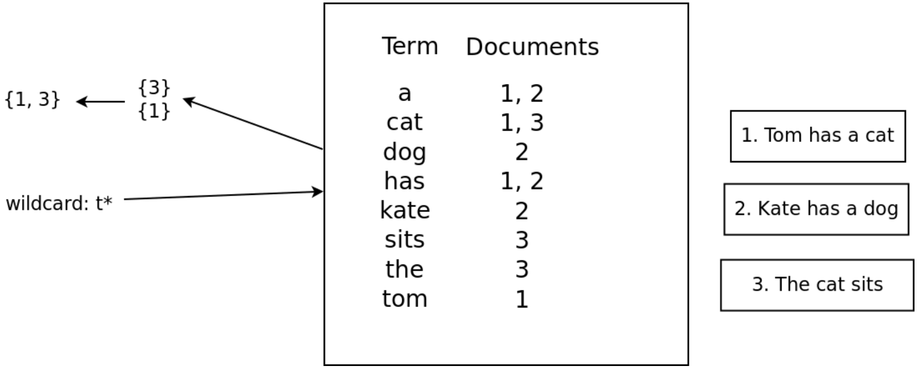
Term	Documents
a	1, 2
cat	1, 3
dog	2
has	1, 2
kate	2
sits	3
the	3
tom	1

Bool



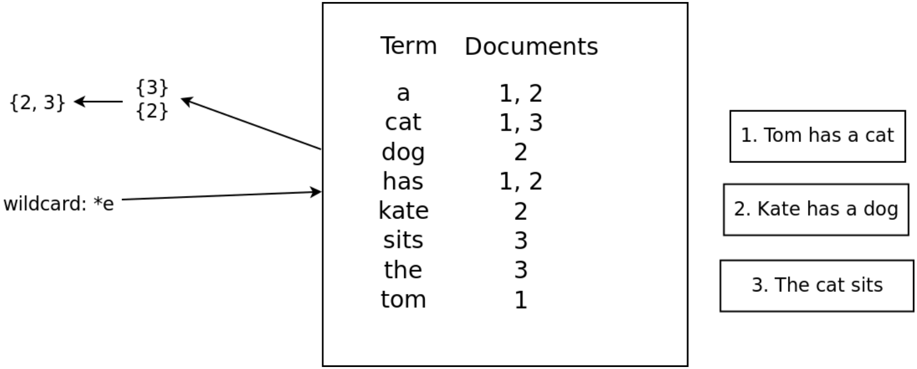
bool.must = and = intersection
{1, 3}
{1, 2}
{1}
bool.must:
term: cat
term: has
Term
Documents
a 1, 2
cat 1, 3
dog 2
has 1, 2
kate 2
sits 3
the 3
tom 1
1. Tom has a cat
2. Kate has a dog
3. The cat sits

Wildcard



{1, 3}
{3}
{1}
wildcard: t*
Term
Documents
a
1, 2
cat
1, 3
dog
2
has
1, 2
kate
2
sits
3
the
3
tom
1
1. Tom has a cat
2. Kate has a dog
3. The cat sits

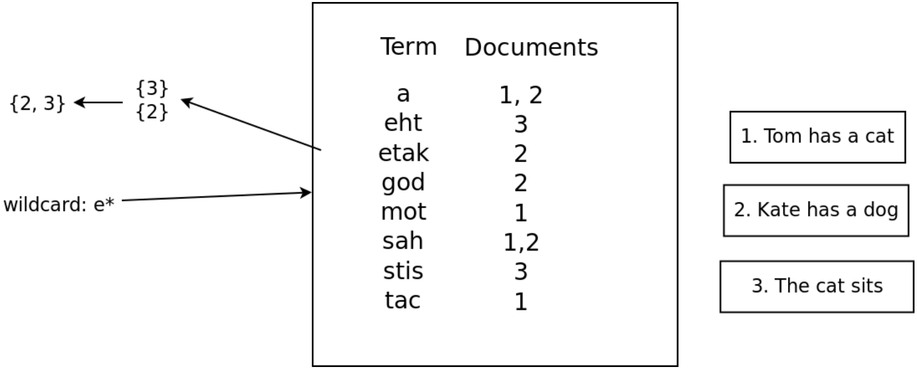
Wildcard



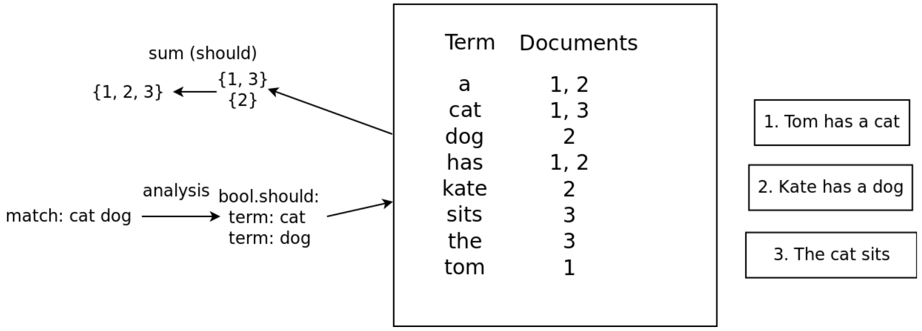
Term	Documents
a	1, 2
cat	1, 3
dog	2
has	1, 2
kate	2
sits	3
the	3
tom	1

1. Tom has a cat
2. Kate has a dog
3. The cat sits

Wildcard


{2, 3}
{3}
{2}
wildcard: e*
Term
Documents
a
1, 2
eht
3
etak
2
god
2
mot
1
sah
1,2
stis
3
tac
1
1. Tom has a cat
2. Kate has a dog
3. The cat sits

Match



Term	Documents
a	1, 2
cat	1, 3
dog	2
has	1, 2
kate	2
sits	3
the	3
tom	1

Analyzers

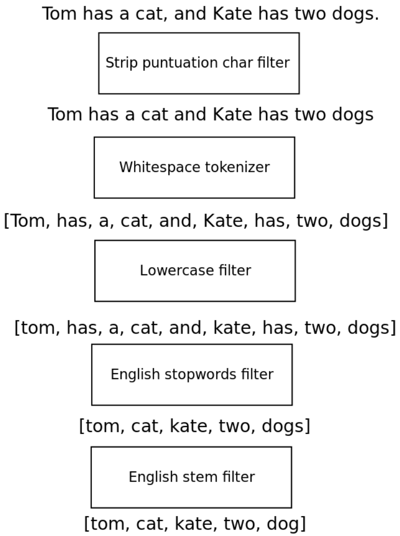
Analazers split input string into stream of terms. Examples:

1. whitespace
2. standard - default analyzer for text fields.
3. keyword - noop

Custom analyzer

1. char_filters - removes / changes / adds characters before text goes to tokenizer
2. tokenizer - splits text into tokens
3. filters - removes / changes / adds tokens

Custom analyzer



Tom has a cat, and Kate has two dogs.

Tom has a cat and Kate has two dogs

[Tom, has, a, cat, and, Kate, has, two, dogs]

[tom, has, a, cat, and, kate, has, two, dogs]

[tom, cat, kate, two, dogs]

[tom, cat, kate, two, dog]

Scoring

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

Scoring

1. score(q,d) - the relevance score of document d for query q
2. queryNorm(q) - query normalization factor
3. coord(q,d) is the coordination factor (rewards for higher percentage of query terms contained in document)
4. tf(t in d) - term frequency for term t in document d,
5. idf(t) - inverse document frequency for term t,
6. t.getBoost() - boost that has been applied to the query,
7. norm(t,d) - field-length norm, combined with the index-time field-level boost

(https://www.elastic.co/guide/en/elasticsearch/guide/current/practical-scoring-function.html)

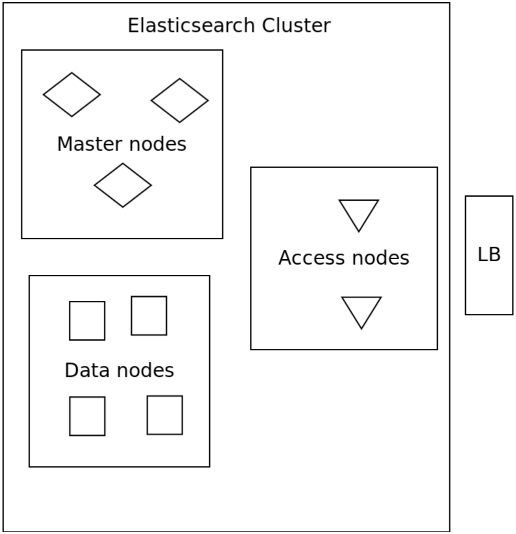
Cluster

1. Master node - keeping, updating, broadcasting state of the cluster (list of nodes, mappings)
2. Data node - storing data and executing searches on shard
3. Access node - forwarding requests to data nodes and merging results
4. Ingest node - preprocess documents before indexing them

Leader election

1. zen discovery
2. leader election
3. quorum

Cluster


Elasticsearch Cluster
Master nodes
Access nodes
LB
Data nodes

Q&A