ADVANCED DATABASES CIS 6930

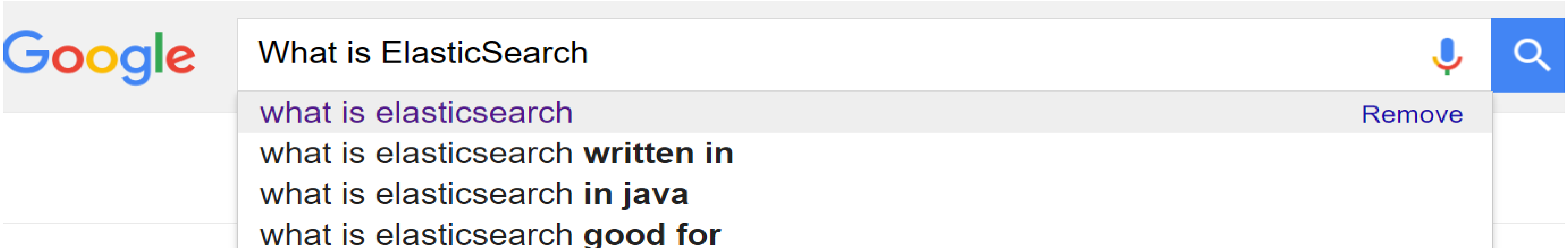
Dr. Markus Schneider

elasticsearch

Group 5

Ajantha Ramineni, Sahil Tiwari,
Rishabh Jain, Shivang Gupta

WHAT IS ELASTIC SEARCH ?



What is ElasticSearch

what is elasticsearch

what is elasticsearch **written in**

what is elasticsearch **in java**

what is elasticsearch **good for**

Elastic Search

Elasticsearch is a search engine based on Lucene. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents.

Key Features

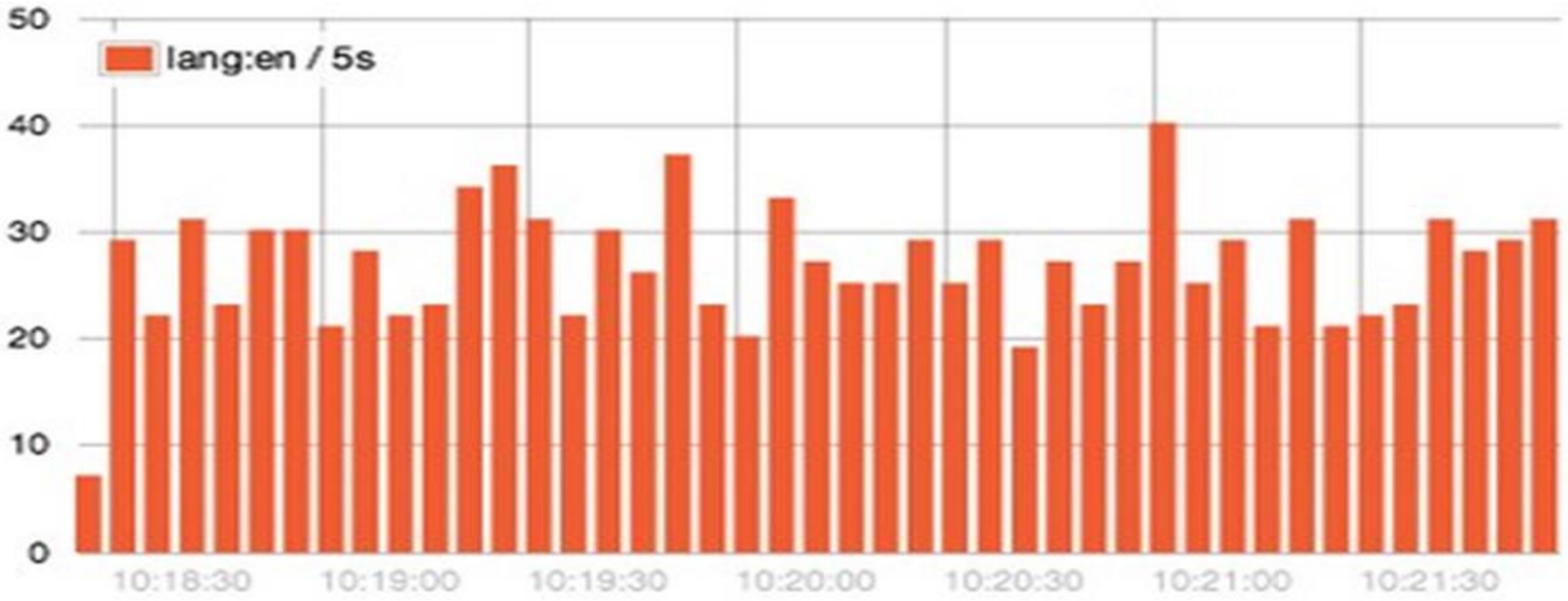
- Real Time data
- Real Time advanced Analytics
- High Availability
- Multi-Tenancy
- Full Text Search
- Document-Oriented
- Conflict Management
- Per-Operation Persistence

Advanced Features

- **Nested documents** (Child-Parent)
  - *Like MySQL joins?*
- **Percolation Index**
  - Store queries in Elastic
  - Send it documents
  - Get returned which queries match
- **Index Warming**
  - Register search queries that cause heavy load
  - New data added to index will be *warmed*
  - So next time query is executed: *pre cached*

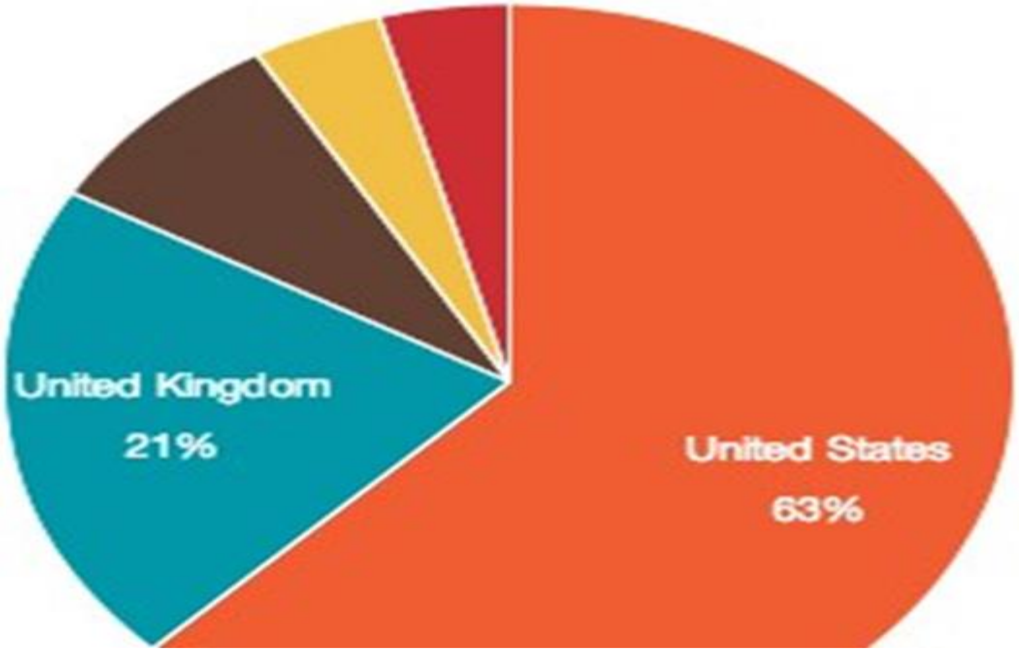
Real-Time data

- Data flows into your system all the time. The question is
- The data accurate. Using Elastic search accurate real time data is achievable.



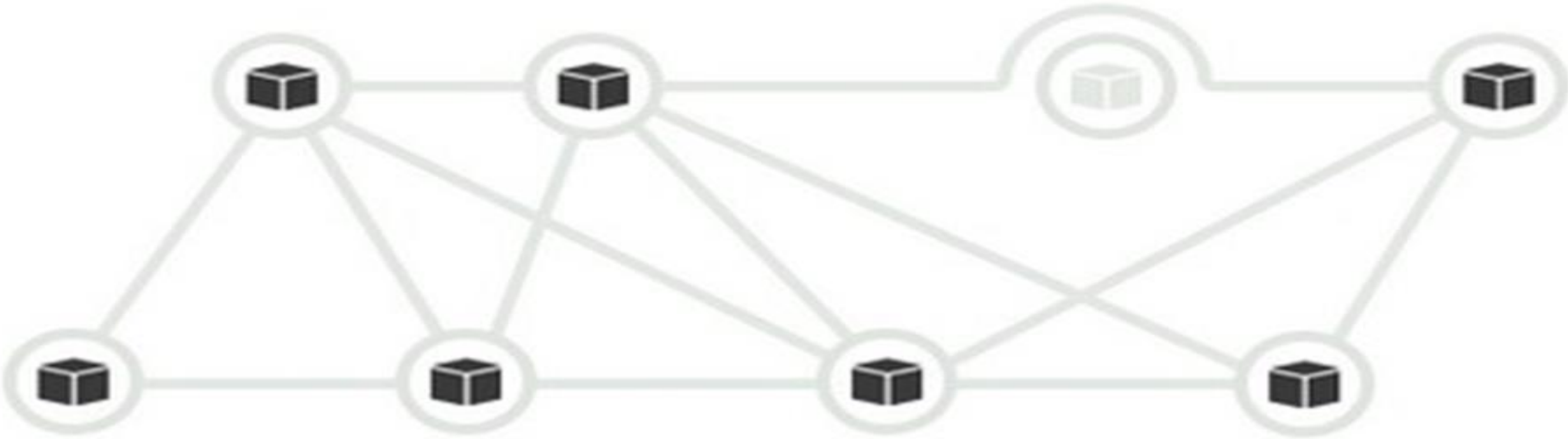
Real Time Analytics

- Search isn’t normal anymore. It’s about exploring the data, Understanding it. Gaining Insights.



High Availability

- Elasticsearch clusters are resilient-they will detect and remove failed nodes and ensure that your data is safe and accessible.

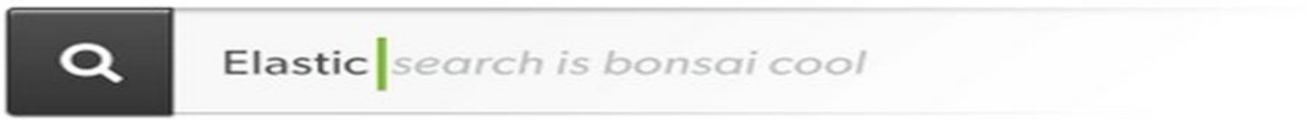


Conflict Management

Optimistic Version control is used to ensure data is never lost in a transaction.

Full Text Search

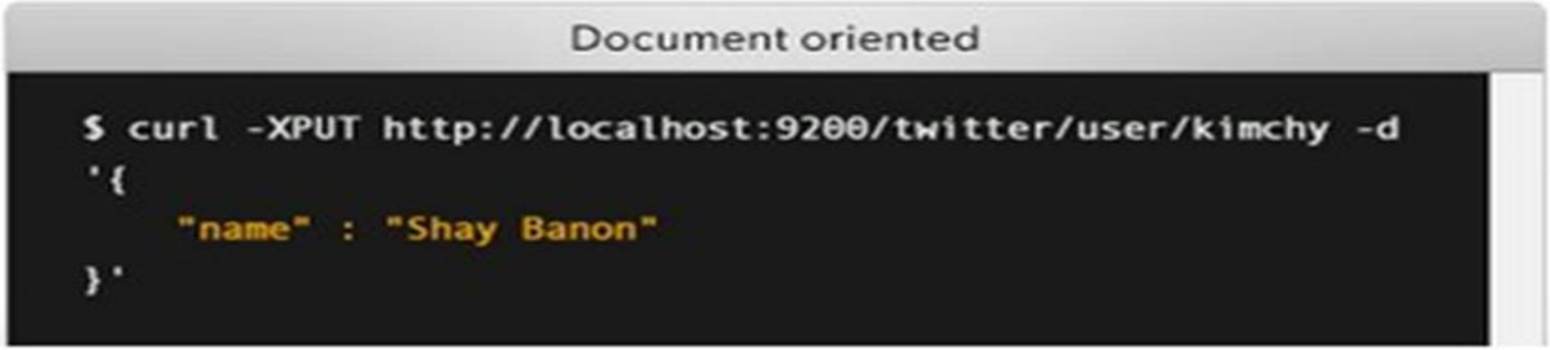
Elastic search uses Lucene behind the scenes to provide the most powerful full text search capabilities available in any open-source project.



Elastic search is bonsai cool

Document Oriented

- Store complex real world entites in Elasticsearch as structured JSON documents.



Document oriented

```
$ curl -XPUT http://localhost:9200/twitter/user/kimchy -d
'{
    "name" : "Shay Banon"
}'
```

Schema Free

Elastic search takes a JSON document and it will detect the data structure, index of the structure , index the data and make it searchable.

```
$ curl -XPUT
http://localhost:9200/twitter/user/
    "name" : "Shay Banon"
}'
$ curl -XPUT http://localhost:920
-d '{
```

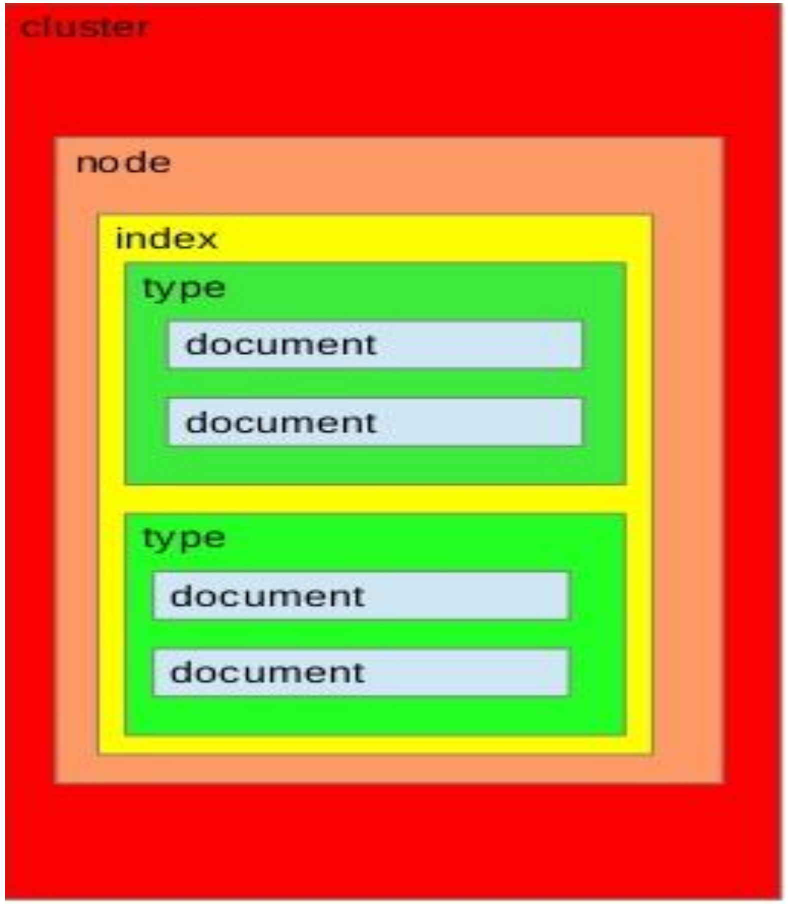
```
url -XPUT http:/
'{
 "user": "kimchy",
```

Terminology

MySQL	Elastic Search
Database	Index
Table	Type
Row	Document
Column	Field
Schema	Mapping
Index	Everything is indexed
SQL	Query DSL
SELECT * FROM table …	GET http://…
UPDATE table SET …	PUT http://…

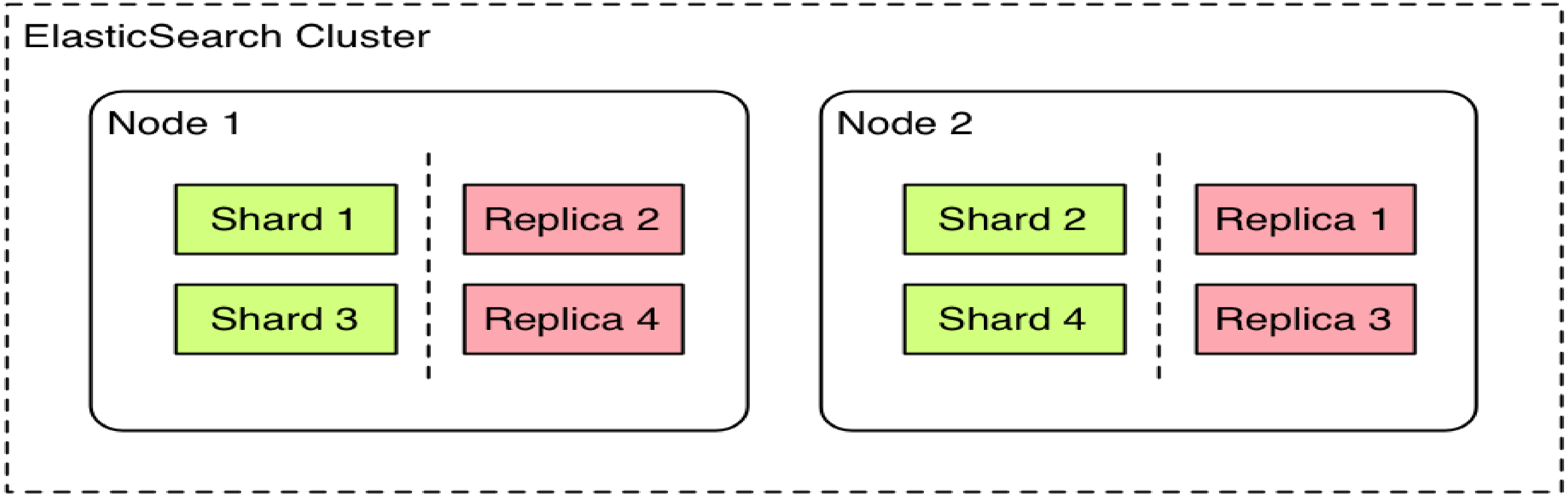
Index, Document and Type

- Index: A collection of documents that have same characteristics
- Document: Basic unit of information.



Node, Cluster and Shard

- Any time that you start an instance of Elasticsearch, you are starting a *node*. A collection of connected nodes is called a cluster.



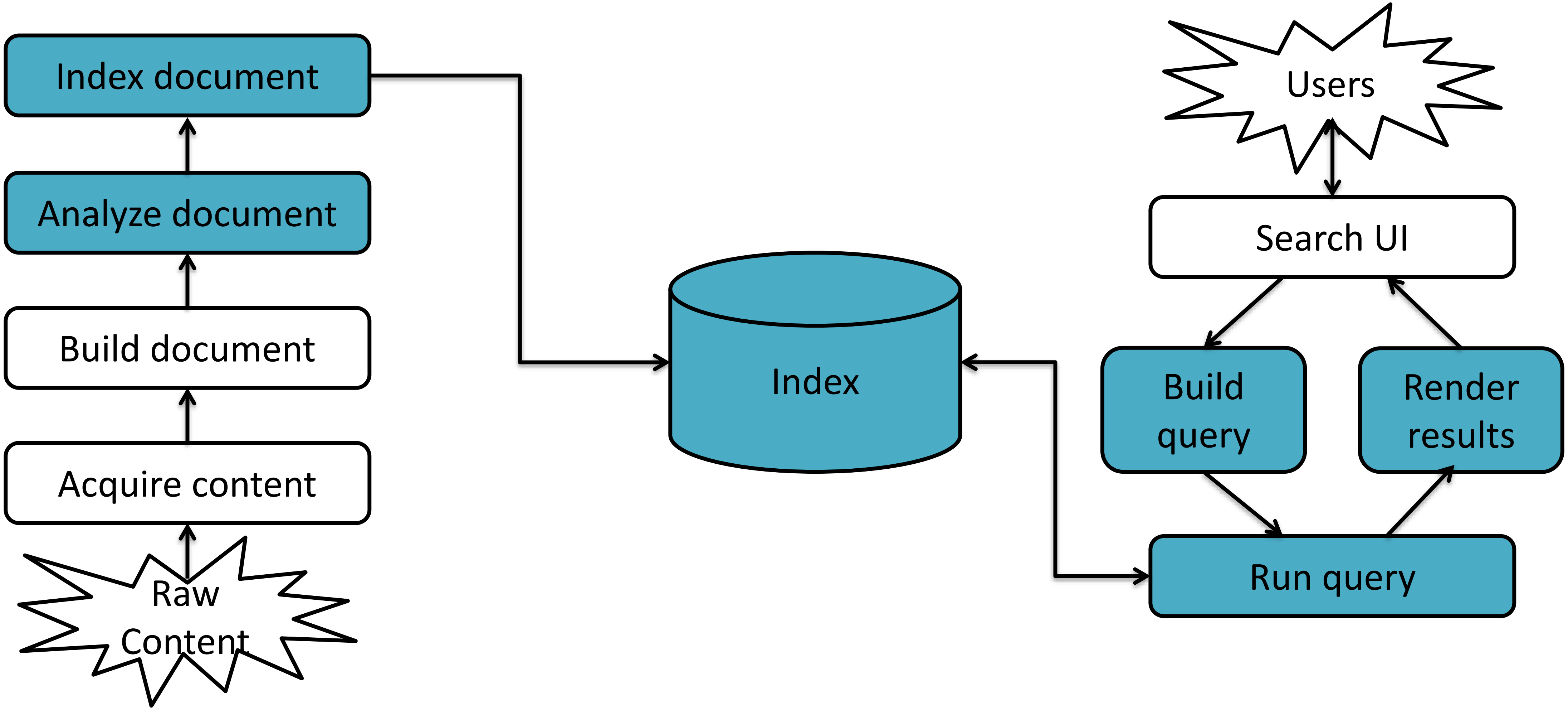
What is Lucene

- High performance, scalable, full-text search library

- Focus: Indexing + Searching Documents

- 100% Java, no dependencies, no config files

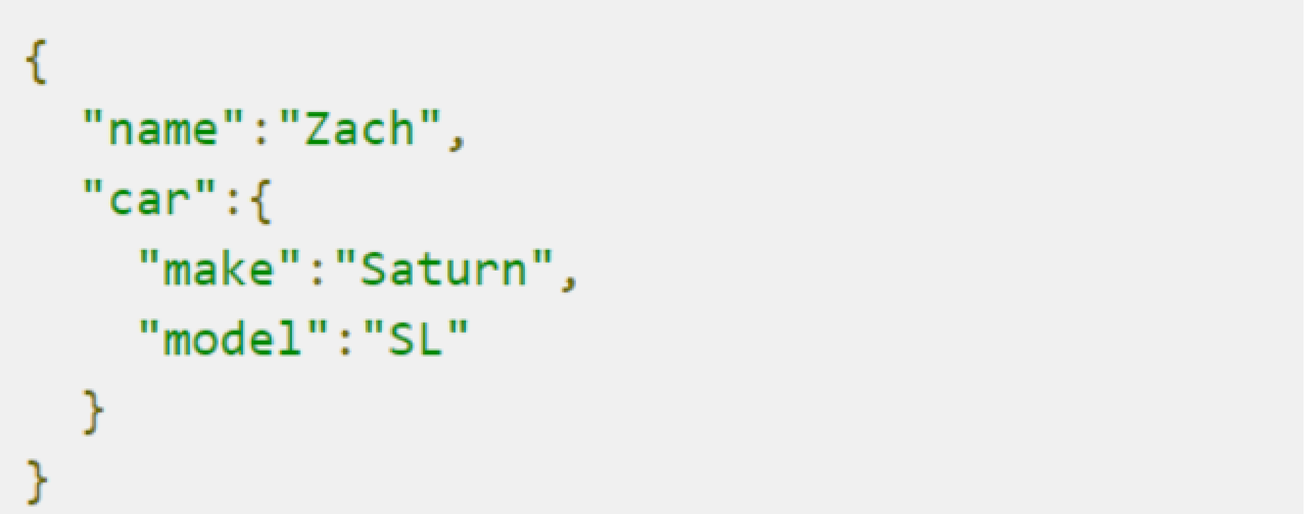
Lucene in a search system



Modeling of Data

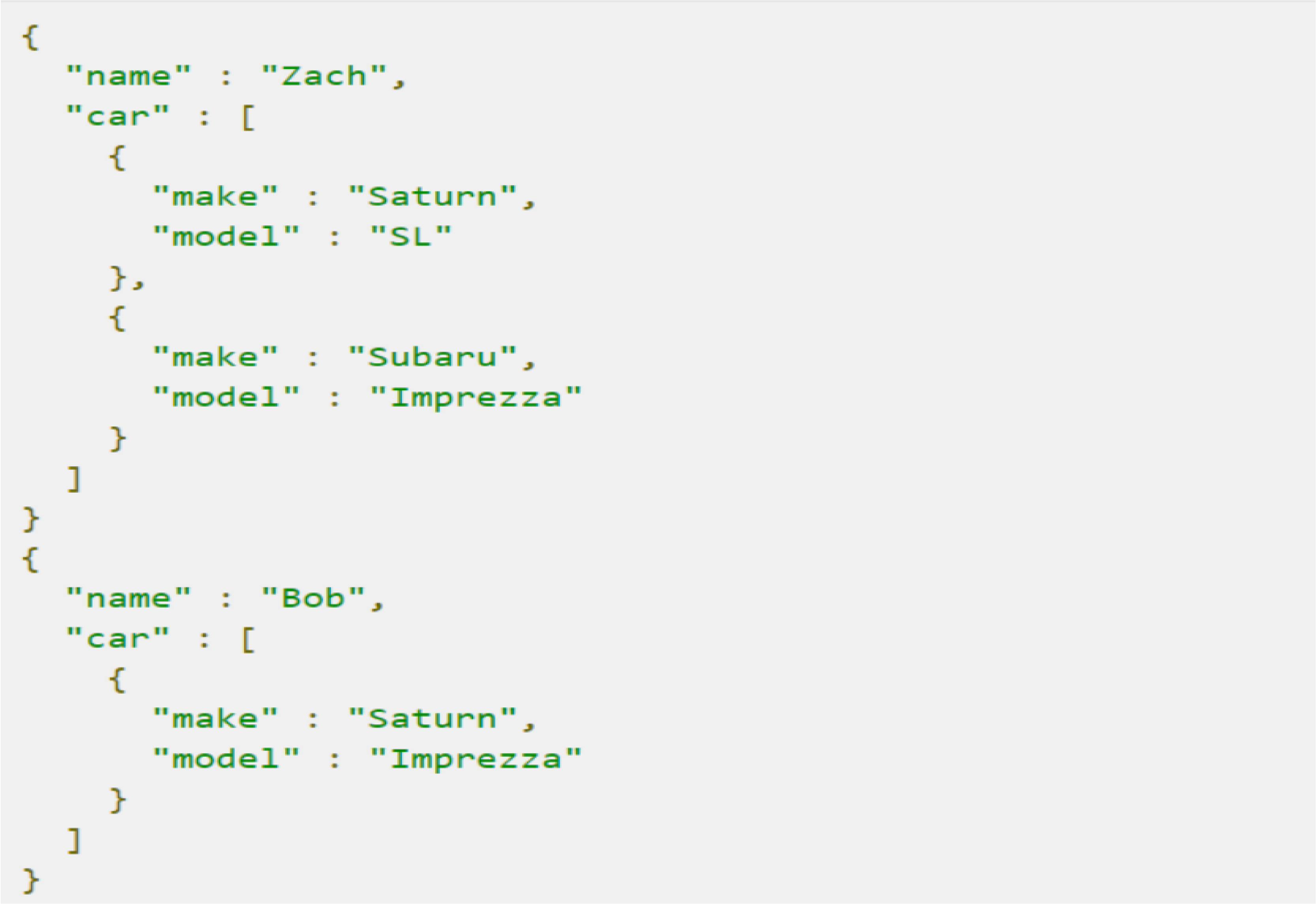
Inner Objects

- JSON objects inside your parent document



```
{
  "name":"Zach",
  "car":{
    "make":"Saturn",
    "model":"SL"
  }
}
```

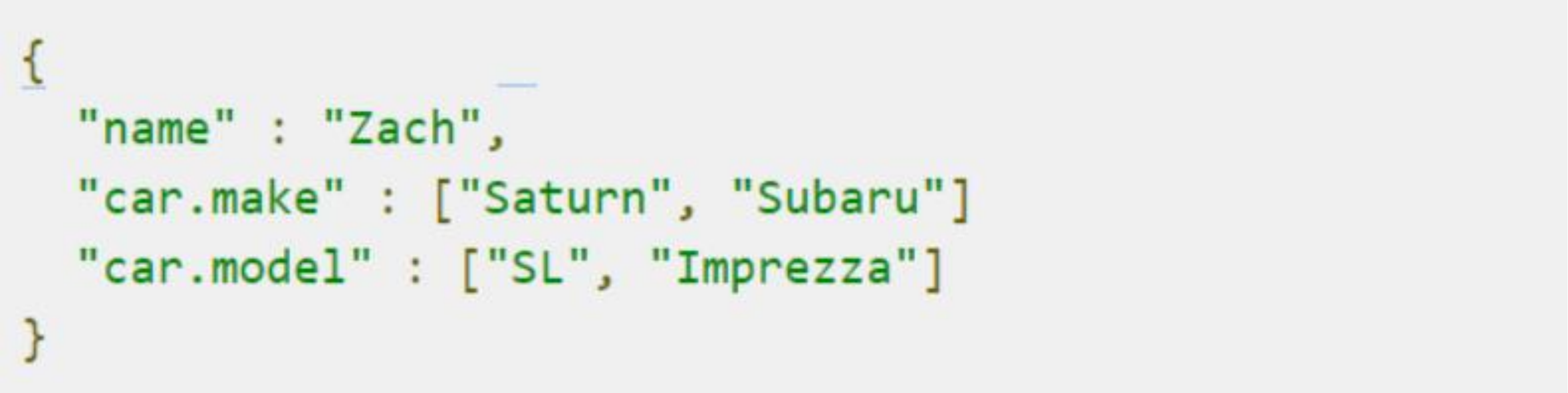
Example:



```
{
  "name" : "Zach",
  "car" : [
    {
      "make" : "Saturn",
      "model" : "SL"
    },
    {
      "make" : "Subaru",
      "model" : "Imprezza"
    }
  ]
}
{
  "name" : "Bob",
  "car" : [
    {
      "make" : "Saturn",
      "model" : "Imprezza"
    }
  ]
}
```

- `query: car.make=Saturn AND car.model=Imprezza`
- If you perform that query, you'll receive both documents as the result which is incorrect.

- Reason: Internally the documents are represented as flattened fields



```
{
  "name" : "Zach",
  "car.make" : ["Saturn", "Subaru"]
  "car.model" : ["SL", "Imprezza"]
}
```

➢ Pros:

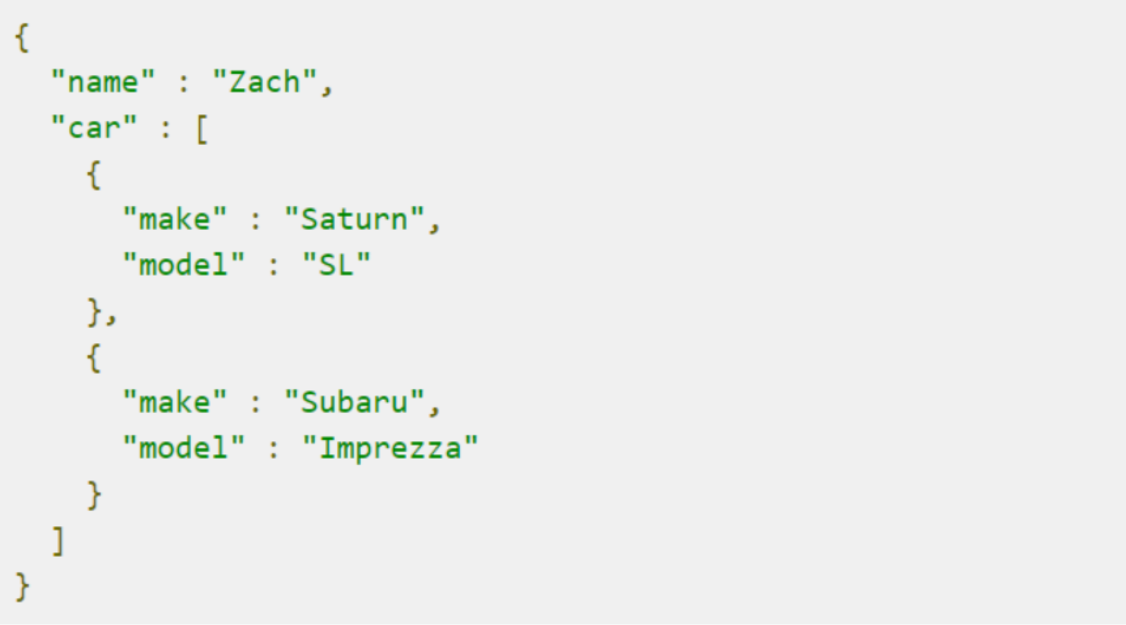
- Easy, fast performance
- No need of special queries

➢ Cons:

- Only applicable when one to one relationships

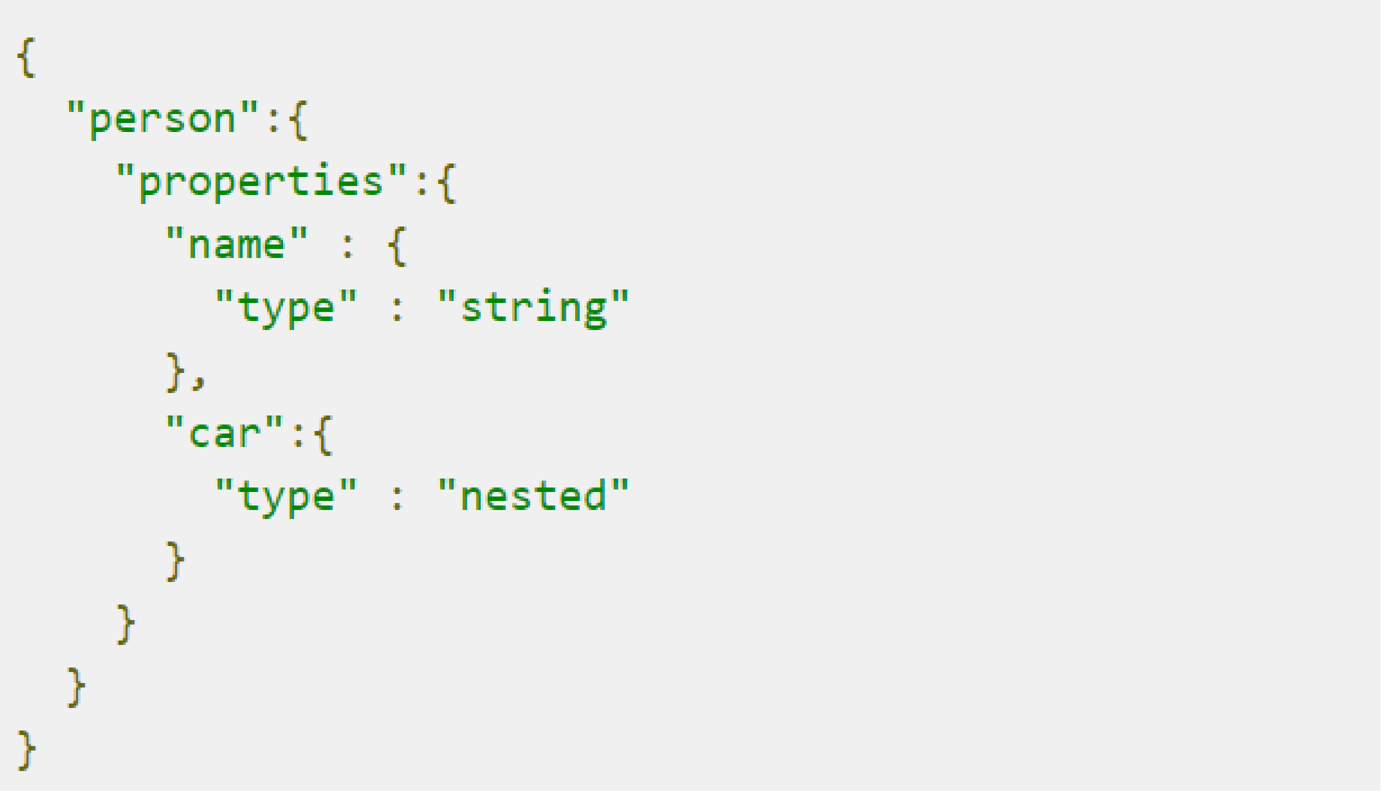
Nested

- As an alternative to inner objects, Elasticsearch provides the concept of "**nested types**".
- Example of a nested document:



```
{
  "name" : "Zach",
  "car" : [
    {
      "make" : "Saturn",
      "model" : "SL"
    },
    {
      "make" : "Subaru",
      "model" : "Imprezza"
    }
  ]
}
```

- At the mapping level, nested types must be explicitly declared (unlike inner objects, which are automatically detected):



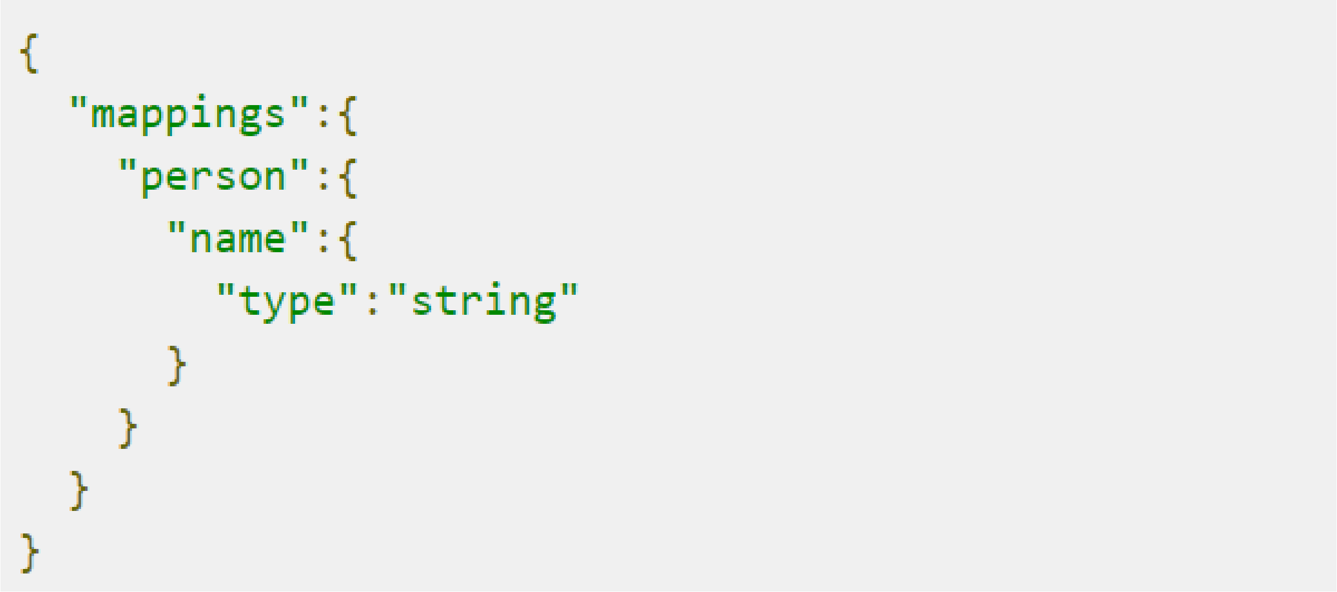
```
{
  "person":{
    "properties":{
      "name" : {
        "type" : "string"
      },
      "car":{
        "type" : "nested"
      }
    }
  }
}
```

- Pros: The earlier search query returns correct results.
- Reason: The root and the nested objects are saved as separate documents on same lucene block on the same shard to improve performance and are related internally.

- Cons:
- A special nested query is required.
- Any update to root or nested object requires reindexing of the entire document to a new lucene block, ie, unnecessary overhead.
- Best suited for data that does not change frequently

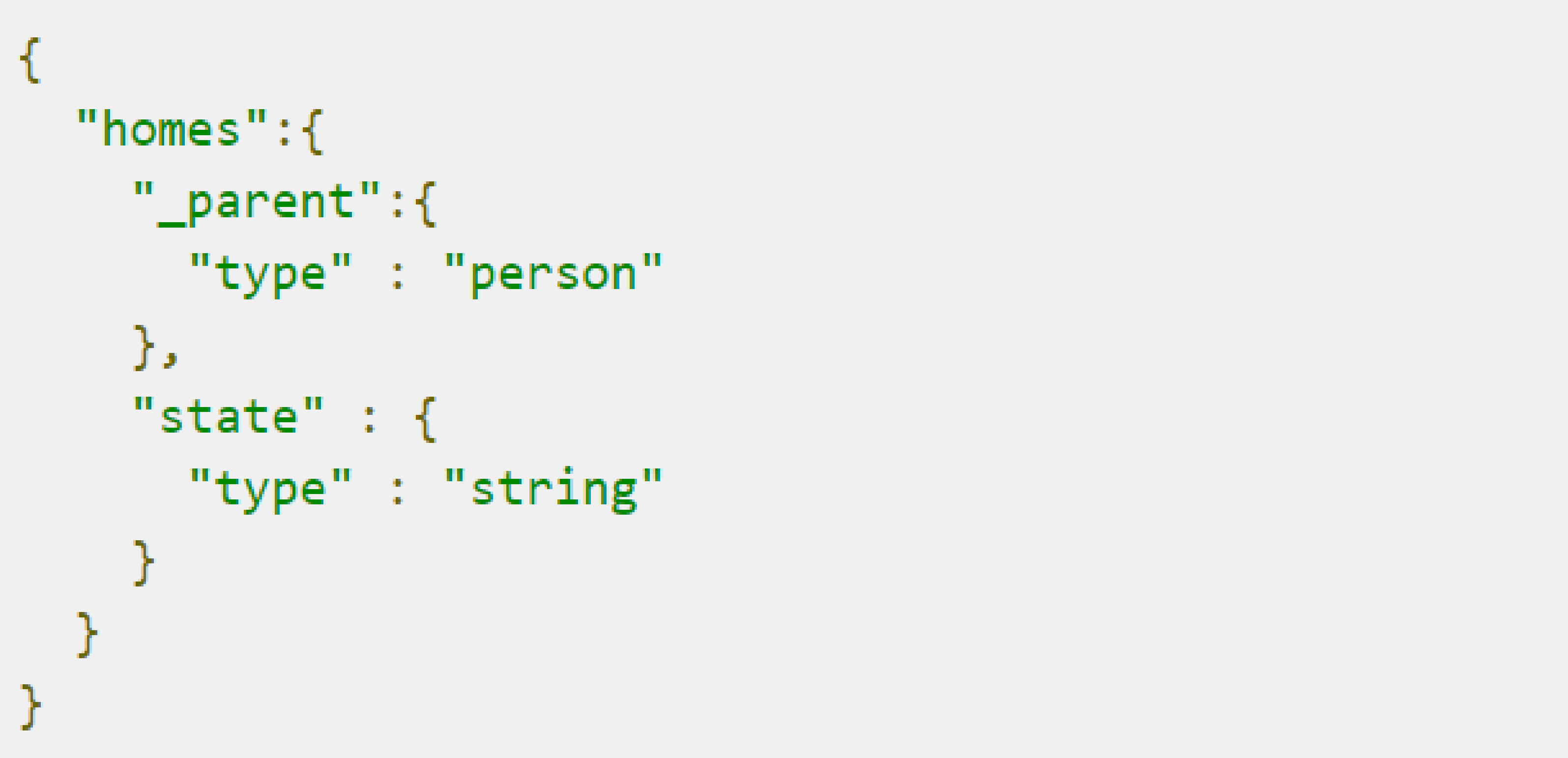
Parent/Child

- The next method that Elasticsearch provides are **Parent/Child types**
- Example of parent mapping:



```
{
  "mappings":{
    "person":{
      "name":{
        "type":"string"
      }
    }
  }
}
```

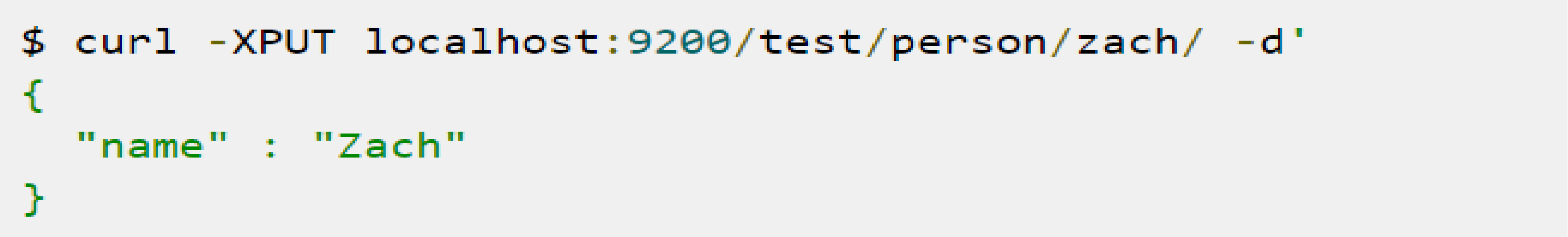
- Example of child mapping:



```
{
  "homes":{
    "_parent":{
      "type" : "person"
    },
    "state" : {
      "type" : "string"
    }
  }
}
```

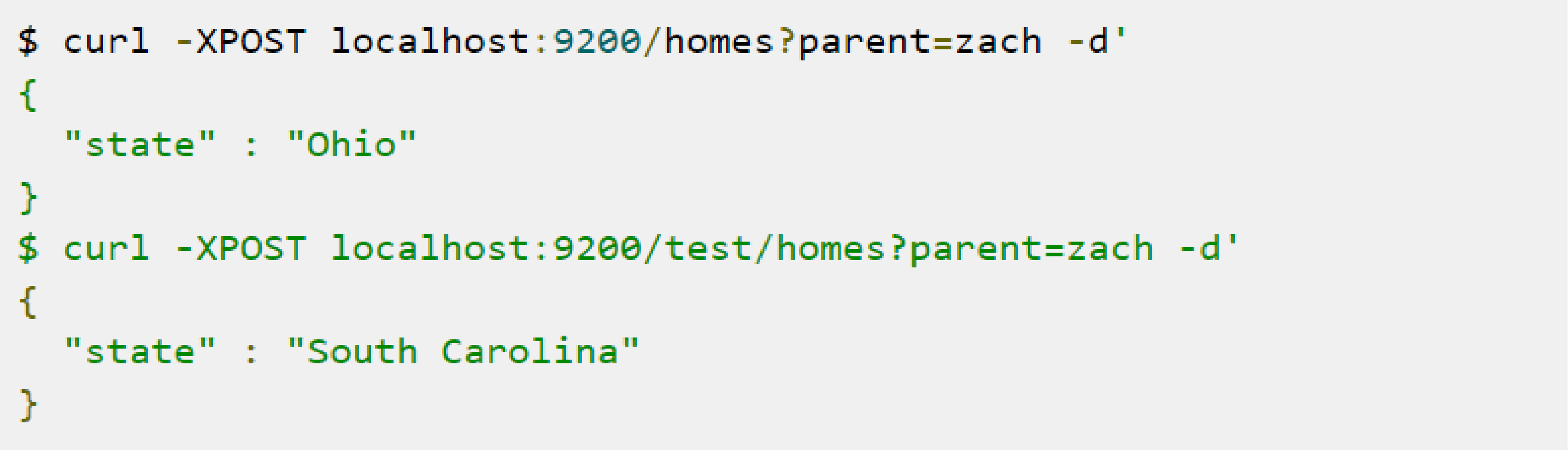
- The children have their own mapping outside the parent, with a special `_parent` property set.

- The parent doc is indexed as normal:



```
$ curl -XPUT localhost:9200/test/person/zach/ -d'
{
  "name" : "Zach"
}
```

- For indexing children documents, you need to specify which parent this child belongs to in the query parameter



```
$ curl -XPOST localhost:9200/homes?parent=zach -d'
{
  "state" : "Ohio"
}
$ curl -XPOST localhost:9200/test/homes?parent=zach -d'
{
  "state" : "South Carolina"
}
```

Pros:

- Saves us from the overhead of reindexing when updating

Cons:

- Less performance
- More memory intensive

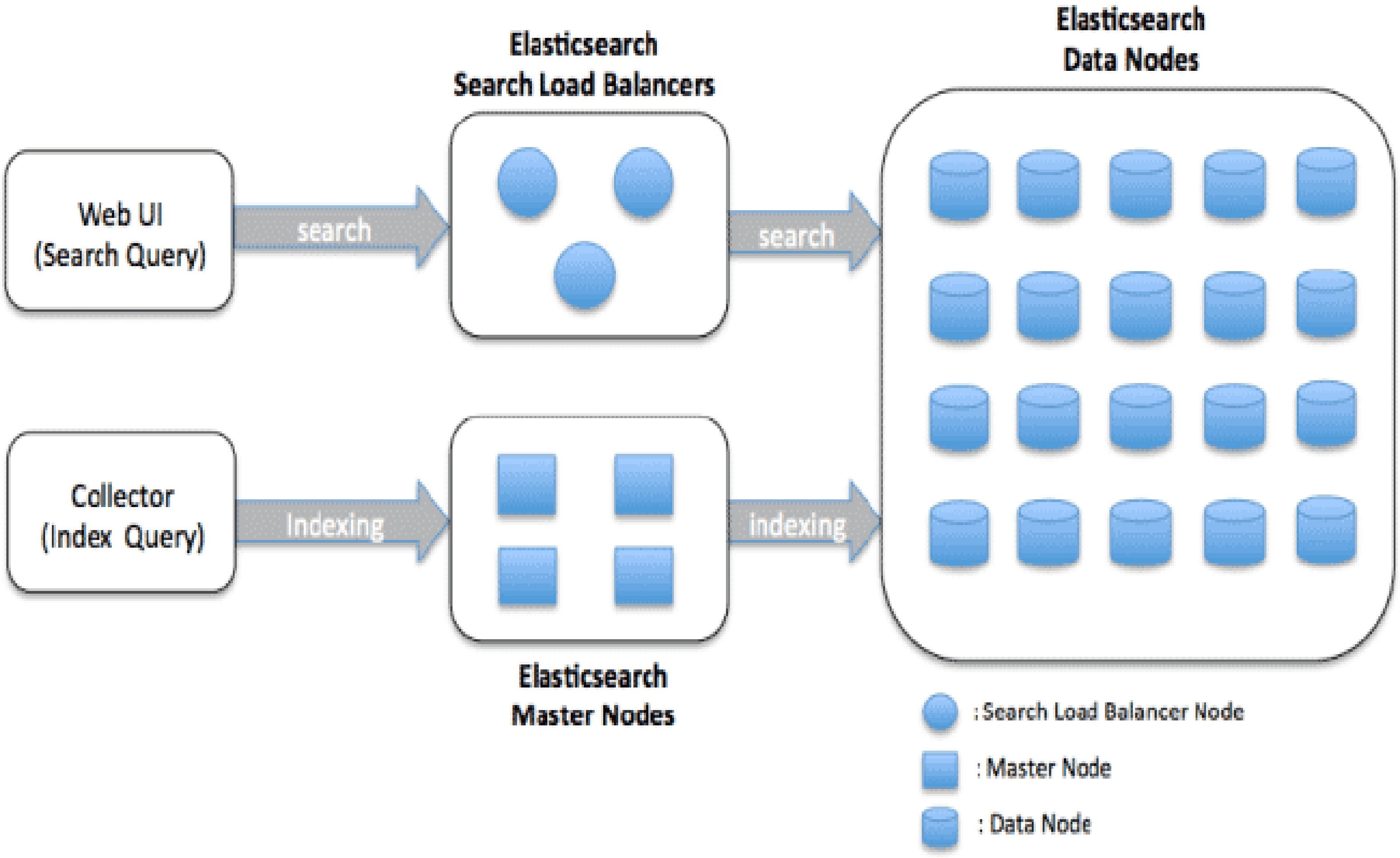
Denormalization

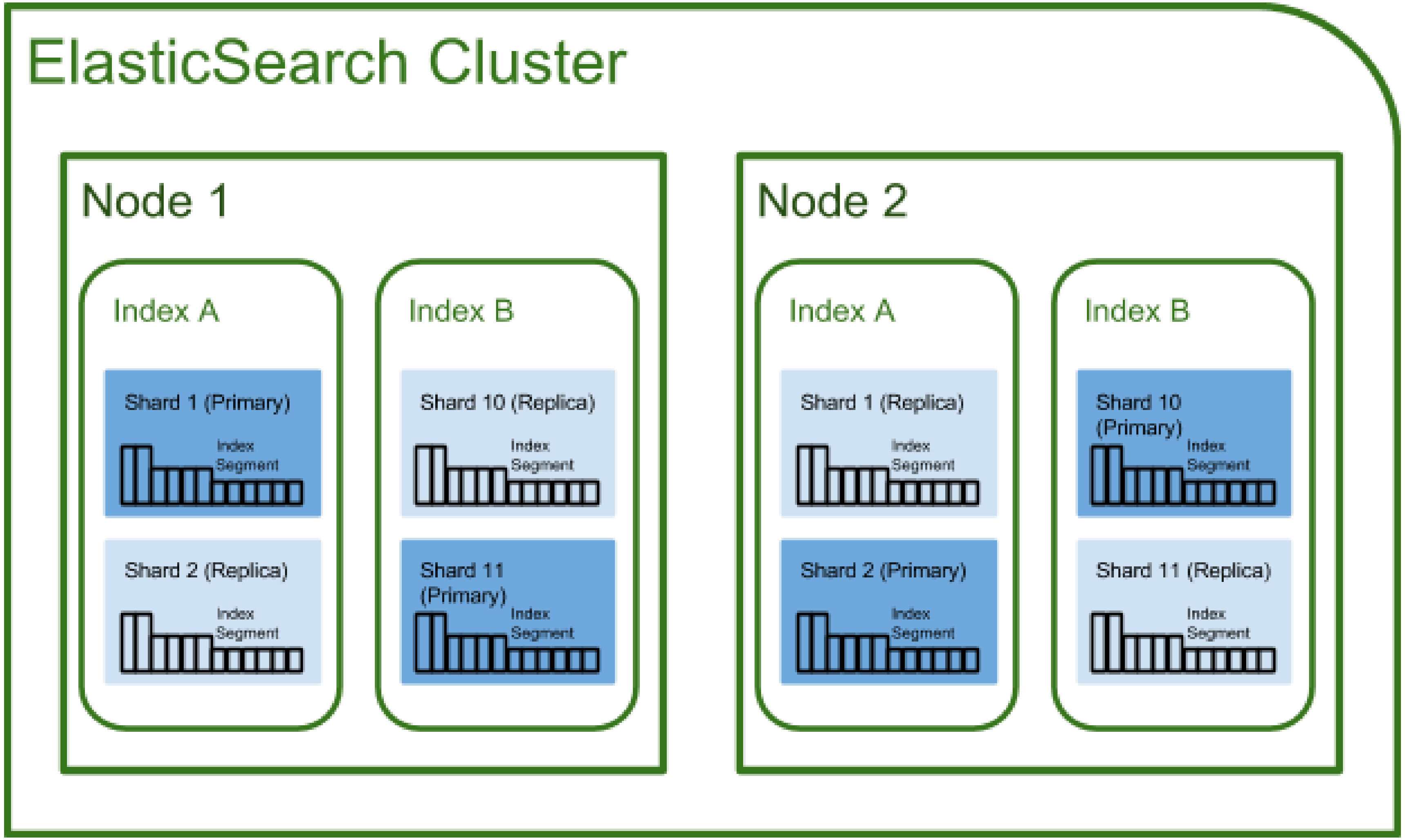
- Relations are not always required
- We should judiciously choose which data to normalize and when we need queries to retrieve children.

- Denormalization provides us with the following powers:
- ▪ We can manage relationships ourselves
- ▪ More flexibility
- ▪ Can be more/less performant depending on the setup

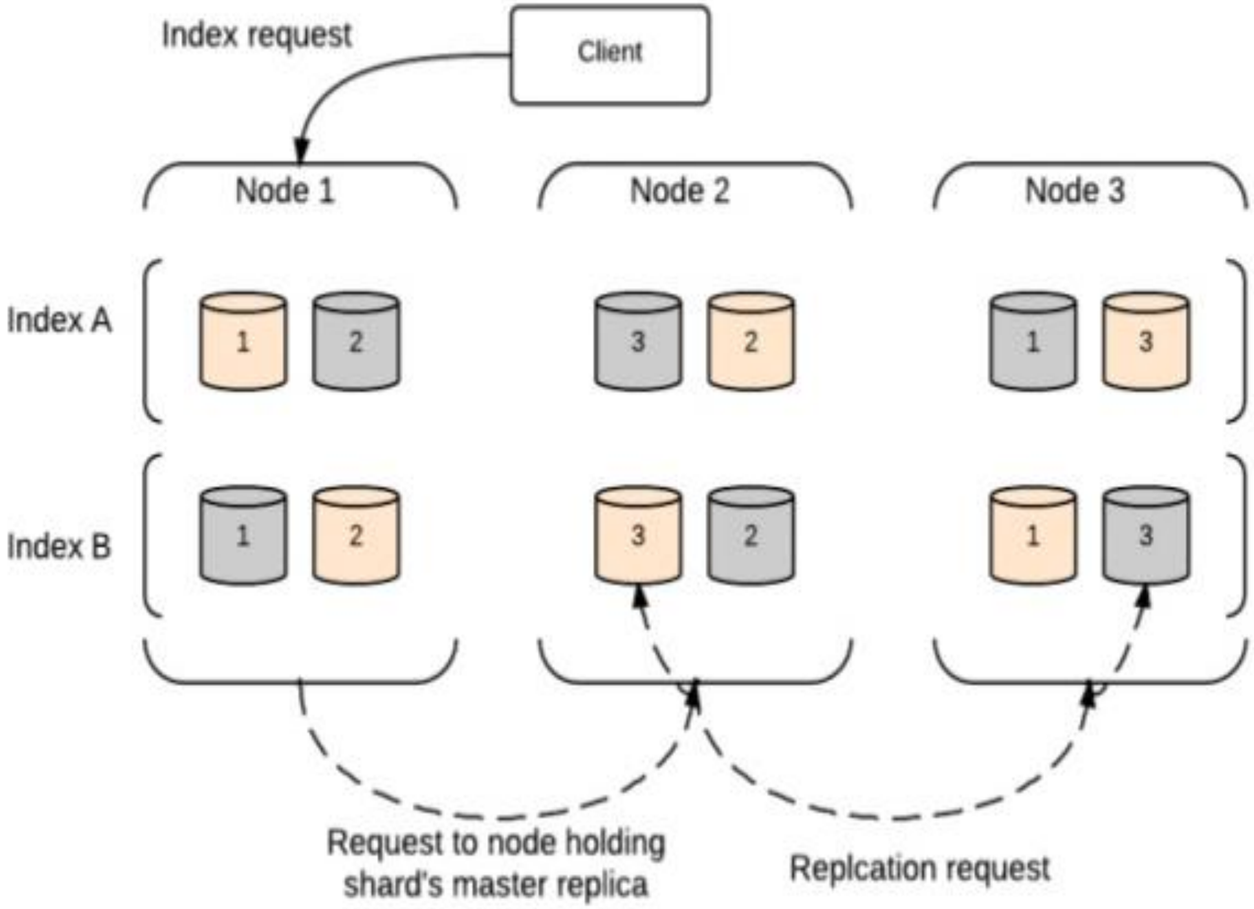
ARCHITECTURE

- Highly Distributed
- Node is single instance of Elasticsearch.
- Communicate each other via network calls.
- There is a master node that organizes the cluster and transfers the request to the other data nodes.
- A node is configured as master node by setting node.master property to be true in elasticsearch.yml file
- Data nodes provide the necessary result transfers to the client.

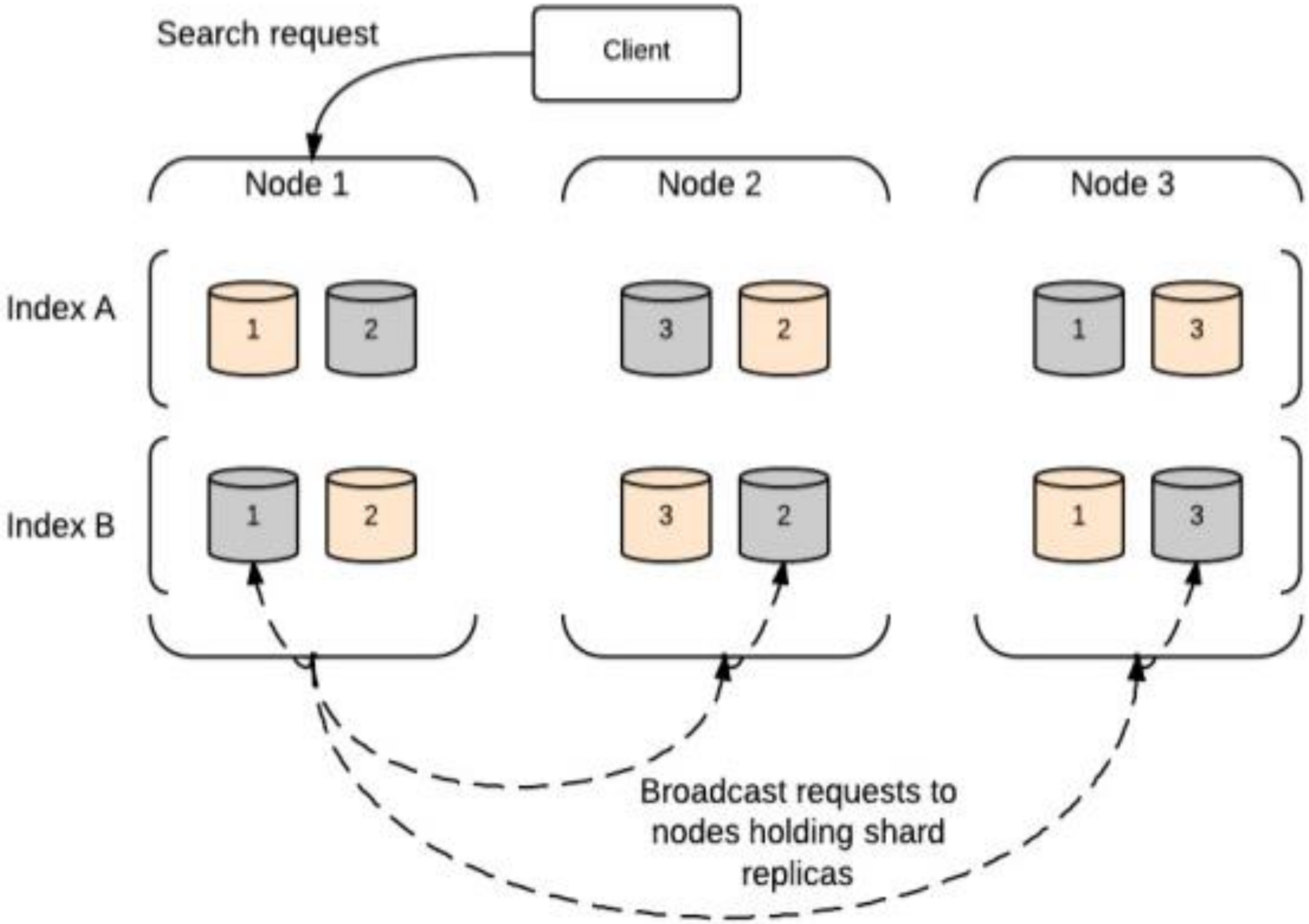

Elasticsearch
Search Load Balancers
Elasticsearch
Data Nodes
Web UI
(Search Query)
search
search
Collector
(Index Query)
Indexing
indexing
Elasticsearch
Master Nodes
● : Search Load Balancer Node
■ : Master Node
: Data Node


ElasticSearch Cluster
Node 1
Index A
Shard 1 (Primary)
Index Segment
Shard 2 (Replica)
Index Segment
Index B
Shard 10 (Replica)
Index Segment
Shard 11 (Primary)
Index Segment
Node 2
Index A
Shard 1 (Replica)
Index Segment
Shard 2 (Primary)
Index Segment
Index B
Shard 10 (Primary)
Index Segment
Shard 11 (Replica)
Index Segment

Index Request



Search Request



QUERY LANGUAGE
AND
FEW OPERATIONS

QUERY LANGUAGE-INTRO.

- Elasticsearch provides a JSON-style domain specific language known as Query DSL.
- Basic queries can be done using only query string parameters in URL.
- Let us take the following example:

```
GET /_search
{
        “query”: { “match_all”: { } }
}
```

- A query DSL consists of two types of clauses:

Leaf query clauses

Compound query clauses

- **Leaf Query Clauses:**
  - These are used to compare field/fields to a query string.

- **Compound Clauses:**
  - Merging other query clauses.
  - Combine a leaf as well as other compound clauses.
  - These queries are nested.

```
ex: {
       “bool”:{
         “must”:    {“match”: {“tweet”:”elasticsearch”}},
         “must_not”: {“match”:  {“name”: “Mary”}},
              “filter” :  { “range”: {“age” : { “gt”:30}}}
        }
    }
```

- Requests are in JSON format.
- No JSON schema required.
- The requests are in the form of REST APIs.
- General request is of the form:

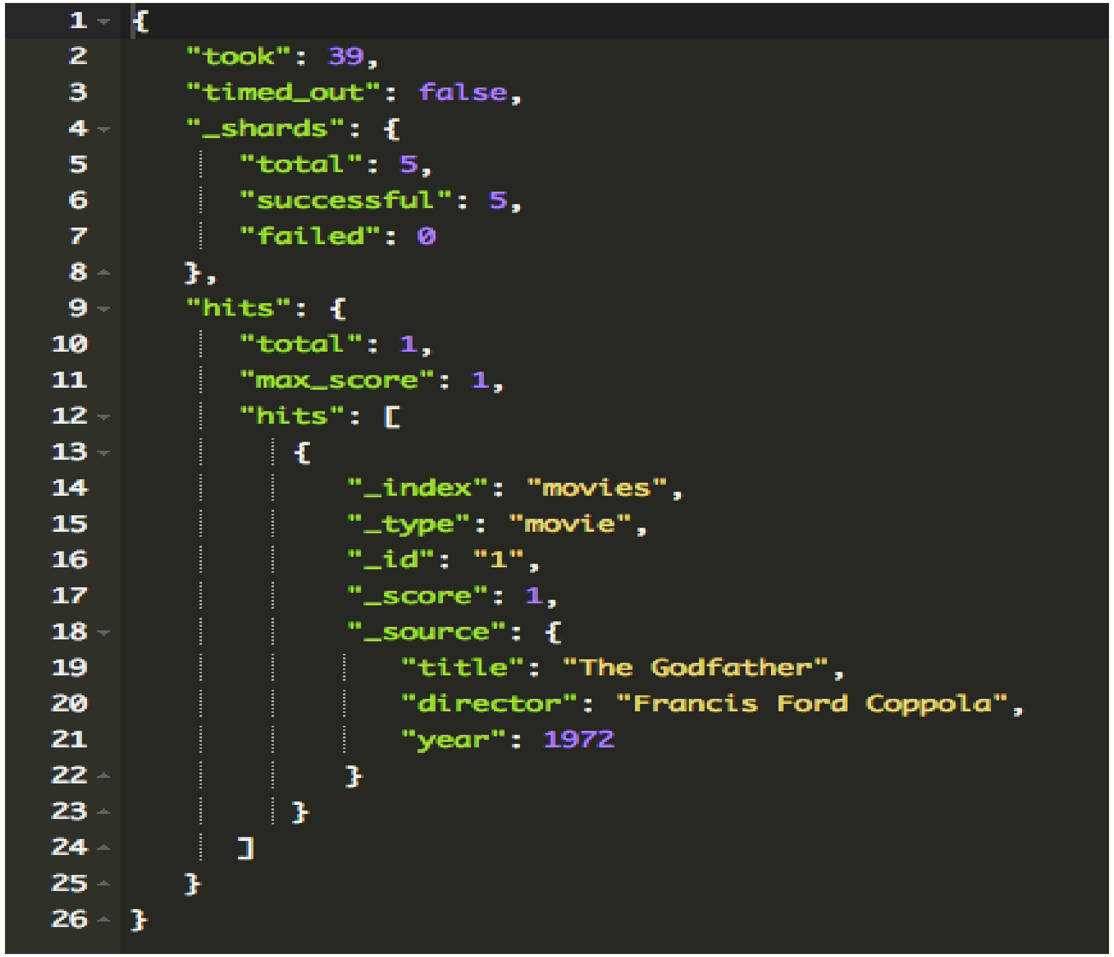
```
    curl –X(GET/POST/PUT/DELETE) “http://{server
name}/<index>/....” –d’
{
    //fields and data here
}
```

INDEX CREATION

```
http://localhost:9200/<index>/<type>/[<id>]
```

```
curl -XPUT  "http://localhost:9200/movies/movie/1" -d‘  {
          "title": "The Godfather",
          "director": "Francis Ford Coppola",
          "year": 1972
}'
```

INDEX CREATION RESPONSE



```
{
    "took": 39,
    "timed_out": false,
    "_shards": {
        "total": 5,
        "successful": 5,
        "failed": 0
    },
    "hits": {
        "total": 1,
        "max_score": 1,
        "hits": [
            {
                "_index": "movies",
                "_type": "movie",
                "_id": "1",
                "_score": 1,
                "_source": {
                    "title": "The Godfather",
                    "director": "Francis Ford Coppola",
                    "year": 1972
                }
            }
        ]
    }
}
```

MECHANISM OF INDEX CREATION

- All nodes in Elasticsearch have metadata about which shard lives in which node.
- Elasticsearch uses the murmur-hash function to determine in which shard document should be indexed in.

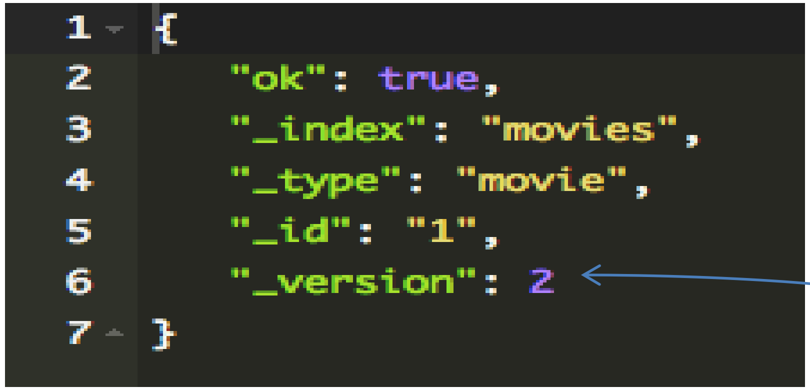
  shard= hash(document_id)%(number of primary shards)

- The memory buffer is refreshed at regular intervals(default: 1second) and contents are written to a new segment.

UPDATE

```
curl -XPUT "http://localhost:9200/movies/movie/1" -d' {
    "title": "The Godfather",
    "director": "Francis Ford Coppola",
    "year": 1972,
    "genres": ["Crime", "Drama"]
}'
```

New field

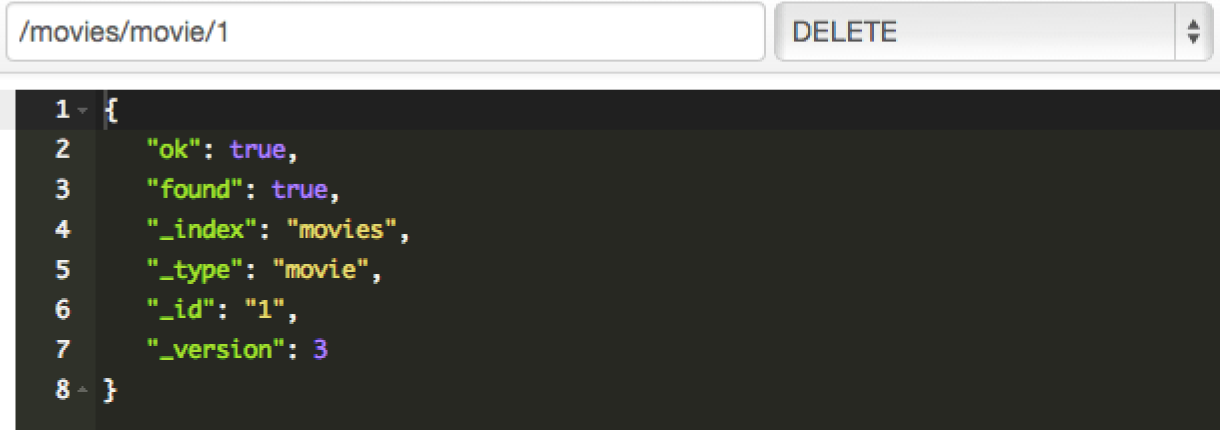


```
{
    "ok": true,
    "_index": "movies",
    "_type": "movie",
    "_id": "1",
    "_version": 2
}
```

Updated Version

DELETE

```
curl -XDELETE "http://localhost:9200/movies/movie/1" -d''
```



/movies/movie/1

DELETE

```
{
   "ok": true,
   "found": true,
   "_index": "movies",
   "_type": "movie",
   "_id": "1",
   "_version": 3
}
```

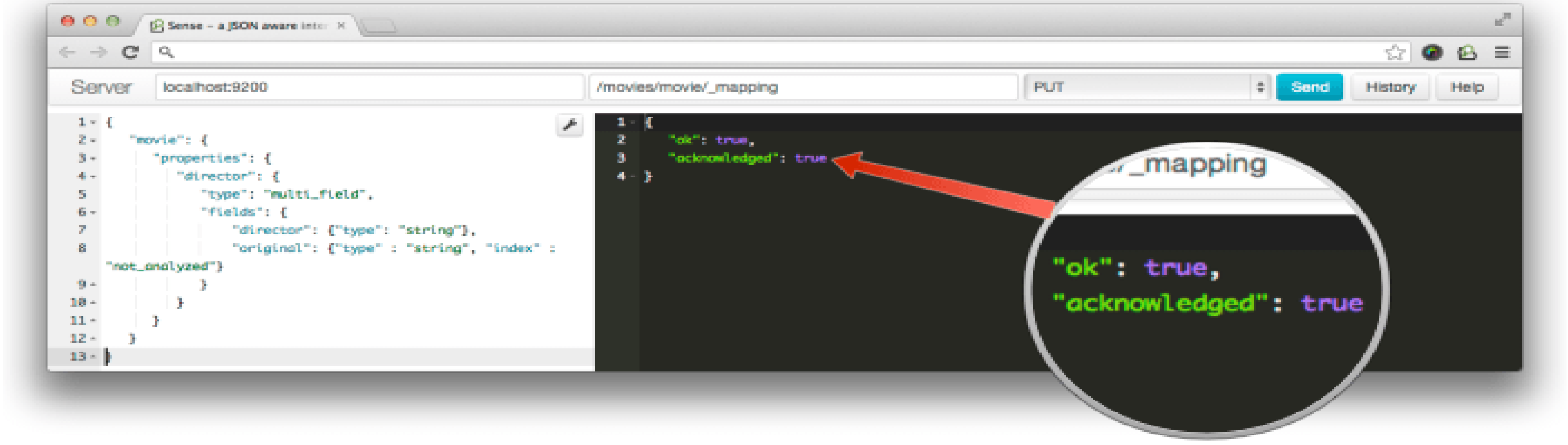
DELETE AND UPDATE MECHANISMS.

- **IMPORTANT: Documents in Elasticsearch are immutable**
- Existence of .del file in disk segment.
- When a delete request is sent, document is not really deleted, but marked as deleted in the .del file. While merging segments, the documents marked deleted won’t appear in new one.

- A version number is given to every newly created document.
- Every change to the document results in a new version number.
- When update is performed, the old version is marked as deleted in the .del file and new version is indexed.

Updating *existing* Mapping

```
curl -XPUT "http://localhost:9200/movies/movie/_mapping" -d'
{
  "movie": {
    "properties": {
      "director": {
        "type": "multi_field",
        "fields": {
            "director": {"type": "string"},
            "original": {"type" : "string", "index" : "not_analyzed"}
        }
      }
    }
  }
}'
```



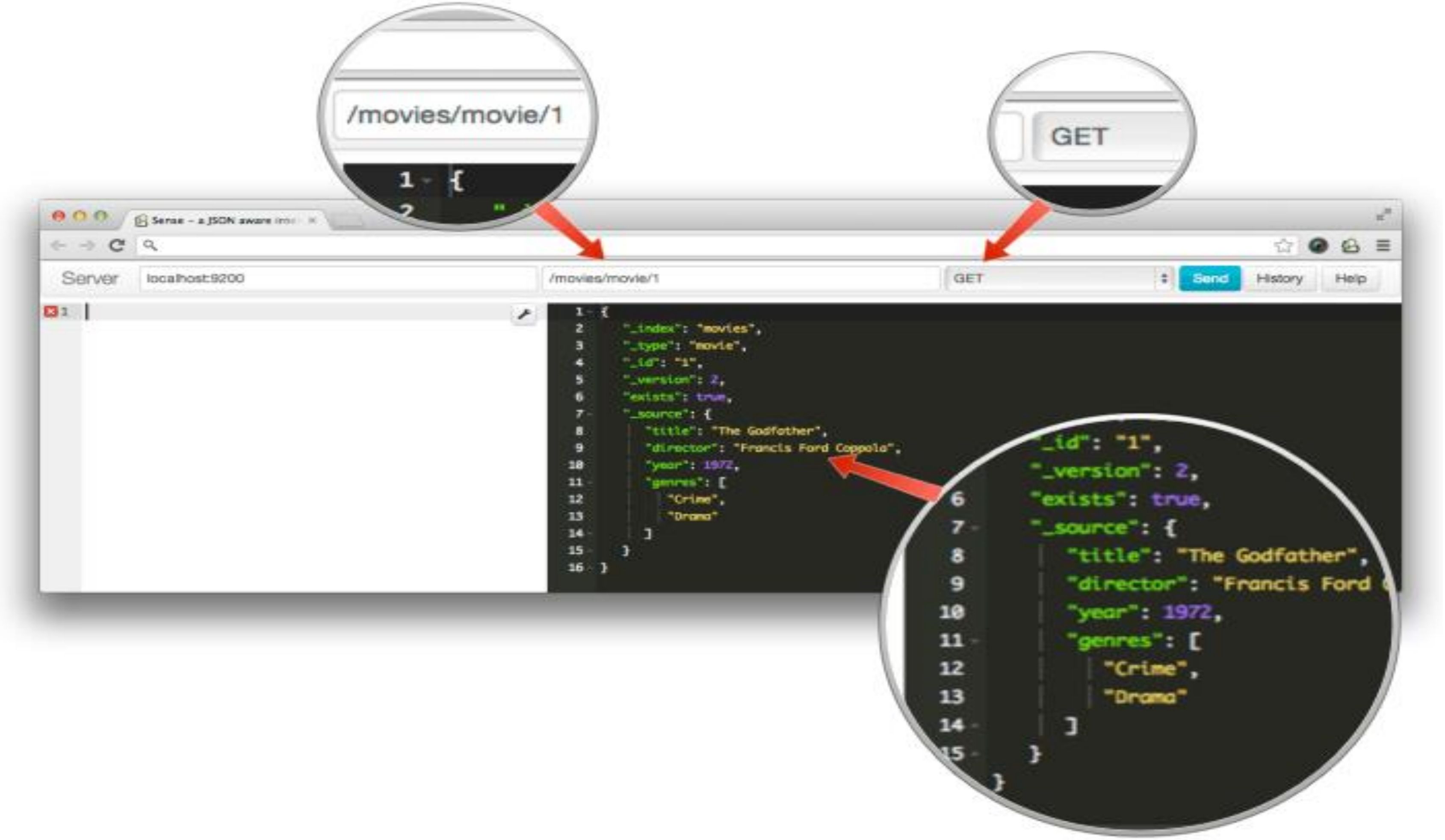
Server localhost:9200 /movies/movie/_mapping PUT

```
{
  "movie": {
    "properties": {
      "director": {
        "type": "multi_field",
        "fields": {
          "director": {"type": "string"},
          "original": {"type" : "string", "index" :
"not_analyzed"}
        }
      }
    }
  }
}
```

```
{
  "ok": true,
  "acknowledged": true
}
```

GET

```
curl -XGET "http://localhost:9200/movies/movie/1" -d"
```



Server localhost:9200 /movies/movie/1 GET

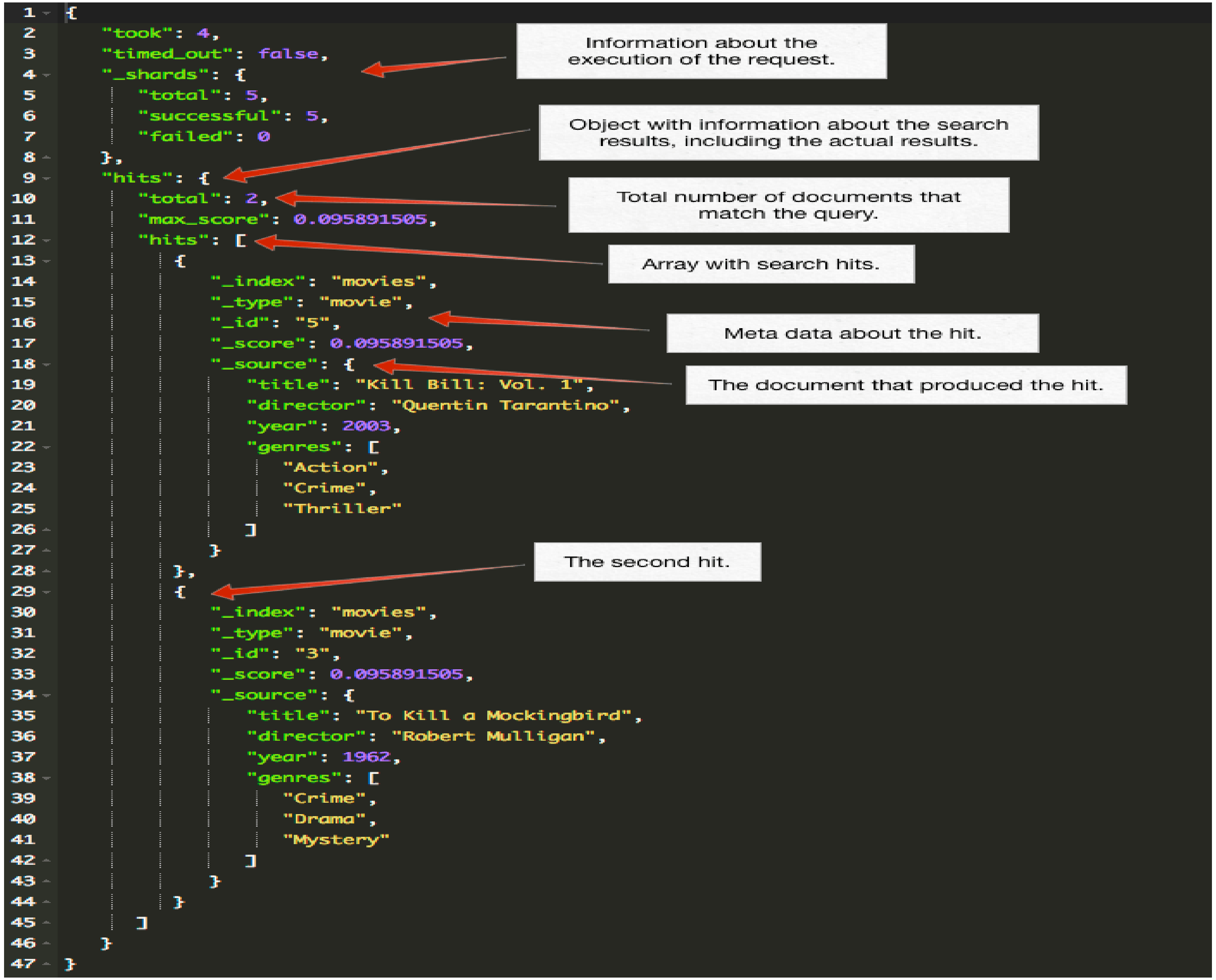
```
{
    "_index": "movies",
    "_type": "movie",
    "_id": "1",
    "_version": 2,
    "exists": true,
    "_source": {
        "title": "The Godfather",
        "director": "Francis Ford Coppola",
        "year": 1972,
        "genres": [
            "Crime",
            "Drama"
        ]
    }
}
```

SEARCH

- Search across all indexes and all types
  - http://localhost:9200/_search
- Search across all types in the movies index.
  - http://localhost:9200/movies/_search
- Search explicitly for documents of type movie within the movies index.
  - http://localhost:9200/movies/movie/_search

```
curl -XPOST "http://localhost:9200/_search" -d'
{
    "query": {
        "query_string": {
            "query": "kill"
        }
    }
}'
```

SEARCH RESPONSE



```
{
    "took": 4,
    "timed_out": false,
    "_shards": {
        "total": 5,
        "successful": 5,
        "failed": 0
    },
    "hits": {
        "total": 2,
        "max_score": 0.095891505,
        "hits": [
            {
                "_index": "movies",
                "_type": "movie",
                "_id": "5",
                "_score": 0.095891505,
                "_source": {
                    "title": "Kill Bill: Vol. 1",
                    "director": "Quentin Tarantino",
                    "year": 2003,
                    "genres": [
                        "Action",
                        "Crime",
                        "Thriller"
                    ]
                }
            },
            {
                "_index": "movies",
                "_type": "movie",
                "_id": "3",
                "_score": 0.095891505,
                "_source": {
                    "title": "To Kill a Mockingbird",
                    "director": "Robert Mulligan",
                    "year": 1962,
                    "genres": [
                        "Crime",
                        "Drama",
                        "Mystery"
                    ]
                }
            }
        ]
    }
}
```

Information about the execution of the request.

Object with information about the search results, including the actual results.

Total number of documents that match the query.

Array with search hits.

Meta data about the hit.

The document that produced the hit.

The second hit.

THE READ OR SEARCH OPERATION

- Read operations consist of two phases:
  - Query Phase
  - Fetch Phase

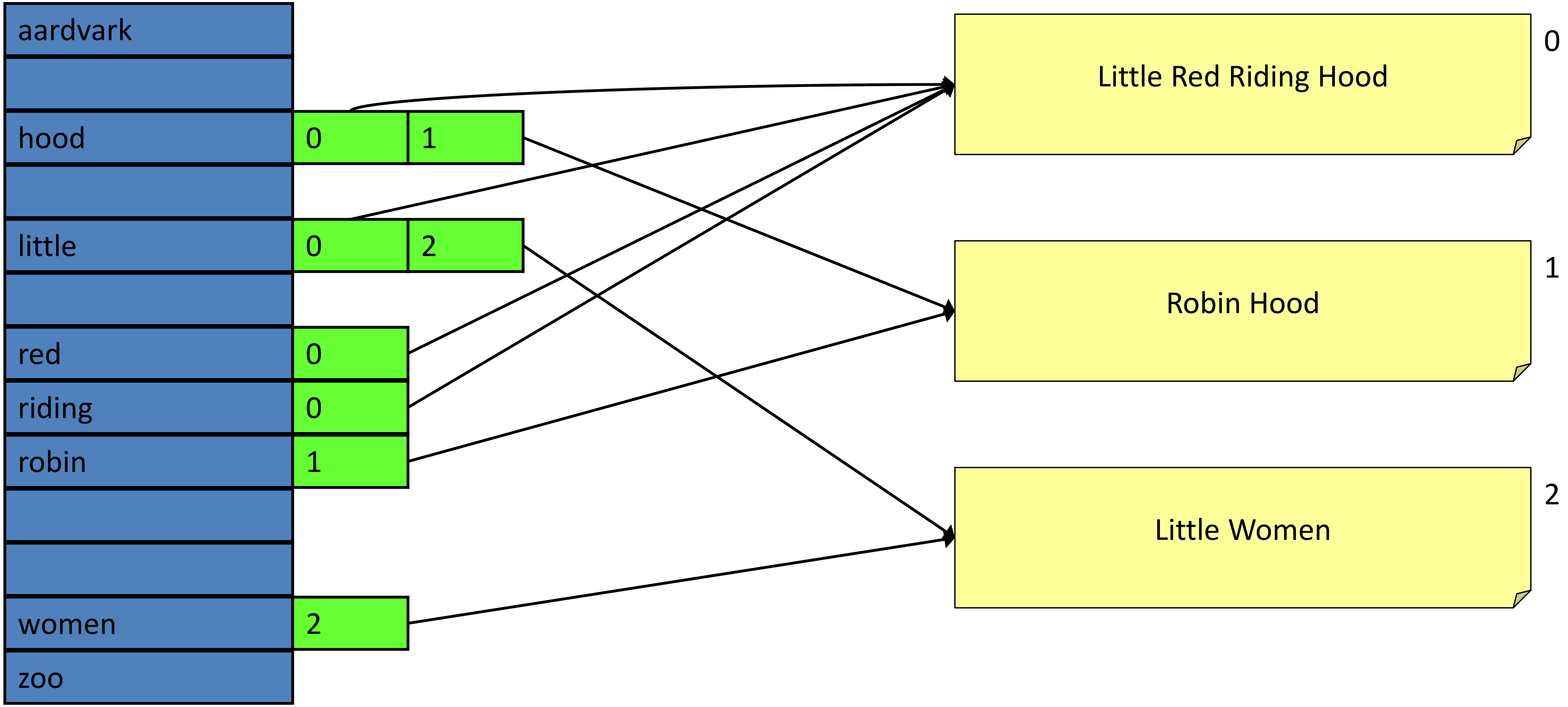
- **Query Phase:**
  - The coordinating node routes the search request to all shards of index.
  - Each shard performs search independently and create a priority queue of results sorted by relevance score.
  - All shards return document ids and relevant scores of the matched documents to the coordinating node.
  - The coordinating node then creates a priority queue and sorts the results globally.

Fetch Phase

- The coordinating node requests original documents from all shards.
- All shards enrich documents and return them to coordinating node.

- Usually searching is carried out in the lucene segments by inverted index.
- The inverted index is composed of two parts:
- Sorted dictionary
- Posting lists

Inverted Index



SEARCH RELEVANCE SCORE

- Relevance score is a score that Elasticsearch assigns to each document returned in their search result.
- Default algorithm used for scoring is tf/idf.
- Where tf or term frequency is the measure of how many times a term appears in a document.
- And idf or inverse document frequency measures how often a term appears in entire index as a percentage of total number of documents in the index.

AGGREGATIONS

- Used for building analytic information over a set of documents.
- Three families of aggregations:
  - Bucketing
    - Bucketing Aggregations can have sub-aggregations. No definite depth.
  - Metric
  - Pipeline

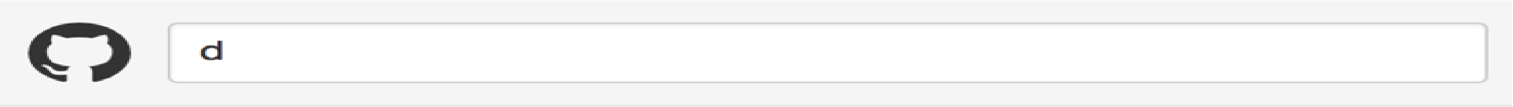
```
"aggregations" : {
    "<aggregation_name>" : {
        "<aggregation_type>" : {
            <aggregation_body>
        }
        [,"meta" : {  [<meta_data_body>] } ]?
        [,"aggregations" : { [<sub_aggregation>]+ } ]?
    }
    [,"<aggregation_name_2>" : { ... } ]*
}
```

Aggregations object holds the aggregations to compute.

Each aggregation has a unique name.

If sub-aggregations are defined under parent aggregation, then these will be computed as well.

AUTO COMPLETION



d

SELECT name

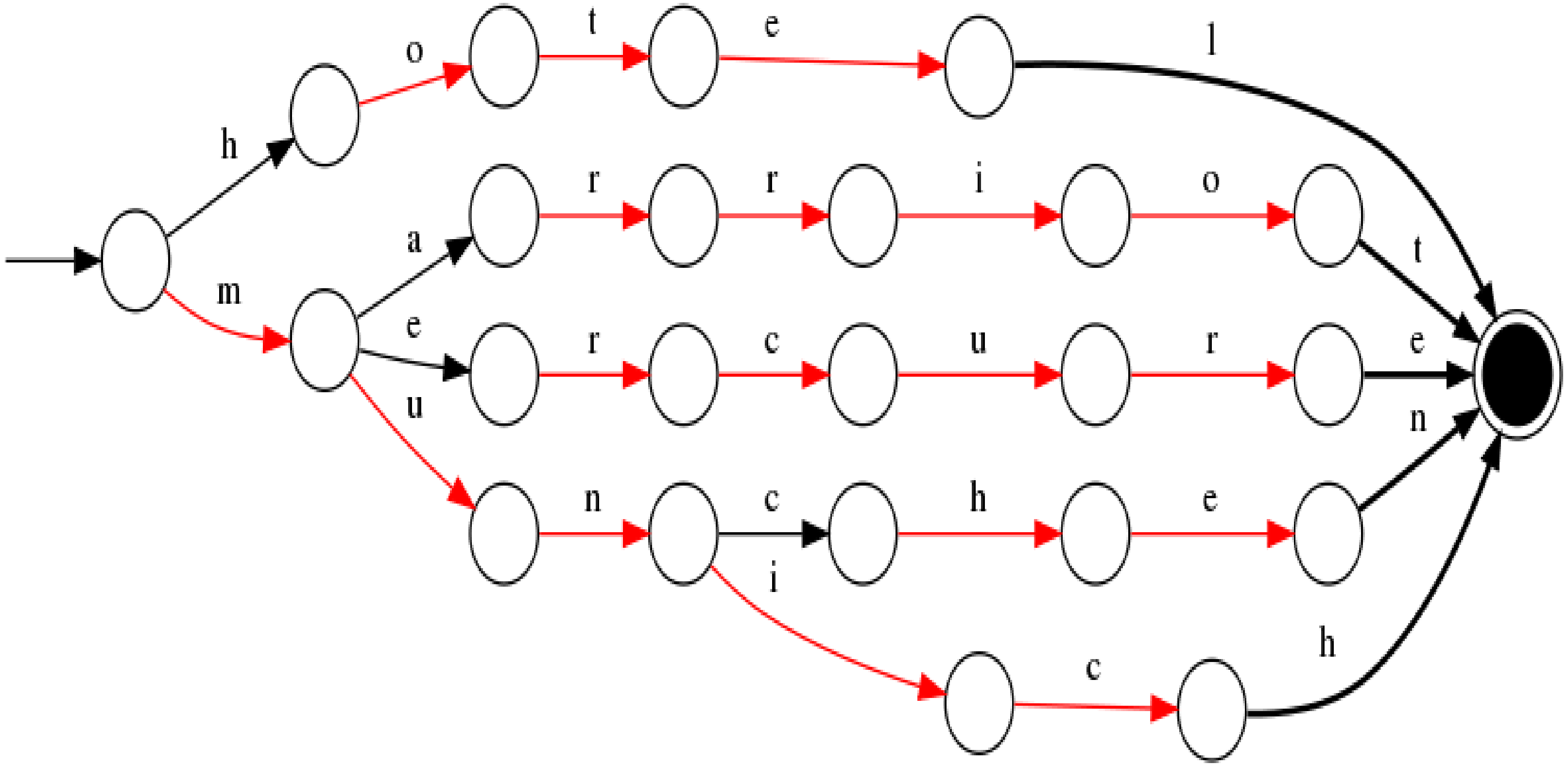
FROM product

WHERE name
LIKE ‘d%’

1k records
records

500k

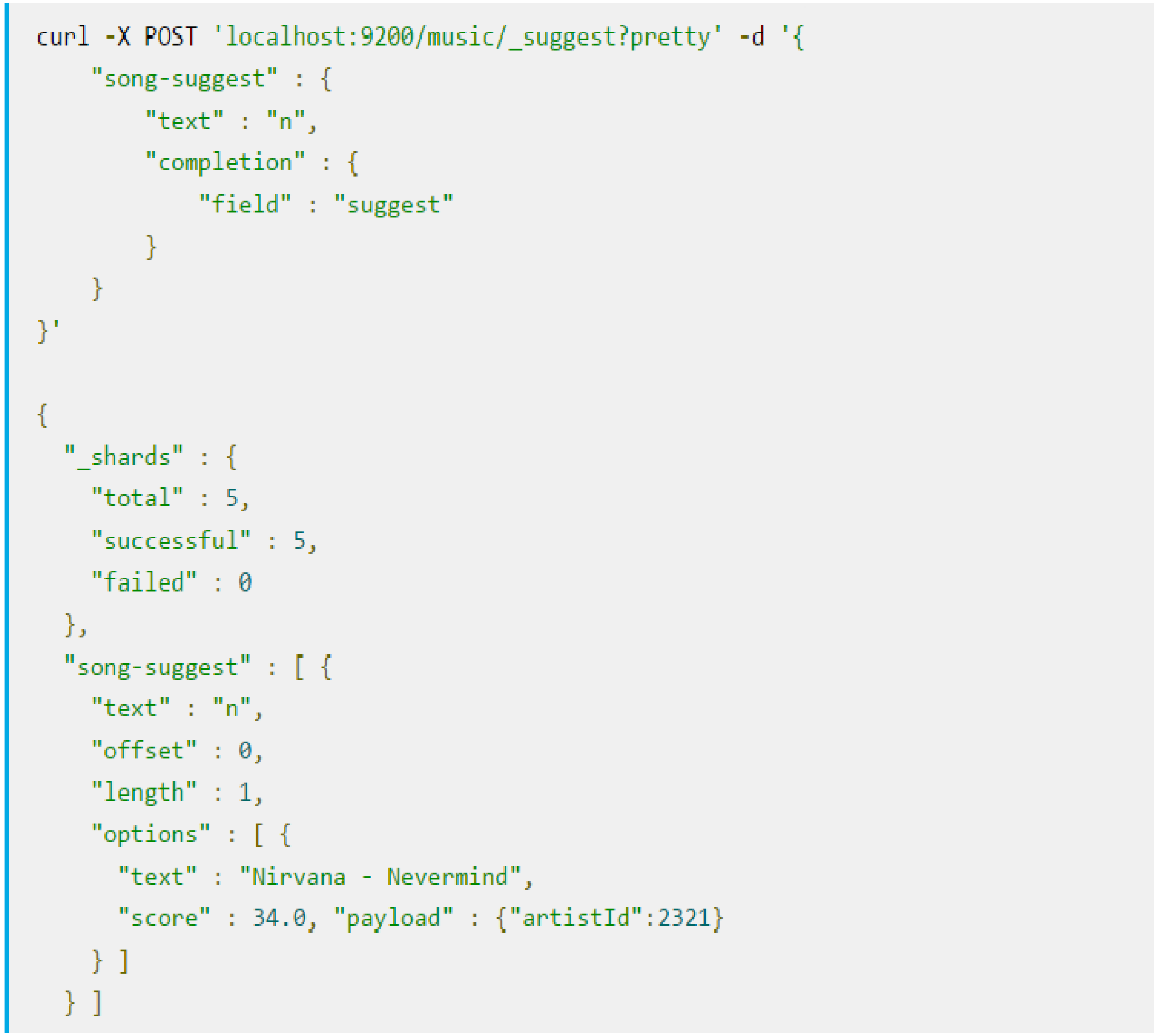
20m
records

- There is a completion suggester that allows basic auto-complete functionality.
- Lucene’s AnalyzingSuggester is used for
- 

Auto Completion - Mapping:

```
curl -X PUT localhost:9200/music
  curl -X PUT localhost:9200/music/song/_mapping -d '{
  "song" : {
    "properties" : {
      "name" : { "type" : "string" },
      "suggest" : { "type" : "completion",
             "analyzer" : "simple",
             "search_analyzer" : "simple",
             "payloads" : true
      }
    }
  }
}
```

Auto Completion - Querying



```
curl -X POST 'localhost:9200/music/_suggest?pretty' -d '{
    "song-suggest" : {
        "text" : "n",
        "completion" : {
            "field" : "suggest"
        }
    }
}'

{
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "failed" : 0
  },
  "song-suggest" : [ {
    "text" : "n",
    "offset" : 0,
    "length" : 1,
    "options" : [ {
      "text" : "Nirvana - Nevermind",
      "score" : 34.0, "payload" : {"artistId":2321}
    } ]
  } ]
```

Ecosystem

- Plugins

  Many third party plugins available

- Clients for many languages

  Ruby, python, php, perl, javascript, .NET, Scala, clojure, go

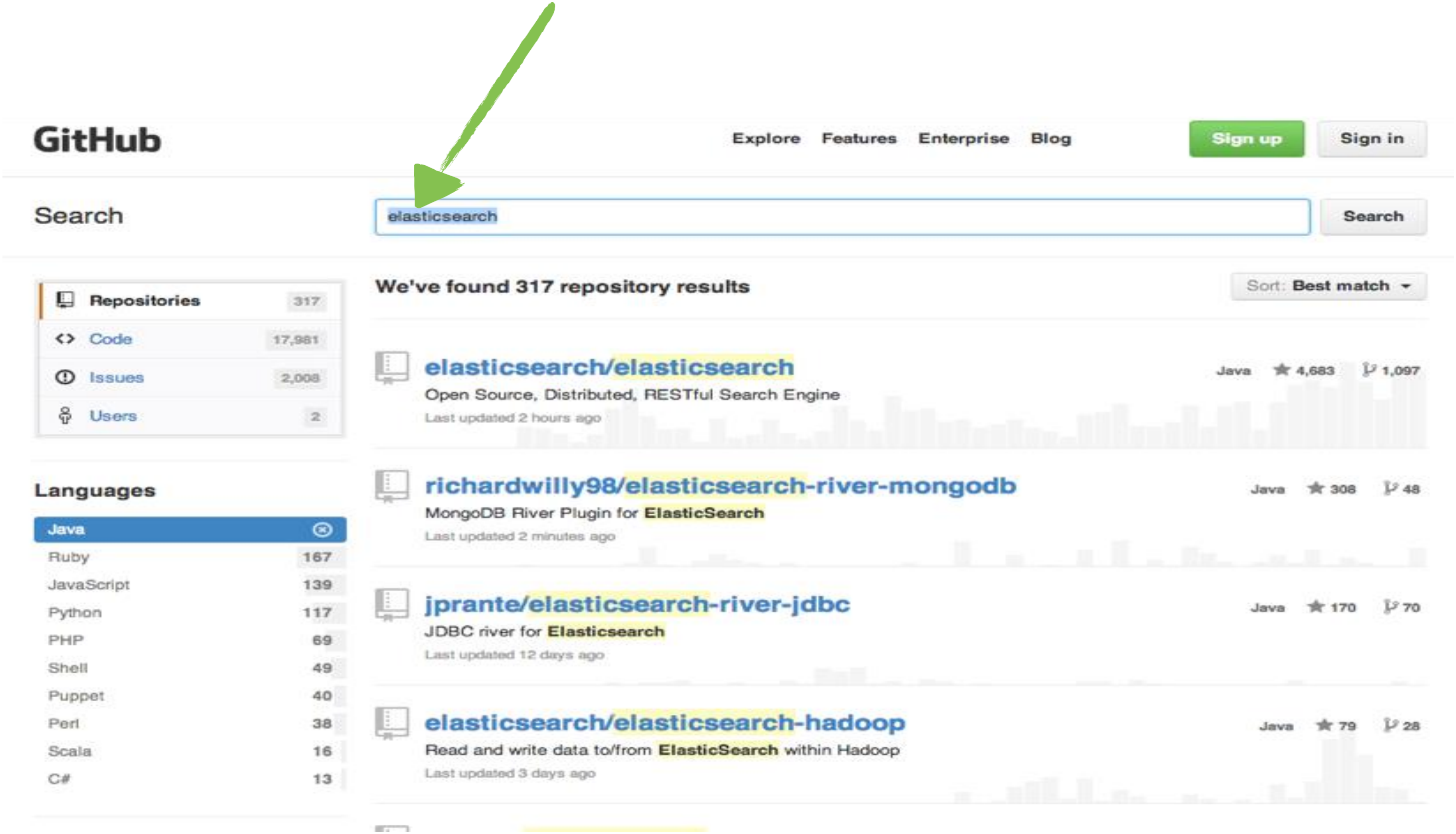
- Kibana

- Logstash

- Hadoop integration

WordPress
GitHub
Goldman Sachs
stackoverflow
NETFLIX
StumbleUpon
mozilla
CERN
theguardian
flickr
f
Etsy
verizon
salesforce
Adobe
The New York Times
ebay
cisco

Search



GitHub

Explore Features Enterprise Blog

Sign up Sign in

Search

elasticsearch

Search

Repositories	317
Code	17,981
Issues	2,008
Users	2

We've found 317 repository results

Sort: Best match

elasticsearch/elasticsearch — Java ★ 4,683 1,097

Open Source, Distributed, RESTful Search Engine

Last updated 2 hours ago

richardwilly98/elasticsearch-river-mongodb — Java ★ 308 48

MongoDB River Plugin for ElasticSearch

Last updated 2 minutes ago

jprante/elasticsearch-river-jdbc — Java ★ 170 70

JDBC river for Elasticsearch

Last updated 12 days ago

elasticsearch/elasticsearch-hadoop — Java ★ 79 28

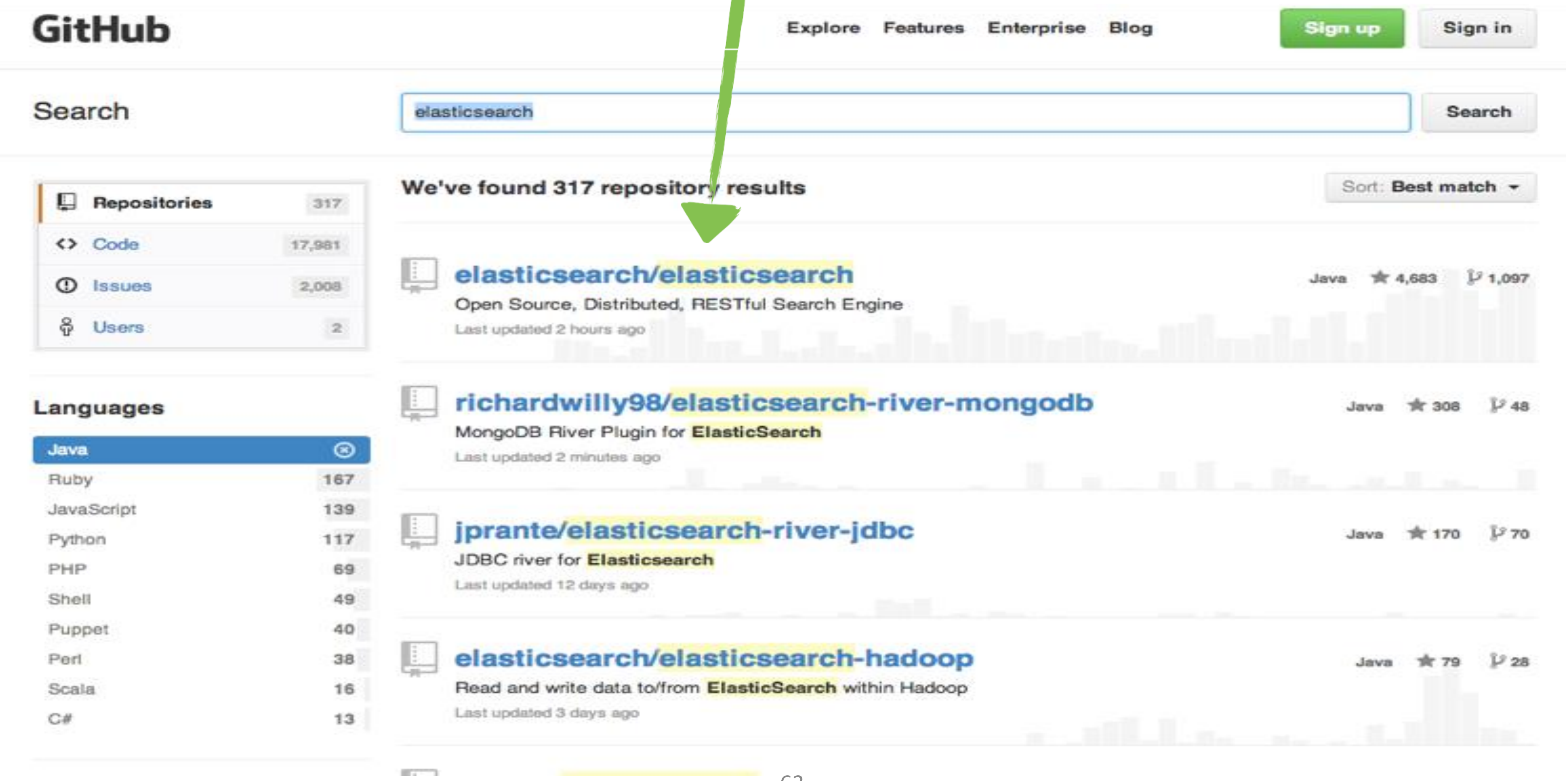
Read and write data to/from ElasticSearch within Hadoop

Last updated 3 days ago

Languages

Java	
Ruby	167
JavaScript	139
Python	117
PHP	69
Shell	49
Puppet	40
Perl	38
Scala	16
C#	13

Enrichment



elasticsearch

Repositories	317
Code	17,981
Issues	2,008
Users	2

Languages

Java	
Ruby	167
JavaScript	139
Python	117
PHP	69
Shell	49
Puppet	40
Perl	38
Scala	16
C#	13

We've found 317 repository results

Sort: Best match

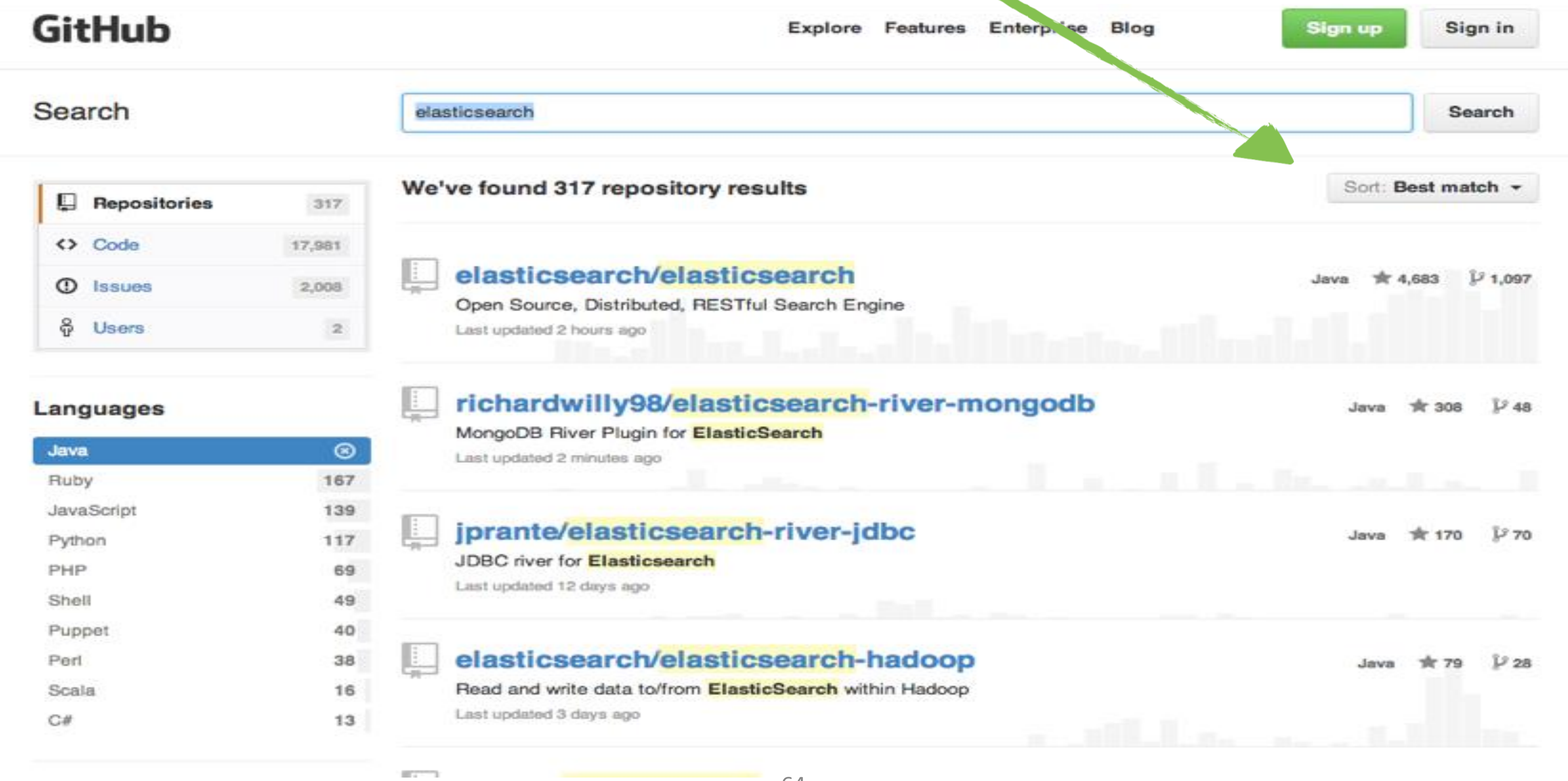
elasticsearch/elasticsearch — Java ★ 4,683 1,097
Open Source, Distributed, RESTful Search Engine
Last updated 2 hours ago

richardwilly98/elasticsearch-river-mongodb — Java ★ 308 48
MongoDB River Plugin for ElasticSearch
Last updated 2 minutes ago

jprante/elasticsearch-river-jdbc — Java ★ 170 70
JDBC river for Elasticsearch
Last updated 12 days ago

elasticsearch/elasticsearch-hadoop — Java ★ 79 28
Read and write data to/from ElasticSearch within Hadoop
Last updated 3 days ago

Sorting



elasticsearch

We've found 317 repository results

Sort: Best match

Repositories	317
Code	17,981
Issues	2,008
Users	2

Languages

Java	
Ruby	167
JavaScript	139
Python	117
PHP	69
Shell	49
Puppet	40
Perl	38
Scala	16
C#	13

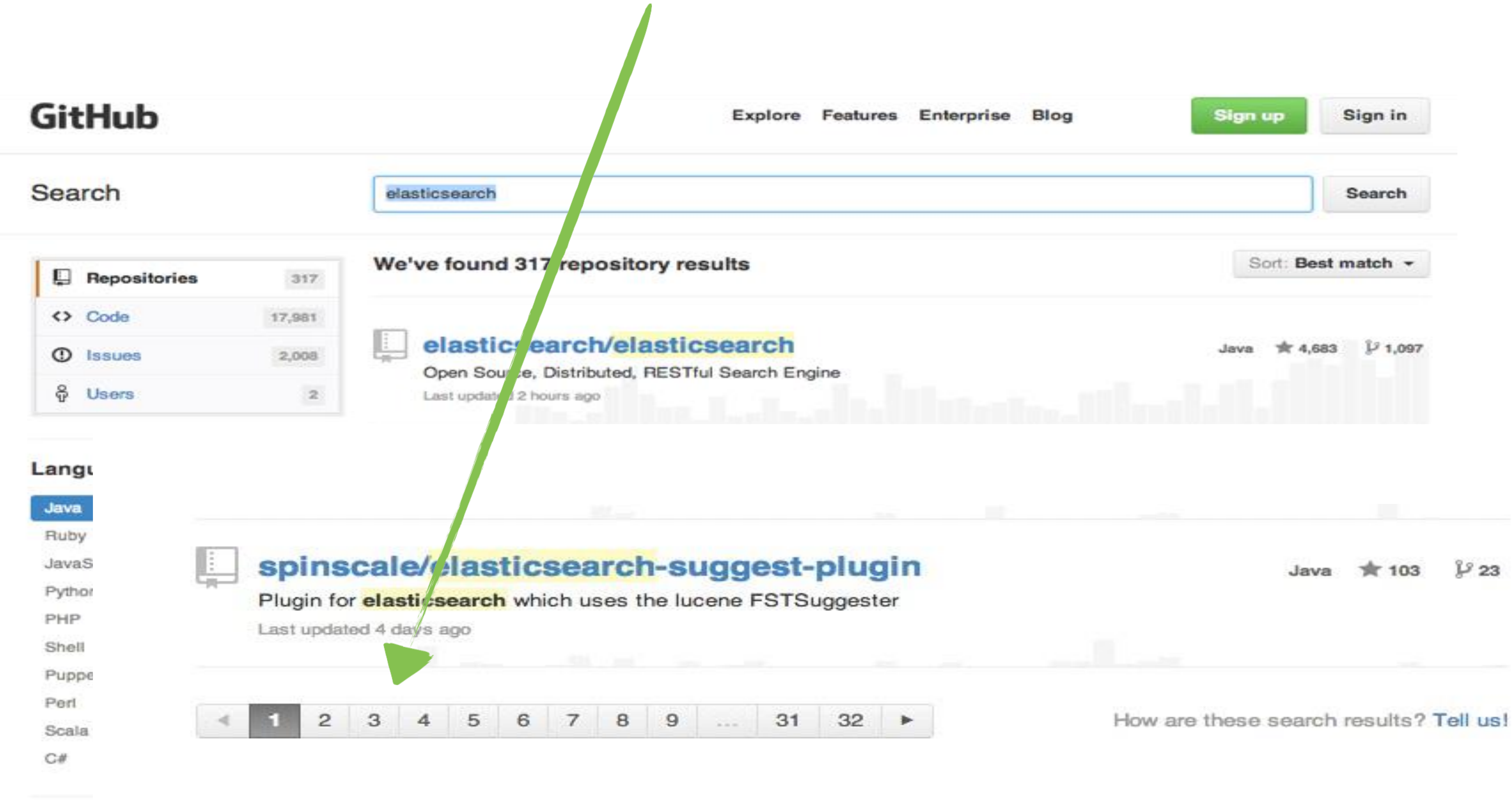
elasticsearch/elasticsearch Java ★ 4,683 1,097
Open Source, Distributed, RESTful Search Engine
Last updated 2 hours ago

richardwilly98/elasticsearch-river-mongodb Java ★ 308 48
MongoDB River Plugin for ElasticSearch
Last updated 2 minutes ago

jprante/elasticsearch-river-jdbc Java ★ 170 70
JDBC river for Elasticsearch
Last updated 12 days ago

elasticsearch/elasticsearch-hadoop Java ★ 79 28
Read and write data to/from ElasticSearch within Hadoop
Last updated 3 days ago

Pagination



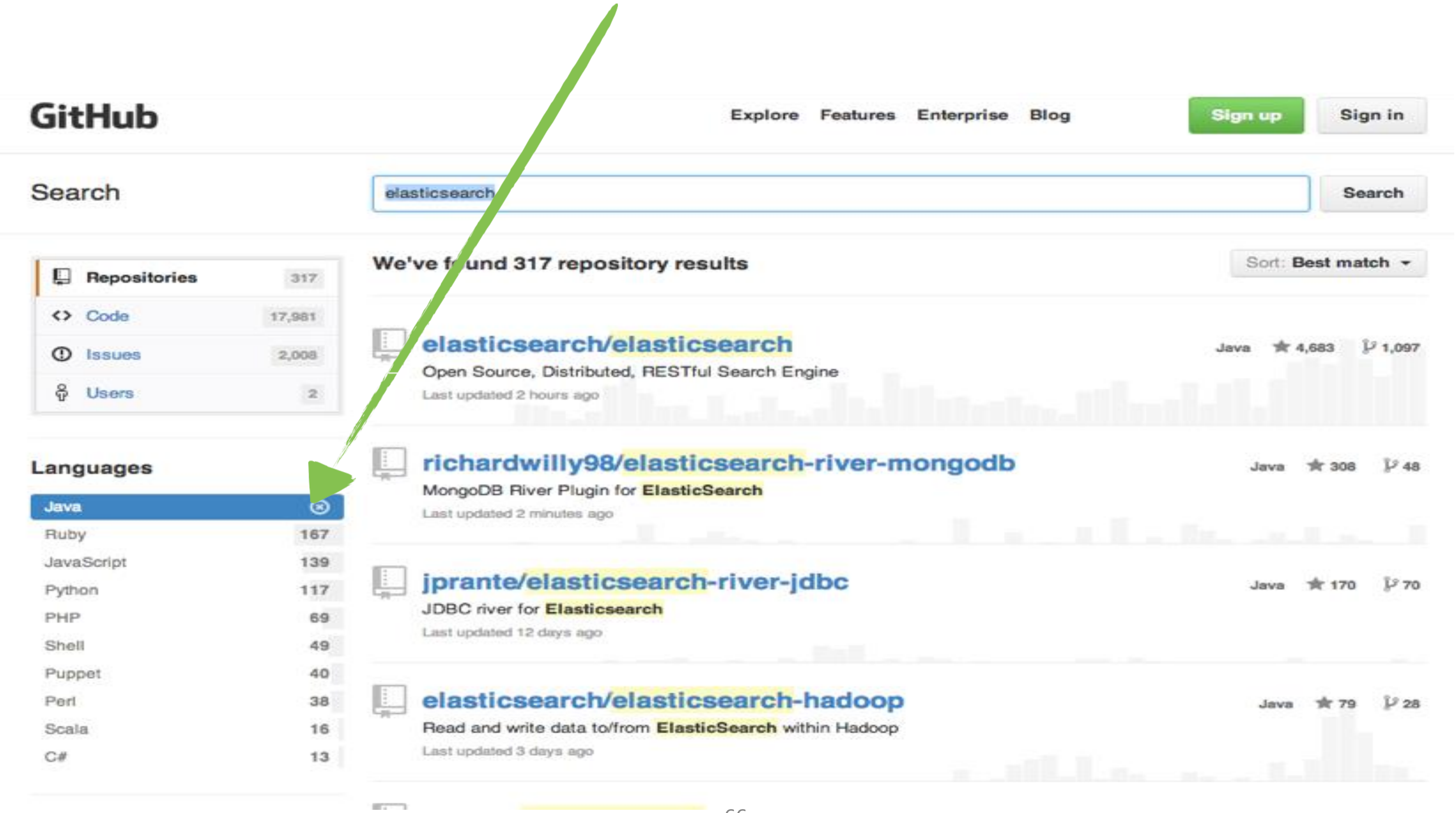
We've found 317 repository results

elasticsearch/elasticsearch — Java ★ 4,683 1,097
Open Source, Distributed, RESTful Search Engine
Last updated 2 hours ago

spinscale/elasticsearch-suggest-plugin — Java ★ 103 23
Plugin for **elasticsearch** which uses the lucene FSTSuggester
Last updated 4 days ago

How are these search results? Tell us!

Aggregation



We've found 317 repository results

Sort: Best match

Repositories	317
Code	17,981
Issues	2,008
Users	2

Languages

Java	
Ruby	167
JavaScript	139
Python	117
PHP	69
Shell	49
Puppet	40
Perl	38
Scala	16
C#	13

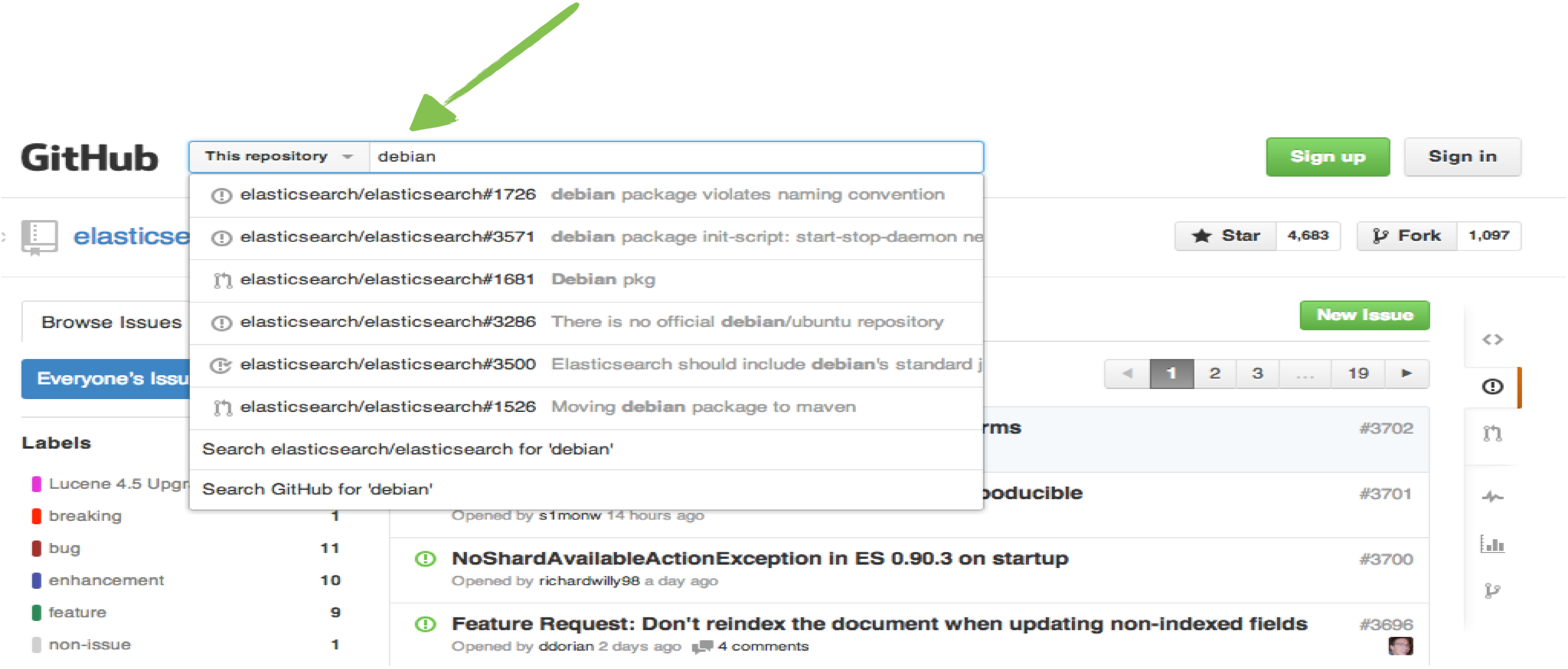
elasticsearch/elasticsearch — Java ★ 4,683 ⑂ 1,097
Open Source, Distributed, RESTful Search Engine
Last updated 2 hours ago

richardwilly98/elasticsearch-river-mongodb — Java ★ 308 ⑂ 48
MongoDB River Plugin for ElasticSearch
Last updated 2 minutes ago

jprante/elasticsearch-river-jdbc — Java ★ 170 ⑂ 70
JDBC river for Elasticsearch
Last updated 12 days ago

elasticsearch/elasticsearch-hadoop — Java ★ 79 ⑂ 28
Read and write data to/from ElasticSearch within Hadoop
Last updated 3 days ago

Suggestions



GitHub

This repository ▾ debian

Sign up Sign in

ⓘ elasticsearch/elasticsearch#1726 **debian** package violates naming convention

ⓘ elasticsearch/elasticsearch#3571 **debian** package init-script: start-stop-daemon ne

elasticsearch/elasticsearch#1681 **Debian** pkg

ⓘ elasticsearch/elasticsearch#3286 There is no official **debian**/ubuntu repository

elasticsearch/elasticsearch#3500 Elasticsearch should include **debian**'s standard j

elasticsearch/elasticsearch#1526 Moving **debian** package to maven

Search elasticsearch/elasticsearch for 'debian'

Search GitHub for 'debian'

elasticse

★ Star 4,683 Fork 1,097

Browse Issues

New Issue

Everyone's Issu

◀ 1 2 3 ... 19 ▶

Labels

Lucene 4.5 Upgr	
breaking	1
bug	11
enhancement	10
feature	9
non-issue	1

rms #3702

oducible #3701

Opened by s1monw 14 hours ago

ⓘ NoShardAvailableActionException in ES 0.90.3 on startup #3700

Opened by richardwilly98 a day ago

ⓘ Feature Request: Don't reindex the document when updating non-indexed fields #3696

Opened by ddorian 2 days ago 4 comments

The New York Times

- 15 million of its articles published over the last 160 years fed into Elasticsearch.
- Typical use cases:
  - Find something you read
  - Find book/movie reviews
  - Serious research
- Why not just use google?
  - Keep the customer on site.
  - There is no google for native apps.
  - They know their content better.

Elasticsearch as a primary data store?

- No transactions
- Relations and constraints
- Robustness
- Security

Conclusion

- Commonly used in addition to another database.
- But if the previously mentioned issues are not a concern, it can be used as a primary database also.

Like with everything else, there's no silver bullet, no one database to rule them all.

REFERENCES

1. https://www.elastic.co/products/elasticsearch

2. https://qbox.io/blog/what-is-elasticsearch

3. https://www.elastic.co/blog/index-vs-type

4. https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html

5. http://exploringelasticsearch.com/overview.html

Thank You