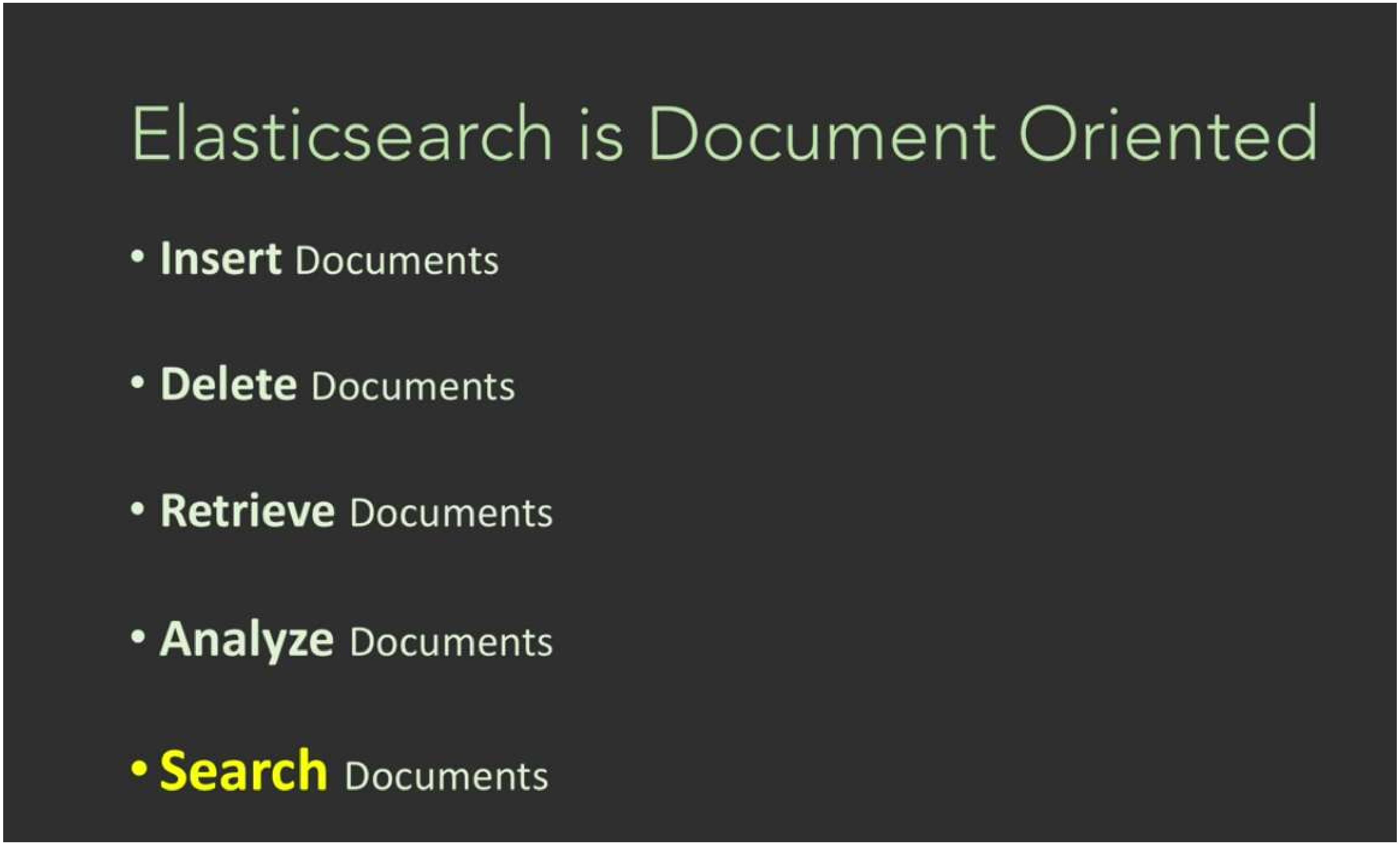
Mastering ELK + EFK

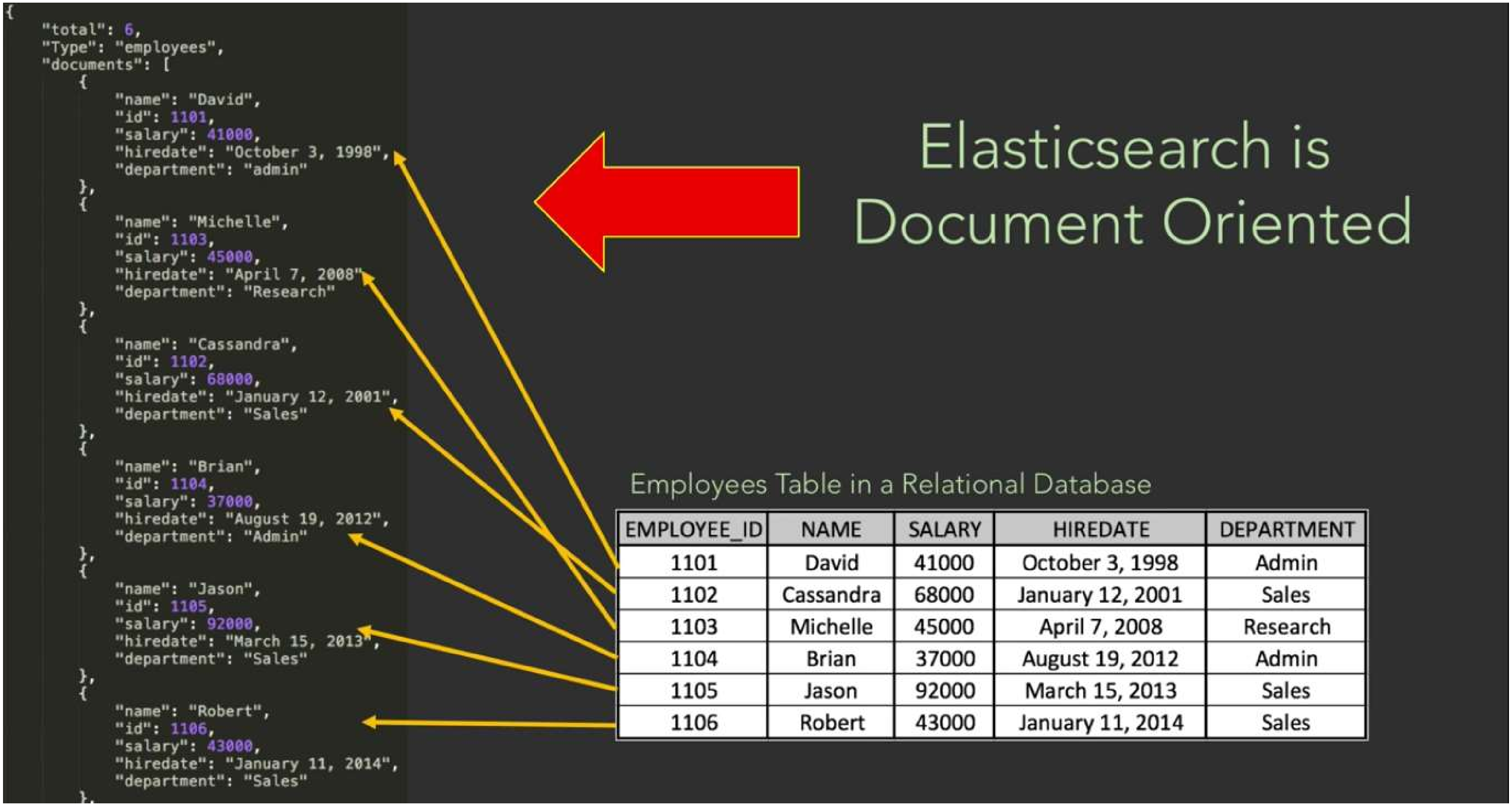
Elasticsearch

- A search engine based on the Lucene library.
- It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents.
- Developed in Java



Elasticsearch is Document Oriented

- **Insert** Documents
- **Delete** Documents
- **Retrieve** Documents
- **Analyze** Documents
- **Search** Documents



```
{
    "total": 6,
    "Type": "employees",
    "documents": [
        {
            "name": "David",
            "id": 1101,
            "salary": 41000,
            "hiredate": "October 3, 1998",
            "department": "admin"
        },
        {
            "name": "Michelle",
            "id": 1103,
            "salary": 45000,
            "hiredate": "April 7, 2008",
            "department": "Research"
        },
        {
            "name": "Cassandra",
            "id": 1102,
            "salary": 68000,
            "hiredate": "January 12, 2001",
            "department": "Sales"
        },
        {
            "name": "Brian",
            "id": 1104,
            "salary": 37000,
            "hiredate": "August 19, 2012",
            "department": "Admin"
        },
        {
            "name": "Jason",
            "id": 1105,
            "salary": 92000,
            "hiredate": "March 15, 2013",
            "department": "Sales"
        },
        {
            "name": "Robert",
            "id": 1106,
            "salary": 43000,
            "hiredate": "January 11, 2014",
            "department": "Sales"
        },
```

Elasticsearch is Document Oriented

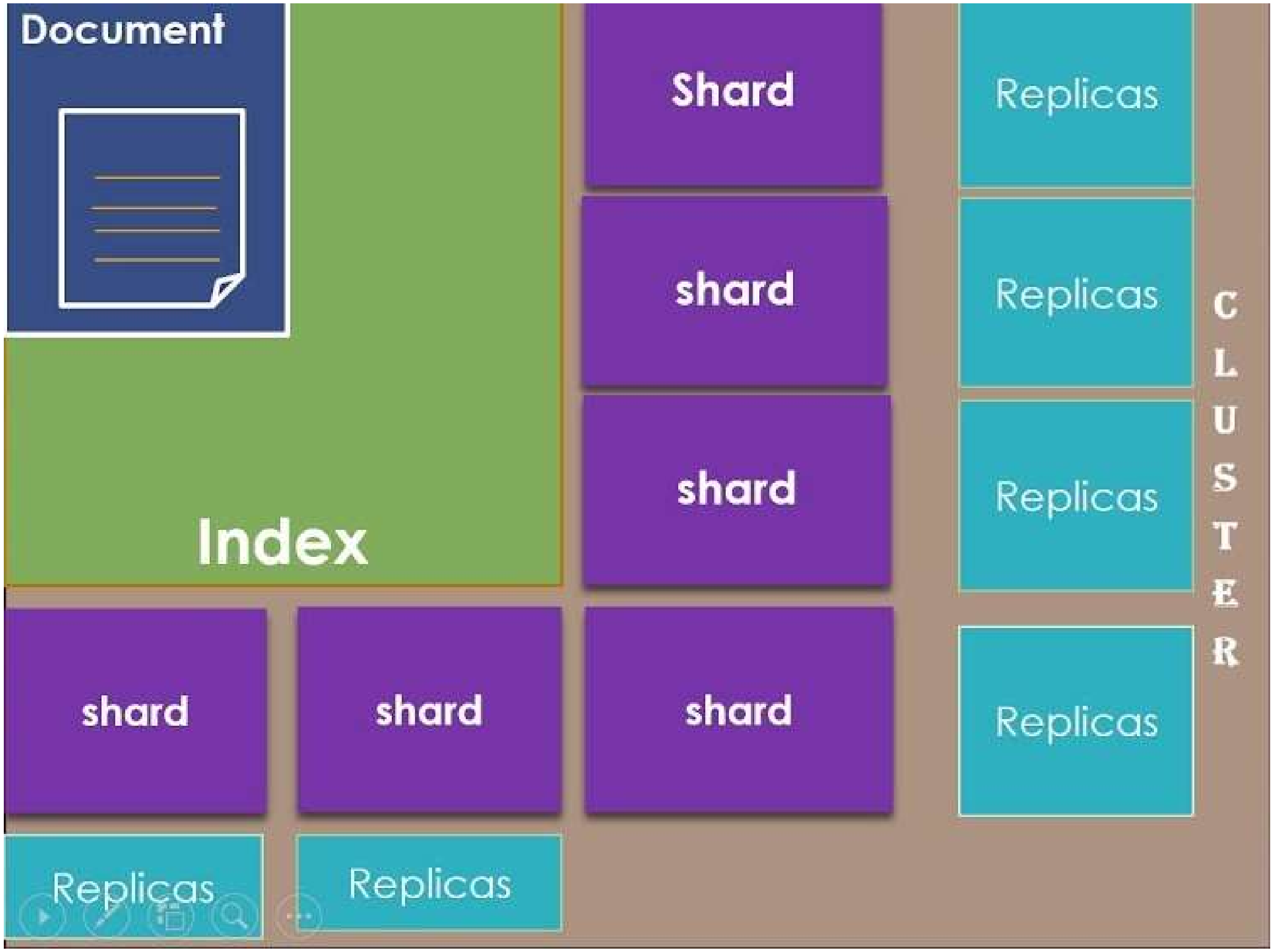
Employees Table in a Relational Database

EMPLOYEE_ID	NAME	SALARY	HIREDATE	DEPARTMENT
1101	David	41000	October 3, 1998	Admin
1102	Cassandra	68000	January 12, 2001	Sales
1103	Michelle	45000	April 7, 2008	Research
1104	Brian	37000	August 19, 2012	Admin
1105	Jason	92000	March 15, 2013	Sales
1106	Robert	43000	January 11, 2014	Sales

General Features

- Scalable up to petabytes of structured and unstructured data.
- Can be used as a replacement of document stores like MongoDB.
- Uses denormalization to improve the search performance.
- Is one of the popular enterprise search engines
- Is currently being used by many big organizations like
  - Wikipedia,
  - The Guardian,
  - StackOverflow,
  - GitHub etc.
- Is an open source and available under the Apache license version 2.0.

Key Concepts

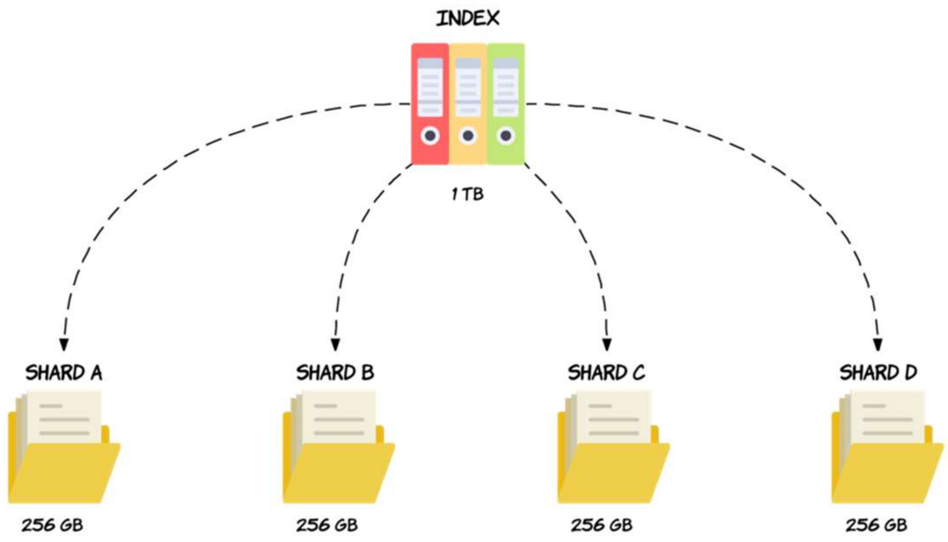


Key Concepts

- Node
  - Single running instance of Elasticsearch.
- Cluster
  - Collection of one or more nodes.
  - Cluster provides collective indexing and search capabilities across all the nodes for entire data.
- Index
  - Collection of different type of documents and their properties.
  - Uses the concept of shards to improve the performance.
- Document
  - Collection of fields in a specific manner defined in JSON format.

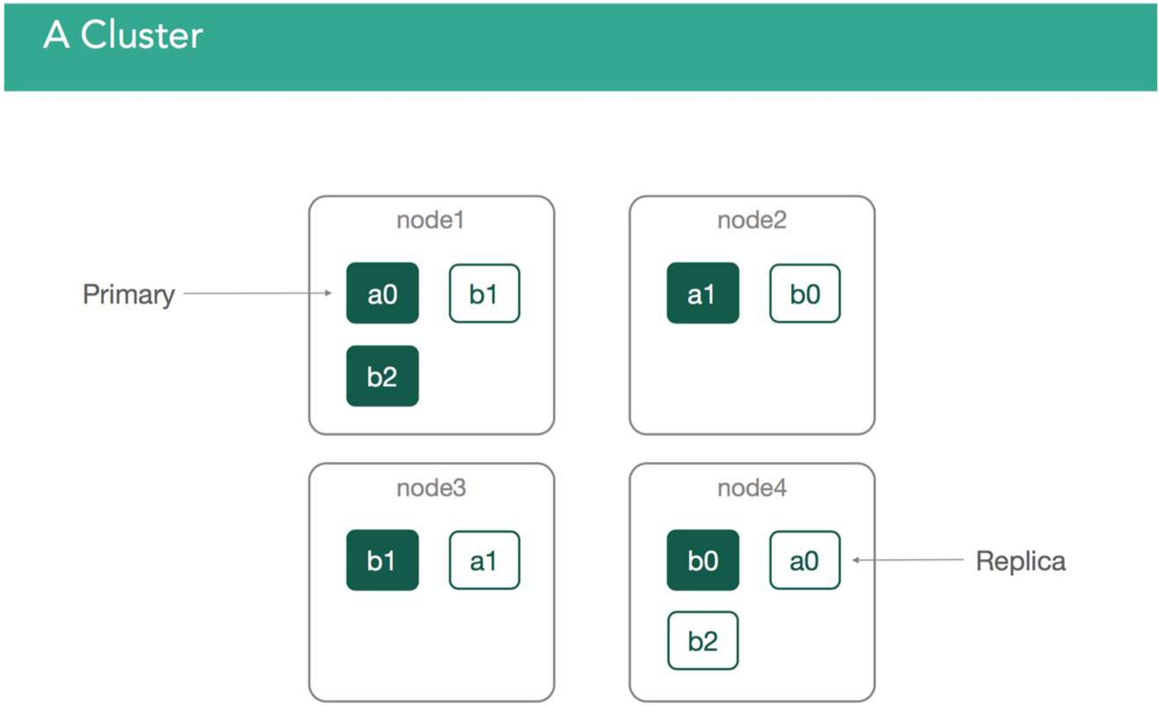
Key Concepts

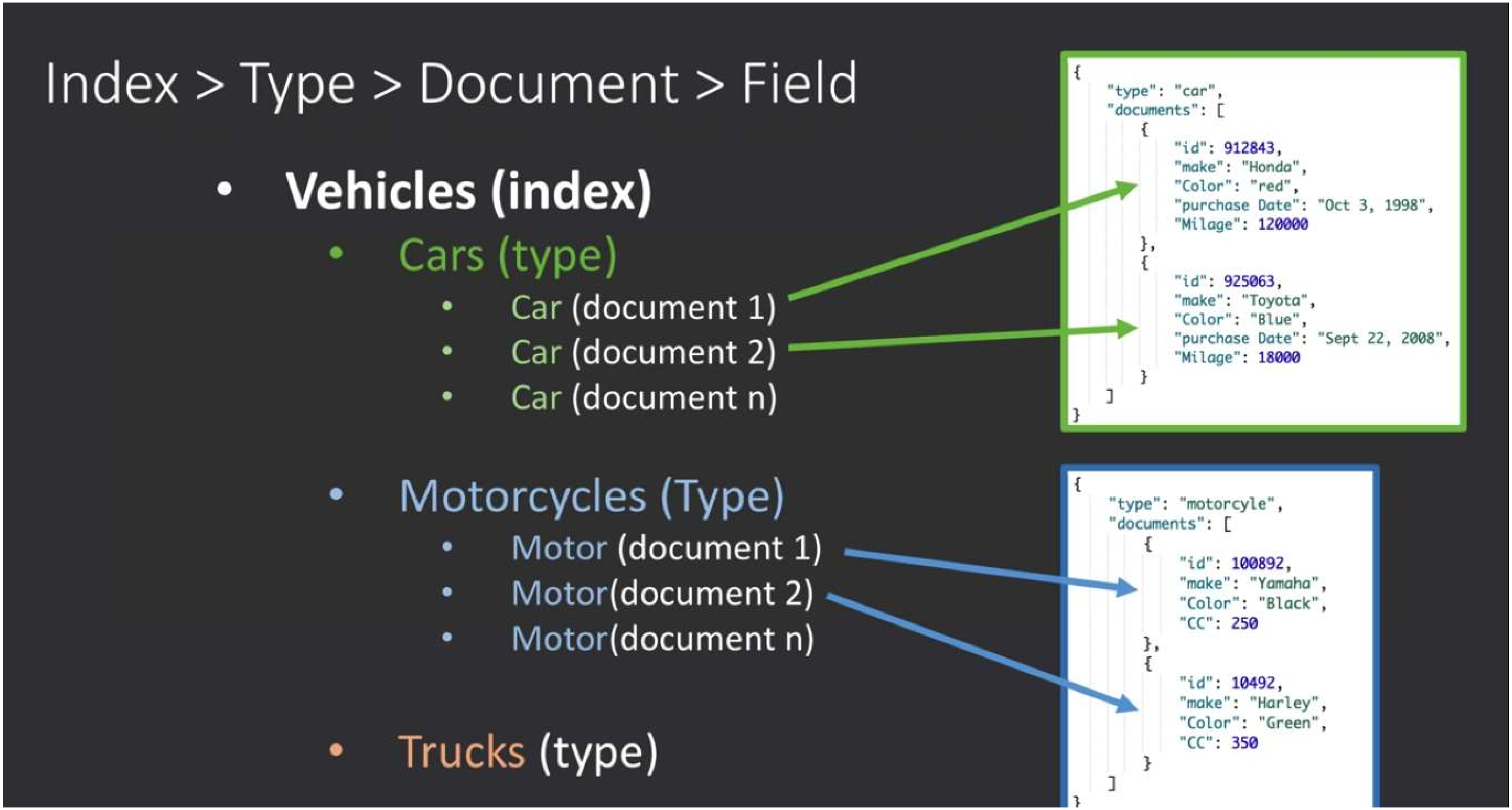
- Shard
  - Indexes are horizontally subdivided into shards.
  - This means each shard contains all the properties of document but contains less number of JSON objects.
  - Primary shard is the original horizontal part of an index and then these primary shards are replicated into replica shards



Key Concepts

- Replicas
  - Replicas of indexes and shards.
  - Helps in increasing the availability of data
  - Improves the performance of searching
    - Parallel search operation in these replicas.





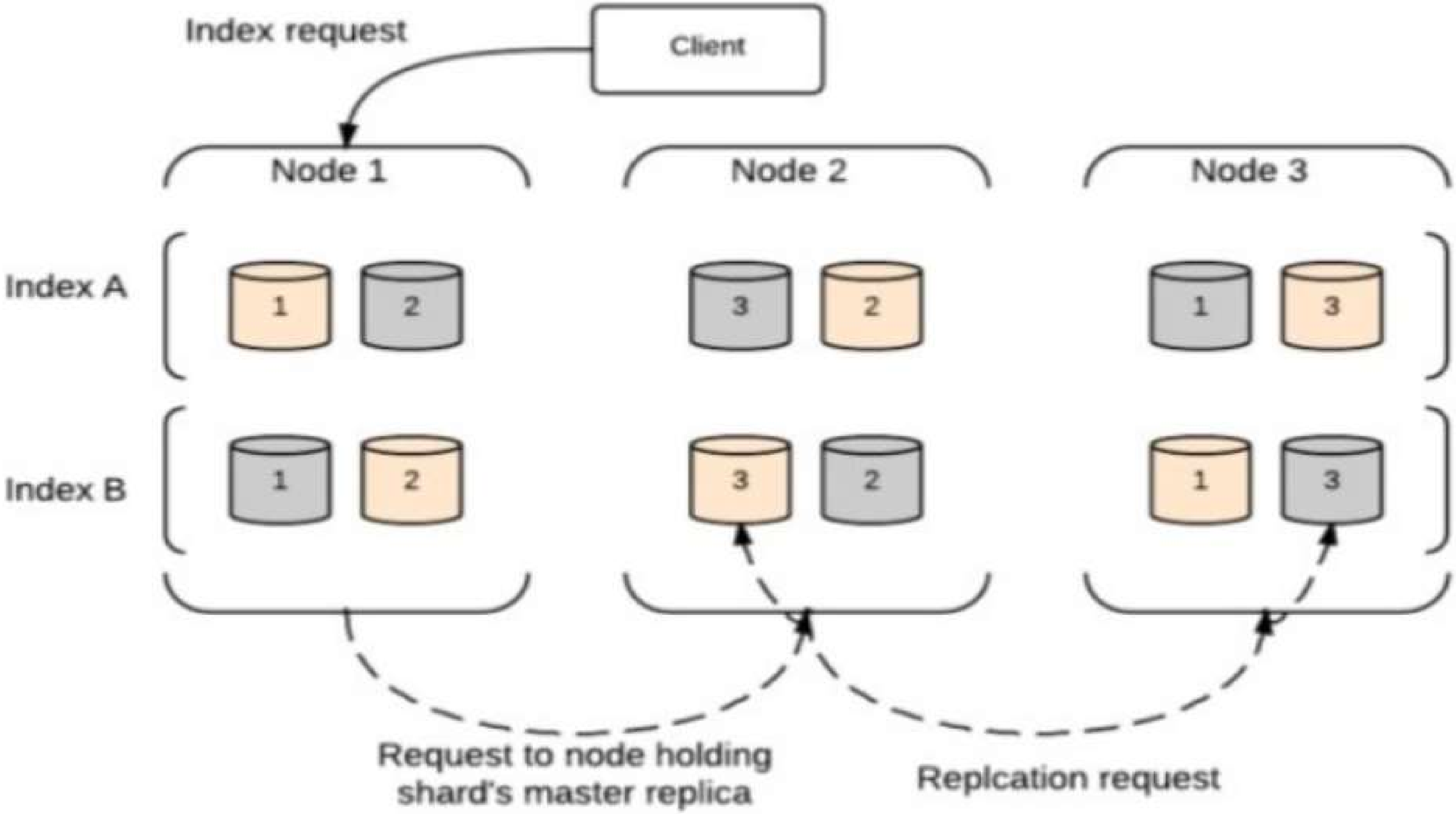
Index > Type > Document > Field

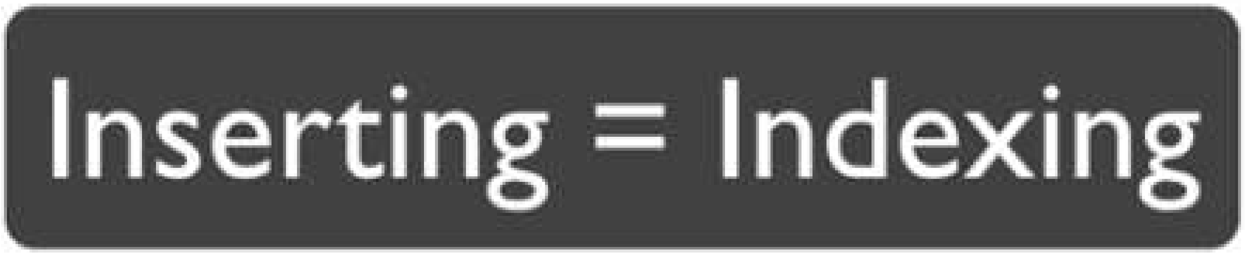
- **Vehicles (index)**
  - Cars (type)
    - Car (document 1)
    - Car (document 2)
    - Car (document n)
  - Motorcycles (Type)
    - Motor (document 1)
    - Motor(document 2)
    - Motor(document n)
  - Trucks (type)

```
{
    "type": "car",
    "documents": [
        {
            "id": 912843,
            "make": "Honda",
            "Color": "red",
            "purchase Date": "Oct 3, 1998",
            "Milage": 120000
        },
        {
            "id": 925063,
            "make": "Toyota",
            "Color": "Blue",
            "purchase Date": "Sept 22, 2008",
            "Milage": 18000
        }
    ]
}
```

```
{
    "type": "motorcyle",
    "documents": [
        {
            "id": 100892,
            "make": "Yamaha",
            "Color": "Black",
            "CC": 250
        },
        {
            "id": 10492,
            "make": "Harley",
            "Color": "Green",
            "CC": 350
        }
    ]
}
```

Index Request





Inserting = Indexing

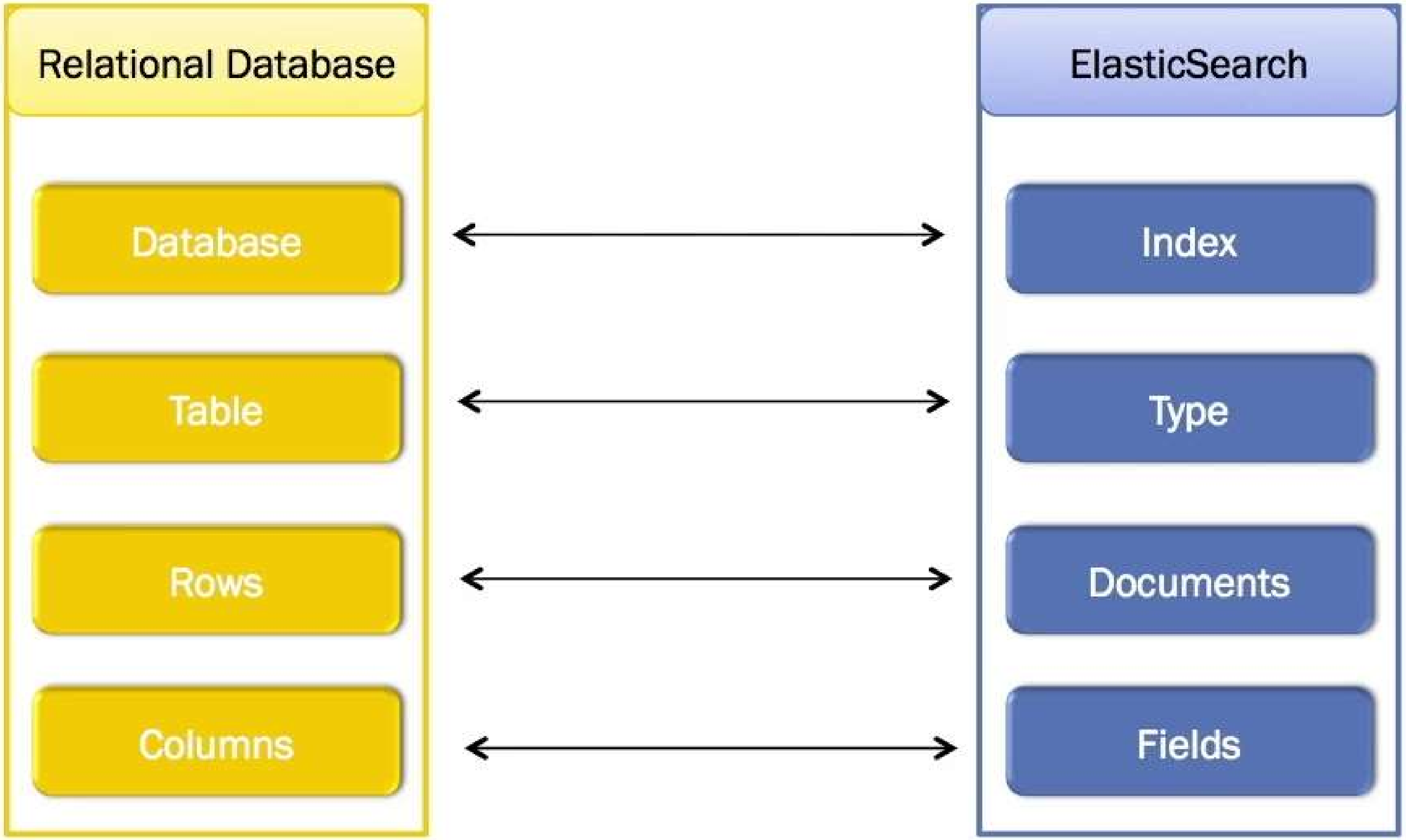
Advantages

- Developed on Java
  - Makes it compatible on almost every platform.
- Real time
  - After one second the added document is searchable in this engine.
- Distributed
  - Makes it easy to scale and integrate in any big organization.
- Uses JSON objects
  - Makes it possible to invoke the Elasticsearch server with a large number of different programming languages.

Disadvantages

- Does not have multi-language support in terms of handling request and response data (only possible in JSON)
- Security
  - Does not provide any built-in authentication or access control functionality.
- Transactions
  - There is no much more support for transactions or processing on data manipulation.

Comparison between Elasticsearch and RDBMS



Relational Database

Database

Table

Rows

Columns

ElasticSearch

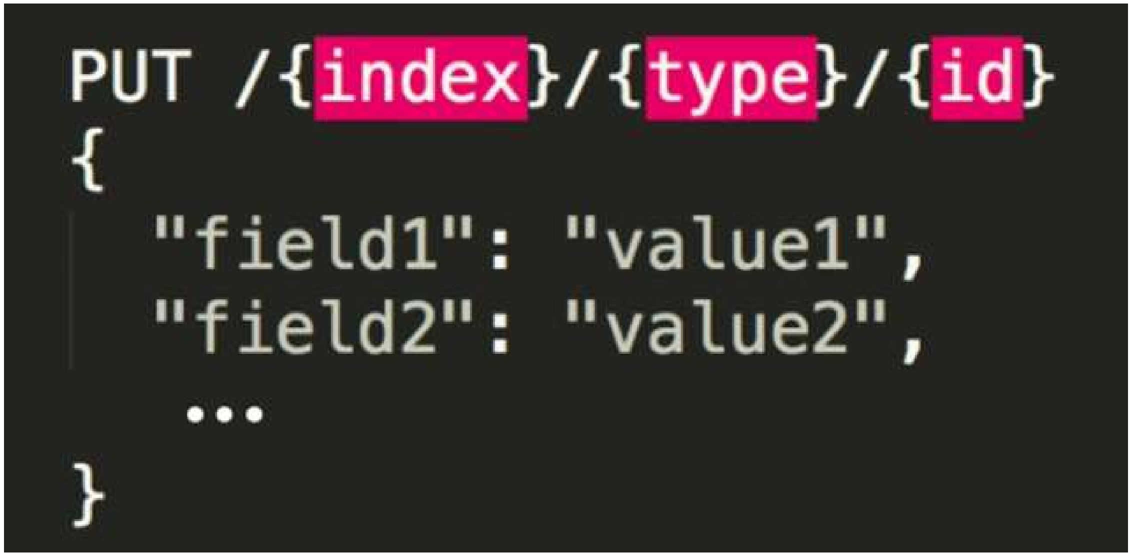
Index

Type

Documents

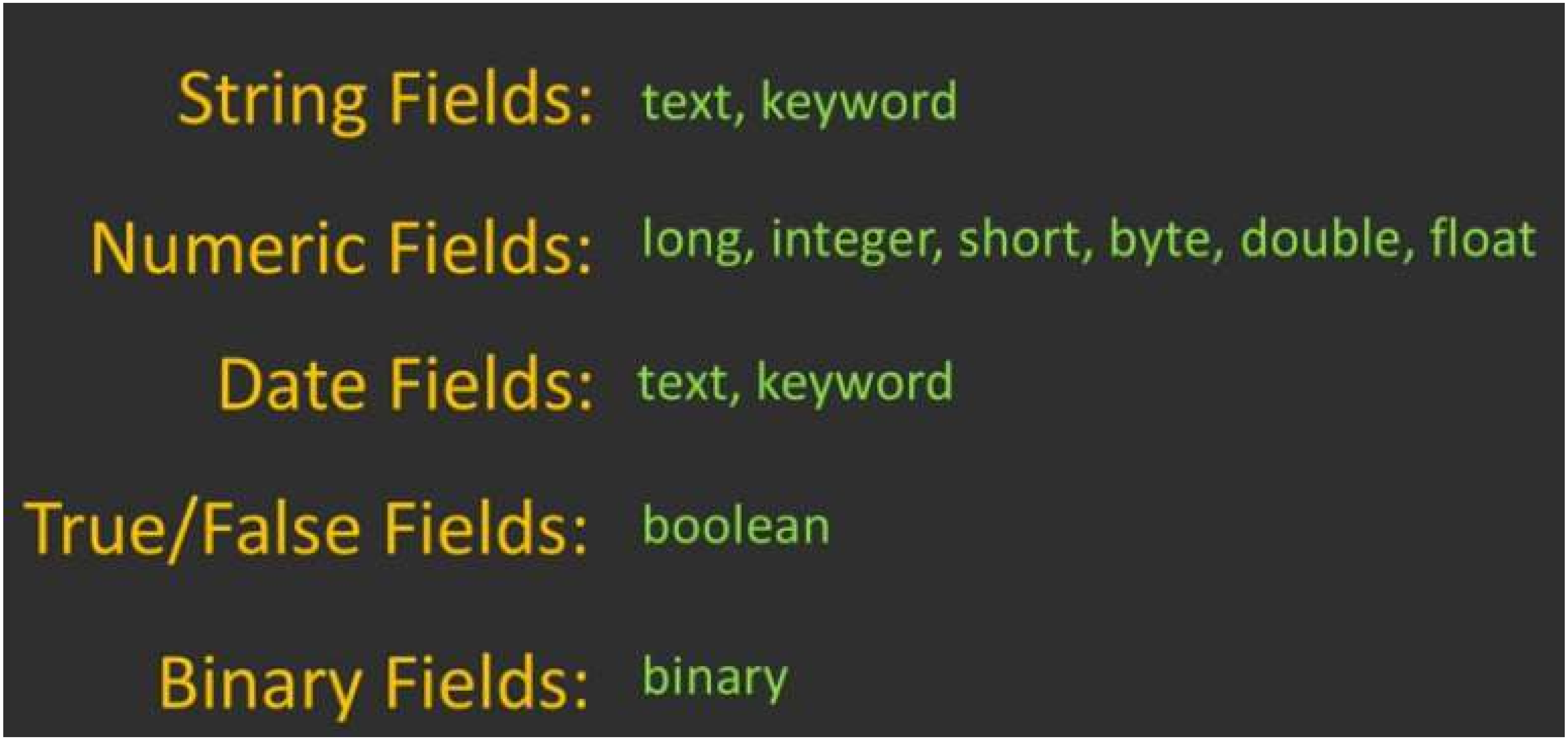
Fields

Syntax for indexing a document

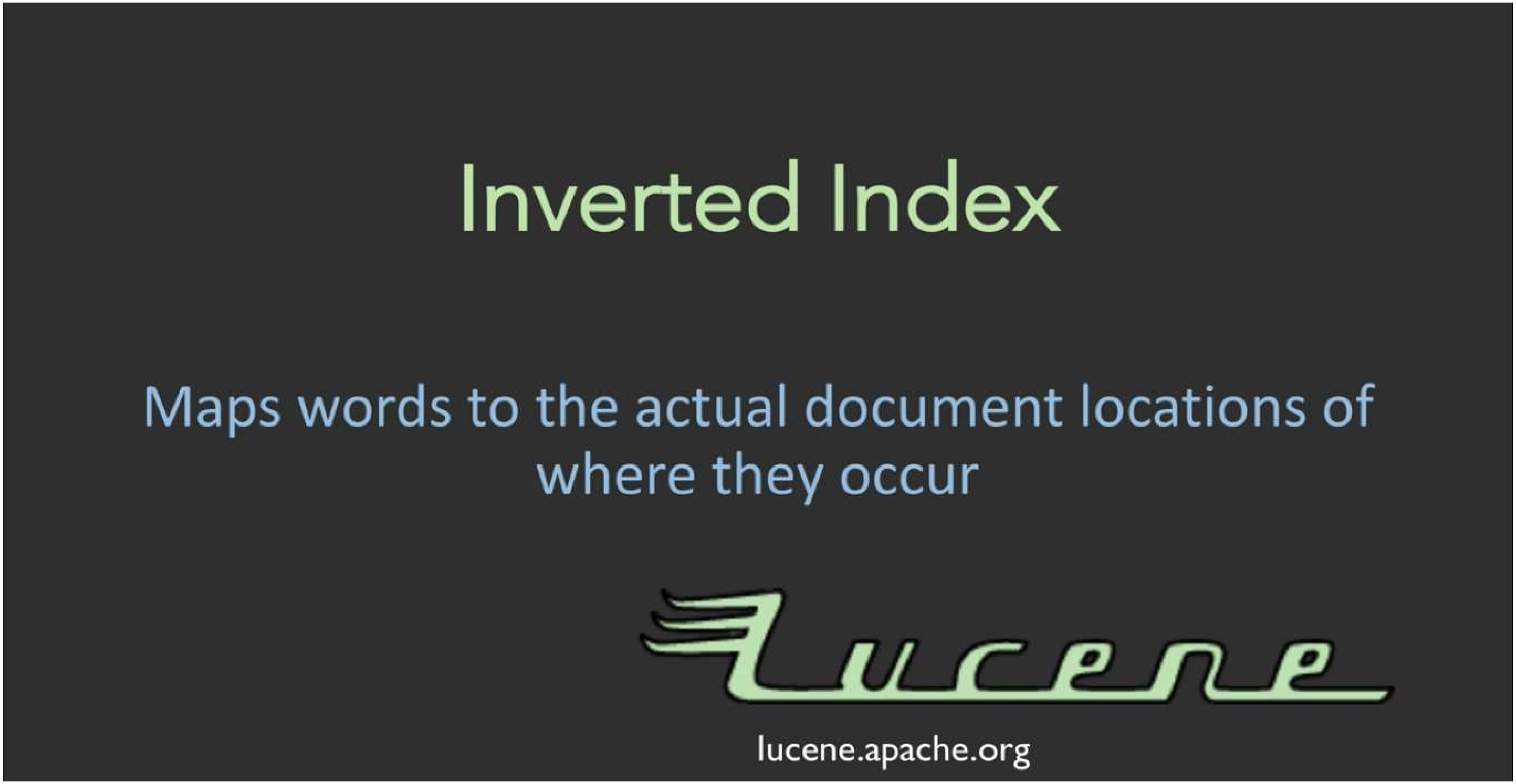


```
PUT /{index}/{type}/{id}
{
  "field1": "value1",
  "field2": "value2",
   ...
}
```

Data Type for Document Fields



String Fields:	text, keyword
Numeric Fields:	long, integer, short, byte, double, float
Date Fields:	text, keyword
True/False Fields:	boolean
Binary Fields:	binary

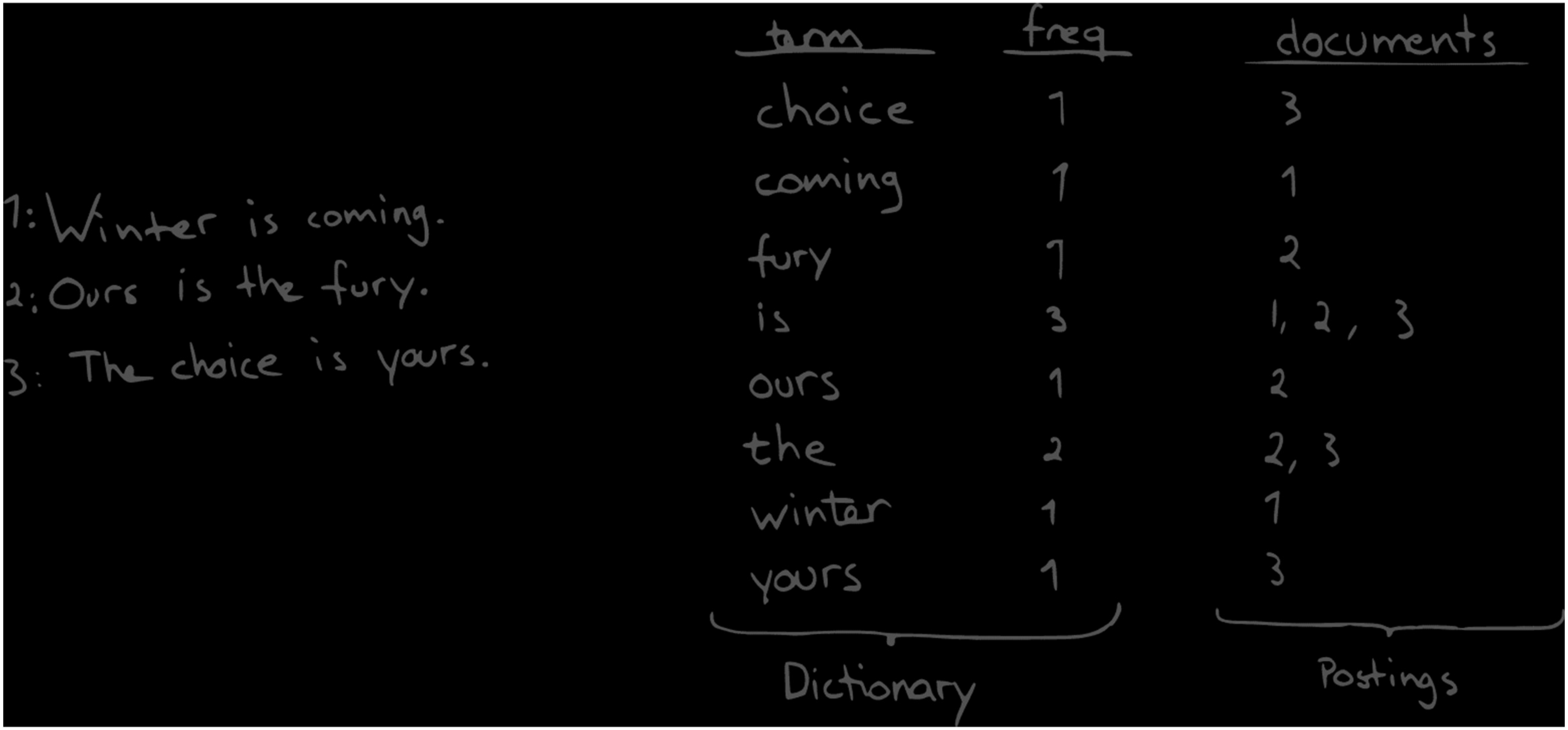


Inverted Index

Maps words to the actual document locations of where they occur

lucene.apache.org

Inverted Index



1: Winter is coming.

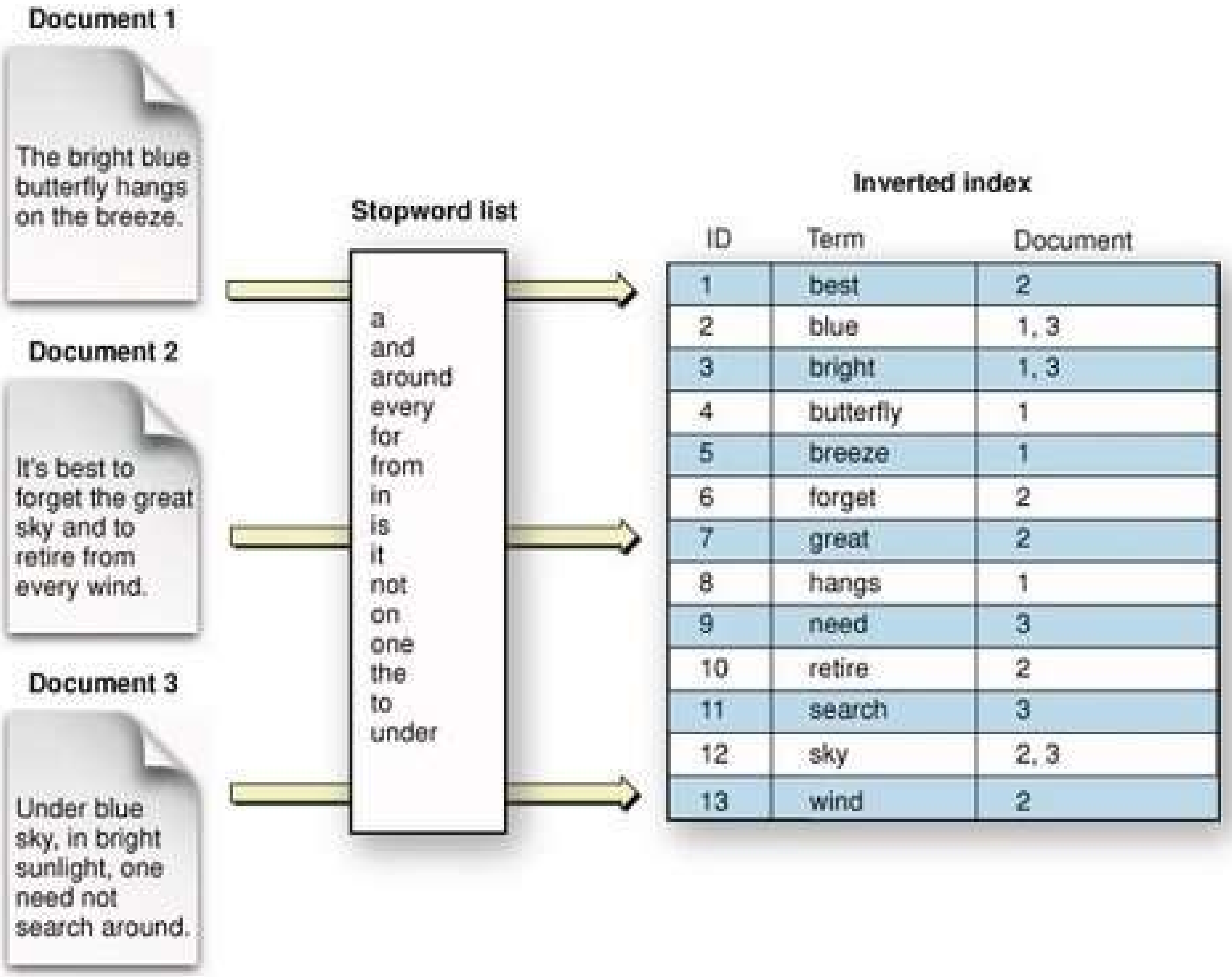
2: Ours is the fury.

3: The choice is yours.

term	freq	documents
choice	1	3
coming	1	1
fury	1	2
is	3	1, 2, 3
ours	1	2
the	2	2, 3
winter	1	1
yours	1	3

Dictionary

Postings



Document 1

The bright blue butterfly hangs on the breeze.

Document 2

It's best to forget the great sky and to retire from every wind.

Document 3

Under blue sky, in bright sunlight, one need not search around.

Stopword list

a
and
around
every
for
from
in
is
it
not
on
one
the
to
under

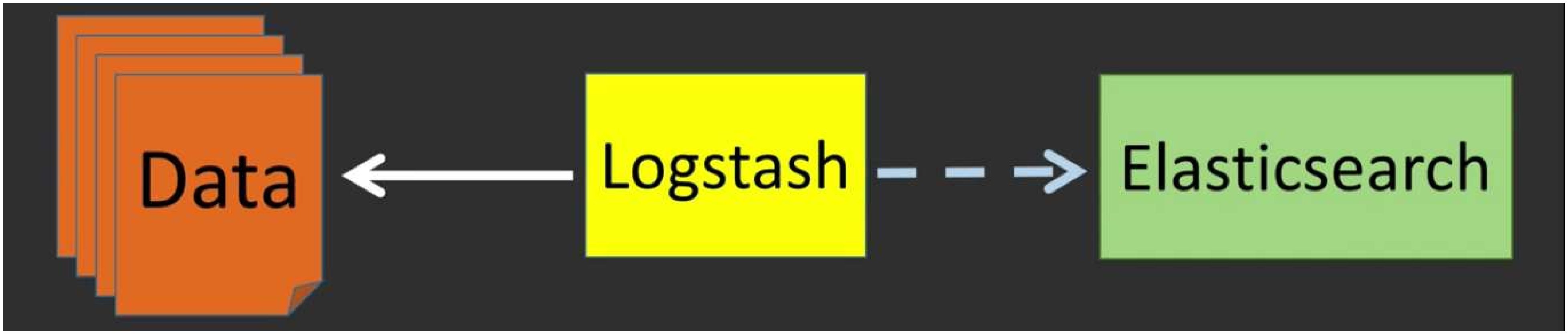
Inverted index

ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

Logstash

- Logstash is a tool for managing events and logs.
- Encompasses a larger system of log collection, processing, storage and searching activities.
- Part of the Elastic Stack along with Beats, Elasticsearch and Kibana
- Processing pipeline that
  - Ingests data from a multitude of sources simultaneously,
  - Transforms it, and then
  - Sends it to Elasticsearch
- Logstash has over 200 plugins, and you can write your own very easily as well.

Logstash



Elastic Search – Populate

- Can use the following command to create an index:
  - PUT school

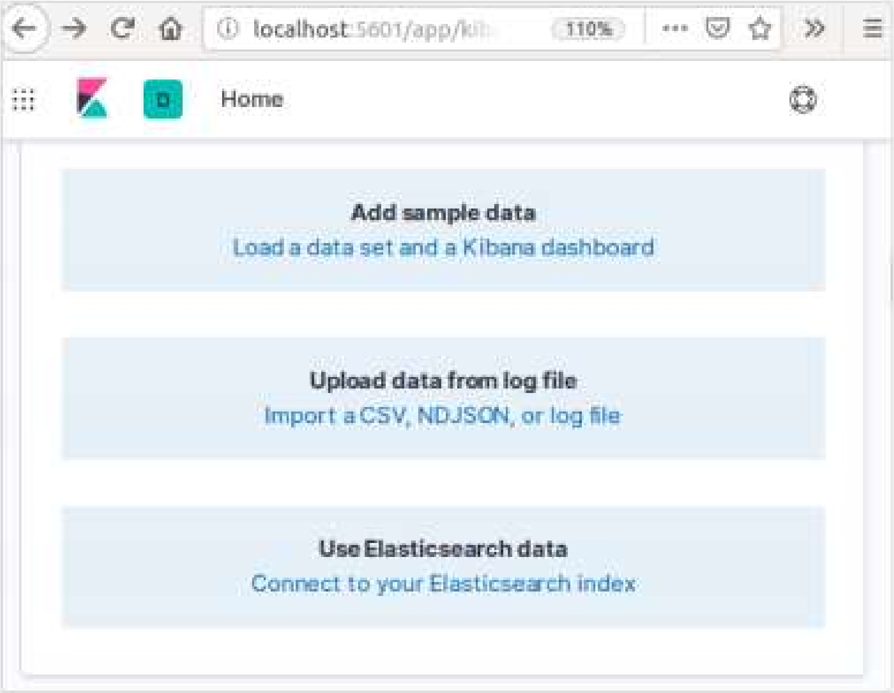
- Response
  - {"acknowledged": true}

Elastic Search – Populate

- Add data
  - POST school/_doc/10
  - {
  - "name":"Saint Paul School", "description":"ICSE Afiliation",
  - "street":"Dawarka", "city":"Delhi", "state":"Delhi", "zip":"110075",
  - "location":[28.5733056, 77.0122136], "fees":5000,
  - "tags":["Good Faculty", "Great Sports"], "rating":"4.5"
  - }

Adding Sample Data in Kibana

- Kibana is a GUI driven tool for accessing the data and creating the visualization.
- In the Kibana home page, choose the following option to add sample ecommerce data:

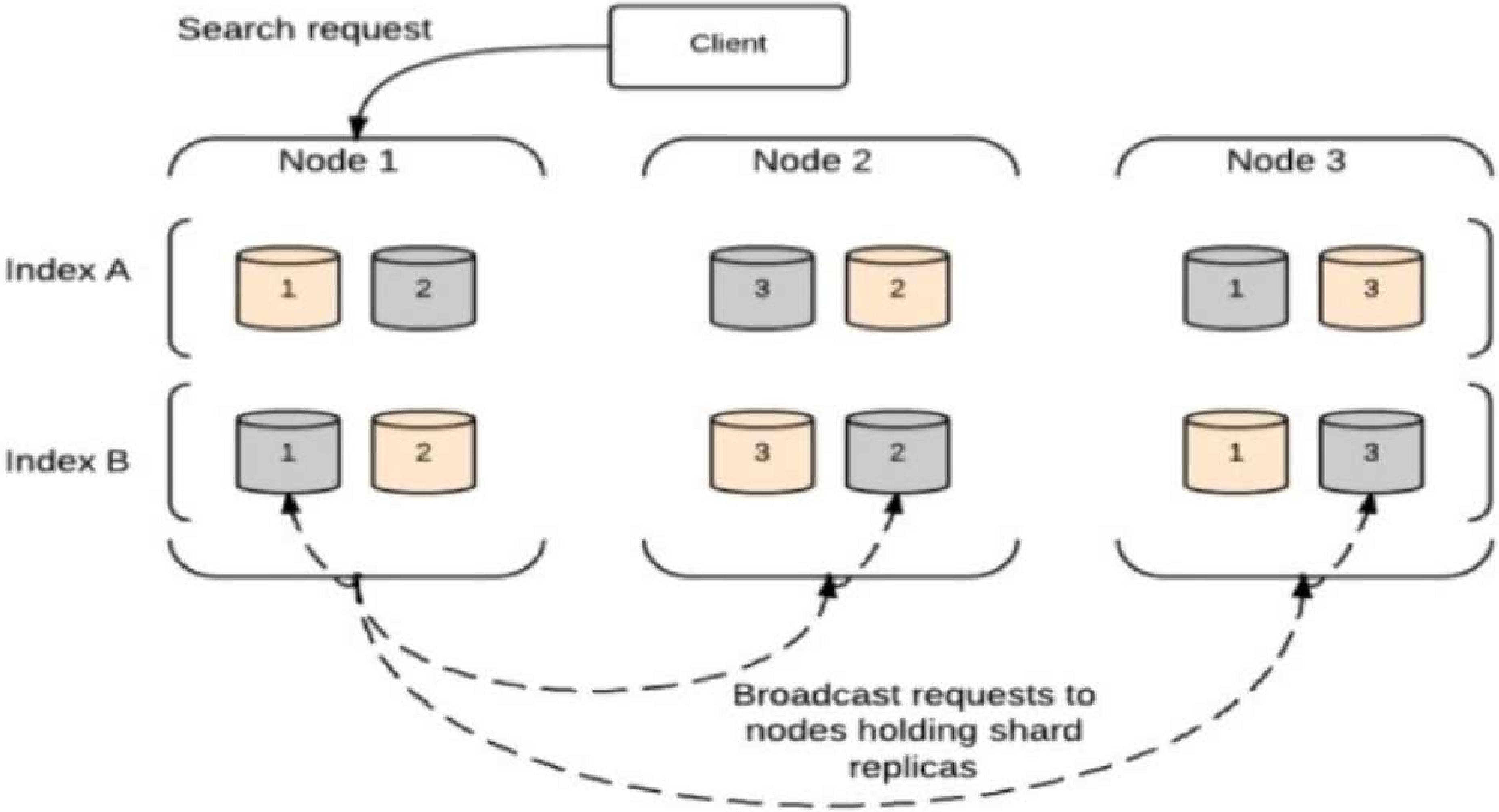


Add sample data
Load a data set and a Kibana dashboard

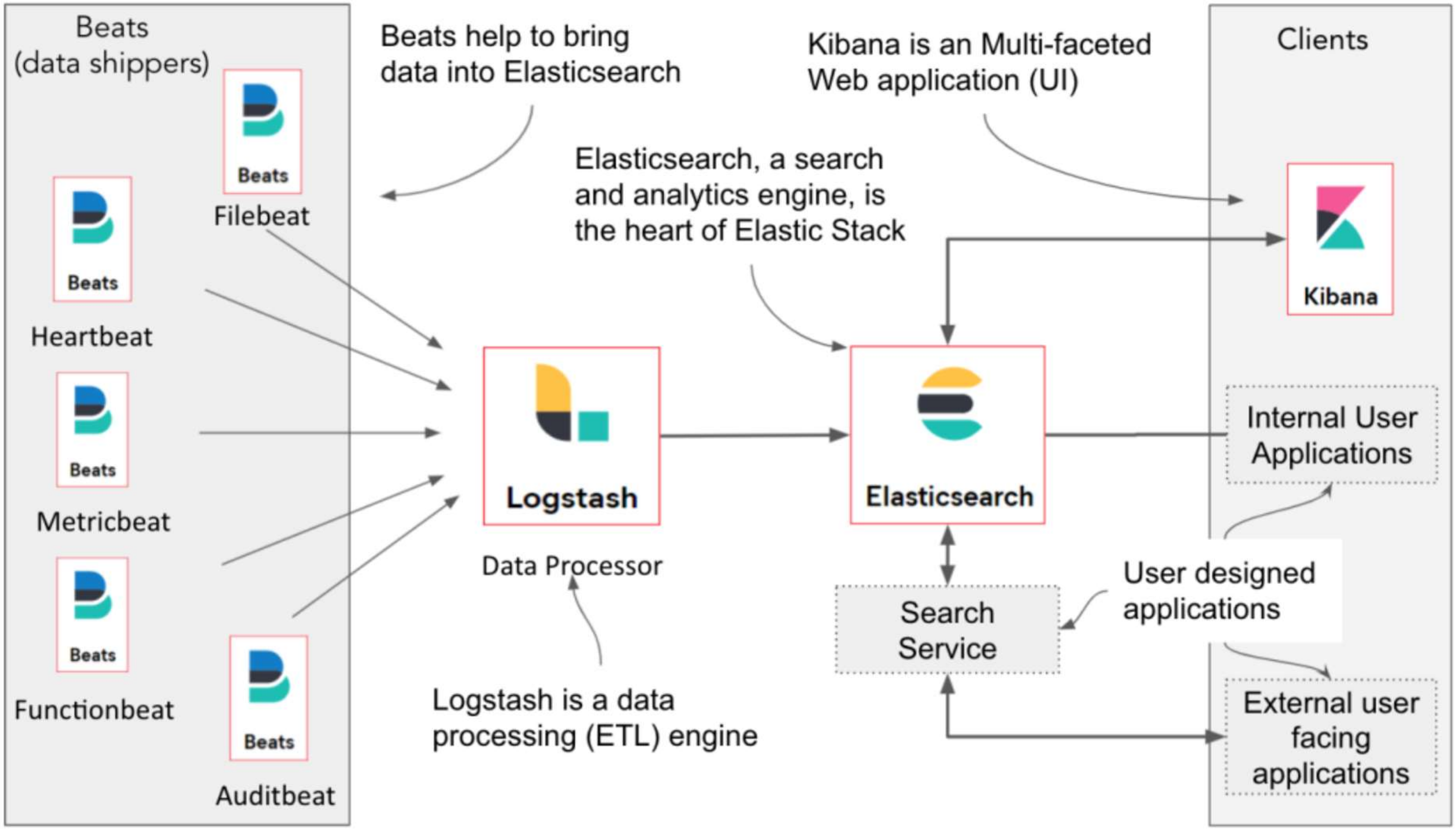
Upload data from log file
Import a CSV, NDJSON, or log file

Use Elasticsearch data
Connect to your Elasticsearch index

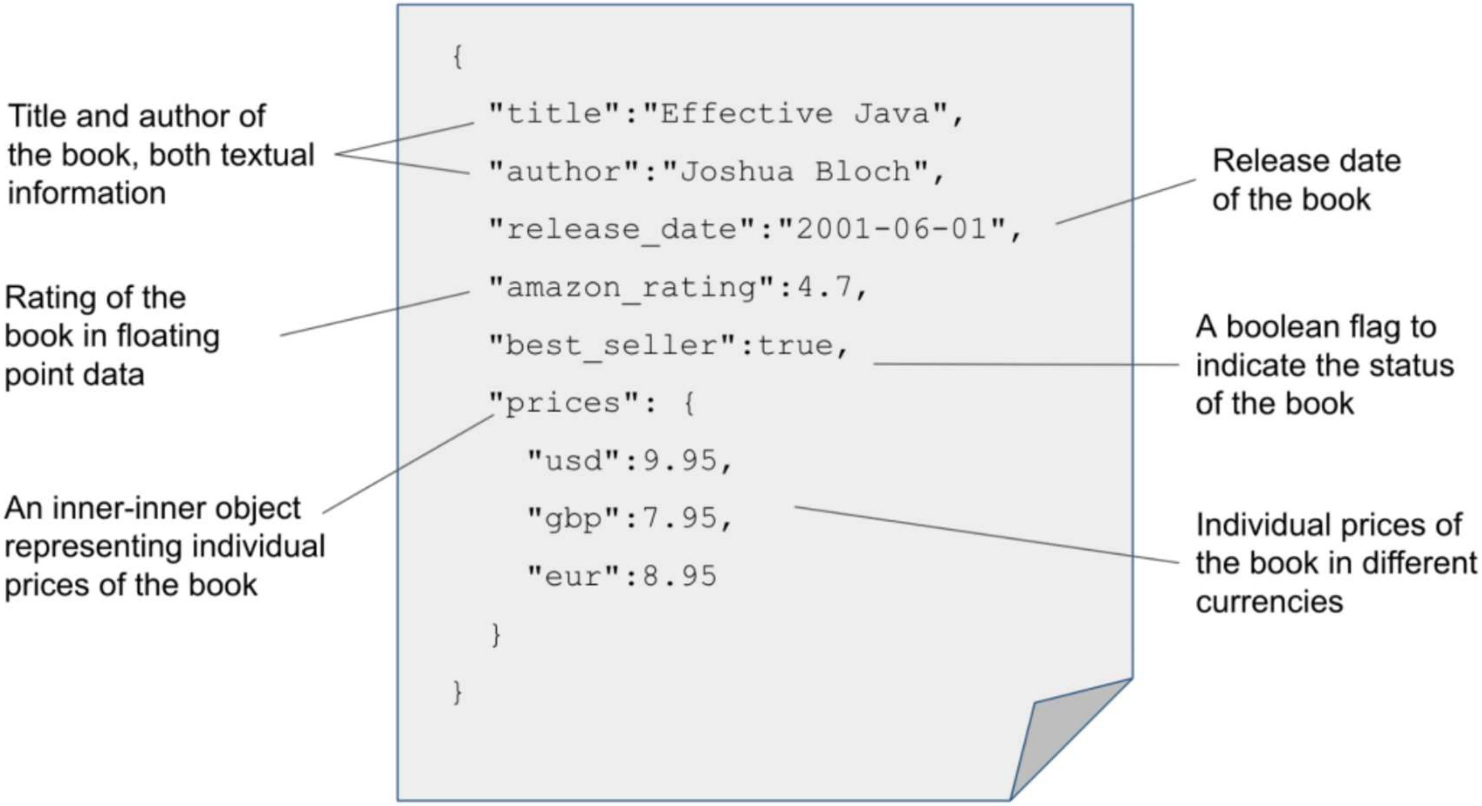
Search Request



Elastic Stack ecosystem



A JSON representation of a book entity



```
{
  "title":"Effective Java",
  "author":"Joshua Bloch",
  "release_date":"2001-06-01",
  "amazon_rating":4.7,
  "best_seller":true,
  "prices": {
    "usd":9.95,
    "gbp":7.95,
    "eur":8.95
  }
}
```

Title and author of the book, both textual information

Release date of the book

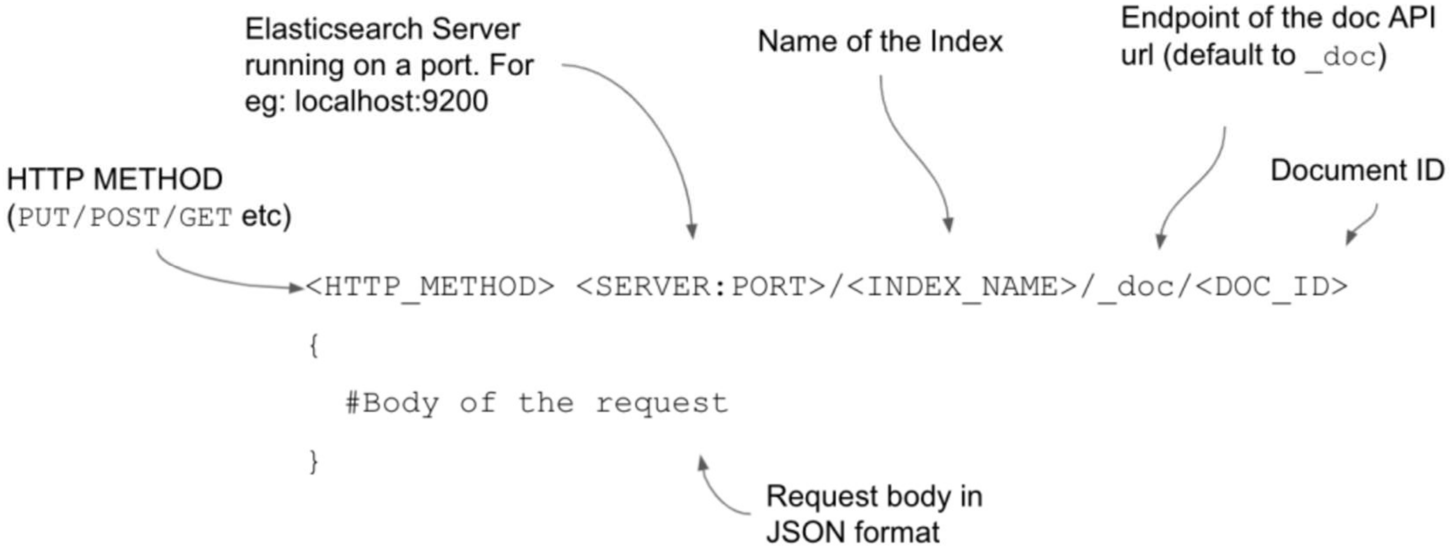
Rating of the book in floating point data

A boolean flag to indicate the status of the book

An inner-inner object representing individual prices of the book

Individual prices of the book in different currencies

Elasticsearch URL invocation endpoint



```
<HTTP_METHOD> <SERVER:PORT>/<INDEX_NAME>/_doc/<DOC_ID>
{
  #Body of the request
}
```

HTTP METHOD (PUT/POST/GET etc)

Elasticsearch Server running on a port. For eg: localhost:9200

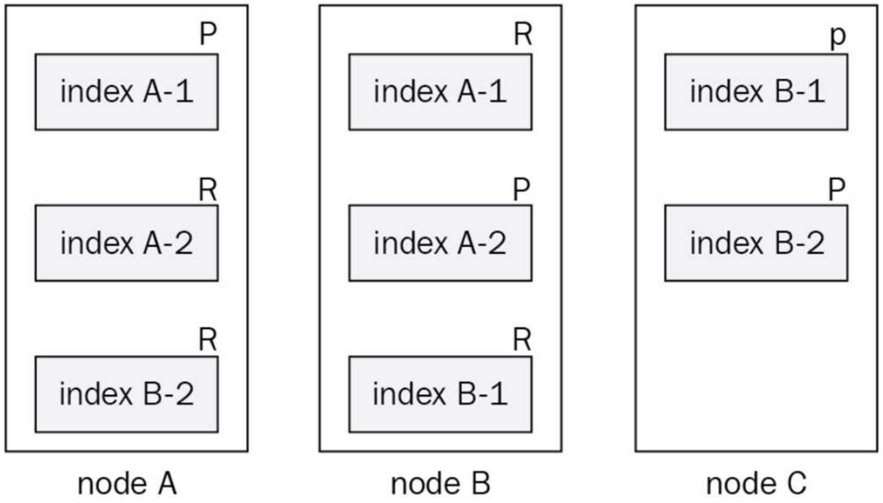
Name of the Index

Endpoint of the doc API url (default to _doc)

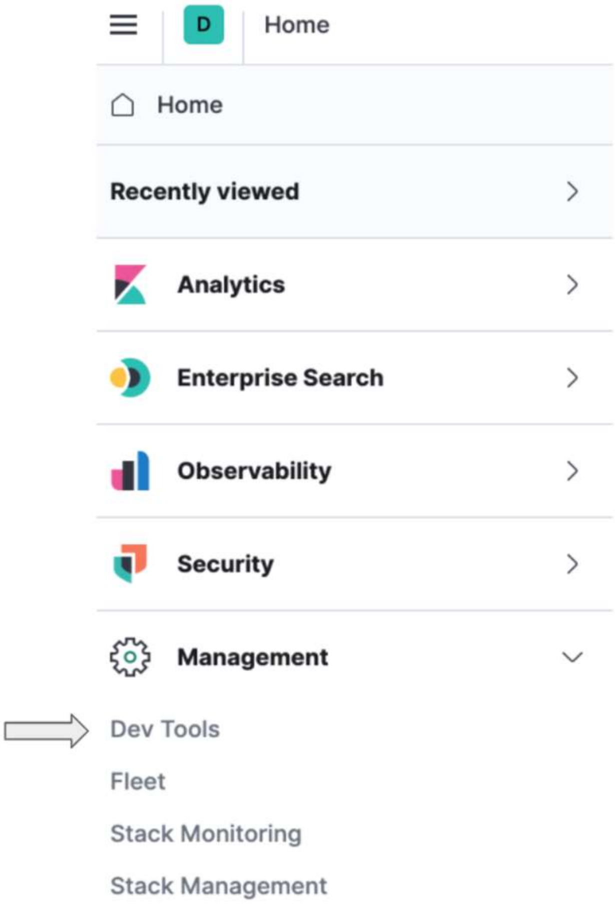
Document ID

Request body in JSON format

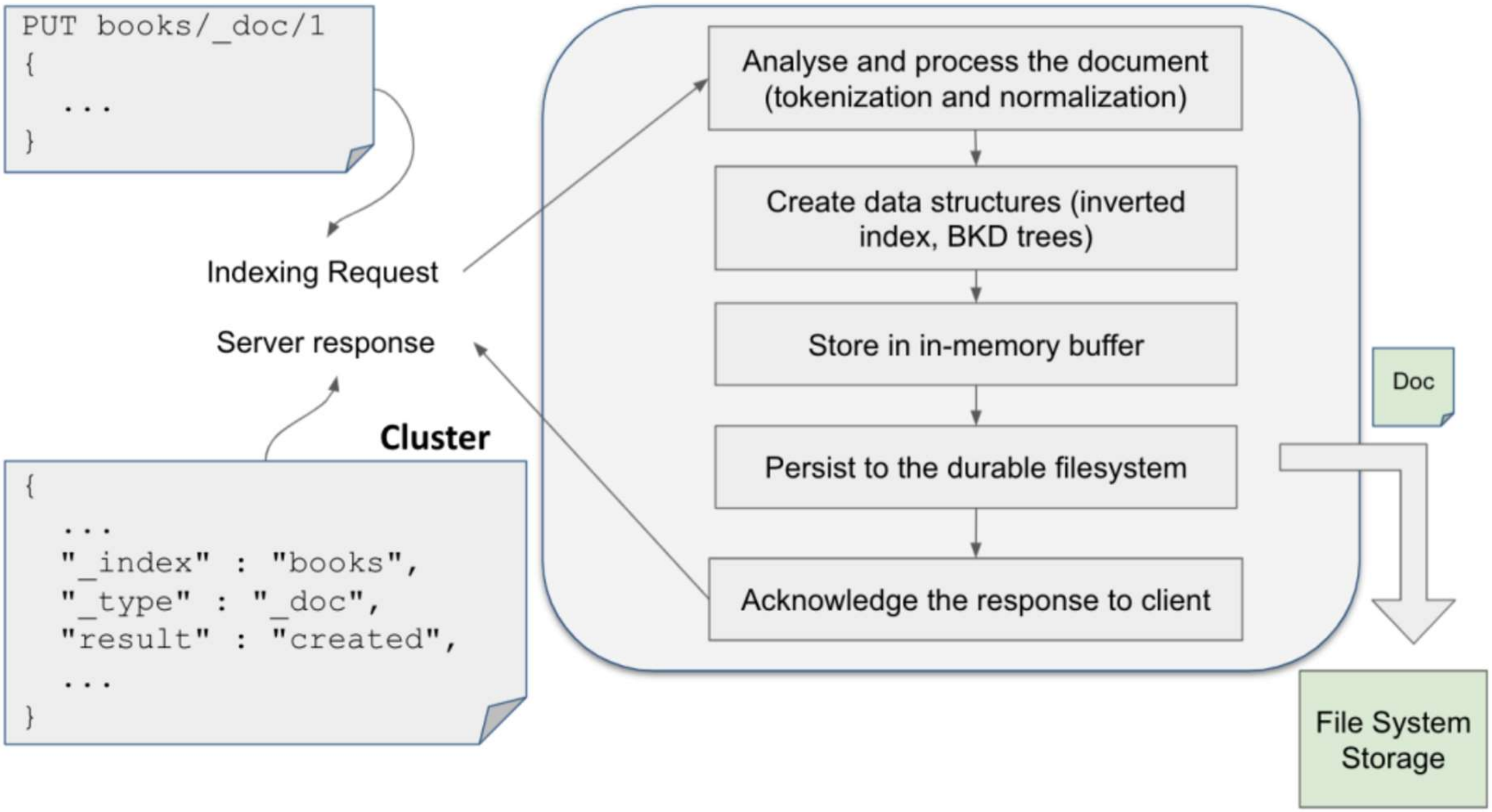
Shards of data distributed across nodes



Accessing the DevTools navigation page



Elasticsearch’s request and response flow



```
PUT books/_doc/1
{
  ...
}
```

Indexing Request

Server response

Cluster

```
{
  ...
  "_index" : "books",
  "_type" : "_doc",
  "result" : "created",
  ...
}
```

Analyse and process the document (tokenization and normalization)

Create data structures (inverted index, BKD trees)

Store in in-memory buffer

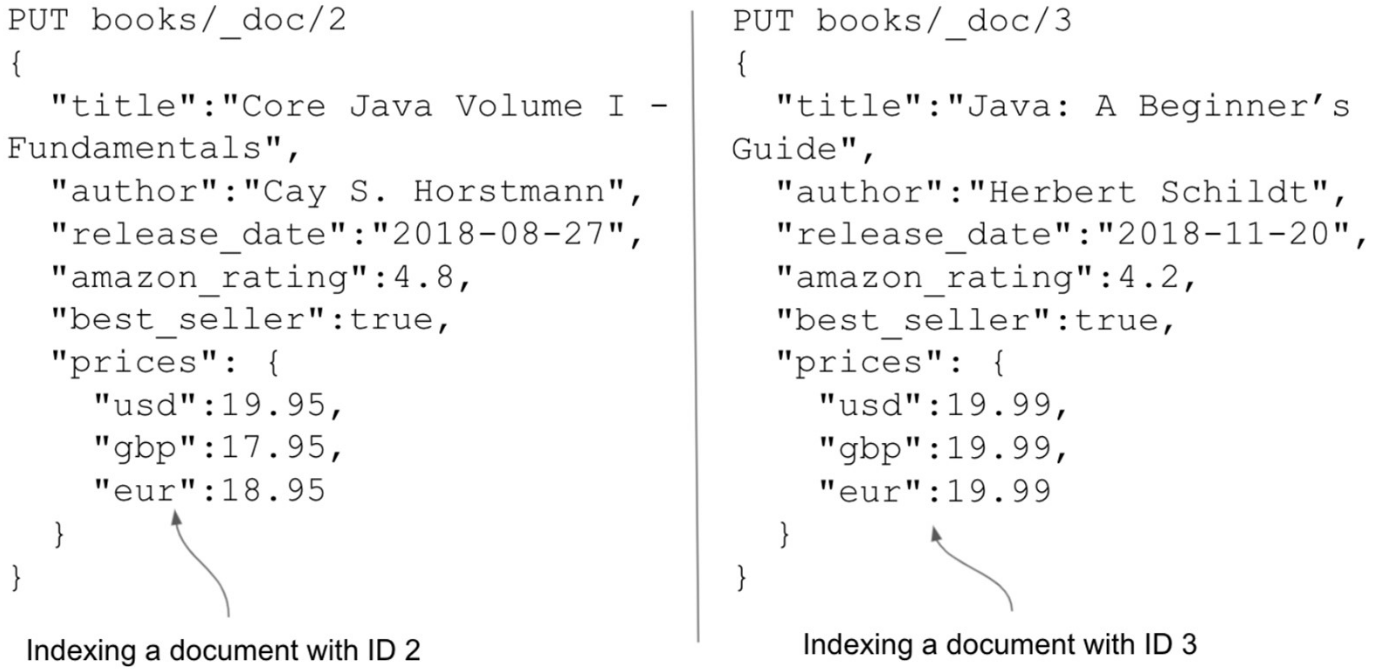
Persist to the durable filesystem

Acknowledge the response to client

Doc

File System Storage

Indexing two more documents using document API



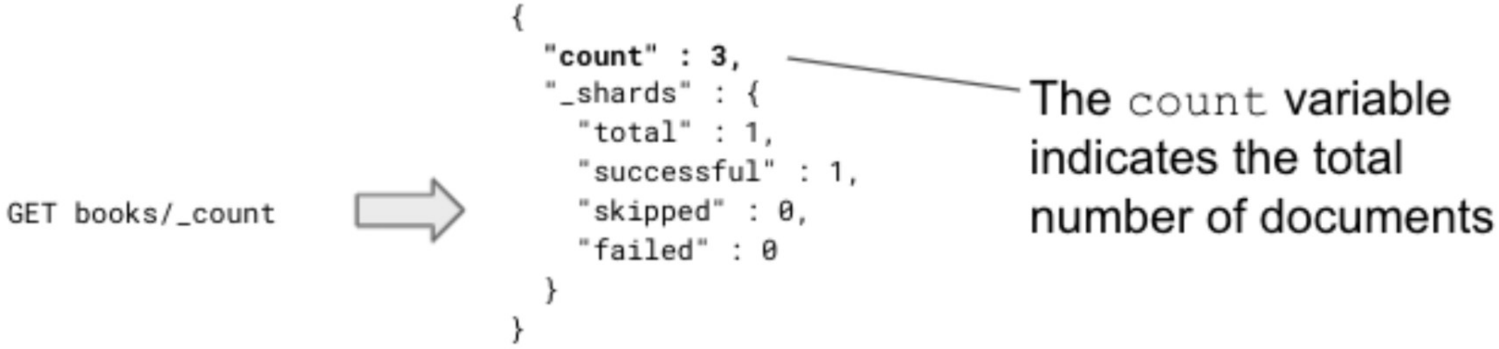
```
PUT books/_doc/2
{
  "title":"Core Java Volume I -
Fundamentals",
  "author":"Cay S. Horstmann",
  "release_date":"2018-08-27",
  "amazon_rating":4.8,
  "best_seller":true,
  "prices": {
    "usd":19.95,
    "gbp":17.95,
    "eur":18.95
  }
}
```

Indexing a document with ID 2

```
PUT books/_doc/3
{
  "title":"Java: A Beginner's
Guide",
  "author":"Herbert Schildt",
  "release_date":"2018-11-20",
  "amazon_rating":4.2,
  "best_seller":true,
  "prices": {
    "usd":19.99,
    "gbp":19.99,
    "eur":19.99
  }
}
```

Indexing a document with ID 3

The JSON response for a _count API invocation

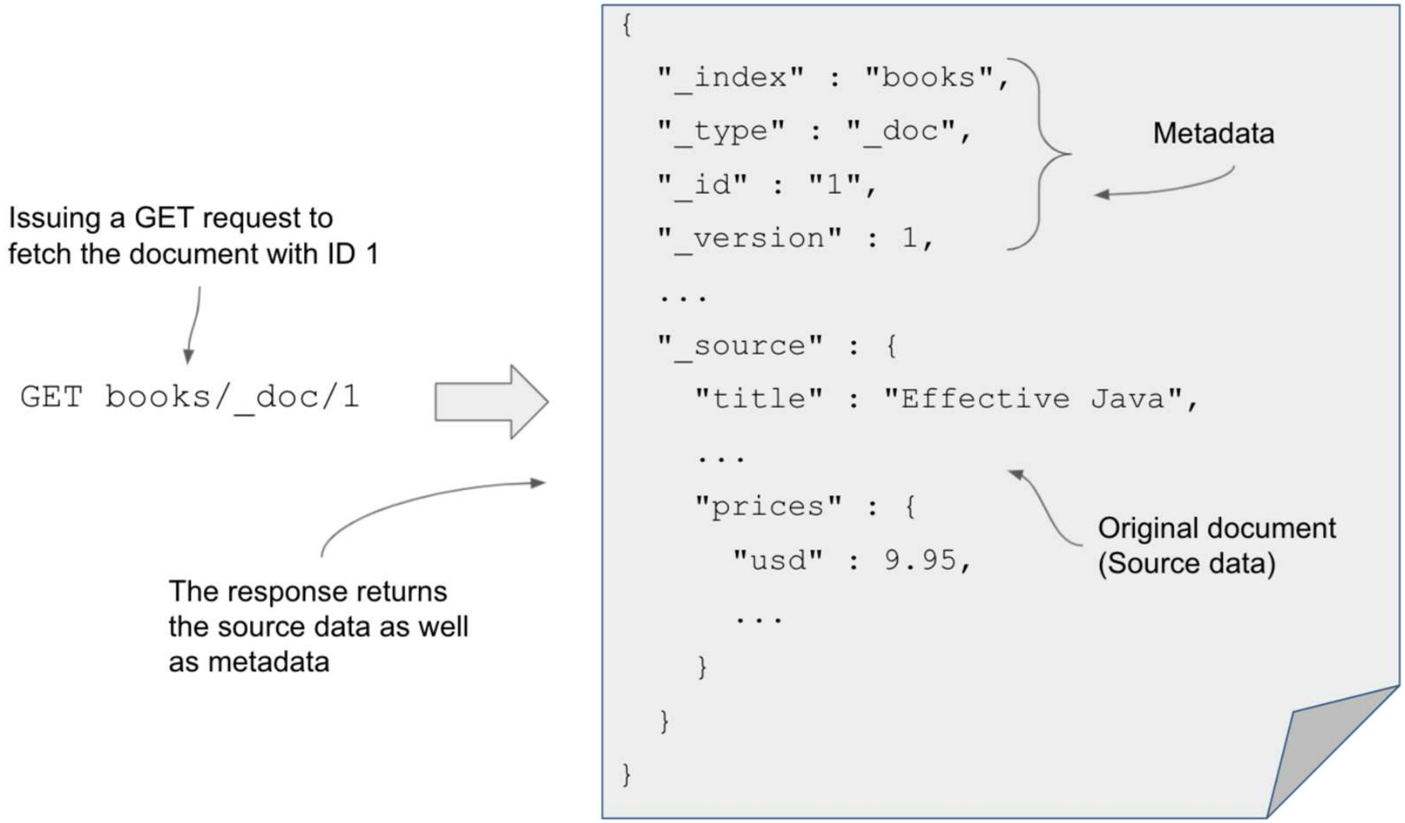


```
GET books/_count
```

```
{
  "count" : 3,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  }
}
```

The `count` variable indicates the total number of documents

Fetching a book document by an ID

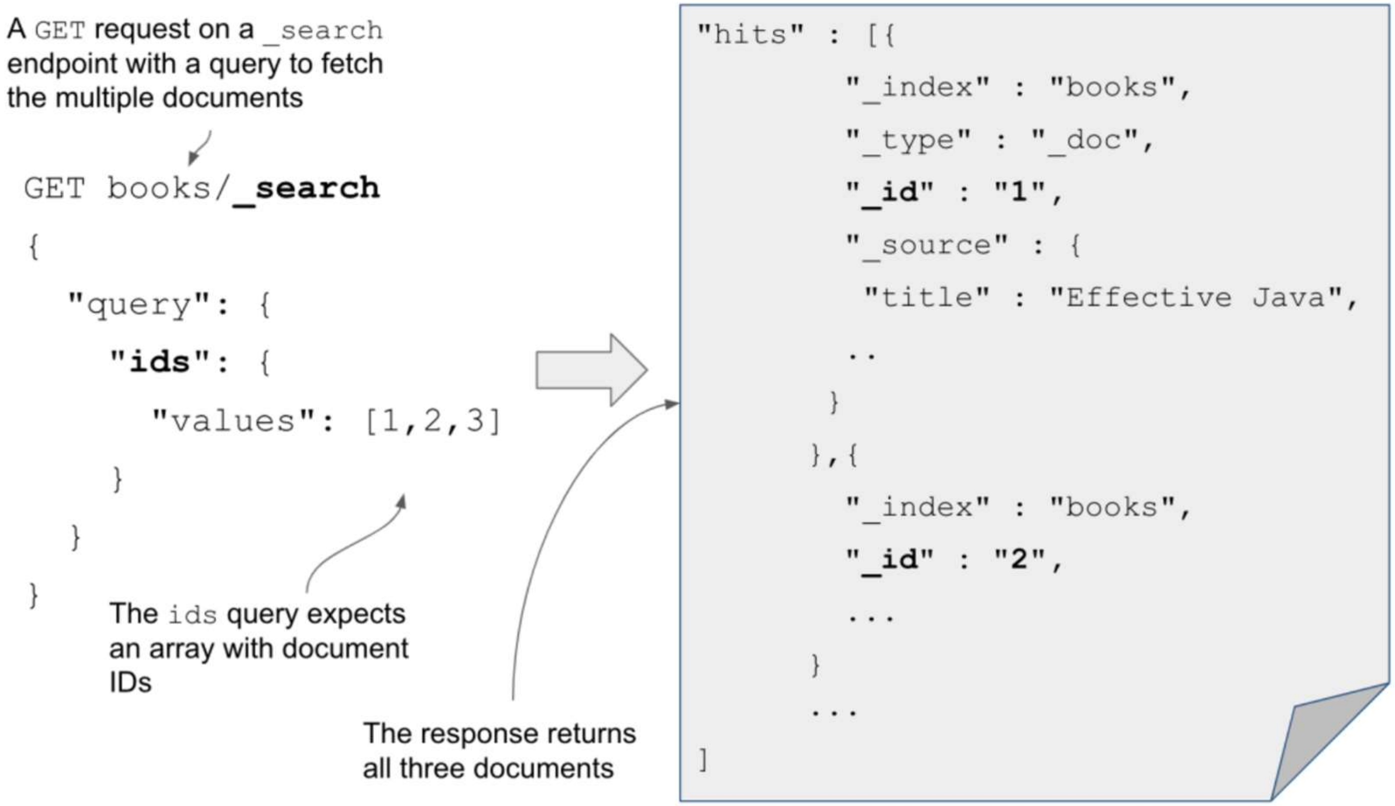


```
GET books/_doc/1
```

```
{
  "_index" : "books",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  ...
  "_source" : {
    "title" : "Effective Java",
    ...
    "prices" : {
      "usd" : 9.95,
      ...
    }
  }
}
```

Issuing a GET request to fetch the document with ID 1
The response returns the source data as well as metadata
Metadata
Original document (Source data)

Retrieving documents given a set of IDs



A GET request on a _search endpoint with a query to fetch the multiple documents

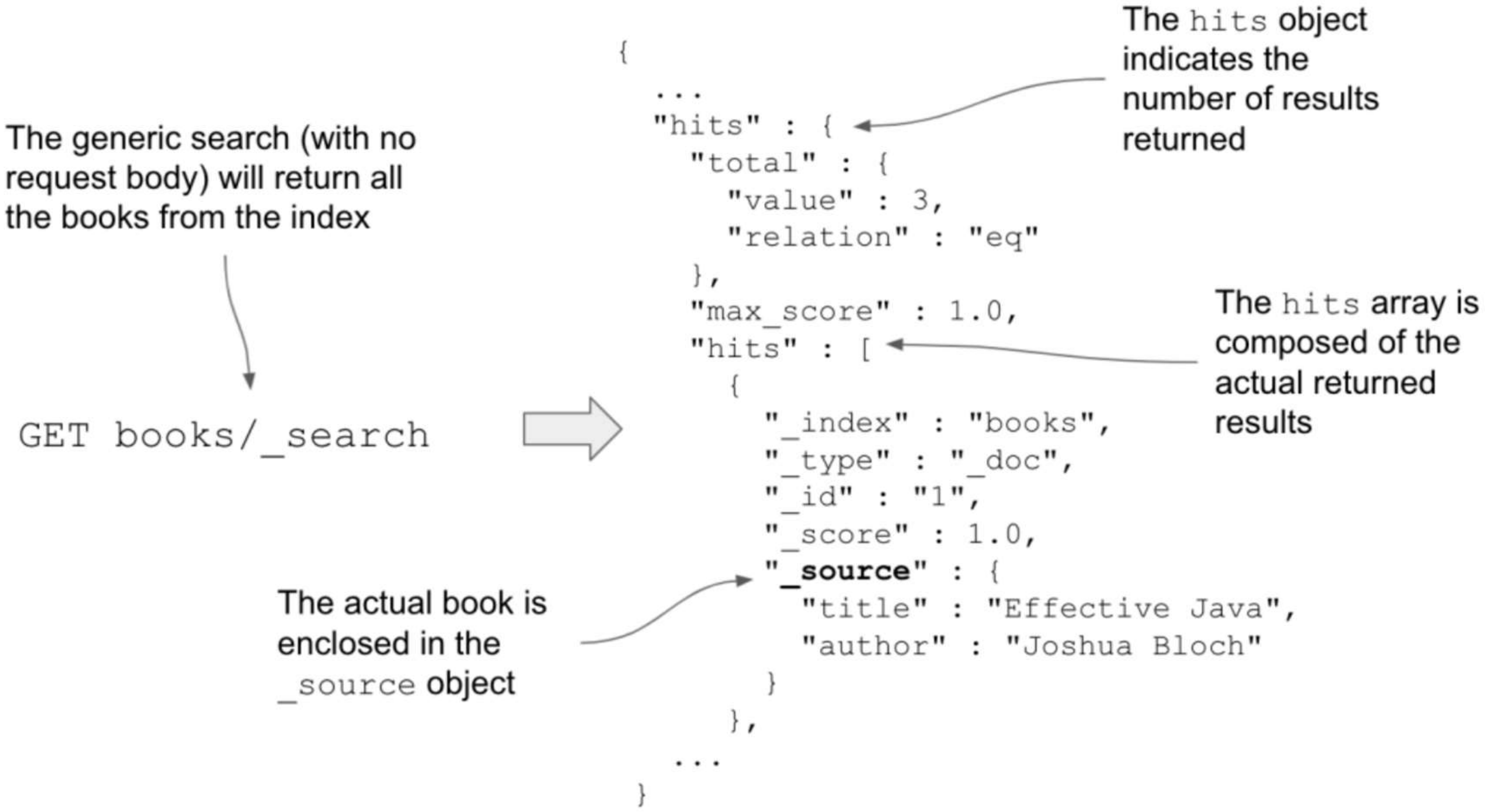
```
GET books/_search
{
  "query": {
    "ids": {
      "values": [1,2,3]
    }
  }
}
```

The ids query expects an array with document IDs

The response returns all three documents

```
"hits" : [{
        "_index" : "books",
        "_type" : "_doc",
        "_id" : "1",
        "_source" : {
         "title" : "Effective Java",
        ..
       }
      },{
        "_index" : "books",
        "_id" : "2",
        ...
      }
      ...
]
```

Retrieving all documents using the search API



The generic search (with no request body) will return all the books from the index

```
GET books/_search
```

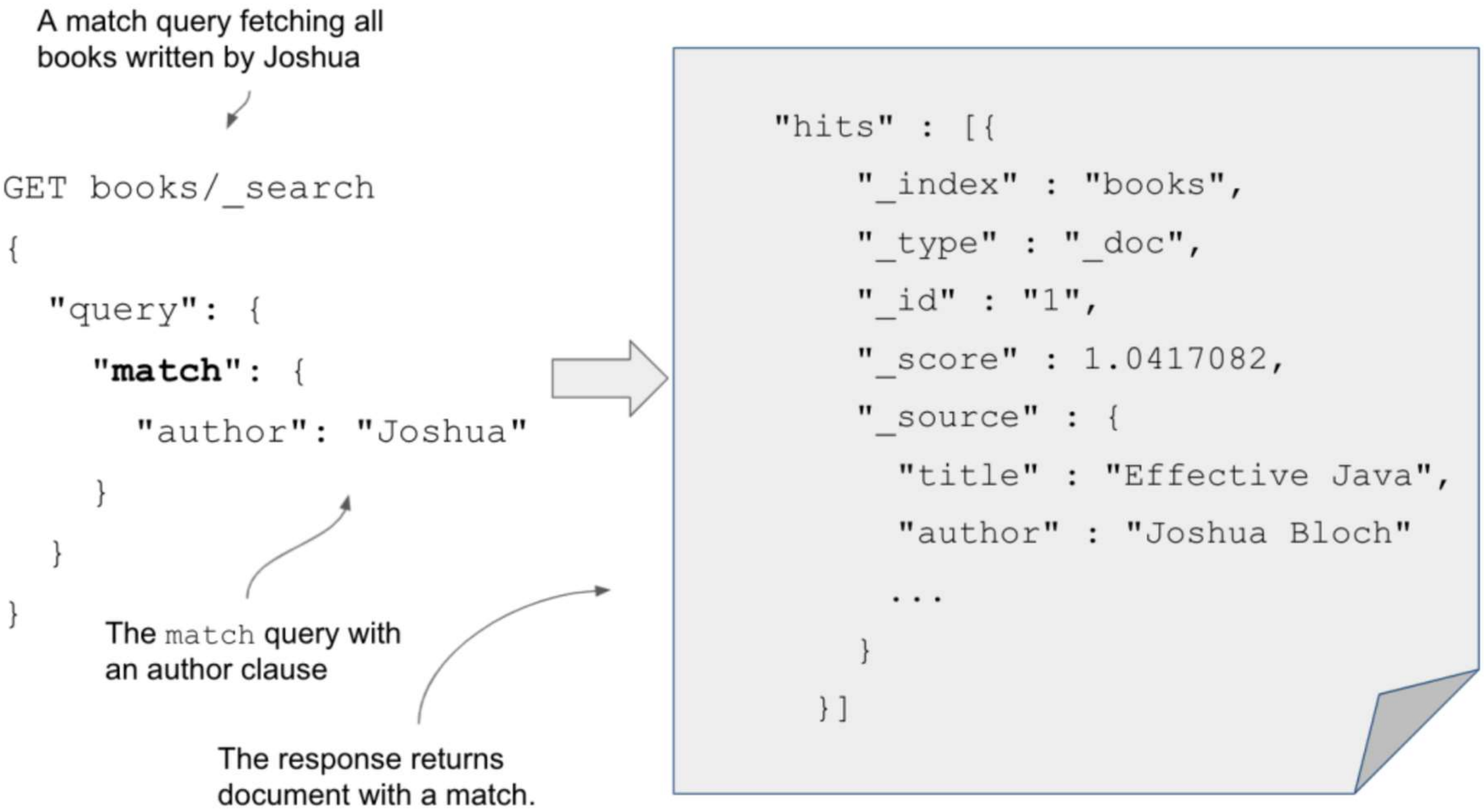
```
{
  ...
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "books",
        "_type" : "_doc",
        "_id" : "1",
        "_score" : 1.0,
        "_source" : {
          "title" : "Effective Java",
          "author" : "Joshua Bloch"
        }
      },
    ...
}
```

The `hits` object indicates the number of results returned

The `hits` array is composed of the actual returned results

The actual book is enclosed in the `_source` object

Fetching books



A match query fetching all books written by Joshua

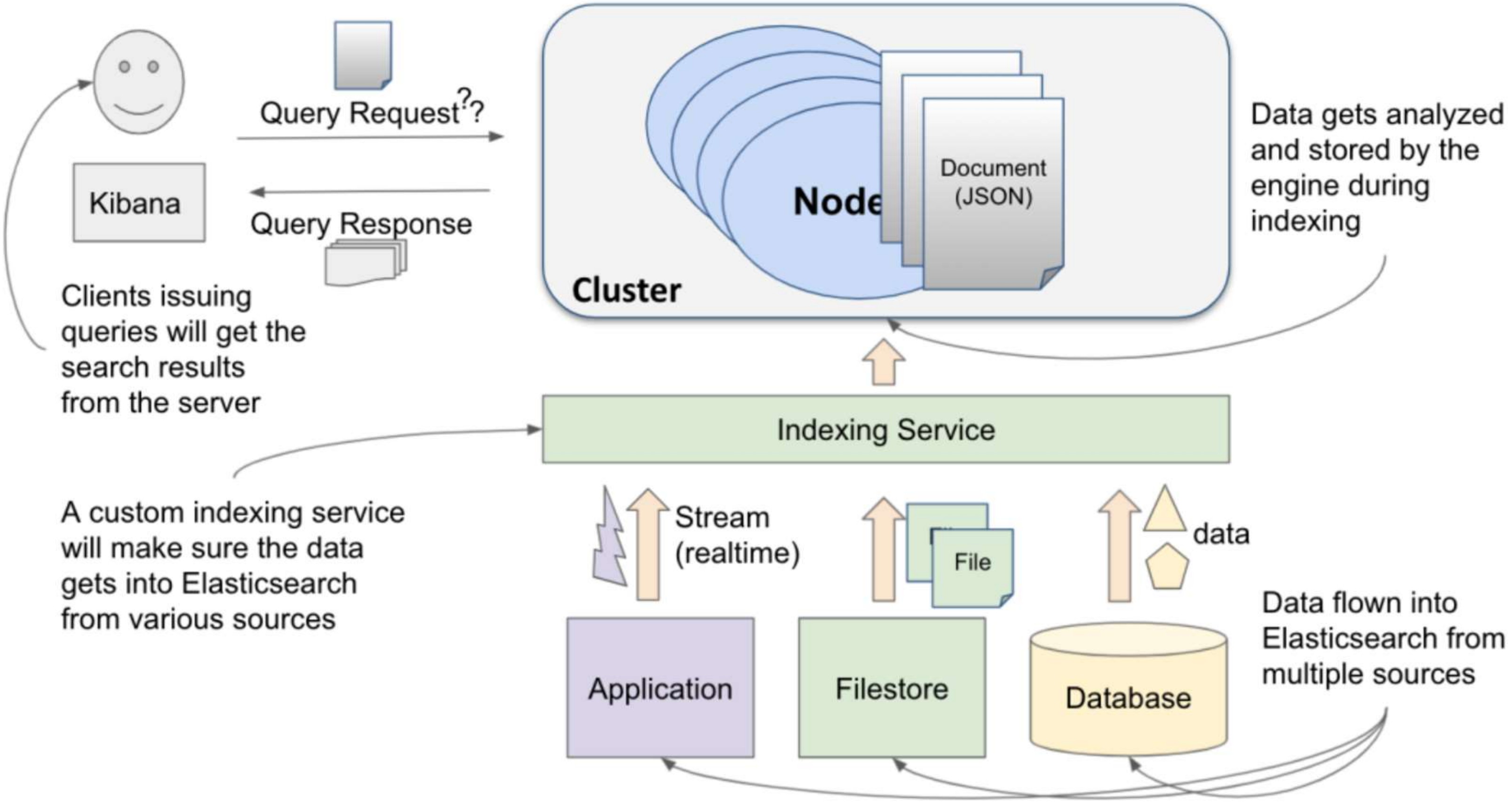
```
GET books/_search
{
  "query": {
    "match": {
      "author": "Joshua"
    }
  }
}
```

The `match` query with an author clause

```
"hits" : [{
    "_index" : "books",
    "_type" : "_doc",
    "_id" : "1",
    "_score" : 1.0417082,
    "_source" : {
      "title" : "Effective Java",
      "author" : "Joshua Bloch"
      ...
    }
  }]
```

The response returns document with a match.

Elasticsearch with data



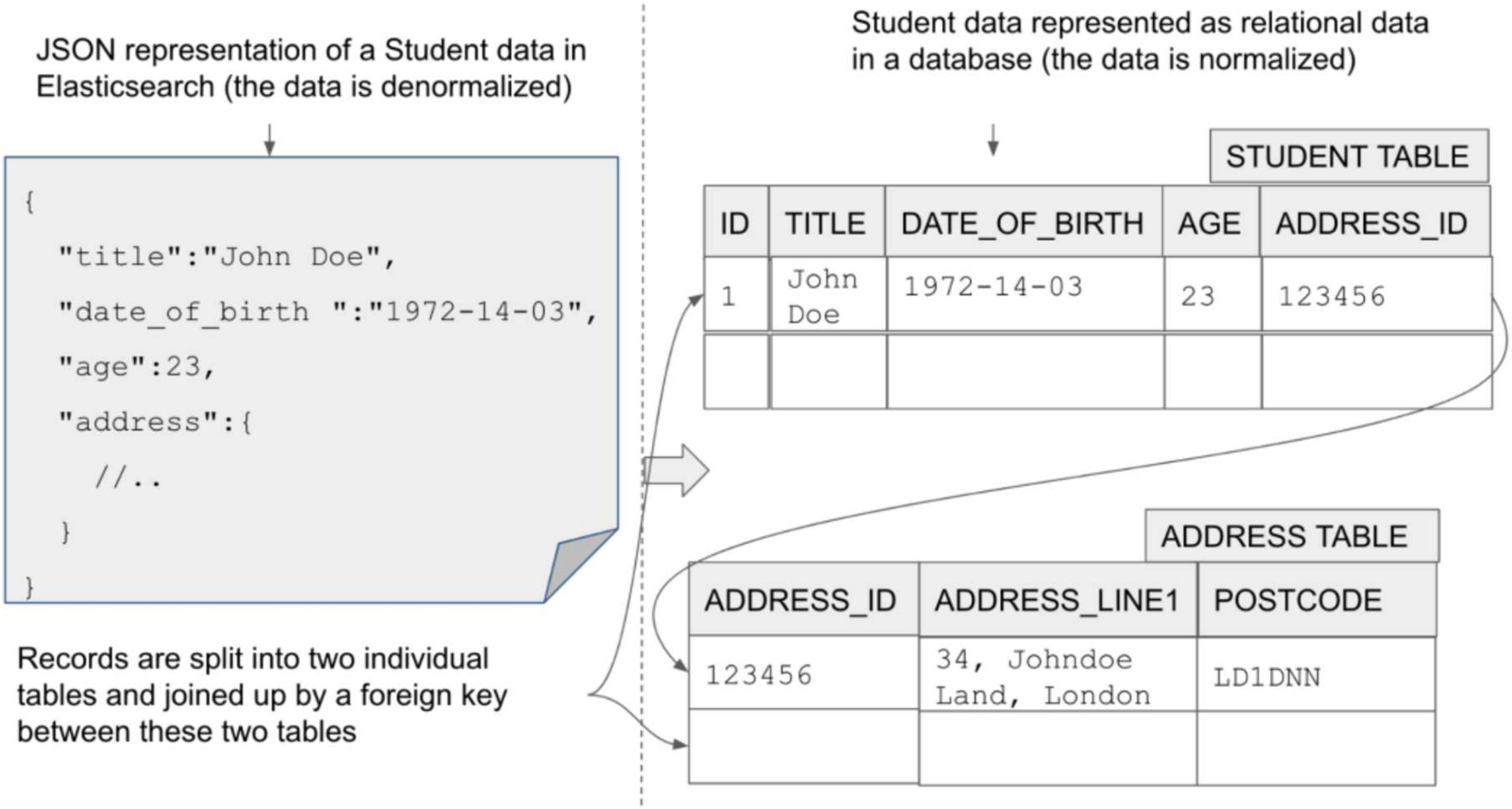
Data gets analyzed and stored by the engine during indexing

Clients issuing queries will get the search results from the server

A custom indexing service will make sure the data gets into Elasticsearch from various sources

Data flown into Elasticsearch from multiple sources

JSON vs. a relational database table structure



JSON representation of a Student data in Elasticsearch (the data is denormalized)

```
{
  "title":"John Doe",
  "date_of_birth ":"1972-14-03",
  "age":23,
  "address":{
    //..
  }
}
```

Student data represented as relational data in a database (the data is normalized)

STUDENT TABLE

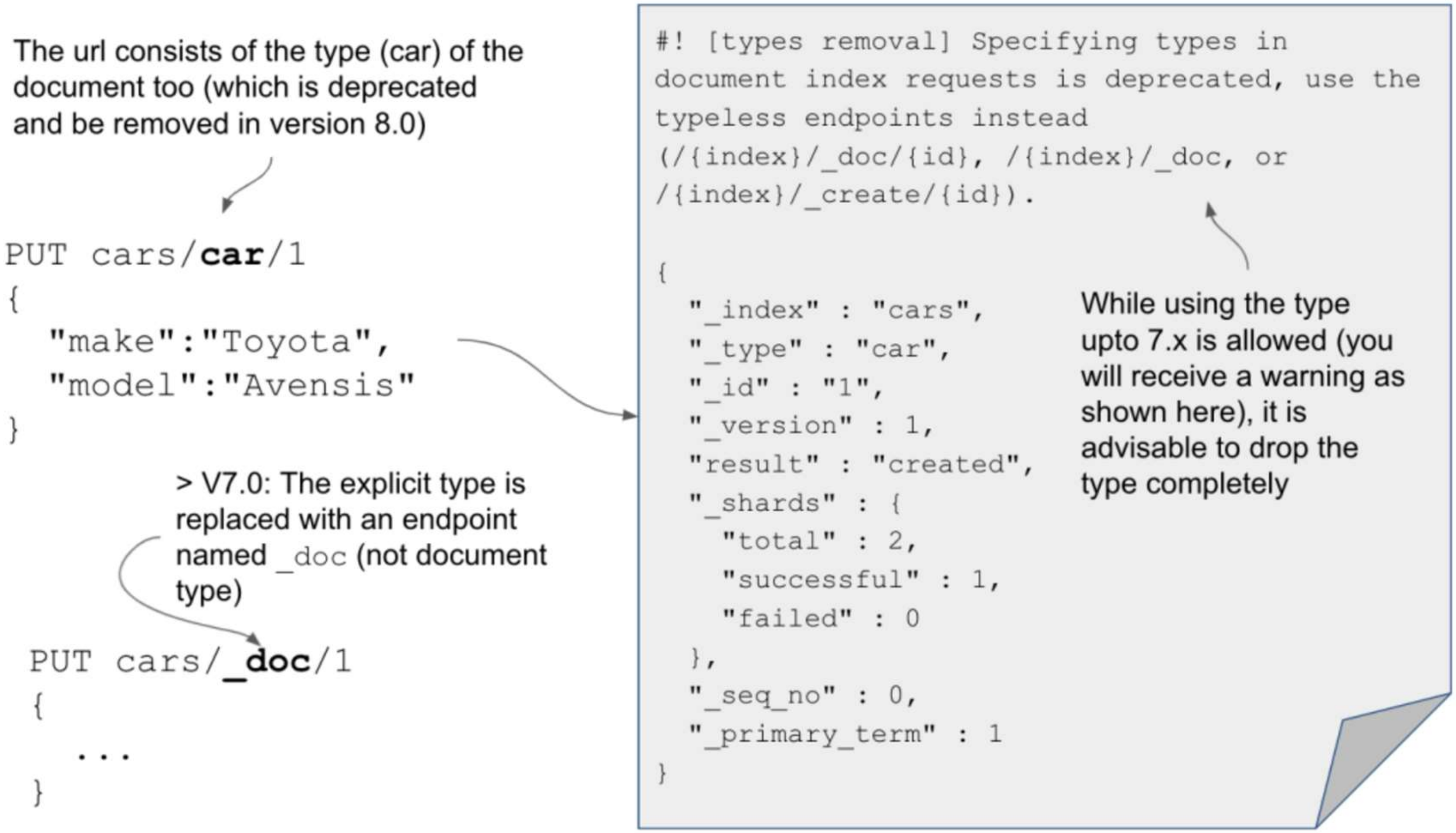
ID	TITLE	DATE_OF_BIRTH	AGE	ADDRESS_ID
1	John Doe	1972-14-03	23	123456

ADDRESS TABLE

ADDRESS_ID	ADDRESS_LINE1	POSTCODE
123456	34, Johndoe Land, London	LD1DNN

Records are split into two individual tables and joined up by a foreign key between these two tables

Removing document types



The url consists of the type (car) of the document too (which is deprecated and be removed in version 8.0)

```
PUT cars/car/1
{
  "make":"Toyota",
  "model":"Avensis"
}
```

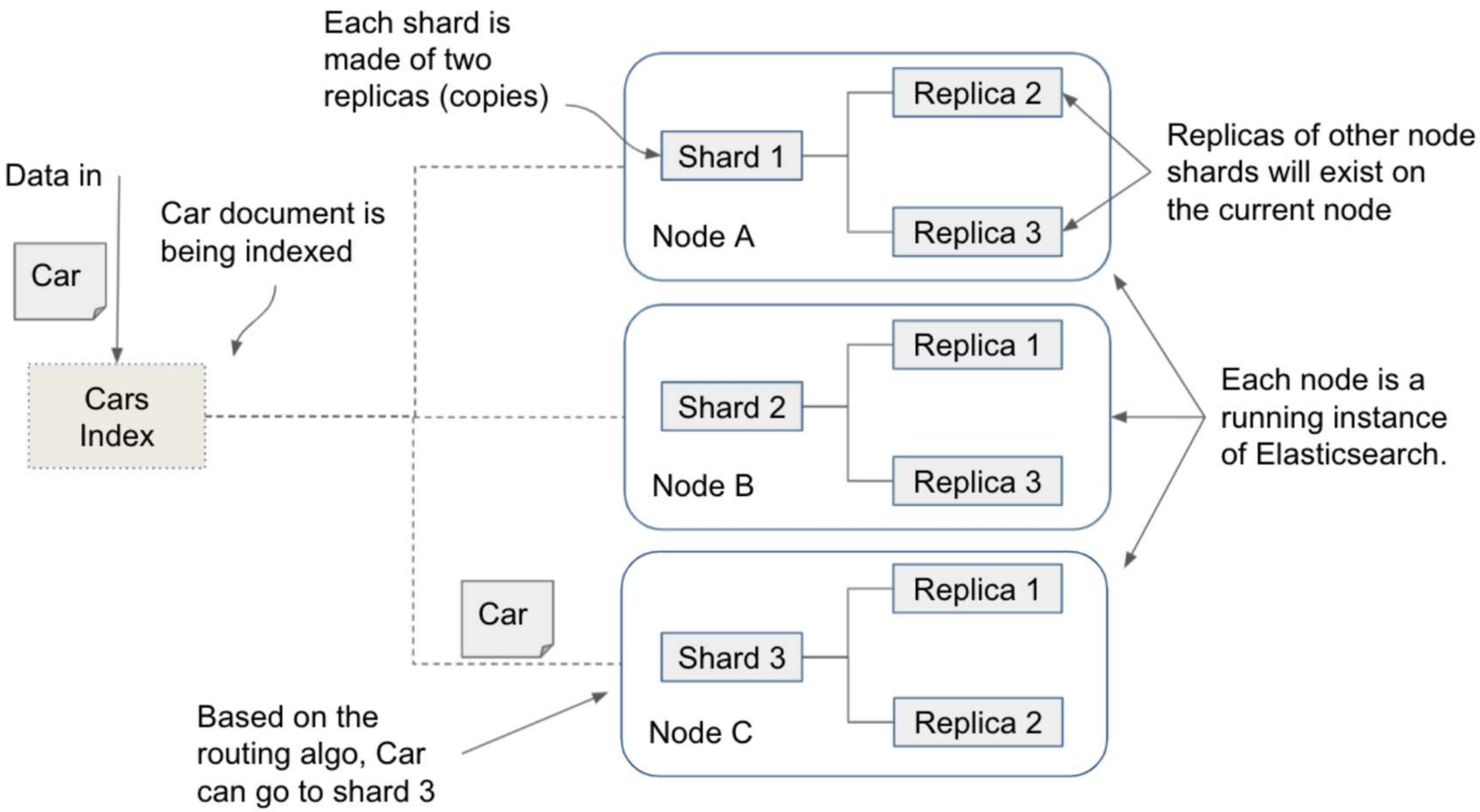
> V7.0: The explicit type is replaced with an endpoint named _doc (not document type)

```
PUT cars/_doc/1
{
   ...
}
```

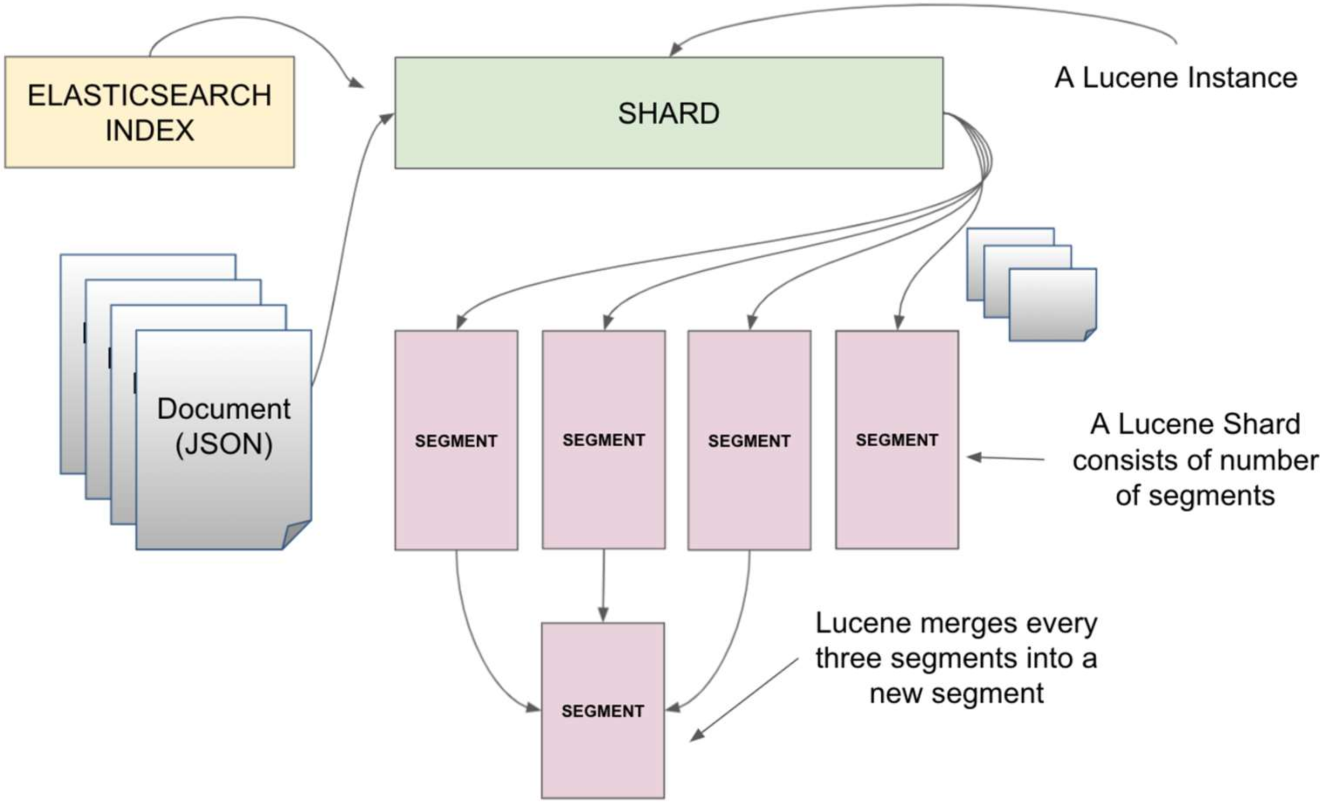
```
#! [types removal] Specifying types in
document index requests is deprecated, use the
typeless endpoints instead
(/{index}/_doc/{id}, /{index}/_doc, or
/{index}/_create/{id}).

{
  "_index" : "cars",
  "_type" : "car",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

While using the type upto 7.x is allowed (you will receive a warning as shown here), it is advisable to drop the type completely

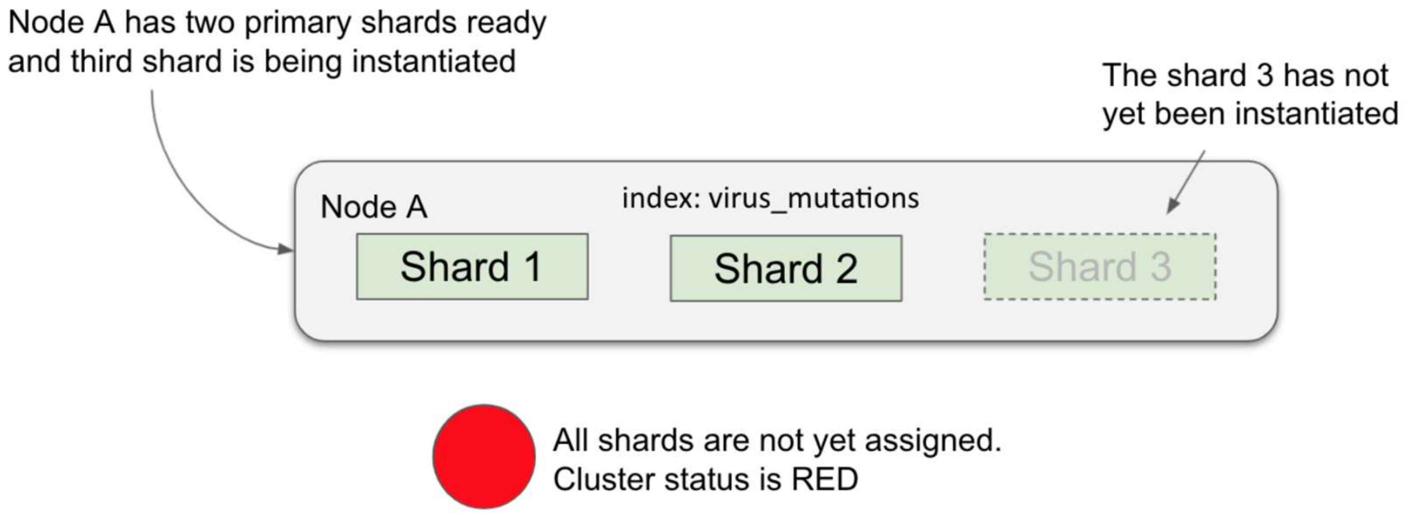
An index


Each shard is made of two replicas (copies)
Data in
Car
Car document is being indexed
Cars Index
Shard 1
Replica 2
Replica 3
Node A
Replicas of other node shards will exist on the current node
Replica 1
Shard 2
Replica 3
Node B
Each node is a running instance of Elasticsearch.
Car
Replica 1
Shard 3
Replica 2
Node C
Based on the routing algo, Car can go to shard 3

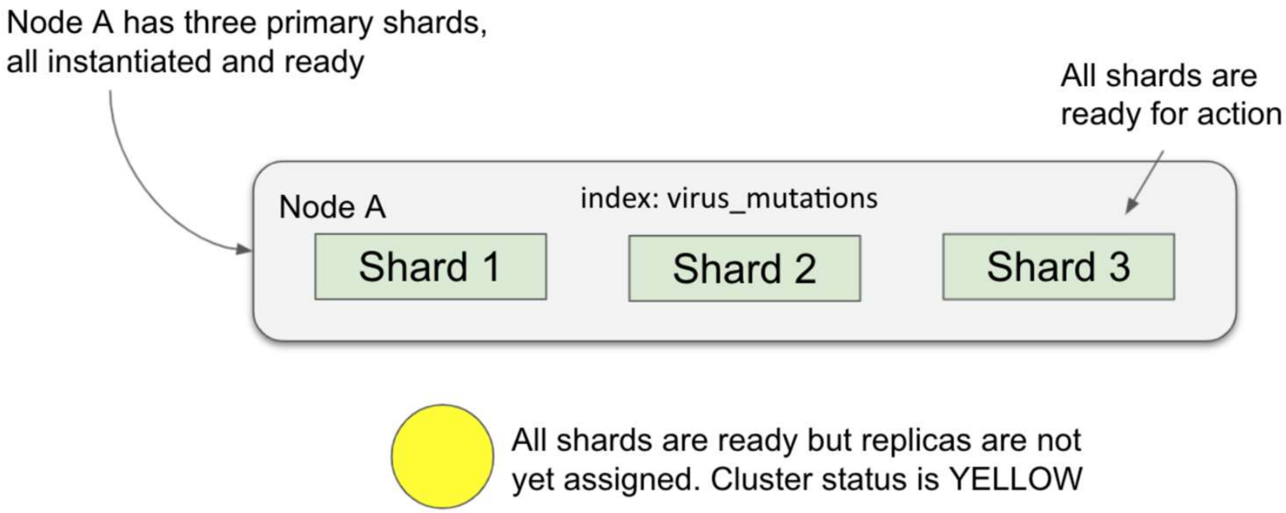
Lucene’s mechanism for indexing documents


ELASTICSEARCH INDEX
SHARD
A Lucene Instance
Document (JSON)
SEGMENT
SEGMENT
SEGMENT
SEGMENT
A Lucene Shard consists of number of segments
SEGMENT
Lucene merges every three segments into a new segment

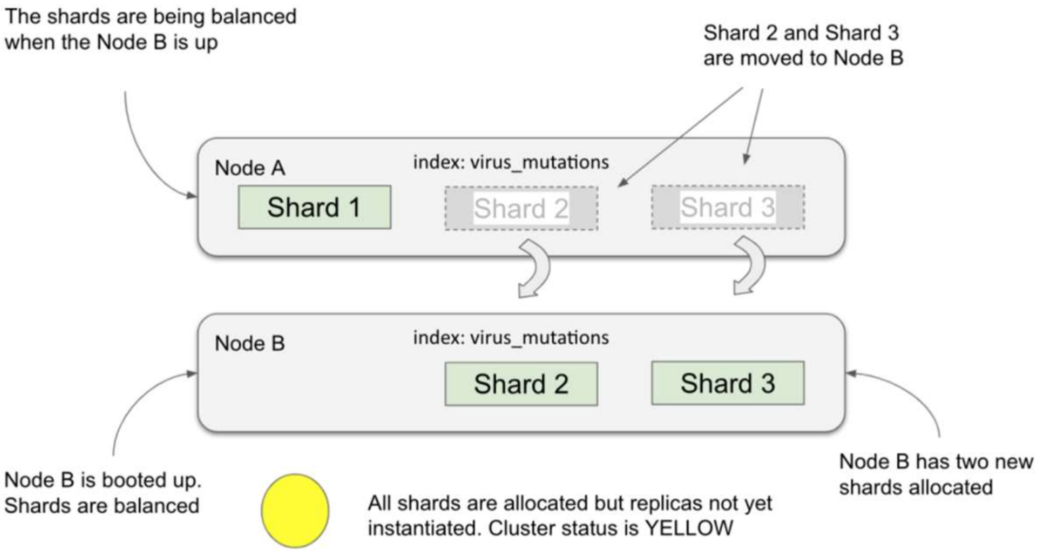
Engine is not ready showing RED status



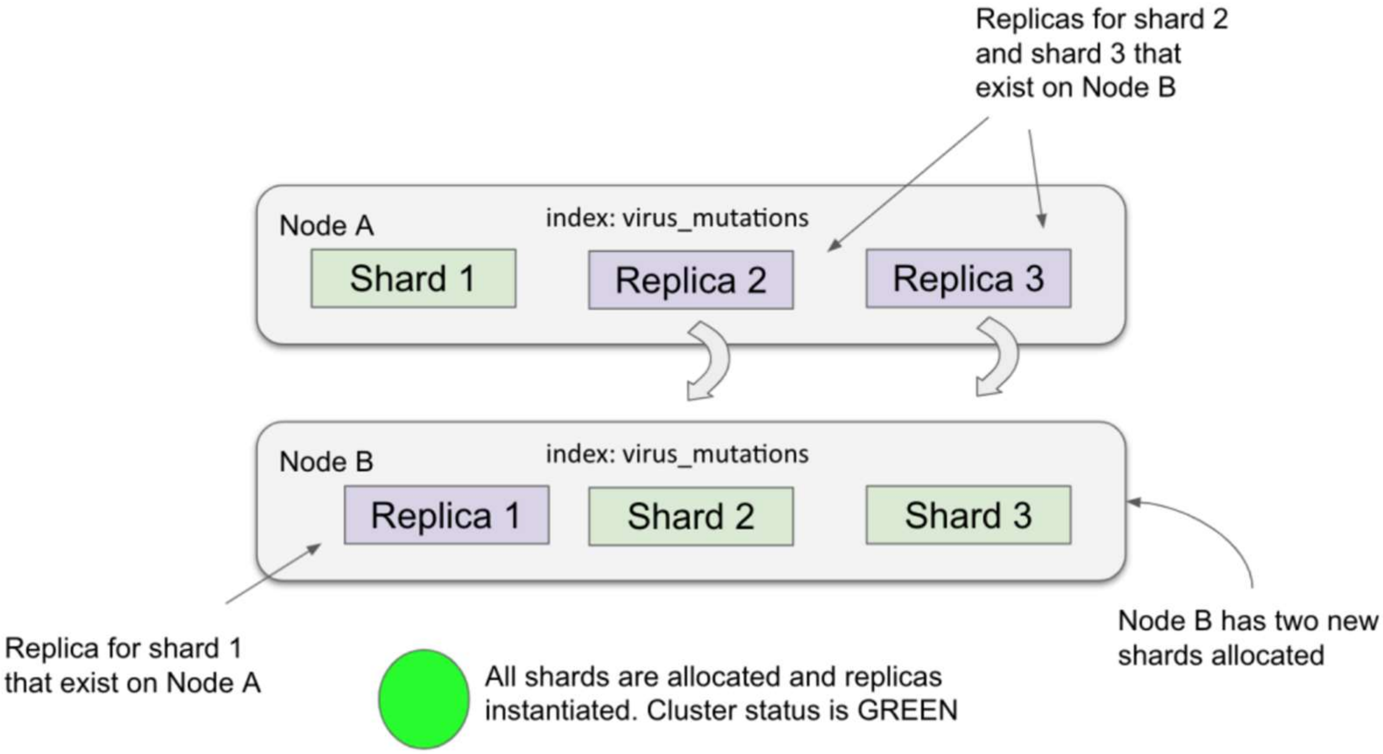
A single node with three shards


Node A has three primary shards,
all instantiated and ready
All shards are
ready for action
Node A
index: virus_mutations
Shard 1
Shard 2
Shard 3
All shards are ready but replicas are not
yet assigned. Cluster status is YELLOW

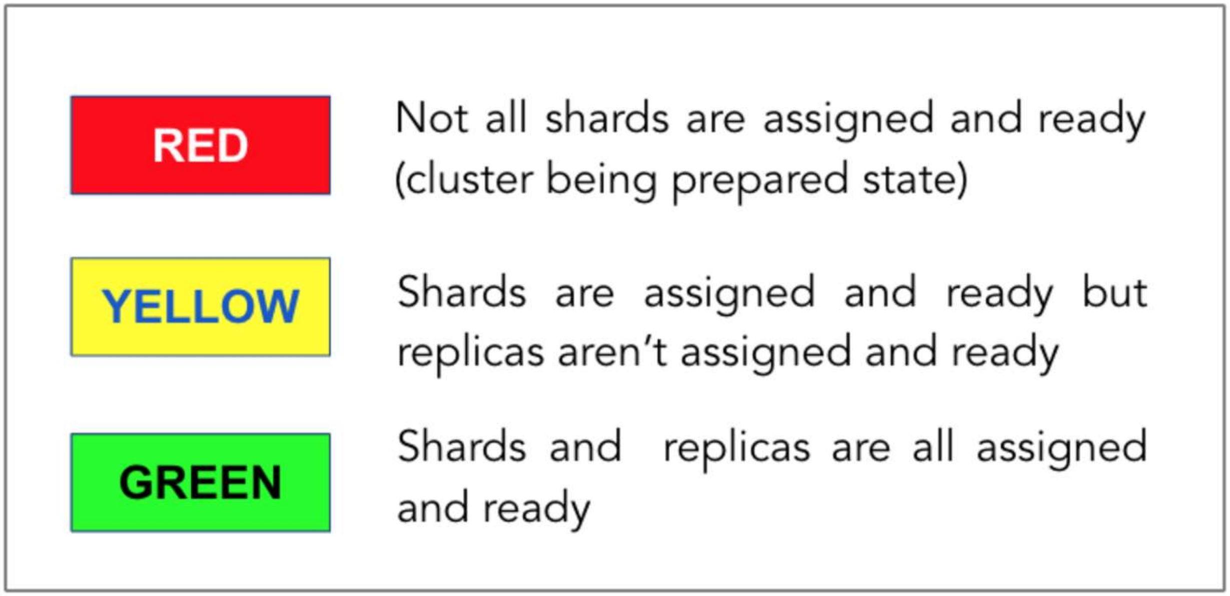
Shards balanced on new node but replicas not assigned


The shards are being balanced when the Node B is up
Shard 2 and Shard 3 are moved to Node B
Node A
index: virus_mutations
Shard 1
Shard 2
Shard 3
Node B
index: virus_mutations
Shard 2
Shard 3
Node B is booted up. Shards are balanced
All shards are allocated but replicas not yet instantiated. Cluster status is YELLOW
Node B has two new shards allocated

All shards and replicas are allocated

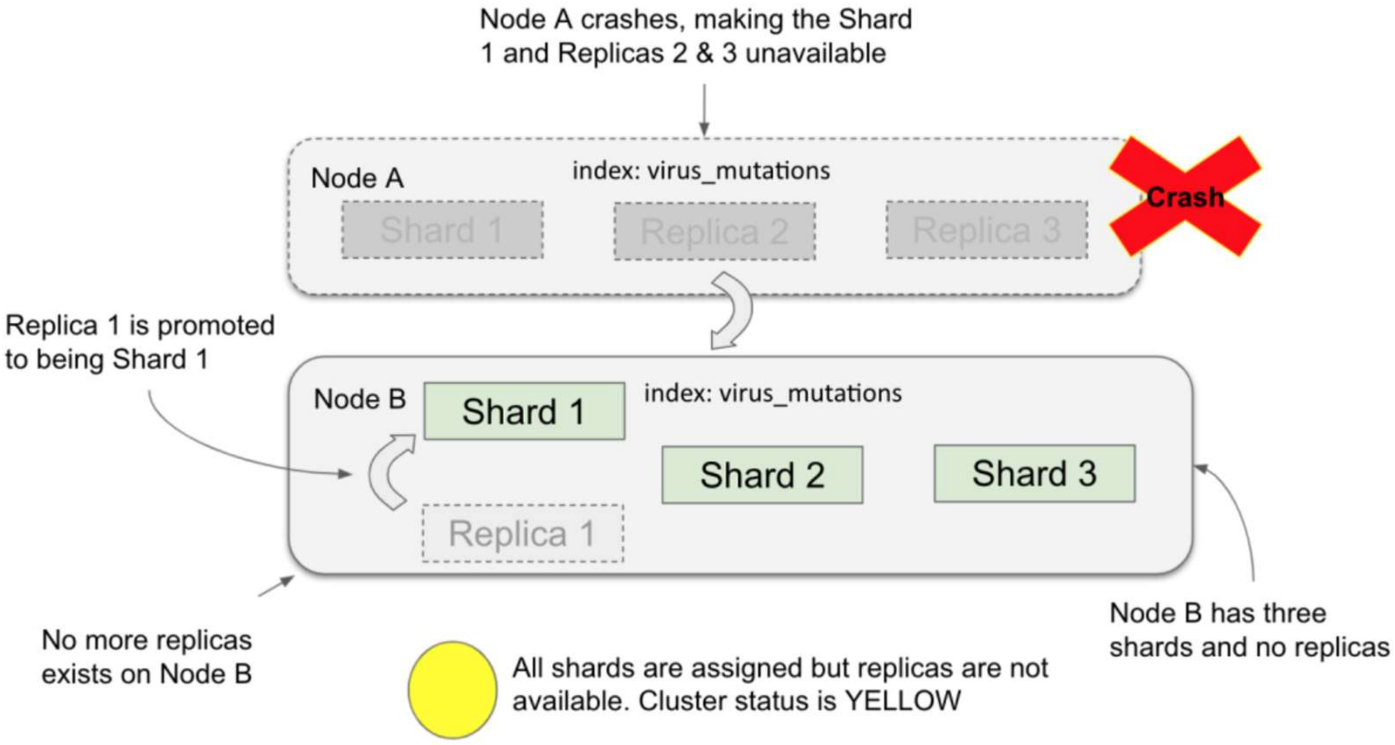


Health of shards using a traffic light signal board



RED	Not all shards are assigned and ready (cluster being prepared state)
YELLOW	Shards are assigned and ready but replicas aren't assigned and ready
GREEN	Shards and replicas are all assigned and ready

Replicas were lost when a node crashed



Node A crashes, making the Shard 1 and Replicas 2 & 3 unavailable

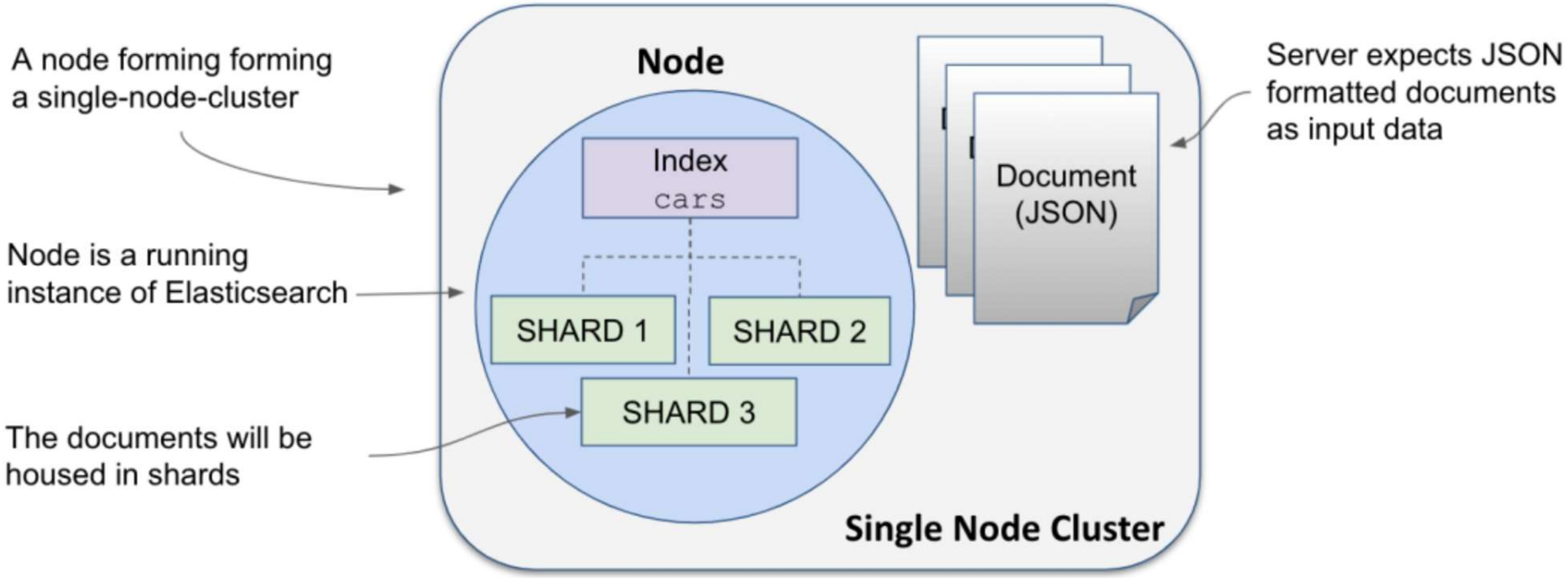
Replica 1 is promoted to being Shard 1

No more replicas exists on Node B

Node B has three shards and no replicas

All shards are assigned but replicas are not available. Cluster status is YELLOW

A single node Elasticsearch cluster



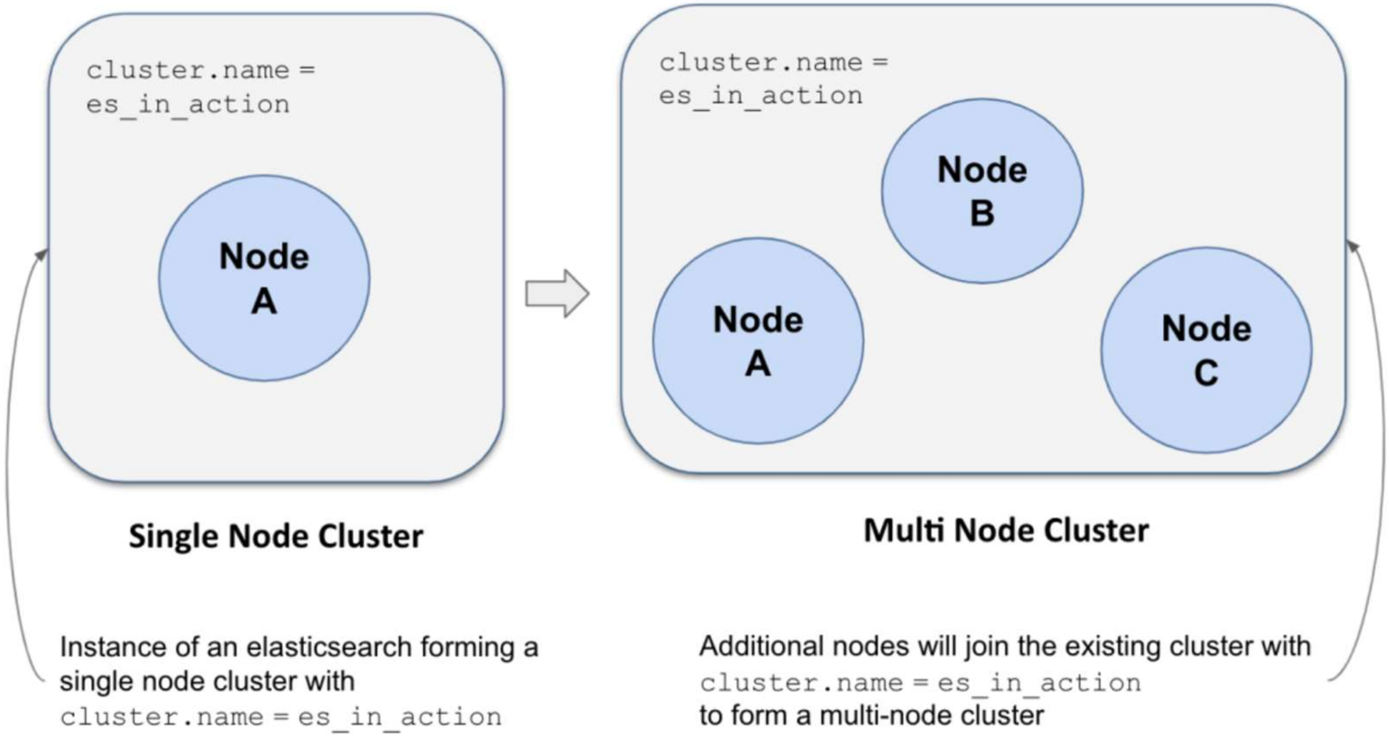
A node forming forming a single-node-cluster

Node is a running instance of Elasticsearch

The documents will be housed in shards

Server expects JSON formatted documents as input data

From a single node to a multiple node



`cluster.name = es_in_action`

Node A

Single Node Cluster

`cluster.name = es_in_action`

Node B

Node A

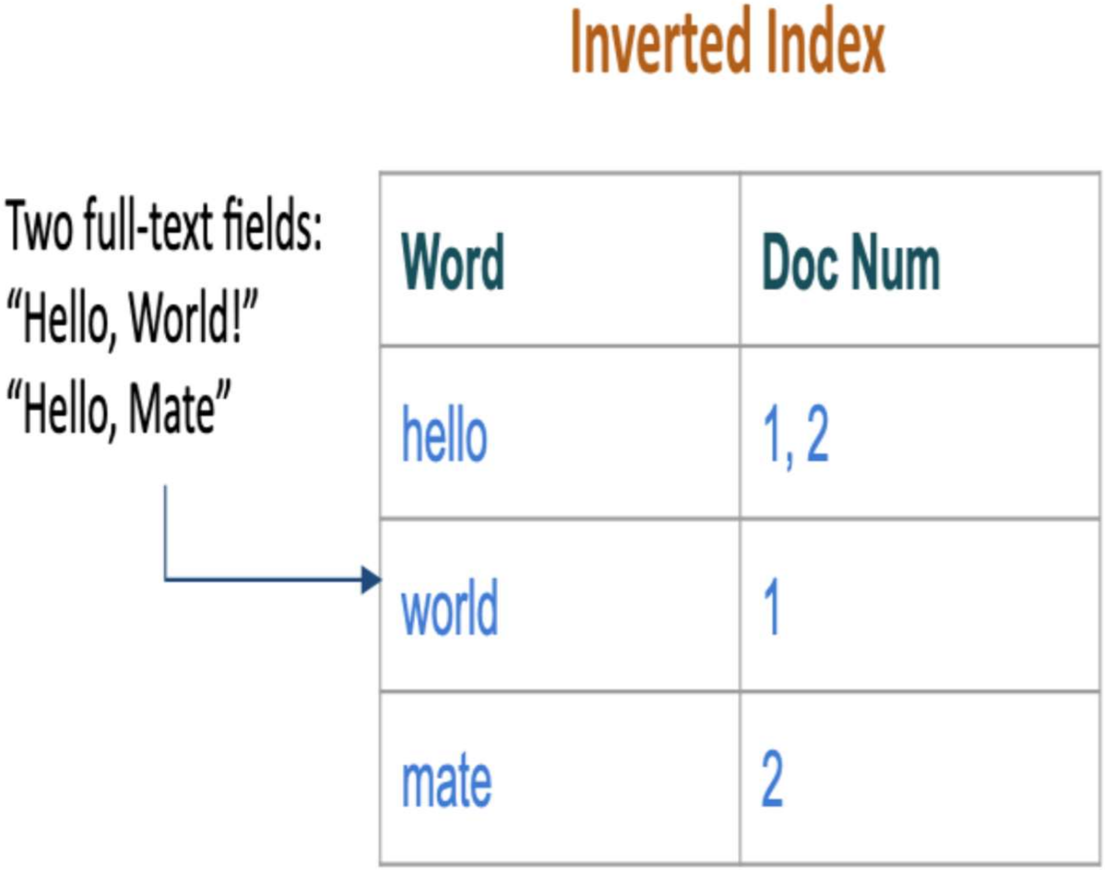
Node C

Multi Node Cluster

Instance of an elasticsearch forming a single node cluster with `cluster.name = es_in_action`

Additional nodes will join the existing cluster with `cluster.name = es_in_action` to form a multi-node cluster

An inverted index data structure



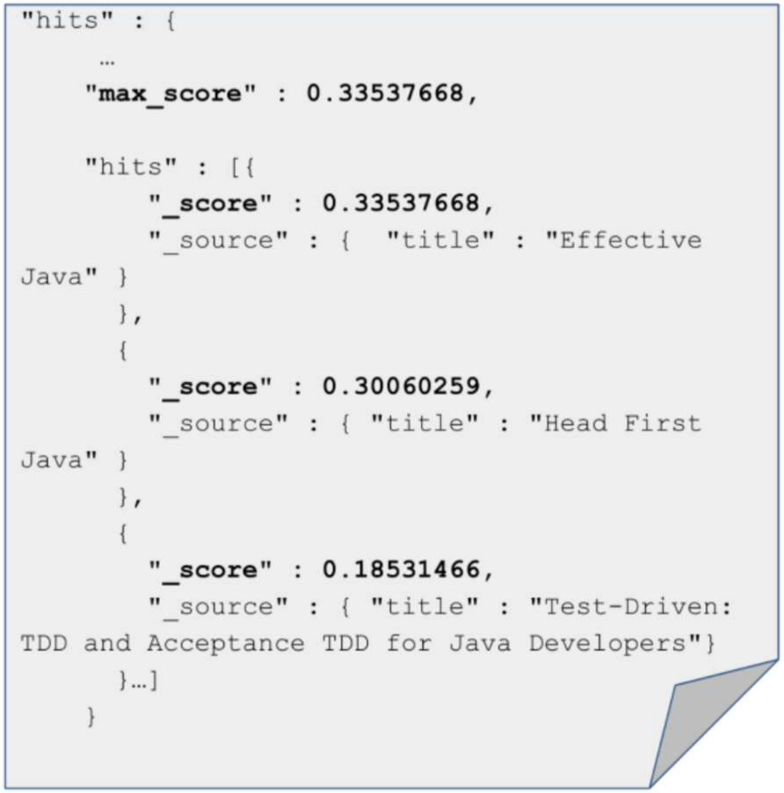
Inverted Index

Two full-text fields:
"Hello, World!"
"Hello, Mate"

Word	Doc Num
hello	1, 2
world	1
mate	2

Relevant results for Java in a title search

```
GET books/_search
{
  "_source": "title",
  "query": {
    "match": {
      "title": "Java"
    }
  }
}
```



```
"hits" : {
     …
    "max_score" : 0.33537668,

    "hits" : [{
        "_score" : 0.33537668,
        "_source" : {  "title" : "Effective
Java" }
      },
      {
        "_score" : 0.30060259,
        "_source" : { "title" : "Head First
Java" }
      },
      {
        "_score" : 0.18531466,
        "_source" : { "title" : "Test-Driven:
TDD and Acceptance TDD for Java Developers"}
      }…]
    }
```