COMP810 Data Warehousing and Big Data

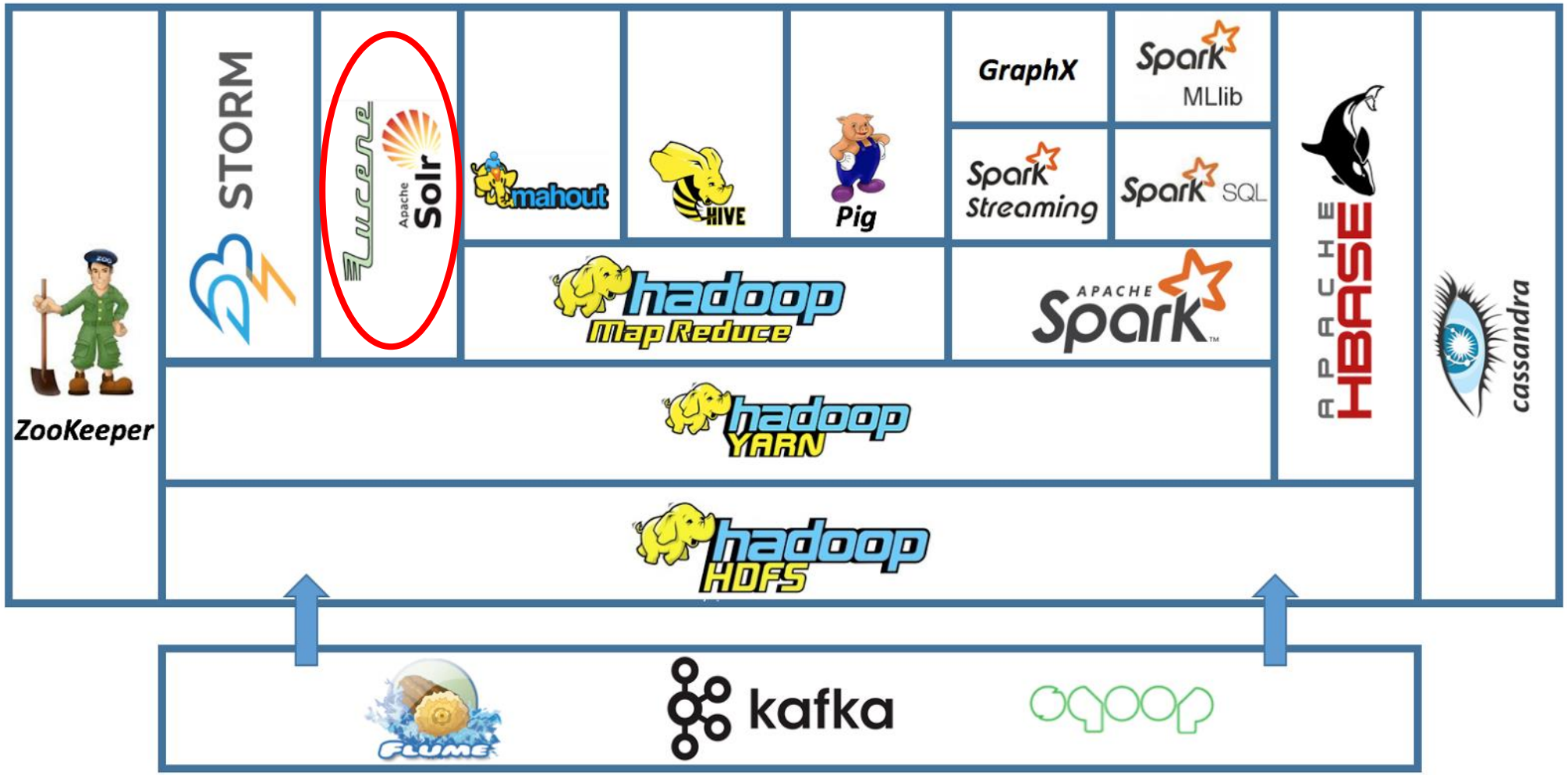
Search and Analyse Big Data with Elasticsearch (I)

Dr Weihua Li

Outline

- Big Data and Search Engine
- Elasticsearch and Kibana
- Elasticsearch Backend Components
- Fundamental Elasticsearch API
- Mapping and Data Type

Apache Hadoop Ecosystem

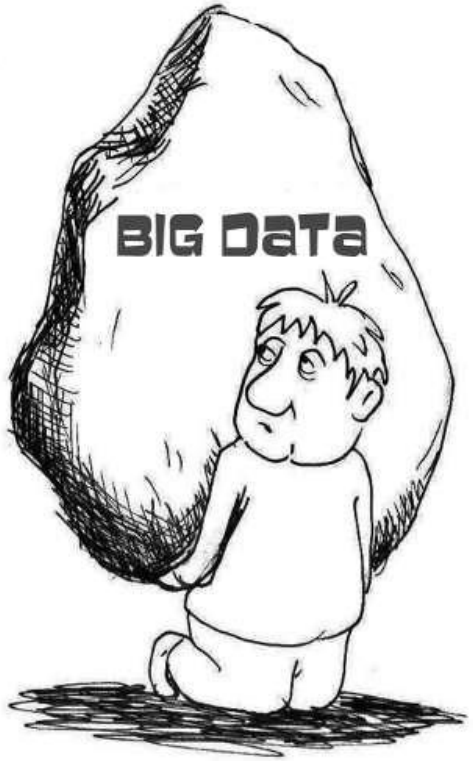


Why Another Set of Tools for Big Data

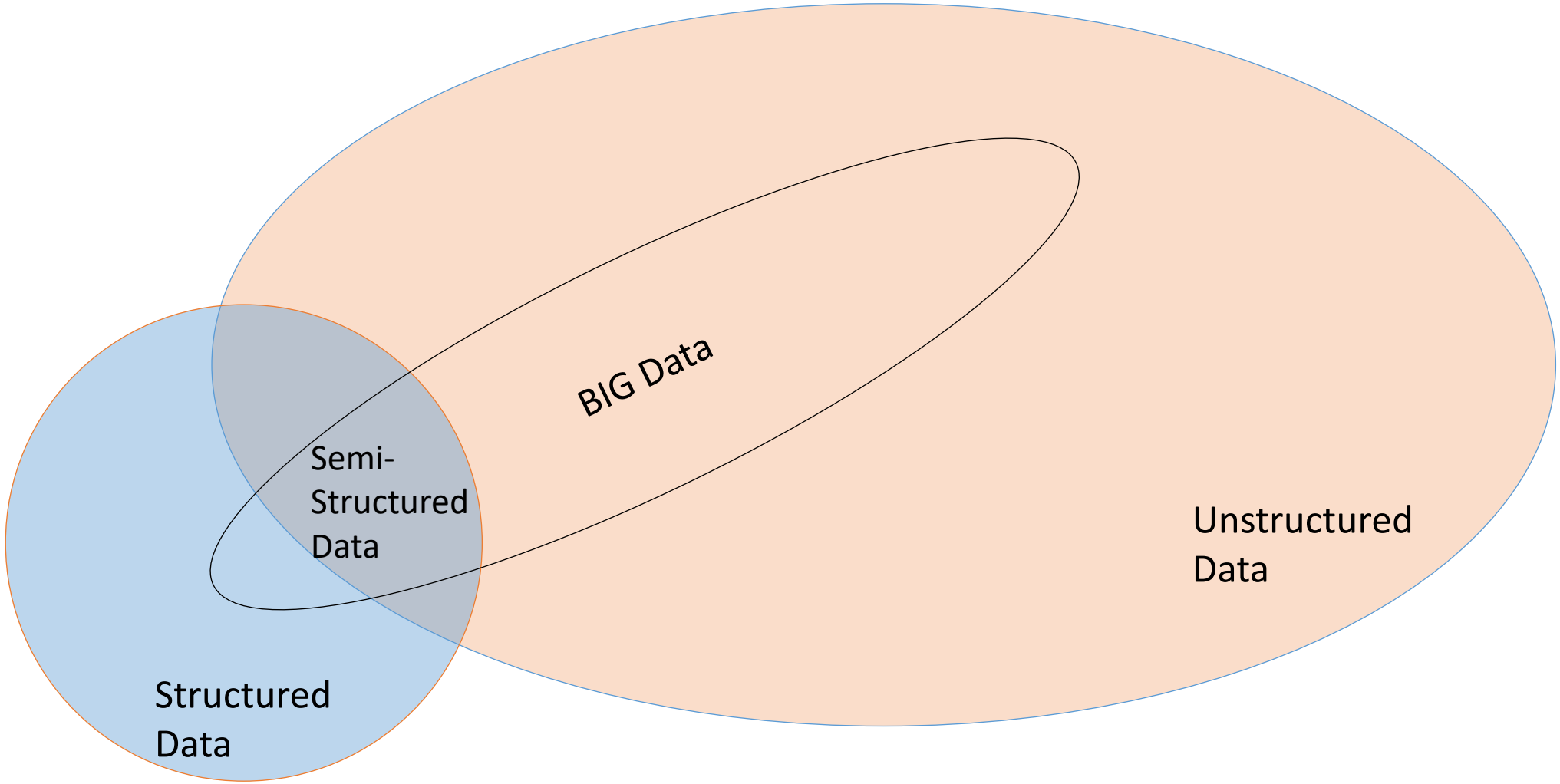
- **Limitations of Traditional RDBMS**
  - Schema on write
  - High cost of storage
  - **Weak support for unstructured data**

Poor choice

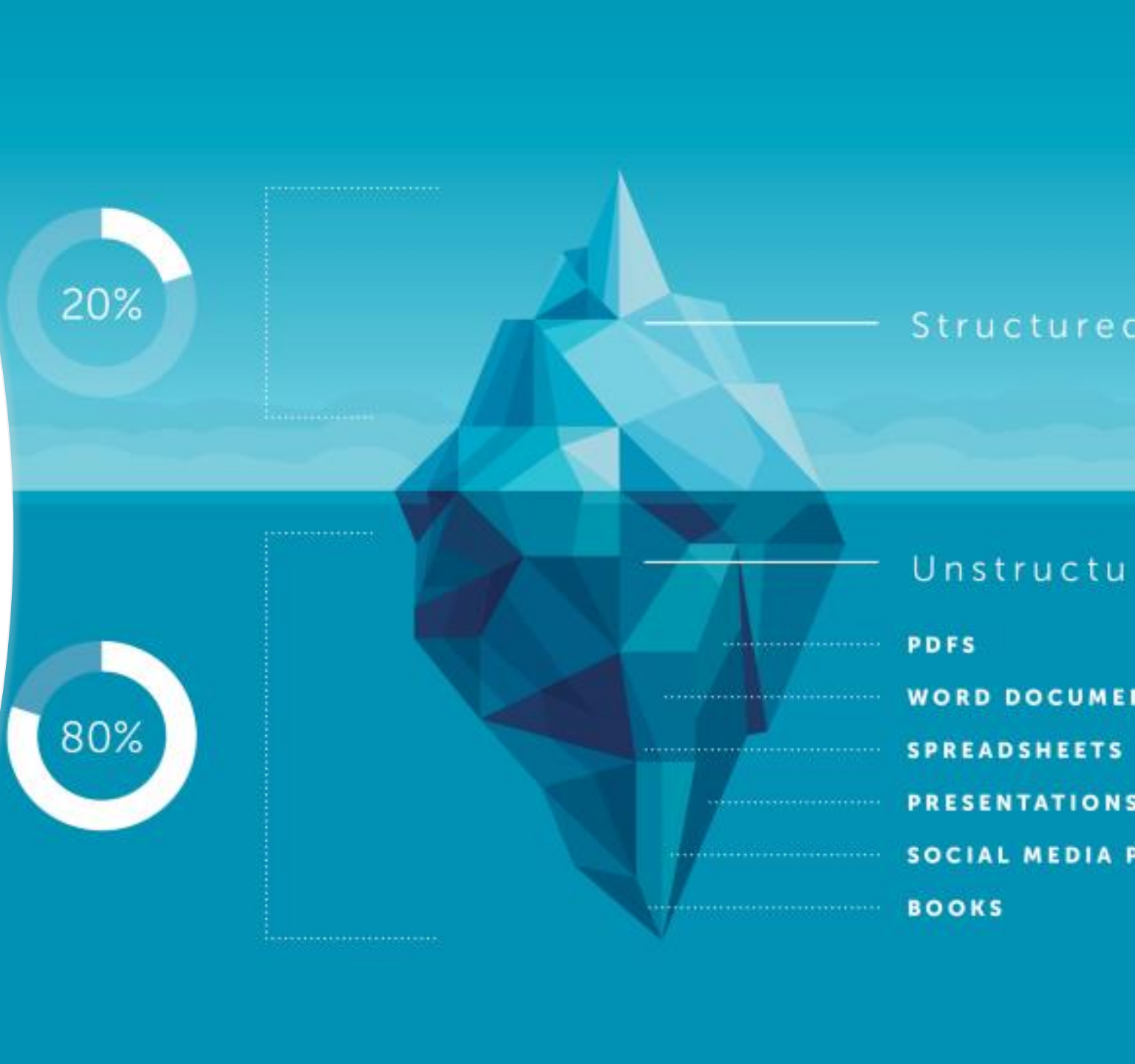
High volume and variety



Unstructured and Structured BIG Data



How to analyse the semi-structured and unstructured Big Data?



What is Elasticsearch

- Elasticsearch is a highly scalable open-source full-text search and analytics engine based on the Apache Lucene library.
- It allows to store, search, and analyse big volumes of data quickly and in near real time.
- It's the most popular search engine and has been available since 2010
- A **distributed**, **real-time**, **document store** / **search engine**

It easily scales to hundreds of servers and TBs of Data

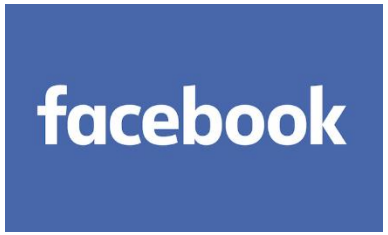
Data is searchable as it is added

Schema-free JSON documents

Best full text search capabilities

Customers of Elasticsearch

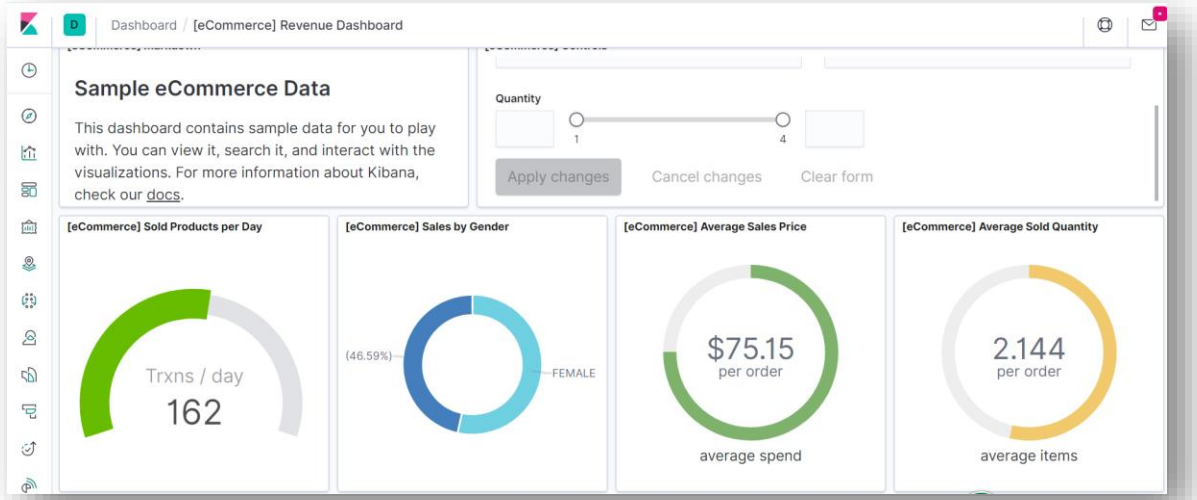
- One of the most popular enterprise search engines
- Well known and widely adopted by customers across industries



Kibana

- Kibana is an open source frontend application that sits on top of the Elastic Stack, providing search and data visualization capabilities for data indexed in Elasticsearch.

- Kibana also acts as the user interface (UI) for monitoring, managing, and securing an Elastic Stack cluster.



Elasticsearch Backend Components

Node

Cluster

Document

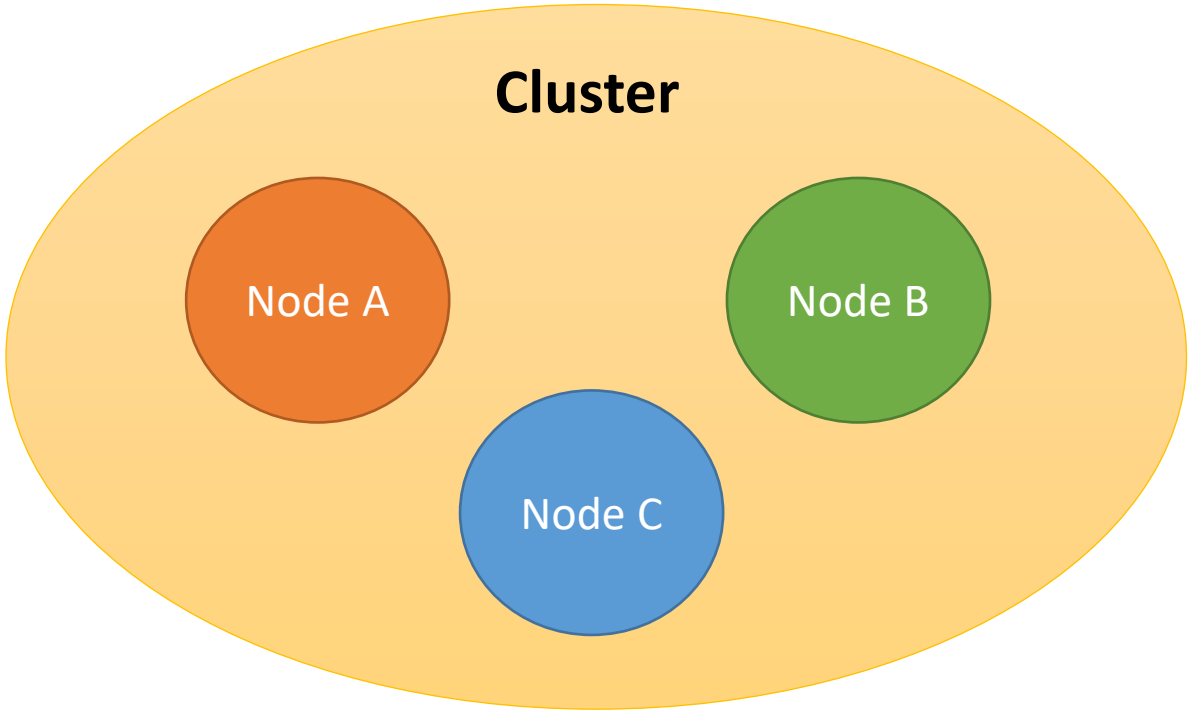
Index

Shard and Replicas

Node and Cluster

- A node is a single server that is part of a cluster, stores our data, and participates in the cluster's indexing and search capabilities.

- A cluster is a collection of one or more nodes that together holds your entire data and provides federated indexing and search capabilities.
- There can be a number of nodes within the same cluster.



Document

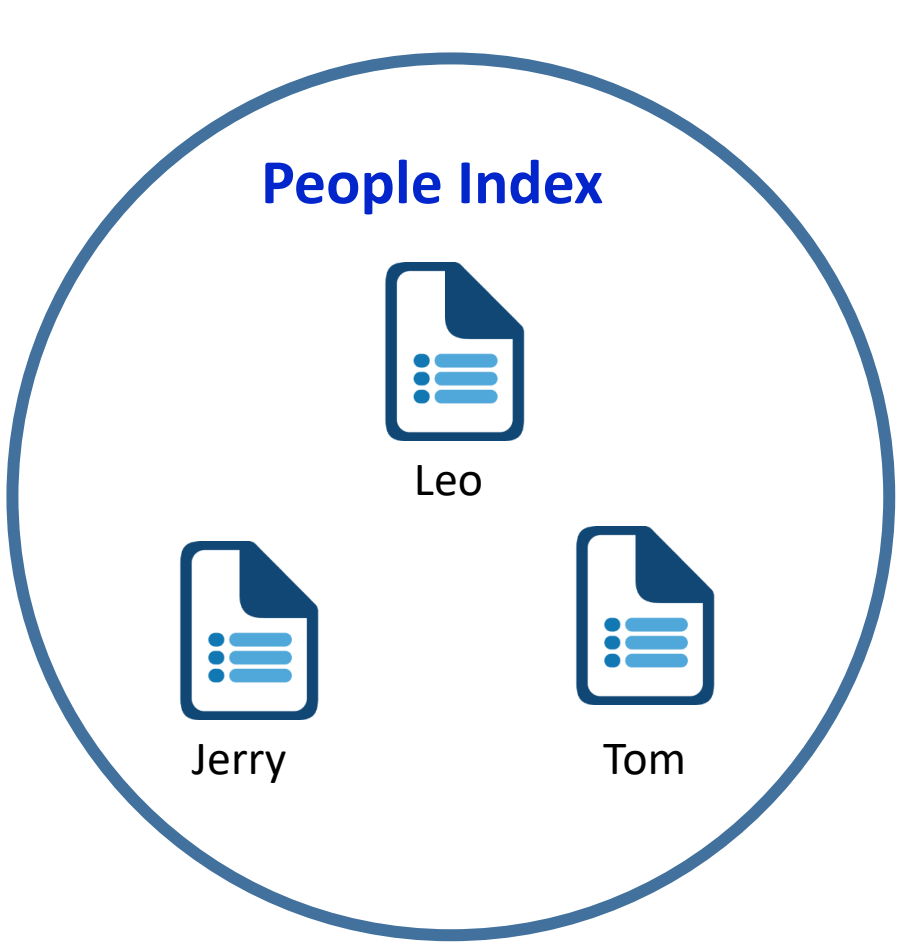
```
{
   “name”: “Leo Li”,
   “country”: “New Zealand”
}
```

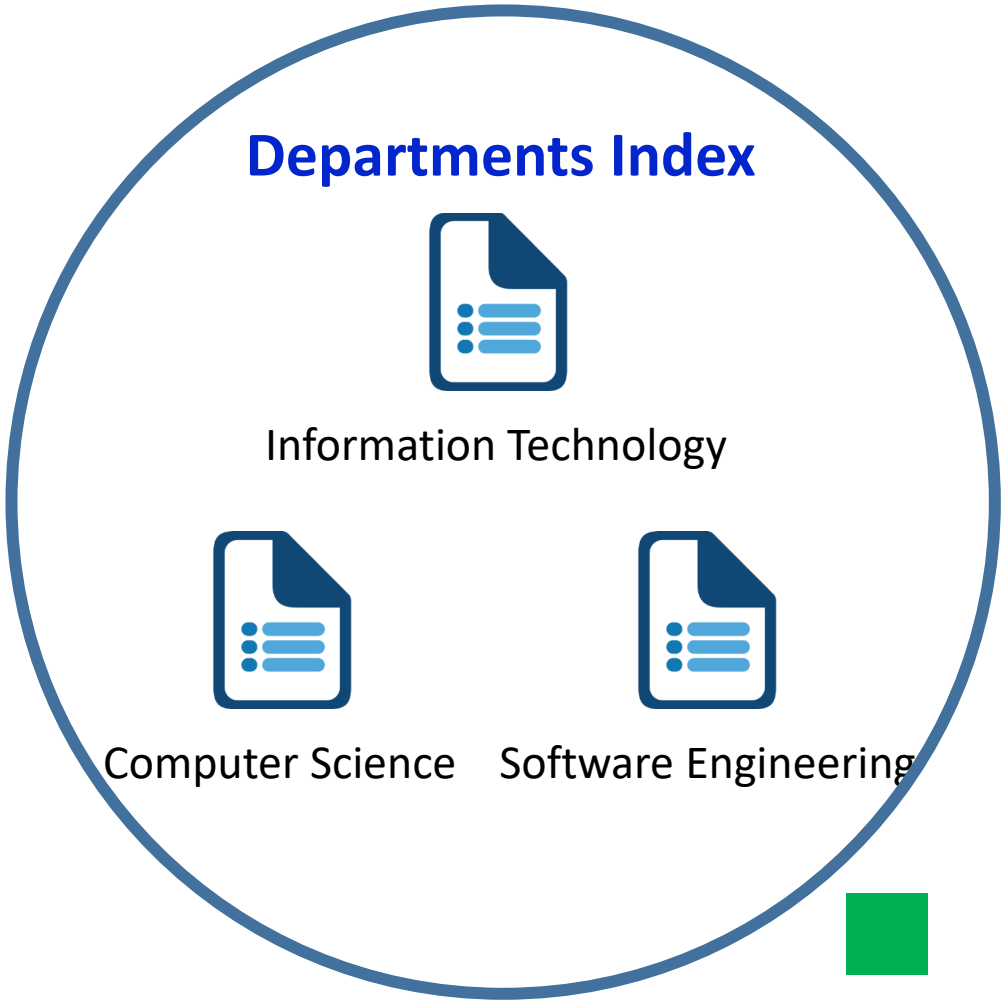
Is stored as

```
{
   “_index”: “people”,
   “_type”: “_doc”,
   “_id”: “123”,
   “_version”: 1,
   “_seq_no”: 0,
   “_primary_term”: 1,
   “_source”: {
      “name”: “Leo Li”,
      “country”: “New Zealand”
    }
}
```

Index

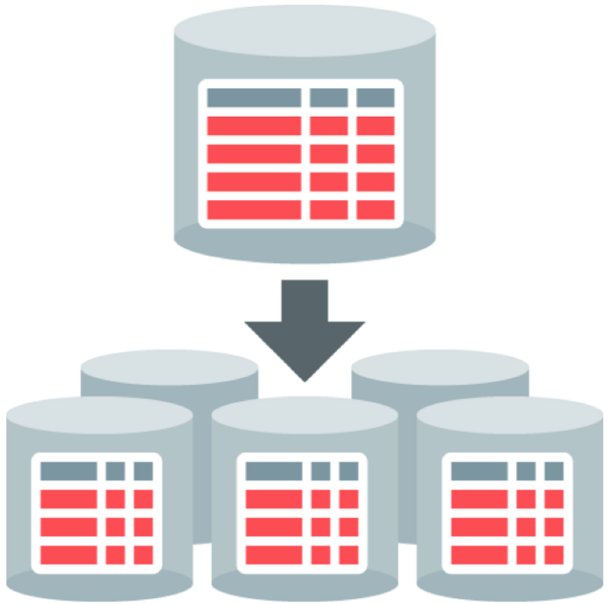
- The index is a collection of documents that have similar characteristics.





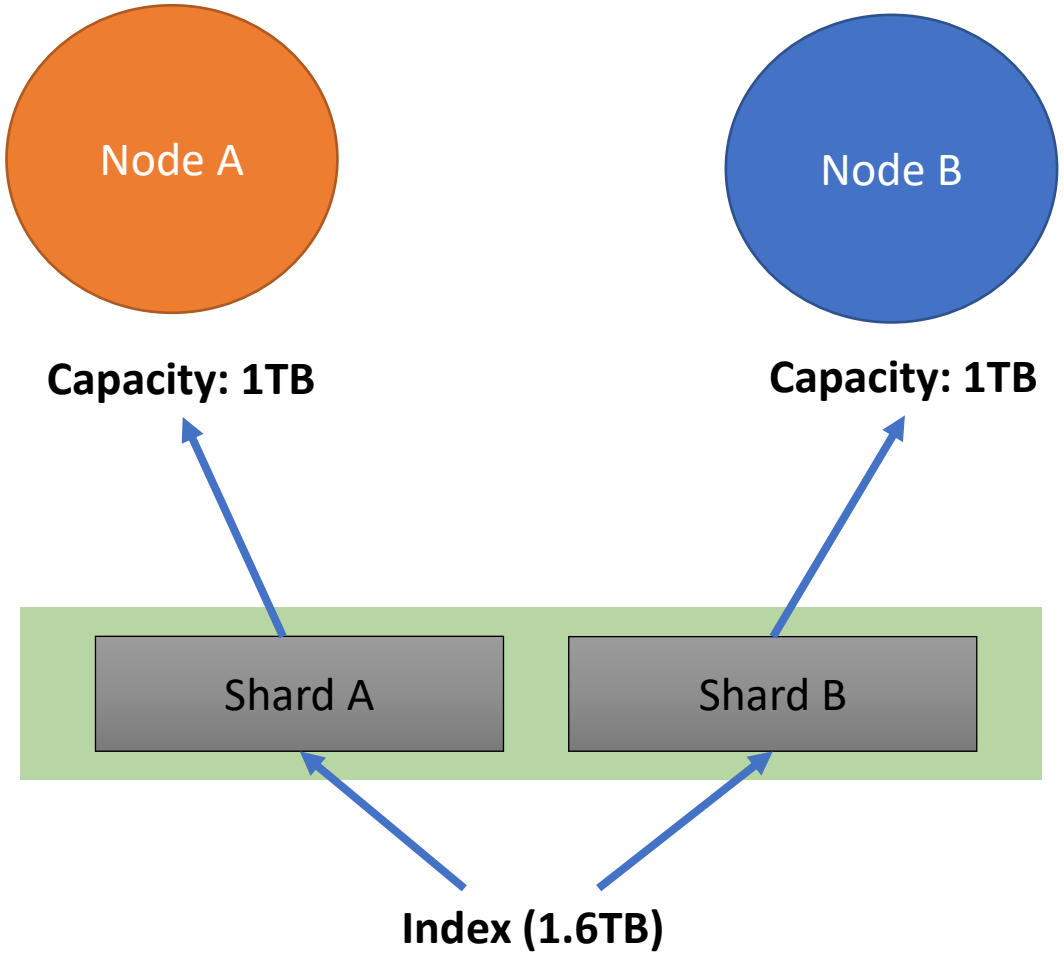
Sharding and Scalability

- Problem
  - If you have 2TB data, but you have a single node with 1TB of space.
- Solution
  - Add an additional node with sufficient capacity.
  - Store data on both nodes, meaning that the cluster now has enough storage capacity.



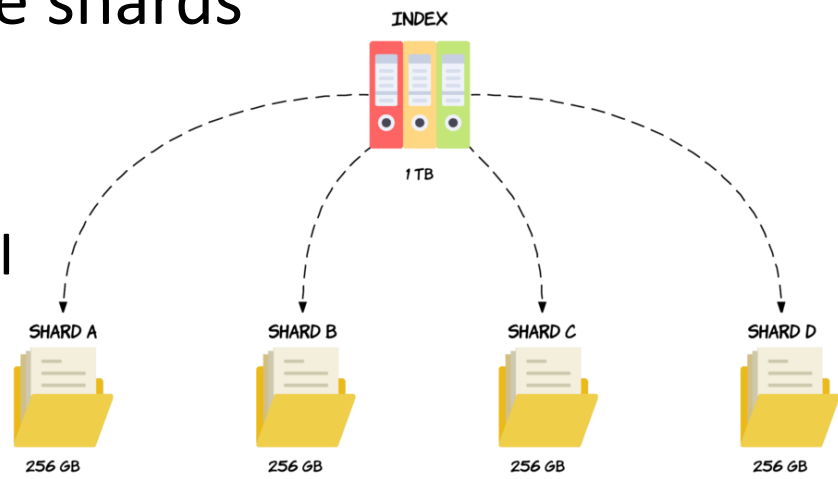
Introduction to Sharding

- Sharding is a way to divide indices into smaller pieces
- Each piece is referred to as a shard
- Sharding is achieved at index level, rather than at node or cluster level.
- The main objective is to horizontally scale the data volume



Sharding

- An index contains a single shard by default (since Elasticsearch 7), and the number of shard can be changed.
- Indices in old versions were created with five shards
  - This also led to over-sharding problem
- Shard control
  - Increase the number of shards with the split API
  - Reduce the number of shards with shrink API
- Optimal shards?
  - No formula yielding an optimal number of shards
  - Add more shards if you anticipate millions of documents

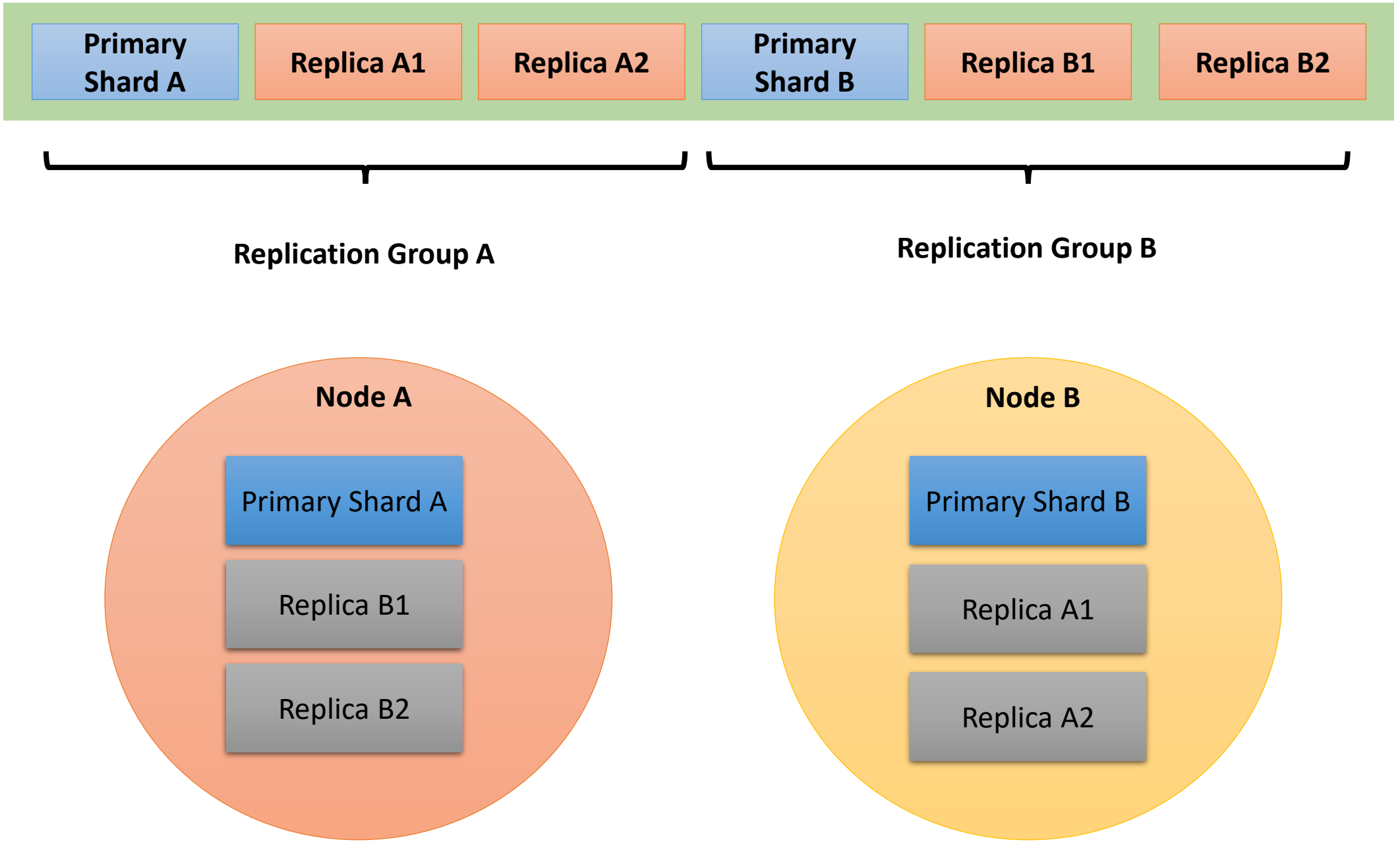


Replication

- What happens is a node's hard drive fails?
- Hardware can fail at any time.
- Fault tolerance using replication.
- Elasticsearch support replication of shards, which is enabled by default.
- With many databases, setting up replication can be complicated
- Replication is extremely easy with Elasticsearch

Theories of Replication

- Replication is configured at the index level
- Replication works by creating copies of shards, referred to as replica shards
- A shard that has been replicated, is called a primary shard
- A primary shard and its replica shards are referred to as replication group
- Replica shards are a complete copy of a shard. It can serve search requests, exactly like the primary shard


Primary Shard A
Replica A1
Replica A2
Primary Shard B
Replica B1
Replica B2
Replication Group A
Replication Group B
Node A
Primary Shard A
Replica B1
Replica B2
Node B
Primary Shard B
Replica A1
Replica A2

Choosing Replica Shards Number

- The idea number of replica shards depends on use cases
  - Is data stored elsewhere, e.g., RDBMS?
  - Is it fine if data is unavailable when being restored?
- For mission critical systems, no downtime.
- Replicate shards once if data loss is not a disaster
- Replicate at least twice for mission critical systems

Define Number of Shards and Replicas

```
PUT /my-index-000001
{
  "settings": {
    "index": {
      "number_of_shards": 3,   1
      "number_of_replicas": 2  2
    }
  }
}
```

Default for number_of_shards is 1

Default for number_of_replicas is 1
(one replica for each primary shard)

Use Elasticsearch API

GET / _cat / indices ? v

Method API Command Parameter

200 - OK 54 ms

```
health status index                           uuid                   pri rep docs.count docs.deleted store.size pri.store.size
yellow open   news                            kZBZ4FreSWu_PZt49rn7fw   1   1       1000            0      2.4mb          2.4mb
yellow open   persondata                      sGRLJM_DRhae9ZpBevt2Nw   1   1          5            0      9.4kb          9.4kb
yellow open   student                         9GlOtJj6QFSvrtntjcPwtw   1   1          1            3      6.2kb          6.2kb
yellow open   bulk_index_test                 SUKn_W1RT1GOyfh-zw5aXQ   1   1          4            0      5.4kb          5.4kb
green  open   .apm-agent-configuration         HBmcVQ94SRidBPIEhq3HZw   1   0          0            0       208b           208b
yellow open   split2-bulk_index_test          OVdGbFhYTmeOmXTjp7biAw   3   1          4            0     15.2kb         15.2kb
green  open   .kibana_1                       C5M-fT1QTKSIPmgTdGhvyA   1   0        321            5      1.1mb          1.1mb
green  open   kibana_sample_data_flights      z5IfwrCqSnWu81uowD09tQ   1   0      13059            0      6.3mb          6.3mb
yellow open   sales                           RXRQQdcnRdaToA5KecP8rw   1   1          1            0      4.1kb          4.1kb
yellow open   nested_aggregation              V_whGggyTIWW_yanACia4w   1   1          8            0      4.9kb          4.9kb
green  open   .apm-custom-link                 4mZvmQxNR46GP7phYiEImg   1   0          0            0       208b           208b
green  open   kibana_sample_data_ecommerce    J1D9D174RciXRB0qMDAmWA   1   0       4675            0      4.5mb          4.5mb
green  open   .kibana_task_manager_1          7lzrCKs2SlGvz0w3eEqdcg   1   0          5            0       76kb           76kb
yellow open   split-bulk_index_test           GiobGOdrRnaDWfqgLFYi6Q   2   1          4            0      5.5kb          5.5kb
green  open   kibana_sample_data_logs         b2WWzZKbR1yW4umLjfYxYg   1   0      14074            0     11.6mb         11.6mb
yellow open   bulk_test                       EvNluUNQT1qjCntcfGRNww   1   1          3            6     11.6kb         11.6kb
```

Use Elasticsearch API - Basics

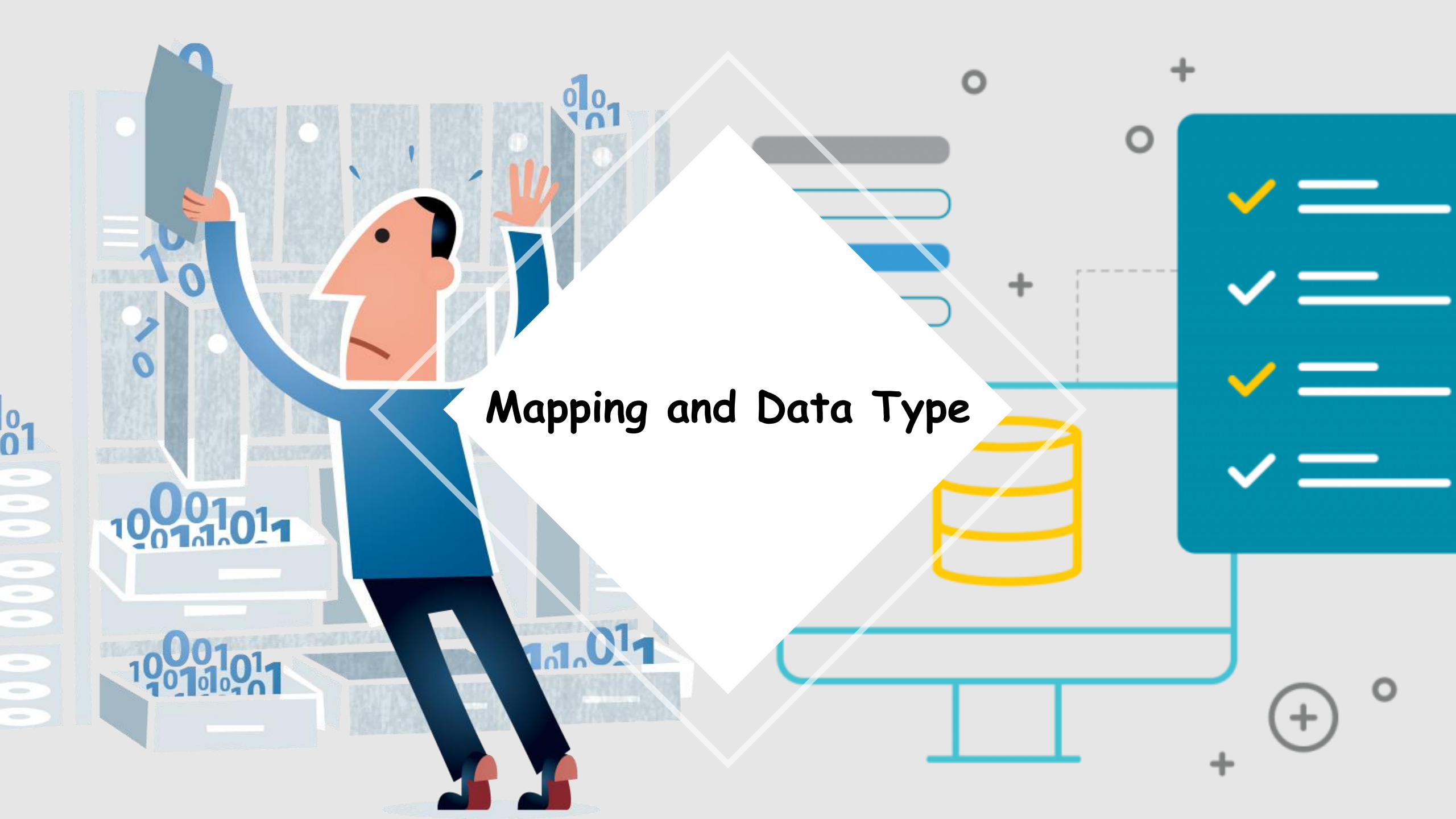
- See the cluster status
  - GET /_cluster/health
  - GET /_cat/nodes?v
  - GET /_cat/indices?v
- Create an index named sales
  - PUT /sales
- Create an document
  - PUT /sales/_doc/123

```
{
   "orderid": "123",
  "orderAmount" : "500"
}
```

- Update a document

```
POST sales/_doc/123/
{
 "name": "Leo",
 "skills": "Data, AI and Programming"
}
```

- Search a document based on id
  - GET /sales/_doc/123
- Delete a document
  - DELETE /sales/_doc/123
- Delete index
  - DELETE /sales/


Mapping and Data Type

Mapping

- Defines the structure of documents (e.g., fields and their **data types**)
- Similar to a table's schema in a relational database

```
CREATE TABLE employees (
  id INT AUTO_INCREMENT PRIMARY KEY,
  first_name VARCHAR(255) NOT NULL,
  last_name VARCHAR(255) NOT NULL,
  dob DATE,
  description TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
PUT /employees
{
  "mappings": {
    "properties": {
      "id": { "type": "integer" },
      "first_name": { "type": "text" },
      "last_name": { "type": "text" },
      "dob": { "type": "date" },
      "description": { "type": "text" },
      "created_at": { "type": "date" }
    }
  }
}
```

MySQL

Elasticsearch

Mapping (Cont.)

- Dynamic Mapping
  - The automatic detection and addition of new fields is called dynamic mapping. Elasticsearch generates field mappings automatically.
  - When a document is indexed, the index, type, and fields will be generated automatically.
- Explicit Mapping
  - We define field mapping ourselves

```
PUT /my-index-000001
{
  "mappings": {
    "properties": {
      "age":    { "type": "integer" },
      "email":  { "type": "keyword"  },
      "name":   { "type": "text"  }
    }
  }
}
```

Mapping (Cont.)

- Create Mapping with dot Notation
- Retrieve Mapping
  - Useful when you use dynamic mapping

```
PUT /reviews
{
  "mappings": {
    "properties": {
      "rating": {"type": "float"},
      "content": {"type": "text"},
      "product_id": {"type": "integer"},
      "author": {
        "properties": {
          "first_name": {"type": "text"},
          "last_name": {"type": "text"},
          "email": {"type": "keyword"}
        }
      }
    }
  }
}
```

```
PUT /reviews_dot_notation
{
  "mappings": {
    "properties": {
      "rating": {"type": "float"},
      "content": {"type": "text"},
      "product_id": {"type": "integer"},
      "author.first_name": {"type": "text"},
      "author.last_name": {"type": "text"},
      "author.email": {"type": "keyword"}
    }
  }
}
```

GET /reviews_dot_notation/_mapping

```
{
  "reviews_dot_notation" : {
    "mappings" : {
      "properties" : {
        "author" : {
          "properties" : {
            "email" : {
              "type" : "keyword"
            },
            "first_name" : {
              "type" : "text"
            },
            "last_name" : {
              "type" : "text"
            }
          }
        },
        "content" : {
          "type" : "text"
        },
        "product_id" : {
          "type" : "integer"
        },
        "rating" : {
          "type" : "float"
        }
      }
    }
  }
}
```

Data Types – JSON Object

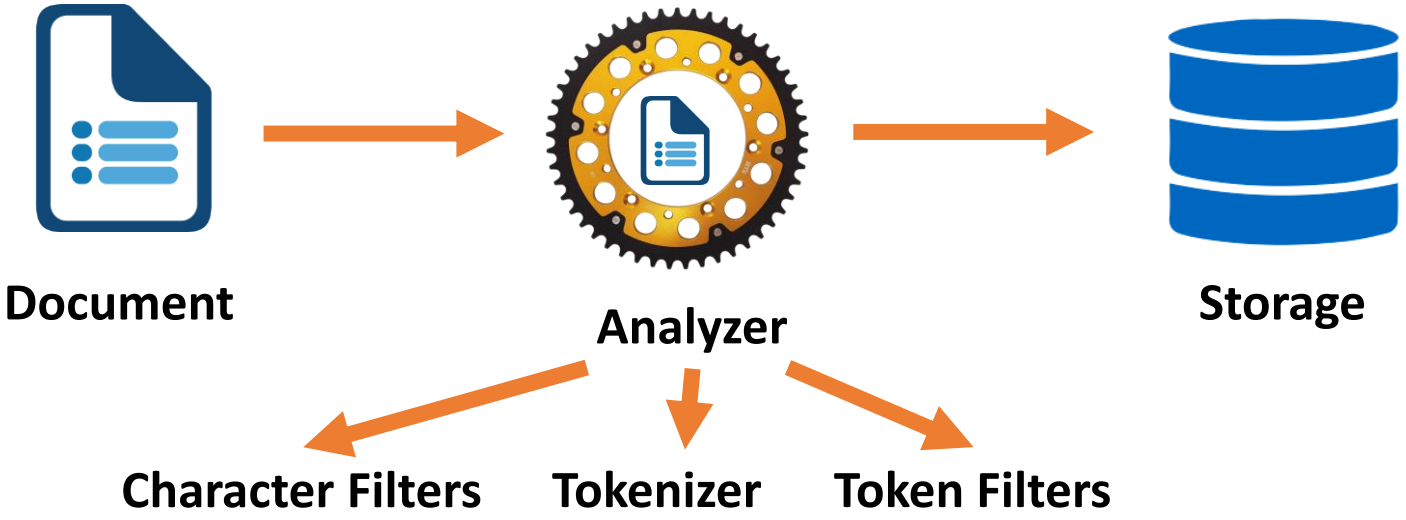
- Documents in Elasticsearch are represented in JSON format.
- JSON documents are hierarchical in nature: the document may contain inner objects
- Internally, the document is indexed as a simple, flat list of key-value pairs

```
PUT my_index/_doc/1
{ (1)
  "region": "US",
  "manager": { (2)
    "age":     30,
    "name": { (3)
      "first": "John",
      "last":  "Smith"
    }
  }
}
```

```
{
  "region":             "US",
  "manager.age":        30,
  "manager.name.first": "John",
  "manager.name.last":  "Smith"
}
```

Data Types – Text Data Type

- Text is a field to index full-text value (for full-text search)
- E.g., the body of an email, the description of a product.
- Processed by analzyer and converted into multiple tokens




```
PUT my_index
{
  "mappings": {
    "_doc": {
      "properties": {
        "full_name": {
          "type":  "text"
        }
      }
    }
  }
}
```

Data Types – Text Data Type

- Example:
  - Use _analyze API with standard analyzer

```
POST _analyze
{
  "text": "I found COMP810 very interesting ^_^",
  "analyzer": "standard"
}
```


```
{
  "tokens" : [
    {
      "token" : "i",
      "start_offset" : 0,
      "end_offset" : 1,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "found",
      "start_offset" : 2,
      "end_offset" : 7,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "comp810",
      "start_offset" : 8,
      "end_offset" : 15,
      "type" : "<ALPHANUM>",
      "position" : 2
    },
    {
      "token" : "very",
      "start_offset" : 16,
      "end_offset" : 20,
      "type" : "<ALPHANUM>",
      "position" : 3
    },
    {
      "token" : "interesting",
      "start_offset" : 21,
      "end_offset" : 32,
      "type" : "<ALPHANUM>",
      "position" : 4
    }
  ]
}
```

Data Types – Keyword Data Type

- Keyword is a field to index structured content such as email addresses, hostnames, zip codes or tags, which matches the exact values.
- Typically used for filtering, aggregation, and sorting
- E.g., searching for articles with a status of *PUBLISHED*
- NOT used for indexing full text content

```
POST _analyze
{
  "text": "I found COMP810 very interesting ^_^",
  "analyzer": "keyword"
}
```

```
{
  "tokens" : [
    {
      "token" : "I found COMP810 very interesting ^_^",
      "start_offset" : 0,
      "end_offset" : 36,
      "type" : "word",
      "position" : 0
    }
  ]
}
```

Understanding Arrays

- In Elasticsearch, there is NO array datatype!
- All values in the “array” must be of the same datatype, e.g., ["one", "two"]
- “Arrays” with a mixture of datatypes are not supported: [10, "some string"]

```
POST /product/_doc
{
  "tags": ["smartphone","desktops","computers"],
  "sales_person_ids": [101, 220, 333]
}
```

Stored as “long” and “text” fields

Understanding Arrays (Cont.)

- How does it work?
- **Concatenation!**

```
POST /_analyze
{
  "text": ["smartphone","desktops","computers"],
  "analyzer": "standard"
}
```

```
{
  "tokens" : [
    {
      "token" : "smartphone",
      "start_offset" : 0,
      "end_offset" : 10,
      "type" : "<ALPHANUM>",
      "position" : 0
    },
    {
      "token" : "desktops",
      "start_offset" : 11,
      "end_offset" : 19,
      "type" : "<ALPHANUM>",
      "position" : 1
    },
    {
      "token" : "computers",
      "start_offset" : 20,
      "end_offset" : 29,
      "type" : "<ALPHANUM>",
      "position" : 2
    }
  ]
```

These three strings are actually concatenated!

Coercion

- Why? Data is NOT always clean.
- Coercion attempts to clean up dirty values to fit the data type of a field. For example:
  - Strings will be coerced to numbers.
  - Floating points will be truncated for integer values.

```
PUT my-index-000001/_doc/1
{
  "number_one": "10" (1)
}

PUT my-index-000001/_doc/2
{
  "number_two": "10" (2)
}
```

(1) The number_one field will contain the integer 10.

(2) This document will be rejected because coercion is disabled.

```
PUT my-index-000001
{
  "mappings": {
    "properties": {
      "number_one": {
        "type": "integer"
      },
      "number_two": {
        "type": "integer",
        "coerce": false
      }
    }
  }
}
```

Coercion (Cont.)

Mapping

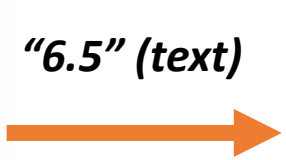
```
{
  "coercion_test" : {
    "mappings" : {
      "properties" : {
        "price" : {
          "type" : "float"
        }
      }
    }
  }
}
```

```
PUT /coercion_test/_doc/1
{
  "price": 6.5
}

PUT /coercion_test/_doc/2
{
  "price": "6.5"
}

PUT /coercion_test/_doc/3
{
  "price": "6.5m"
}
```

"6.5" (text)



Coercion

Inspect mapping and coerce into field data type if possible

6.5 float

Storage

Coercion Failed!

References

- Bo Andersen, *Complete Guide to Elasticsearch*
- Giovanni Pagano Dritto,
  an Overview on Elasticsearch and it usage, 28 Mar 2019
- Elasticsearch Reference,
  https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html