

# Introduction to Data Engineering 9

Paul Blondel

UTSEUS, Shanghai

May 25th, 2021

*'The cost of managing traditional databases is high. Mistakes made during routine maintenance are responsible for 80 percent of application downtime.'*

Dev Ittycheria, President and CEO of MongoDB

- 1 Recall
- 2 NoSQL
- 3 MongoDB: Document based NoSQL
- 4 ElasticSearch: A different kind of Document-based NoSQL DB

1 Recall

2 NoSQL

3 MongoDB: Document based NoSQL

4 ElasticSearch: A different kind of Document-based NoSQL DB

## The main aspects of Relational Databases:

- Very structured
  - ▶ **Tables**
  - ▶ **Typed attributes**
  - ▶ Integrity **constraints** and **relations**
    - ★ Uniqueness of **primary keys** (integrity)
    - ★ Existence of **foreign keys** (relations)
    - ★ Etc.
- Can be **formalized with diagrams**
  - ▶ UML, MERISE, etc
- Use **SQL** language for DB operations

idetudiant	nom	prenom	ville
1	Perrier	Jean	Rennes
2	Martin	Aline	Mulhouse

Figure: SQL example: "SELECT nom,prenom FROM etudiant;"

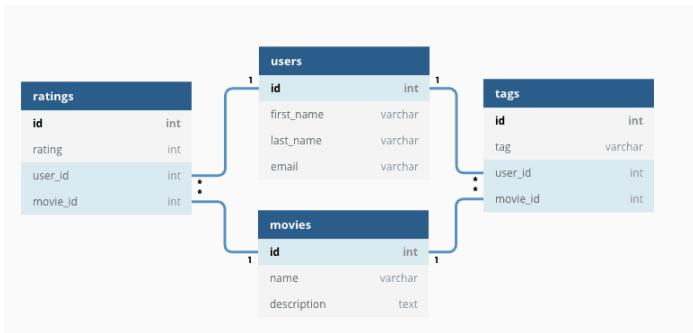


Figure: Example of a relational database diagram

- There are four tables
- Each table has a set of typed fields
- Tables have relations

Relation Databases have pros, and cons...

- Relation Databases are **difficult to distribute on several servers**
- They face a **scaling problem**
  - ▶ A bigger database = a more powerful server = limitations
- They have a **rigid definition** (schema):
  - ▶ Changing the schema is hard (require "migrations" = dangerous)
- They are **slow** (because of all the relations to consider)

A complex Relational Database is hard to maintain...

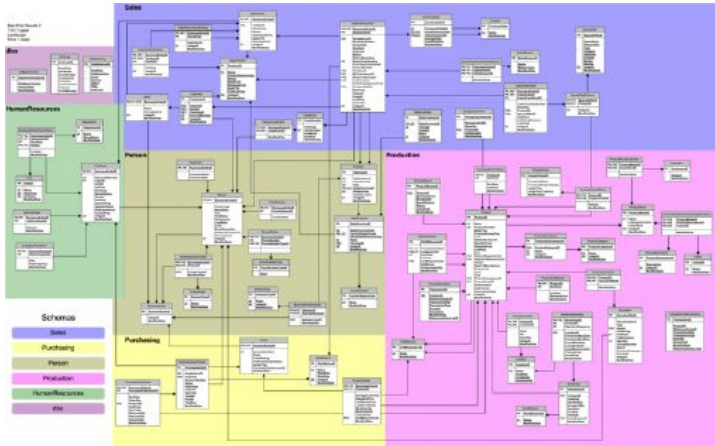


Figure: Example of a complex Relational Database



All these cons make it extremely difficult to deal with Big Data's three Vs...

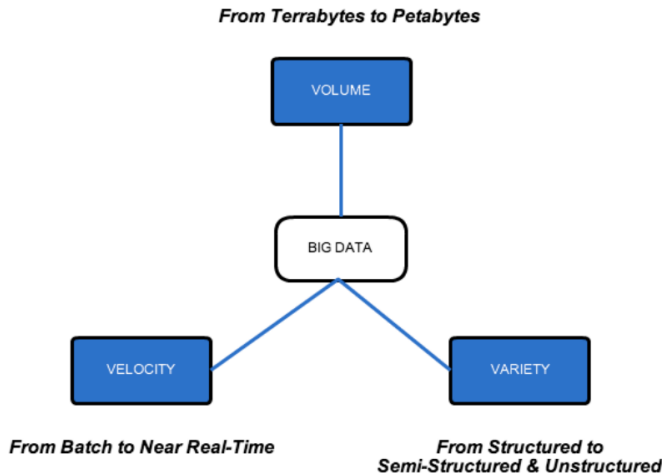


Figure: The three Vs.

Biggest issue of Relational DB with Big Data is: **scaling**

## Why?

Relational DB **cannot easily distribute the storage capacity on other nodes because of the complex relations and constraints** existing between the tables of a relational database

Indeed, we can only add storage capacity on the same server and also add more processing power (vertical scaling):



**Figure:** Vertical scaling of a server containing a Relational DB

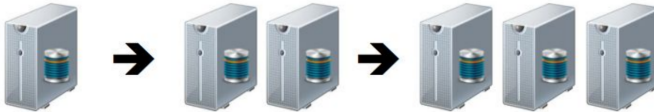
1 Recall

2 NoSQL

3 MongoDB: Document based NoSQL

4 ElasticSearch: A different kind of Document-based NoSQL DB

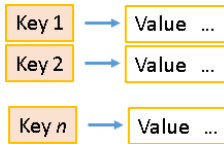
In the previous course, we saw that **the way to deal with Big Data is horizontal scaling** (adding nodes in a cluster):



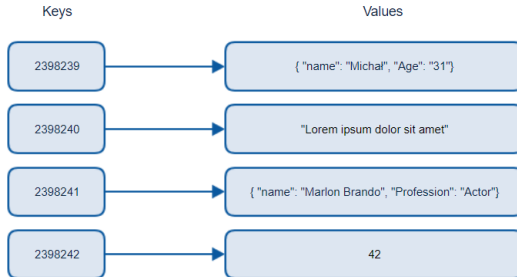
**Figure:** Horizontal scaling of a cluster: adding more nodes to increase capacity

- Storing data on a cluster of servers and **benefit from horizontal scaling is possible using NoSQL databases**
- NoSQL means « Not-Only SQL » and not No SQL (NOTE: some NoSQL databases partially understand SQL)
- **NoSQL databases are schemeless**
- Data can be stored with different scheme on the fly, **there are no fixed schemes**
- **Removing the constraints of Relational DB permits to distribute the DB on multiples server nodes**

In a NoSQL database, the information is stored as **key and value pairs**:



Example:



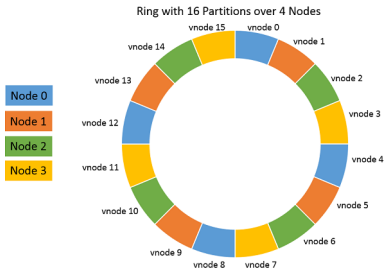
**Figure:** Contrary to Relational DB, all the information is stored in the "values field" and can be various (JSON, strings, numbers, etc)

A NoSQL database **can be easily distributed on different nodes of a cluster using of a hash function:**

It is easy to know in which node (or server) is located the data by computing the result of a hash function:

A hash function is a function **taking the key in input and giving a number in output which is the node id of the cluster:**

Example:  $\text{HASH}(\text{key}) \rightarrow [1 \ 15]$



- Example: we want the values of key 2398239,  $\text{HASH}(2398239)=12$ . So the values are in the virtual node 12 (a partition of Node 0)
- Here we have 16 virtual nodes,  $\text{HASH}$  can be just the modulo of 16 ( $\text{HASH}(X)=X \bmod 16$ )
- If we want to add a new nodes to our cluster: we just have to change the hash function ( $\text{HASH}(X) = X \bmod 20$ )
- Adding nodes to a NoSQL cluster is very easy... It scales horizontally easily.



Different types of NoSQL databases, for example:

- **Document DB:** MongoDB, CouchDB, ElasticSearch, etc.
- **Column DB:** Cassandra, etc.
- **Key-value purely based:** Redis, etc.
- **Cache system:** Redis, etc.
- **Graph:** Neo4j, etc.

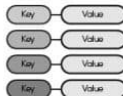
## Document



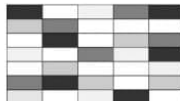
## Graph



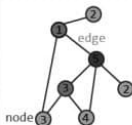
## Key-Value



## Wide-Column



```
{
  "user": {
    "id": "143",
    "name": "improgrammer",
    "city": "New York"
  }
}
```



1	Fruit	A Foo	B Baz	
2	City	E DC	D PLA	G FLD
3	State	A NZ	C CL	



There are many NoSQL databases but **no NoSQL standard**

Common points between NoSQL DBs:

- They have an **implicit schema**:
  - ▶ Data **schema not predefined on the server side**
  - ▶ The **client application structures the data**
  - ▶ Some exceptions: Cassandra V2
- There are **no relations**:
  - ▶ No relationship between data or between elements of two collections
  - ▶ Some exceptions: Neo4 and Hive in some cases

In this course

We will see **MongoDB** and **ElasticSearch**

## Advantages and disadvantages of NoSQL and Relational Databases.

### NoSQL:

- **Advantages:**

- ▶ Can **deal with large volumes of structured, semi-structured, and unstructured data**
- ▶ Set of **functions or API easier to use than SQL**
- ▶ **Efficient horizontal scaling** (instead of expensive vertical scaling)

- **Disadvantages:**

- ▶ **Less support** since NoSQL databases are usually open-source
- ▶ NoSQL databases **require technical skill** in order to install and maintain
- ▶ **Less mature:** they are still growing and many features have to be implemented

## Relational databases:

- **Advantages:**

- ▶ Can **handle very complex queries, database transactions, and routine analysis of data**
- ▶ Respect "ACID" (**Atomity, Consistency, Isolation, Durability**): properties ensuring reliable database transactions
- ▶ Can **handle constraints** (Ex: make sure data can only be deleted if some conditions are met)

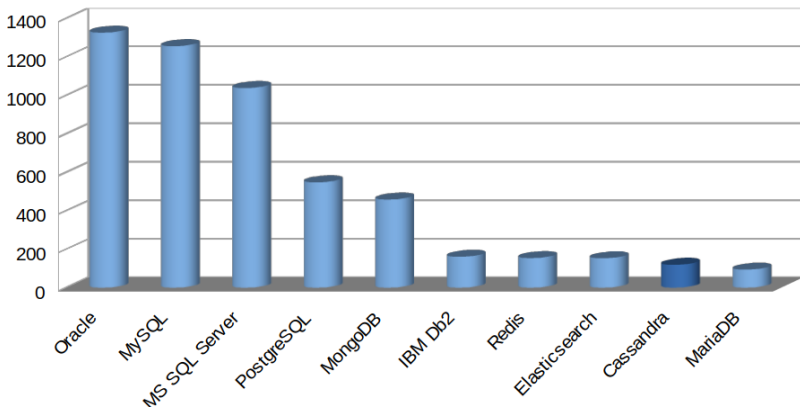
- **Disadvantages:**

- ▶ Cannot store **too complex or too large** images, numbers, designs and multimedia products
- ▶ Can **become very costly in maintenance and fragile**

- 1 Recall
- 2 NoSQL
- 3 MongoDB: Document based NoSQL
- 4 ElasticSearch: A different kind of Document-based NoSQL DB

MongoDB: A document-based NoSQL database and the most popular NoSQL database.

DB-Engines Ranking Score (Dec, 2020)



- Documents are stored using a **hierarchical representation**
  - ▶ Documents are written with the **BSON** syntax
    - ★ **BSON = Binary JSON**
    - ★ **BSON: extension of the JSON format** containing additional types
- The DB is **schemaless**
  - ▶ **No mandatory attribute**
  - ▶ **No fixed type** for an attribute
  - ▶ **No need to perform complex and dangerous DB migrations**
- MongoDB documents are **similar Python dictionaries**
  - ▶ Set of key/value pairs.

## Note

Because NoSQL DBs are schemaless, constraints and data relations must be explicitly handled on the application-side.



Example of a BSON MongoDB document:

```
{ 'name' : 'Jean', 'height':170}  
{ 'name' : 'Jacques', 'height' : 180, 'job' : 'teacher'}  
{ 'type' : 'car', 'brand' : 'renault', 'price' : 1500 }  
{ 'type' : 'house'}
```

## Properties of MongoDB documents:

- Keys...

- ▶ are **string** of characters
- ▶ are **case sensitives**
- ▶ must **be unique**:
  - ★ {"name": "Romain", "height":185, "name": "Tavenard"} **not** valid

- Values...

- ▶ are **case sensitives**: {"name": "Roman"} != {"name": "roman"}
- ▶ are **type sensitives**: {"height": "185"} != {"height": 185}

MongoDB documents are stored in collections  
(a **collection** = set of documents)

- **Collection:**

- ▶ Example: [{"name": "Romain", "height": 185}, {"name": "Paul", "height": "172"}, {"name": "Romain", "height": 163, "weight\_kg": 65}]
- ▶ No schema: keys, values and types can change from one doc to another

Comparison of naming with Relational Databases:

Relational DB	Document-based DB
Table	Collection
Recording	Document

## Focus on the **JSON** format:

- Recall: JSON means **JavaScript Object Notation**
- Example of a JSON document: `{"name": "Romain", "height": 185}`
- Data types:
  - ▶ null: `{"x": null}`
  - ▶ Boolean: `{"x": true}`
  - ▶ Number: `{"x": 3.14}`
  - ▶ String: `{"x": "abcdef"}`
  - ▶ Array: `{"x": [1, 5, 7]}`
  - ▶ Date: `{"x": new Date()}`

How to interrogate the DB?

- **No need of SQL**
- Can read/write the DB using **JavaScript** or **Python**
- Languages allowing more than data access
  - ▶ Definition of variables
  - ▶ Loops
  - ▶ Etc.

How to use MongoDB on Linux:

Launch the MongoDB daemon (depend on your system). On GNU/Linux systems using "systemd":

```
$ systemctl start mongod.service
```

Launch mongo:

```
$ mongo
```

Create a database (or use it, if it already exist):

```
> use my_db
```

Create a collection:

```
> db.createCollection("my_collection")
```

Display the databases:

```
> show dbs
```

Display the collections:

```
> show collections
```

We can interrogate MongoDB with Python using the "pymongo" module:

Let's connect to the MongoDB and fetch the database « my\_db » we just created:

```
$ python
> from pymongo import MongoClient
> client = MongoClient()
> col = client.my_db
```

Let's see how to perform the basic **CRUD** operations using pymongo  
(**CRUD** = **C**reate **R**emove **U**pdate **D**ele~~t~~e)

### Create:

Insert a single document using `insert_one(document)`:

```
> result = col.insert_one({'x':1})  
> result.inserted_id  
ObjectId('583c16b9dc32d44b6e93cd9b')
```

Insert multiple documents using `insert_many(documents)`:

```
> result = col.insert_many([{'x': 2}, {'x': 3}])  
> result.inserted_ids  
[ObjectId('583c17e7dc32d44b6e93cd9c'),  
ObjectId('583c17e7dc32d44b6e93cd9d')]
```



## Update:

Update a single document matching a filter using **update\_one(filter, update, upsert=False)**:

```
> result = col.update_one({'x': 1}, {'x': 3})
```

Update one or more documents matching a filter using **update\_many(filter, update, upsert=False)**:

```
> result = col.update_many({'x': 1}, {'x': 3})
```

## Read:

Query the database using `find(filter=None, projection=None, skip=0, limit=0, no_cursor_timeout=False)`:

The filter argument is a prototype document that all results must match:

```
> result = col.find({'x': 1})
```

Get a single document from the collection using `find_one(filter=None)`:

```
> result = col.find_one()
```

## Delete:

Delete a single document matching a filter using **delete\_one(filter)**:

```
> result = col.delete_one({'x': 1})  
> result.deleted_count
```

Delete one or more documents matching a filter using **delete\_many(filter)**:

```
> result = col.delete_many({'x': 1})  
> result.deleted_count
```

pymongo also provides **find\_one\_and\_delete()** and **find\_one\_and\_replace()** functionality.

## More information

For more information see the official documentation:

<https://pymongo.readthedocs.io/en/stable/tutorial.html>

- 1 Recall
- 2 NoSQL
- 3 MongoDB: Document based NoSQL
- 4 ElasticSearch: A different kind of Document-based NoSQL DB

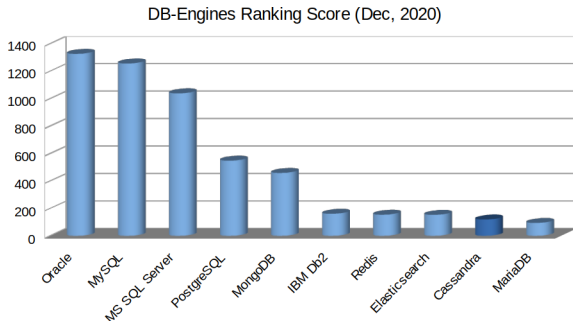
## ElasticSearch...

- is a different kind of document-based NoSQL database
- is a also powerful **real-time distributed search and analysis tool**

ElasticSearch is used for:

- Full text search
- Structured search
- Analysis
- All three combined...

ElasticSearch is also a very popular NoSQL DB:



It is used by:

- Wikipedia (<http://fr.wikipedia.org>)
- The Guardian (<http://www.theguardian.com>)
- StackOverflow (<http://stackoverflow.com/>)
- GitHub (<https://github.com/>)
- Goldman Sachs (<http://www.goldmansachs.com/>)

ElasticSearch is...

- A distributed **real-time** document DB where **all fields are undefined and searchable**
- A distributed **search engine with real-time analysis**
- Capable of supporting a scalability of **hundreds of servers and petabytes of structured or unstructured data**
- Allow to:
  - ▶ Perform and combine **various searches** on structured, unstructured, geolocation or indicator data
  - ▶ **Explore trends and identify patterns** in the data

## Why Elasticsearch?

Most databases are inadequate at extracting actionable data. They **cannot do full-text search, handle synonyms, and sort documents by relevance**. Besides, they **do not do it in real-time**.

How are stored the documents?

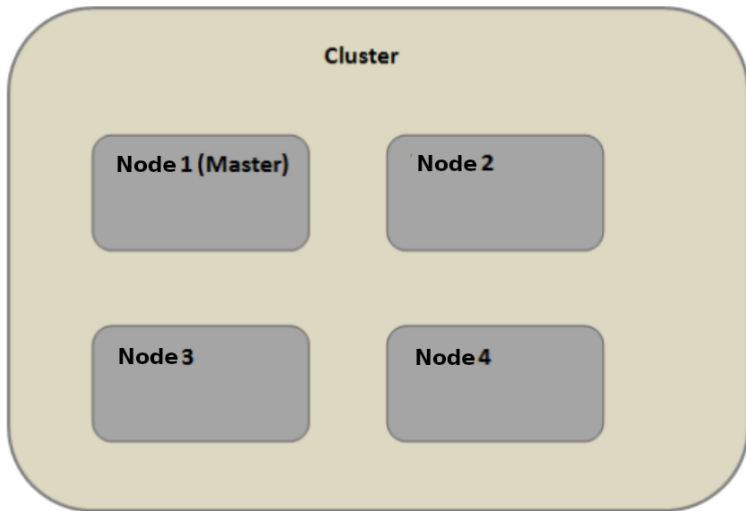
- The **content** of each document is indexed
- A document **has a Type** (which defines its mapping)
- **Types** are contained in an **Index**

Comparison ElasticSearch VS Relational Databases VS MongoDB:

Relational DB	Database	Tables	Rows	Columns
MongoDB	Database	Collections	Documents	Fields
Elasticsearch	Index	Types	Documents	Fields



Alike MongoDB, **documents are stored in a cluster**. And we can horizontally scale the cluster by adding nodes:

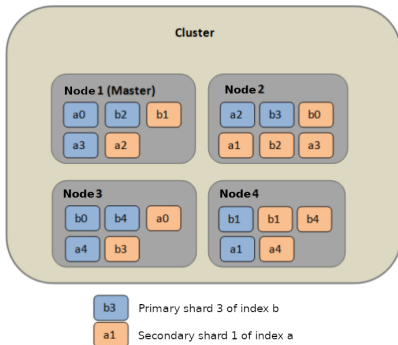


ElasticSearch is not really a DataBase, it is an index.

- An index is a **logical storage space for documents of the same type** split into one or more Primary Shards
- An index can be **replicated on zero or more Secondary Shards**

## Shards

- **Primary Shards:** This is a partition of the index  
(Default: 5 Primary Shards)
- **Secondary Shards:** Copies of the Primary Shards  
(Zero to several times number of Primary Shards)



Why we have shards?

- It's **faster to write and read** big amount of data
- It is possible to write on different nodes in the same time, and collect from different locations: **no funnel effect**
- **Shards are replicated** to make the cluster more robust

## ElasticSearch Document

- 1 document = a **simple record** in an **ElasticSearch** shard
- Documents are structured as JSON object and must belong to a **Type** (defining its structure)

Example:

```
{
  "nom": "Paris",
  "codePostal": "75000",
  "monuments": [
    {
      "nom": "Arc de Triomphe"
    },
    {
      "nom": "Tour Eiffel"
    }
  ]
}
```

ElasticSearch offers a **REST API** to perform operations through HTTP (**GET, PUT, POST and DELETE methods** can be performed)

API calls are **performed on an URL address with the following syntax:**

`http://localhost:9200/[index]/[type]/[id]/[action]`

- index: Name of the index
- type: Name of the document type
- id: ID of the document
- action: Action to perform

## Note

To call the API through HTTP we can use the "curl" command (or Python+requests)

Let's see how create an Index named "articles":

```
curl -X PUT "localhost:9200/articles?pretty" -H 'Content-Type: application/json' -d'
{
  "settings" : {
    "index" : {
      "number_of_shards" : 3,
      "number_of_replicas" : 2
    }
  }
}
```

And a type:

```
curl -XPUT "http://localhost:9200/articles/_doc/_mapping" -d
{
  "_doc": {
    "properties": {
      "title": {
        "type": "string"
      },
      "description": {
        "type": "string"
      },
      "author": {
        "type": "string"
      }
    }
  }
}
```

## Let's see how to index a document (= insert a document):

```
curl -XPOST 'localhost:9200/articles/_doc/1?pretty' -d '{"title": "python tuples",
"description": "practical operations with python tuples","author": "santosh"}'
-H 'Content-Type: application/json'
```

This will return :

```
% Total      % Received % Xferd Average Speed   Time    Time       Time  Current Dload  Upload   Total   Spent
100   377   100   222   100   155     222    155   0:00:01  --:--:--   0:00:01  1008{
"_index" : "articles",
"_type" : "_doc",
"_id" : "1",
"_version" : 1,
"result" : "created",
"_shards" : {
  "total" : 2,
  "successful" : 2,
  "failed" : 0
},
"_seq_no" : 0,
"_primary_term" : 1
}
```

If it succed it will return a HTTP 200 code.

Let's see how to delete a document:

```
curl -XDELETE 'localhost:9200/articles/_doc/1?pretty'
```

% Total Dload	% Received Upload	% Xferd Total	Average Spent	Speed Left	Speed	Time	Time	Time	Current
100	241	100	241	0	0	241	0	0:00:01	--:--:-- 0:00:01 1928{

```
"_index" : "articles",
"_type" : "_doc",
"_id" : "1",
"_version" : 2,
"result" : "deleted",
"_shards" : {
  "total" : 2,
  "successful" : 2,
  "failed" : 0
},
"_seq_no" : 1,
"_primary_term" : 1
}
```



## And finally an example of a search:

```
curl -XPOST "https://localhost:9200/_search" -d'{ "query": { "query_string": { "query": "hello" } } }'  
Results:  
{  
  "took": 12,  
  "timed_out": false,  
  "_shards": {  
    "total": 12,  
    "successful": 12,  
    "failed": 0  
  },  
  "hits": {  
    "total": 1,  
    "max_score": 0.19178301,  
    "hits": [  
      {  
        "_index": "my-first-index",  
        "_type": "message",  
        "_id": "AUqiBnvdK4Rpq0ZV4-Wp",  
        "_score": 0.19178301,  
        "_source": {  
          "text": "Hello world!"  
        }  
      }  
    ]  
  }  
}
```

## Official doc

More information about the Elasticsearch REST API on this website:  
<https://www.elastic.co/guide/en/elasticsearch/reference/current/docs.html>

As mentioned above, it's possible to use **ElasticSearch** with **Python**:

```
settings = {
    "settings": {
        "number_of_shards": 3,
        "number_of_replicas": 2
    },
    "mappings": {
        "profile": {
            "properties": {
                "name": {
                    "type": "string"
                },
                "age": {
                    "type": "integer"
                },
                "address": {
                    "type": "string"
                }
            }
        }
    }
}

from elasticsearch import Elasticsearch
# Connect to the elastic cluster
es=Elasticsearch([{'host':'localhost','port':9200}])
es.indices.create(index='people', body=settings)
```

```
document = {
    'name': 'Jean',
    'age': 19,
    'address': 'Paris',
}
res = es.index(index="people", id=1, body=document)
print(res['_result'])

res = es.get(index="people", id=1)
print(res['_source'])

es.indices.refresh(index="people")

res = es.search(index="people", body={"query": {"query_string": {"query" : "Jean"}}})

print("Got %d Hits:" % res['_hits']['_total']['_value'])
for hit in res['_hits']['_hits']:
    print("(%(timestamp)s %(author)s: %(text)s" % hit["_source"])
```

## Official doc

More information about the Python Elasticsearch API here:  
<https://elasticsearch-py.readthedocs.io/en/6.8.2/api.html>