



# MOTEUR DE RECHERCHE ELASTICSEARCH

Illustration avec la suite ELK  
Durée : 5 jours





# MODULE 1

## Introduction

# Elasticsearch ?

---

- Elasticsearch est un moteur de recherche distribué en temps réel.
- Il permet d'explorer les données très rapidement et de "clusteriser" facilement.
- Il est utilisé pour de la recherche dans le texte, recherche par mot clé, etc.

# Elasticsearch ?

---

- Un moteur de recherche open source.
- Développé en Java → JVM nécessaire
- ElasticSearch est basé sur **Apache Lucene**.
- Créateur **Shay Banon**
- La première version : Février 2010
- NoSQL oriented document
- Distribuée et scalable
- HTTP/REST/JSON
- Sans schéma (pas de définition stricte du contenu des index)

# Pour quels besoins ?

---

- 2 use cases principaux :
  - Recherche
  - Statistiques
- Exemple:
  - Recherche partielle de texte (livres, documents, posts blog ...)
  - Recherche de texte / Recherche structuré de données (produit, profile utilisateur, log d'application...)
  - Agrégation de données (statistique, mesure, analyse...)
  - Recherche géo localisée

# Exemple

<https://github.com/search?utf8=✓&q=java>

**GitHub** [Explore](#) [Features](#) [Enterprise](#) [Blog](#) [Sign up](#) [Sign in](#)

Search

java

Search

📁 Repositories194,880

<> Code129,302,934

🔔 Issues1,023,909

👤 Users4,322

Languages

Java118,632

JavaScript15,068

HTML2,586

CSS1,863

Shell1,270

Ruby1,169

Python910

C++757

Scala724

C702

We've found 194,880 repository results

Sort: Best match ▾

pubnub/java

Java ★ 118 📄 249

PubNub **Java**-based APIs for core **Java**, Android, J2ME, BlackBerry

Updated 2 days ago

hmkcode/Java

JavaScript ★ 101 📄 413

**Java** related code

Updated on 22 Sep 2014

agileorbit-cookbooks/java

Ruby ★ 203 📄 421

Chef **Java** Cookbook

Updated 8 days ago

MBENGUE Mohamadou

# Qui utilise Elasticsearch ?

---



# Autres besoins





# Atouts ?

---

- **Simplicité** : Sa mise en place est très simple.
- **Rapidité** : Les recherches sont traitées en quasi temps réel grâce à la parallélisation des traitements.
- **Scalabilité** : Le rajout de nouveau nœud permet d'augmenter la capacité de traitement et d'être en haute disponibilité.
- Pas de schéma établi(**schemaless**)
- **Sauvegarde** : Les données sont automatiquement sauvegardées et répliquées.
- **Accessibilité** : API REST / JAVA

# Pourquoi Elasticsearch ?

---

- La plupart des BDD sont inadéquates à extraire des données exploitables.
- Bien sûr, elle peuvent filtrer par date ou par valeurs exactes mais ne peuvent pas faire une recherche en plein texte, gérer les synonymes et trier des documents par pertinence.
- Et surtout, elle ne le font pas en temps réel et sans grosses requêtes ponctuelles.

# Lucene

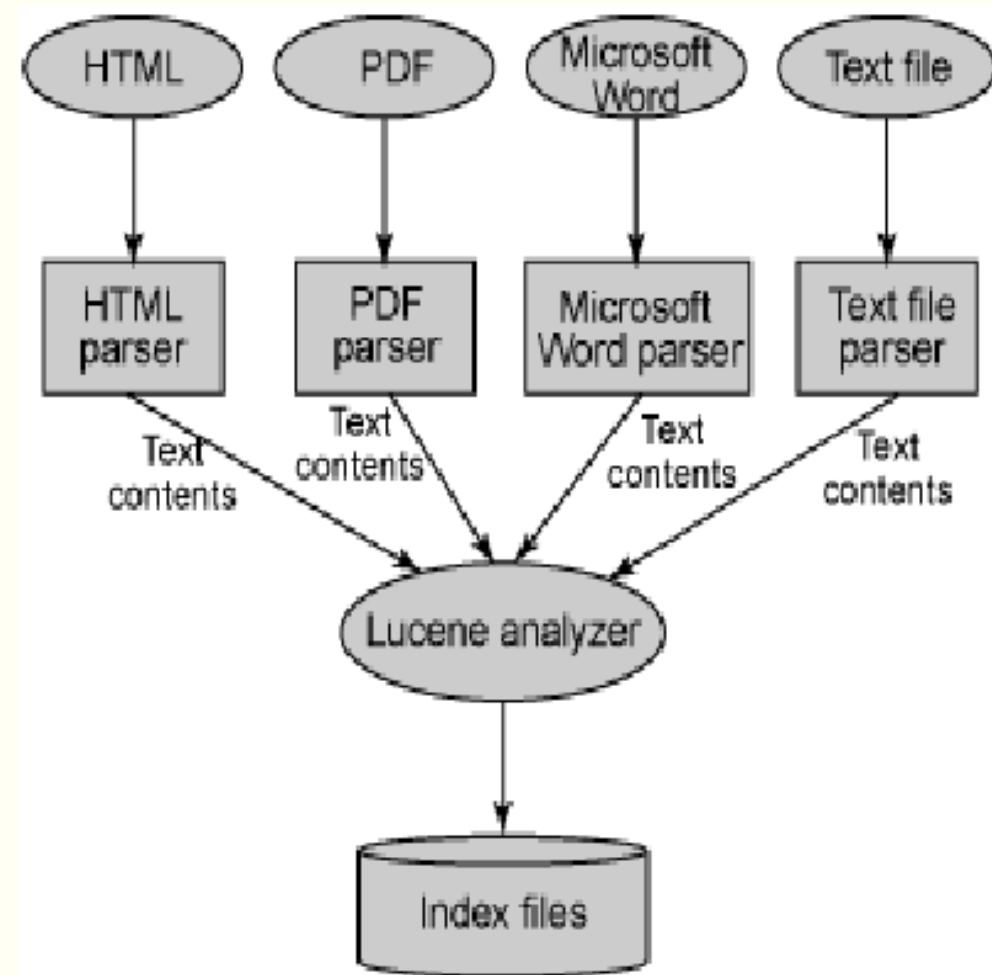
---

ElasticSearch est basé sur **Apache Lucene**

**Lucene** est un  
moteur de recherche  
libre écrit en Java  
qui permet  
d'indexer et  
de rechercher du texte.

## Limite:

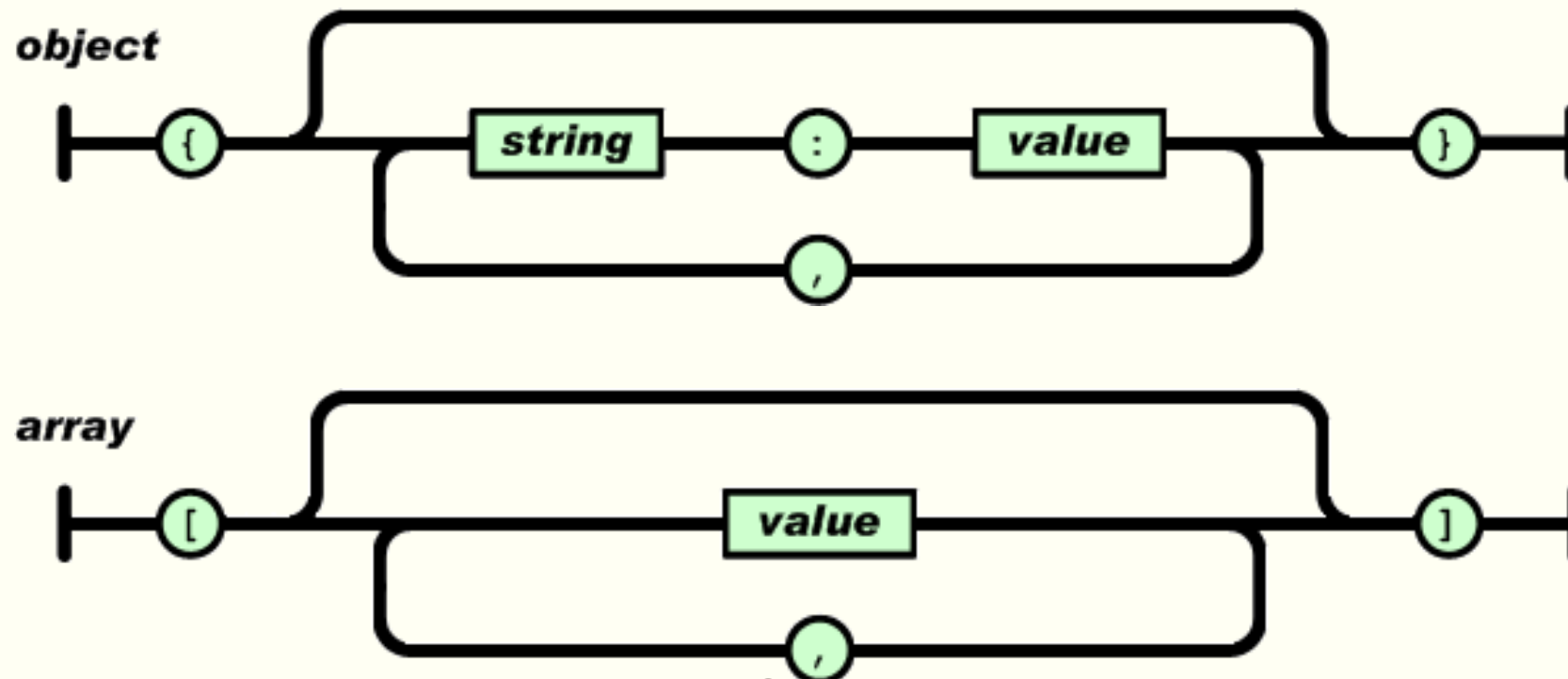
- Scalabilité verticale avec Lucene



# Format JSON

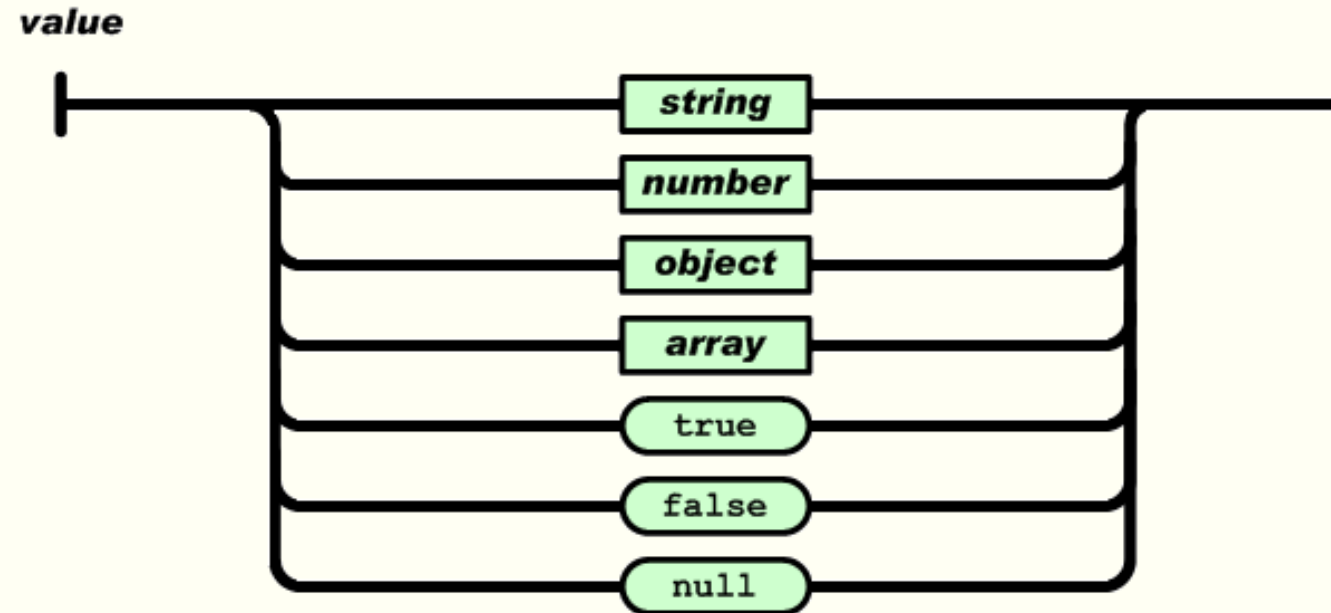
---

- Official Site: <http://json.org/>
- Un document **json** est de la forme: **{ }** (object) ou **[ ]** (tableau).



# Value JSON?

---



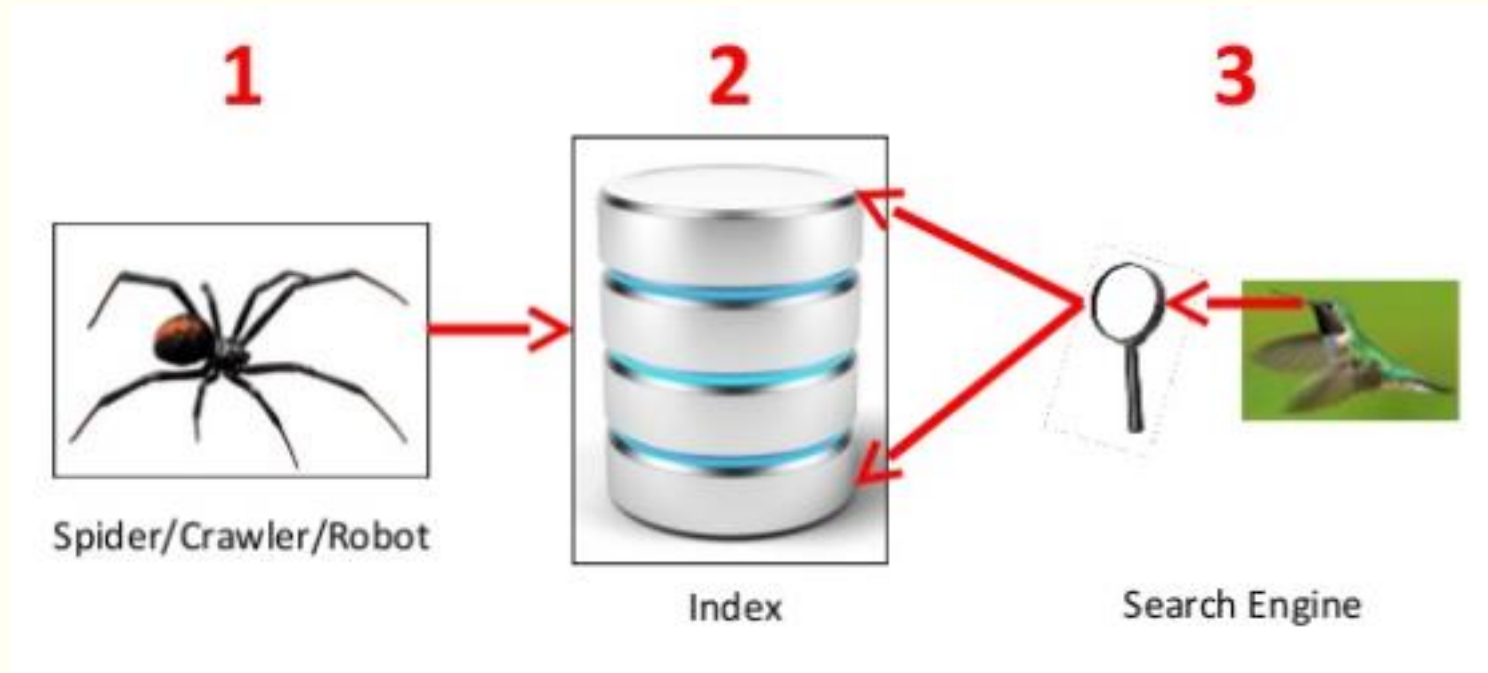
```
{ "firstName":"John" , "lastName":"Doe" }
```

```
[ "Text goes here", 29, true, null ]
```

# Composition d'un moteur de recherche?

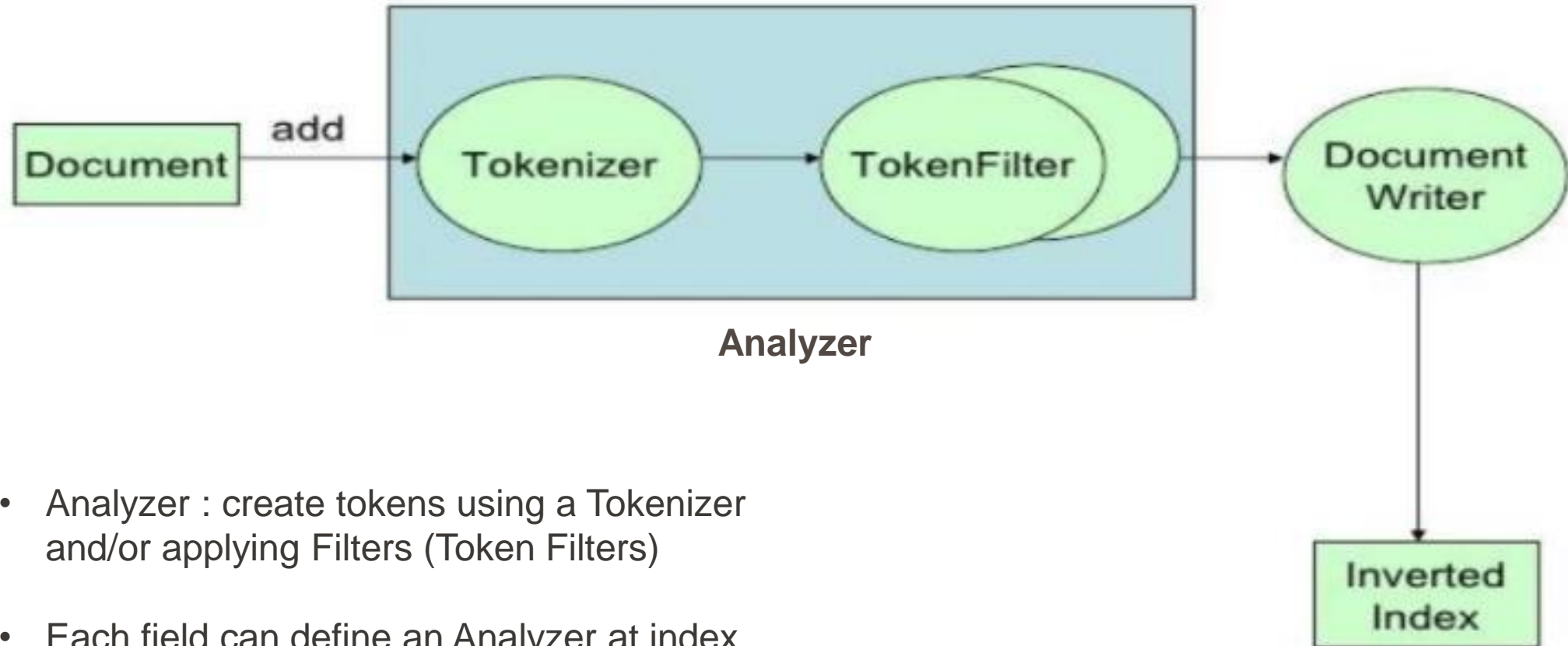
---

- Un moteur d'indexation
- Un moteur de recherche dans les index



# Processus de création de l'index

---



- Analyzer : create tokens using a Tokenizer and/or applying Filters (Token Filters)
- Each field can define an Analyzer at index time/query time or the both at same time

# Principe de fonctionnement d'un index inversé

## Exemple 1

- d1 = "c'est ce que c'est"
- d2 = "c'est ceci"
- d3 = "ceci est une orange"

	C'	est	ce	que	ceci	une	orange
d1	1	1	1	1	0	0	0
d2	1	1	0	0	1	0	0
d3	0	1	0	0	1	1	1



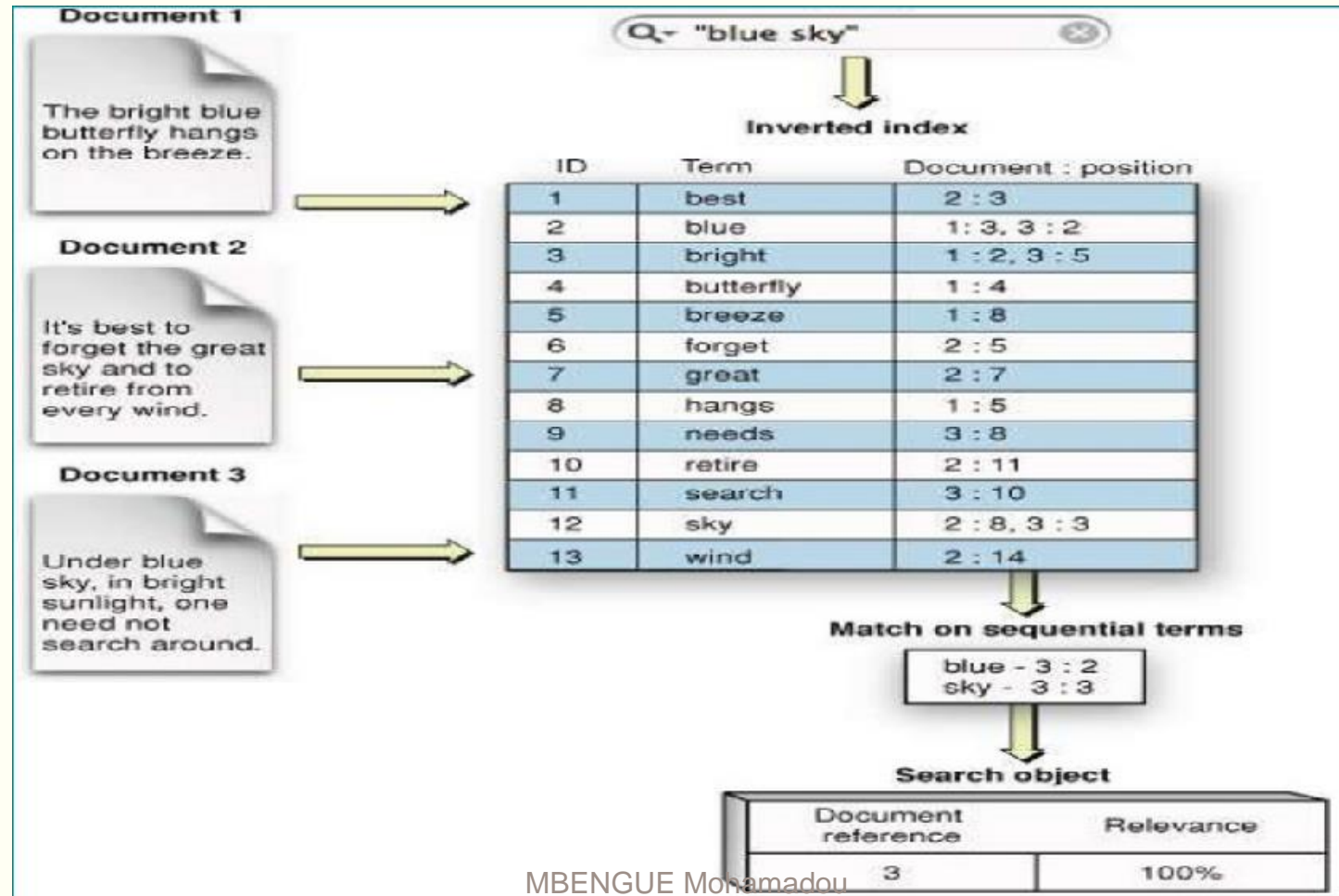
	d1	d2	d3
C'	1	1	0
est	1	1	1
ce	1	0	0
que	1	0	0
ceci	0	1	1
une	0	0	1
orange	0	0	1

$$recherchesur(\{ceci, est\}) = \{D2, D3\} \cap \{D1, D2, D3\} = \{D2, D3\}$$



# Principe de fonctionnement d'un index inversé

## ▪ Exemple 2



# ELK c'est quoi?

---

- **Elasticsearch** - Base NoSql distribuée et moteur de recherche Lucene
- **Logstash** - ETL spécialisé dans la gestion des logs
- **Kibana** - Interface graphique pour Elasticsearch

# LogStash

---

**LogStash est un ETL (Extract-Transform-Load)**

Il permet nativement de :

- Récupérer les logs provenant de sources variées,
- Transformer les logs vers de multiples formats,
- Sauvegarder le résultat de la transformation vers différents systèmes de stockage.

# LogStash

---

LogStash propose par défaut:

- **41 entrées** : syslog, zeromq, file, collectd, pipe, eventlog, etc...
- **20 codecs** : json, json\_lines, multiline, etc...
- **50 filtres** : grok, date, geoip, mutate, etc...
- **55 sorties** : elasticsearch, stdout, rabbitmq, graphite, etc...

# Kibana

---

## Kibana est l'interface web de référence d'ElasticSearch

Depuis la version 4, les opérations dans kibana se décomposent en 3 parties :

- **Discover:** permet de visualiser les données des index elasticsearch.
- **Visualize:** cœur de kibana pour mettre en forme et agréger les données dans des vues.
- **Dashboard:** pages de synthèse des vues.

# Architecture générale

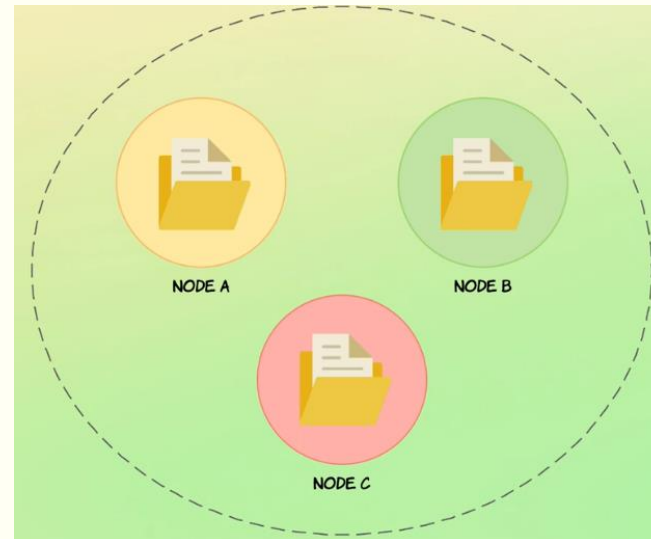
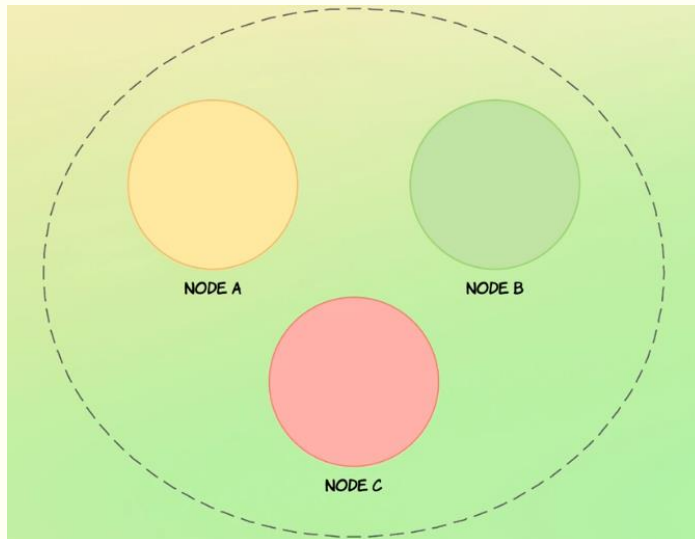
---

## Nœud

- Correspond à un processus d'Elasticsearch en cours d'exécution.

## Cluster

- Un cluster est composé d'un à plusieurs nœuds. Un nœud maître est choisi, il sera remplacé en cas de défaillance.



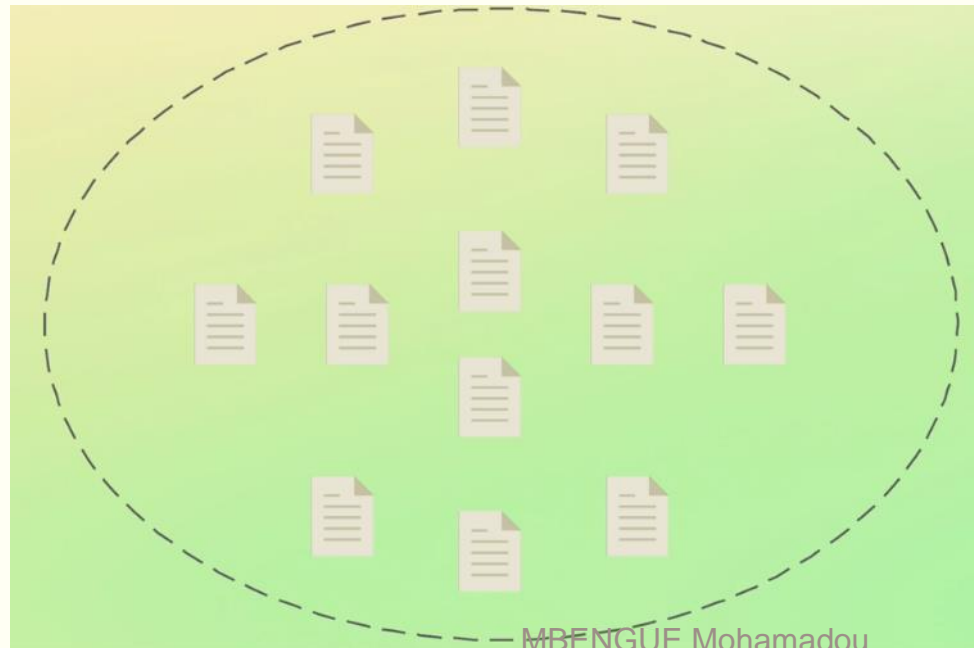
MBENGUE Mohamadou  
Chaque nœud contient une partie des données

# Architecture générale

---

## Index

- Un index est un espace logique de stockage de documents de même type, découpé sur un à plusieurs Primary Shards.
- Un index peut être répliqué sur zéro ou plusieurs Secondary Shards.



# Architecture générale

---

## Primary Shards

- C'est une partition de l'index.
- Par défaut, l'index est découpé en 5 Shards Primary.
- Il n'est pas possible de changer le nombre de Shards après sa création.

## Secondary Shards

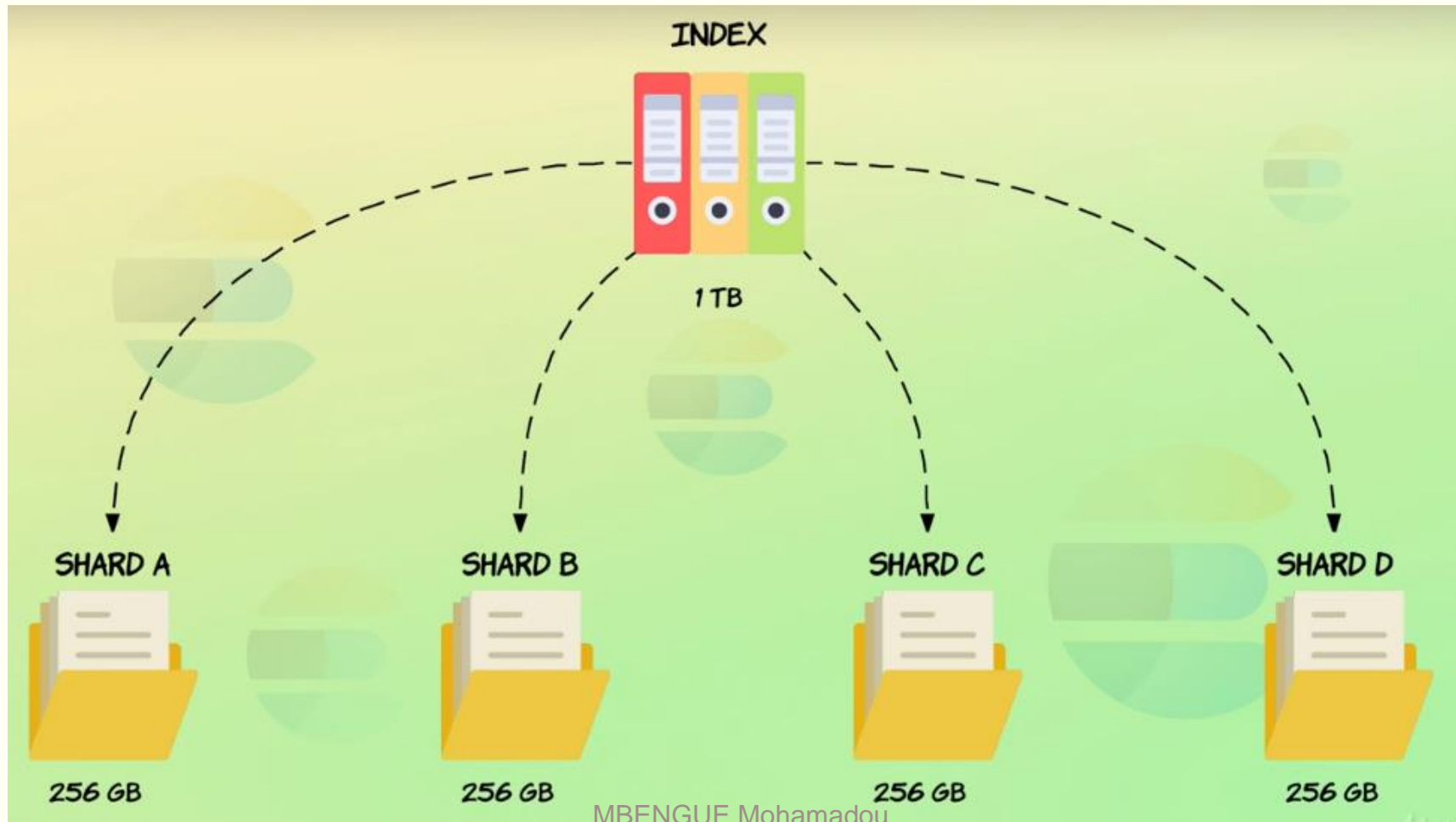
- Il s'agit de copies de Primary Shards.
- Il peut y en avoir zéro à plusieurs par Primary Shard.
- Ce comportement est adopté à des fins de performance et de sécurisation des données.



# Architecture générale

---

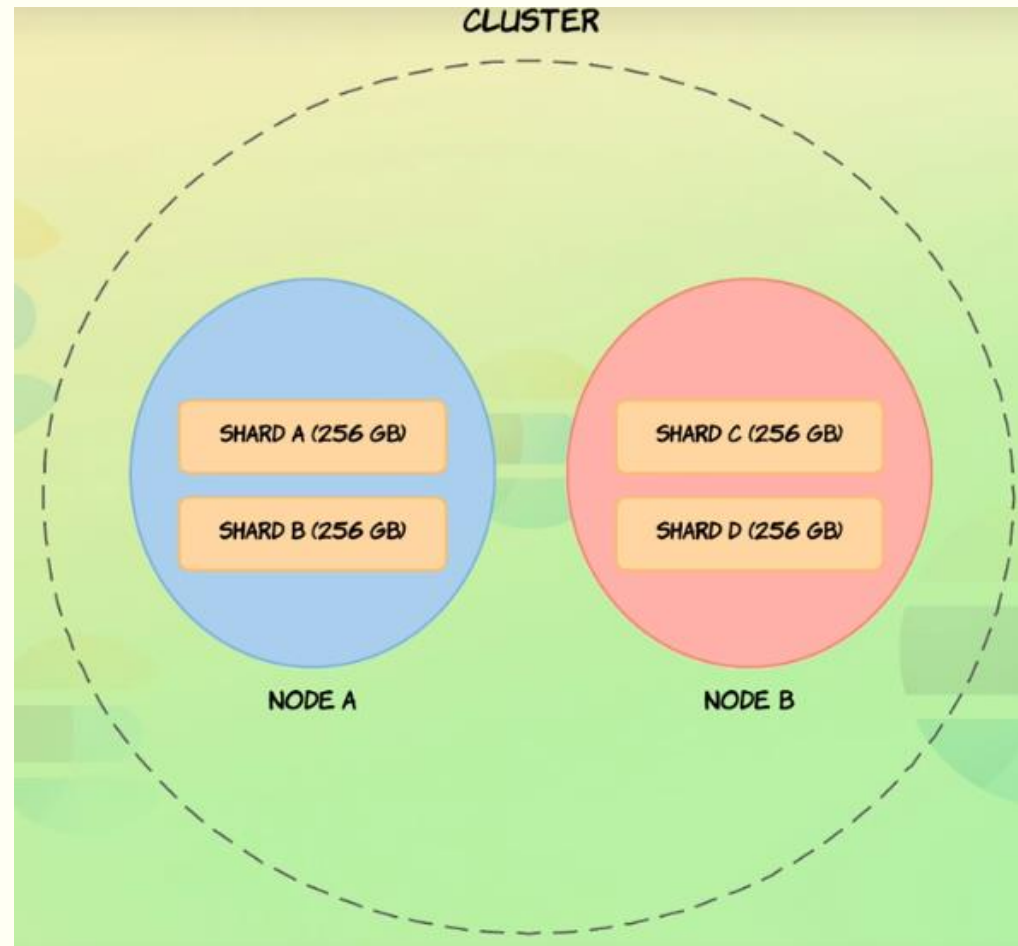
## Sharding



# Architecture générale

---

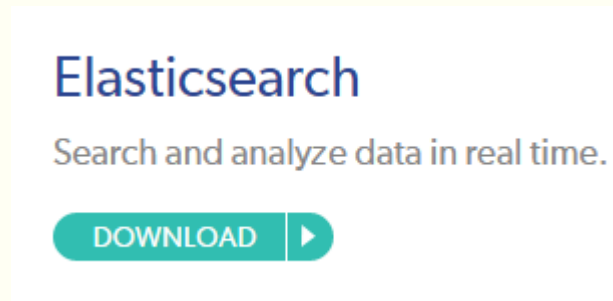
## Sharding vs distribution



# Mise en place d'ElasticSearch

---

- Télécharger ElasticSearch(Zip) sur le site <https://www.elastic.co/downloads>



- Décompressez le zip dans un répertoire (exemple : c:\serveur\elasticHome) que nous appellerons ES\_HOME

# Mise en place d'ElasticSearch

---

## La distribution contient :

- **bin** : fichier de commandes
- **config** : contient les fichiers *elasticsearch.yml* et *logging.yml*
- **lib** : contient les librairies

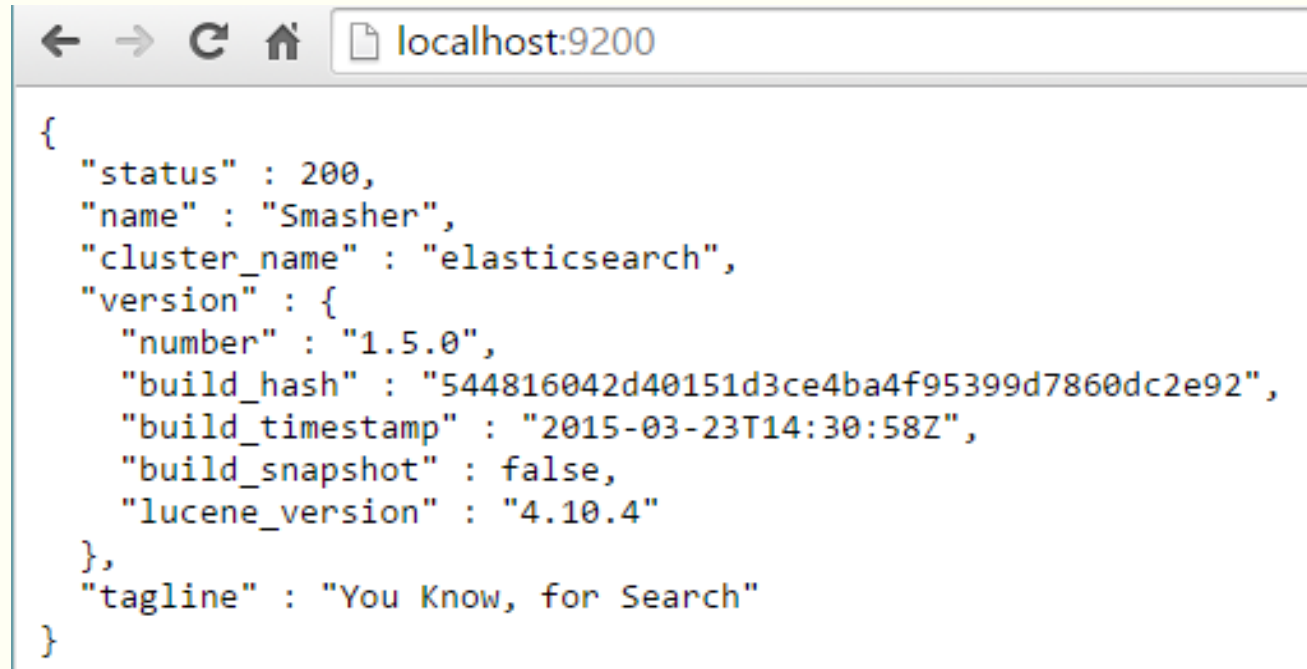
## Au lancement, ElasticSearch va créer de nouveaux répertoires :

- **data** : destiné à contenir les données indexées
- **logs** : qui contient les fichiers de journalisation
- **work** : qui contient des fichiers temporaires nécessaires au fonctionnement du moteur de recherche

# Démarrez Elasticsearch

---

- Exécuter le script **ES\_HOME/bin/elasticsearch.bat**
- Vérifier qu'ES s'est correctement lancé.
- Ouvrir l'URL <http://localhost:9200/> dans un navigateur Web.



```
{
  "status" : 200,
  "name" : "Smasher",
  "cluster_name" : "elasticsearch",
  "version" : {
    "number" : "1.5.0",
    "build_hash" : "544816042d40151d3ce4ba4f95399d7860dc2e92",
    "build_timestamp" : "2015-03-23T14:30:58Z",
    "build_snapshot" : false,
    "lucene_version" : "4.10.4"
  },
  "tagline" : "You Know, for Search"
}
```

# Installation de Kibana / Sense

---

## ▪ Installation de Kibana 4.3.1

- Lien : <https://www.elastic.co/downloads/kibana>
- Décompressez le zip
- Lancer kibana-4.x.x/bin/[kibana.bat](#)

## ▪ Installation du plugin Sense

- Ouvrir un terminal windows tapez la commande suivante.

```
C:\prog\serveurs\kibana-4.3.1\bin>kibana plugin --install elastic/sense
Installing sense
Attempting to extract from https://download.elastic.co/elastic/sense/sense-latest.tar.gz
Downloading 318236 bytes.....
Extraction complete
Optimizing and caching browser bundles...
Plugin installation complete

C:\prog\serveurs\kibana-4.3.1\bin>
```

# Mise en route

<http://localhost:5601/app/sense>

The screenshot shows the Elasticsearch Sense web interface. At the top, a 'Server' dropdown is set to 'localhost:9200'. To its right are buttons for 'Development Trial', 'Dashboards', and icons for history, settings, and help. The main area is split into two panes. The left pane contains a request editor with the following text:

```
1 GET _search
2 {
3   "query": {
4     "match_all": {}
5   }
6 }
```

The right pane shows a response table with one row and one column, containing the number '1'. Several callout boxes with green arrows provide instructions:

- 'The URL to the Elasticsearch cluster.' points to the 'localhost:9200' dropdown.
- 'A history of requests made can be found here.' points to the history icon in the top right.
- 'A request is made up of a HTTP verb, a relative URL and the (optional) request body.' points to the request text in the left pane.
- 'Press the green arrow to send the request.' points to the green arrow icon in the top right of the left pane.
- 'Responses from Elasticsearch is presented here.' points to the response table in the right pane.
- 'Write requests here' points to the left pane.

At the bottom center, the text 'MBENGUE Mohamadou' is displayed.

# Disposition de titre et de contenu avec liste

---

**Question ?**



# Disposition de titre et de contenu avec liste

---

**TP**

**Installation**



# MODULE 2

Manipulations de base

# RESTful API en JSON sur HTTP

---

Cette API utilise le format JSON pour :

- les requêtes,
- les réponses

et supporte les principales méthodes HTTP

- PUT : création ou modification d'un document
- GET : récupération d'un document
- HEAD : test si un document existe
- DELETE : suppression d'un document
- POST : création

Retourne

- un code de retour HTTP (200, 404, etc.)
- une réponse encodé en JSON (sauf pour les requêtes HEAD)

# API REST – Utilisation

---

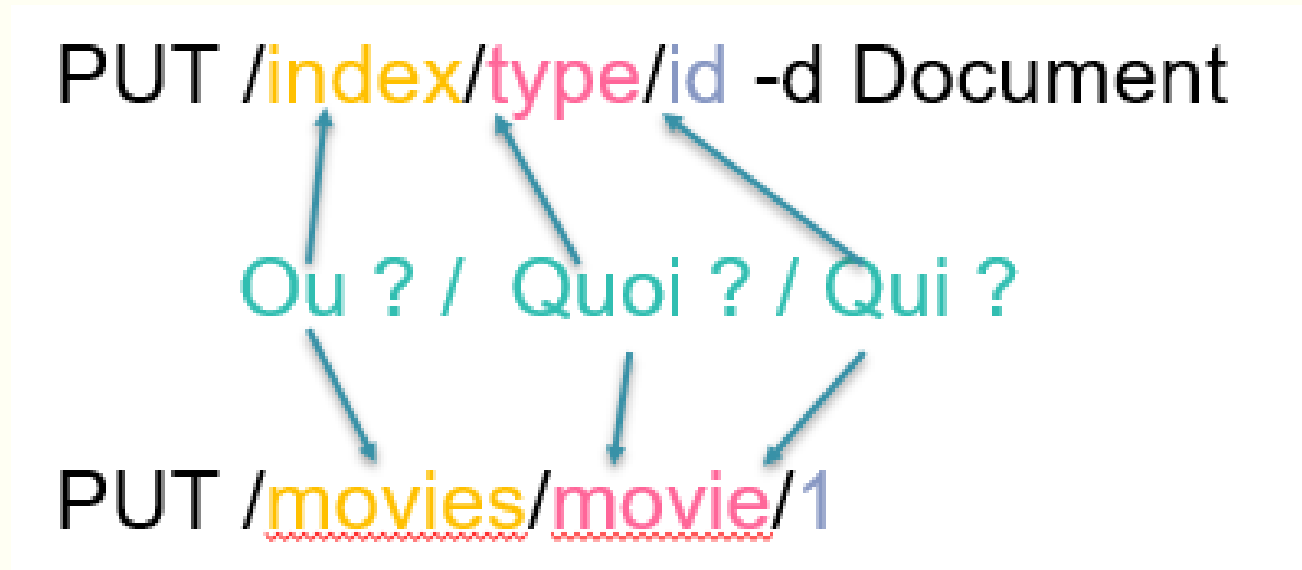
Elle est utilisée de la façon suivante :

`http://host:port/[index]/[type]/[_action|id] -d document`

- **index** : nom de l'index sur lequel porte l'opération
- **type** : une famille de document
- **\_action** : nom de l'action à effectuer
  - Dans ES, les actions sont préfixées de underscore "\_"
- **id** : identifiant du document
- **document** : Un élément typé et identifié(json)

# Exemple

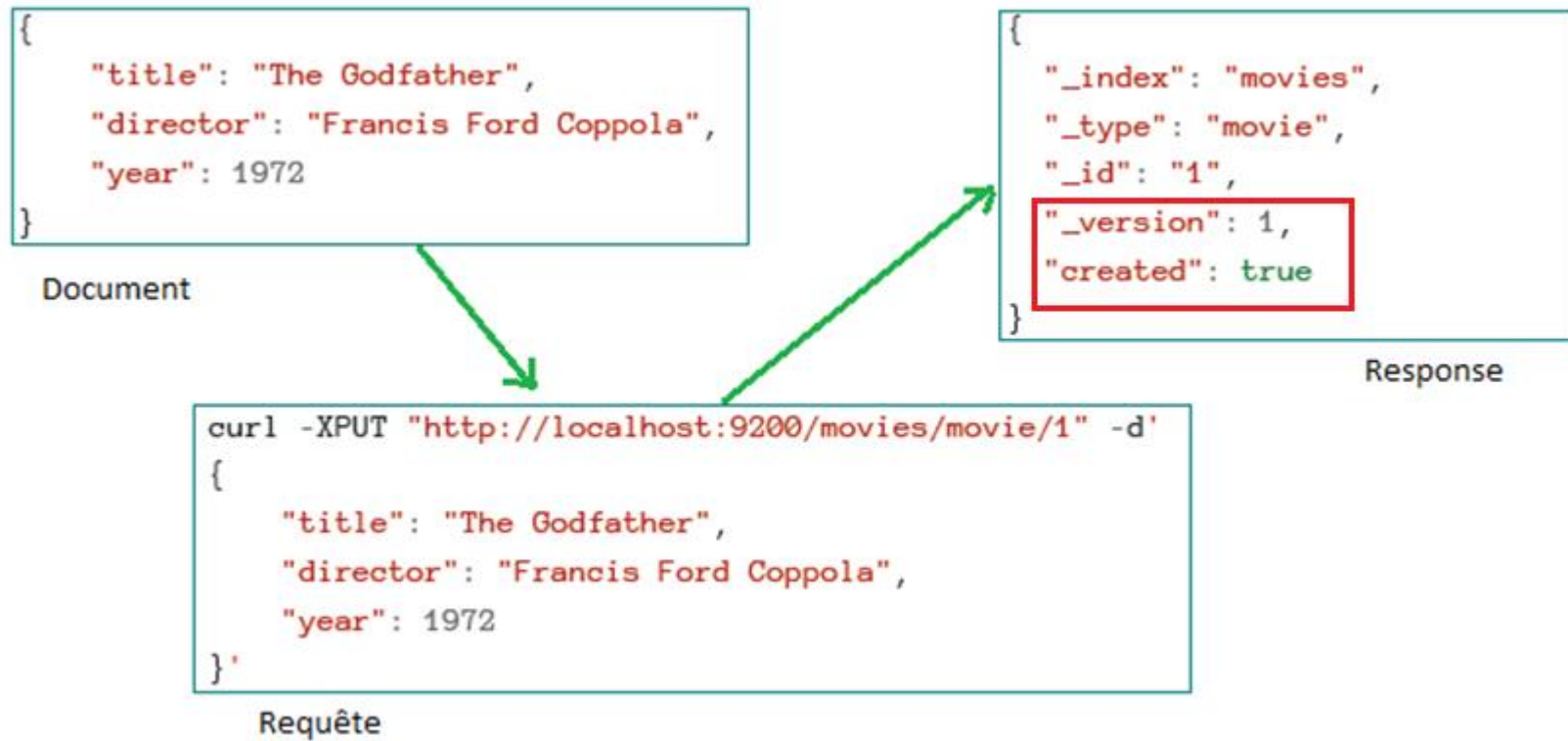
---



- Indexation d'un document de type **movie** dans un index nommé **movies**

# Index Création

---



Indexation d'un document de type **movie** dans un index nommé **movies**

# Indexation

---

## Indexation avec POST



```
1 POST http://localhost:9200/biblio/livres
2 {
3   "titre": "Shining",
4   "auteur": "Stephen King",
5   "genre": "fantastique",
6   "date_parution": "27/01/1977"
7 }
```

# Indexation

---

## Indexation avec PUT

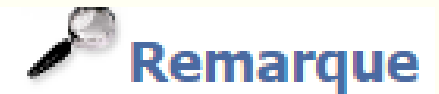


```
1 PUT http://localhost:9200/biblio/livres/1
2 {
3   "titre": "Shining",
4   "auteur": "Stephen King",
5   "genre": "fantastique",
6   "date_parution": "27/01/1977"
7 }
```



# Indexation

## Réponse de l'API



HTTP 200 : Document modifié

HTTP 201 : Document créé

HTTP 409 : Document déjà existant dans le cas d'une opération de création

```
1 {  
2   "_index": "biblio",  
3   "_type": "livres",  
4   "_id": "1",  
5   "_version": 1,  
6   "_shards": {  
7     "total": 2,  
8     "successful": 1,  
9     "failed": 0  
10  },  
11  "created": true  
12 }
```

# Indexation

---

## Version d'un document

Un numéro de version est assigné à la création d'un document.

Ce numéro de version est incrémenté pour chaque opération de ré-indexation, modification ou suppression.

# Indexation

---

## Modification d'un document



## Fondamental

Même méthodes que pour une création, si le champ donné existe déjà, il est mis à jour, sinon il est ajouté au document.

Attention si le **\_update** n'est pas ajouté, le document est remplacé entièrement (supprimé et créé avec uniquement les champs donnés)

```
1 POST http://localhost:9200/biblio/livres/1/_update
2 {
3   "doc" : {
4     "date_parution" : "28/01/1977"
5   }
6 }
```

# Indexation

---

## Suppression de documents



## Fondamental

Il est possible de supprimer un ou plusieurs document en changeant spécifiant son ID, ou de supprimer intégralement l'index en ne spécifiant rien.

```
1 DELETE http://localhost:9200/biblio/livres/3
2 DELETE http://localhost:9200/biblio/livres
```



# Recherche

---

- 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"
    }
  }
}
```

MBENGUE Mohamadou

# Recherche Simple

---

- Json → Transformation données en informations
- Tous les champs sont indexés
- ES peut rechercher l'info dans tous les champs
- Une recherche peut être :
  - Structuré (↔ Requête SQL)
  - Full-Text (*relevance*)
  - Une combinaison des deux

# Structure d'une recherche simple

---

Deux formes :

- Query-String

```
GET /_all/tweet/_search?q=tweet:elasticsearch
```

- Request-Body

```
GET /_search
{
  "query": YOUR_QUERY_HERE
}
```

# Recherche Simple

---

## Récupérer un document par son ID



Fondamental

Il suffit de changer le type de méthode. Pas besoin de le suffixé par `_search` puisqu'il ne s'agit pas d'une recherche, mais d'une récupération directe.

```
1 GET http://localhost:9200/biblio/livres/1
```

La réponse est soit HTTP 200 : OK si le document existe, soit HTTP 404 : NOT FOUND si celui si est introuvable

```
1 HTTP 200: OK
2 {
3   "_index": "biblio",
4   "_type": "livres",
5   "_id": "1",
6   "_version": 4,
7   "found": true,
8   "_source": {
9     "date_parution": "28/01/1977"
10  }
11 }
12
13 HTTP 404: NOT FOUND
14 {
15   "_index": "biblio",
16   "_type": "livres",
17   "_id": "2",
18   "found": false
19 }
```



# Recherche Simple

## Recherches simples



Les URL doivent systématiquement finir par `_search`.

Les méthodes HTTP reconnues sont GET et POST.

- GET permet d'envoyer des requêtes sans corps, avec des critères basiques en utilisant le paramètre `q`.
- POST permet d'envoyer des requêtes DSL avancées formalisées en JSON en envoyant un corps de requête.

La spécification de l'index et du type est optionnelle. Cela permet de définir directement la portée de la recherche dans l'URL de la requête. Il est possible de définir une recherche parmi plusieurs index ou types en les séparant par une virgule.

Quelques exemples d'URL de requête de recherche :

```
1 http://localhost:9200/_search
2 http://localhost:9200/index1/_search
3 http://localhost:9200/index1/type1/_search
4 http://localhost:9200/index1/type1,type2/_search
5 http://localhost:9200/index1,index2/type1/_search
6 http://localhost:9200/_all/type1/_search
7
8 http://localhost:9200/index1/_search?q=field:value
9 http://localhost:9200/index1/_search?q=value
```

# Recherche Simple

---

## Recherche simple

 **Exemple**

```
1 GET http://localhost:9200/biblio/_search
```

```
1 {
2   "took": 11,
3   "timed_out": false,
4   "_shards": {
5     "total": 5,
6     "successful": 5,
7     "failed": 0
8   },
9   "hits": {
10    "total": 1,
11    "max_score": 1,
12    "hits": [
13      {
14        "_index": "biblio",
15        "_type": "livres",
16        "_id": "1",
17        "_score": 1,
18        "_source": {
19          "titre": "Shining",
20          "auteur": "Stephen King",
21          "genre": "fantastique",
22          "date_parution": "28/01/1977"
23        }
24      }
25    ]
26  }
27 }
```

# Recherche Simple

---

- `took` : temps de la requête en millisecondes
- `time_out` : La recherche a-t-elle dépassé le temps imparti
- `hits` : Les résultats de la recherches
- `hits.total` : nombre de résultats correspondant aux critères
- `hits.hits` : résultat sous la forme d'un tableau de document
- `_score` : notion particulière d'Elasticsearch qui indique le taux de pertinence d'un document par rapport à une requête

# Recherche en texte intégral : Query-String

---

- La plupart du temps, les coordonnées (champs) du document ne sont pas connues.
- Il nous faut donc retrouver le document avec l'aide de quelques mots-clés.
- Les mots-clés sont passés dans le paramètre *q* (pour « *query* ») de l'URL.
- Pratique pour la recherche en ligne de commande.

# Disposition de titre et de contenu avec liste

---

**Question ?**

# Disposition de titre et de contenu avec liste

---

**Tuto 1**

**Crud**

**Exo 1**

**Cinémathèque**

**Tuto 2**

**Overview**

# Introduction à la recherche

---

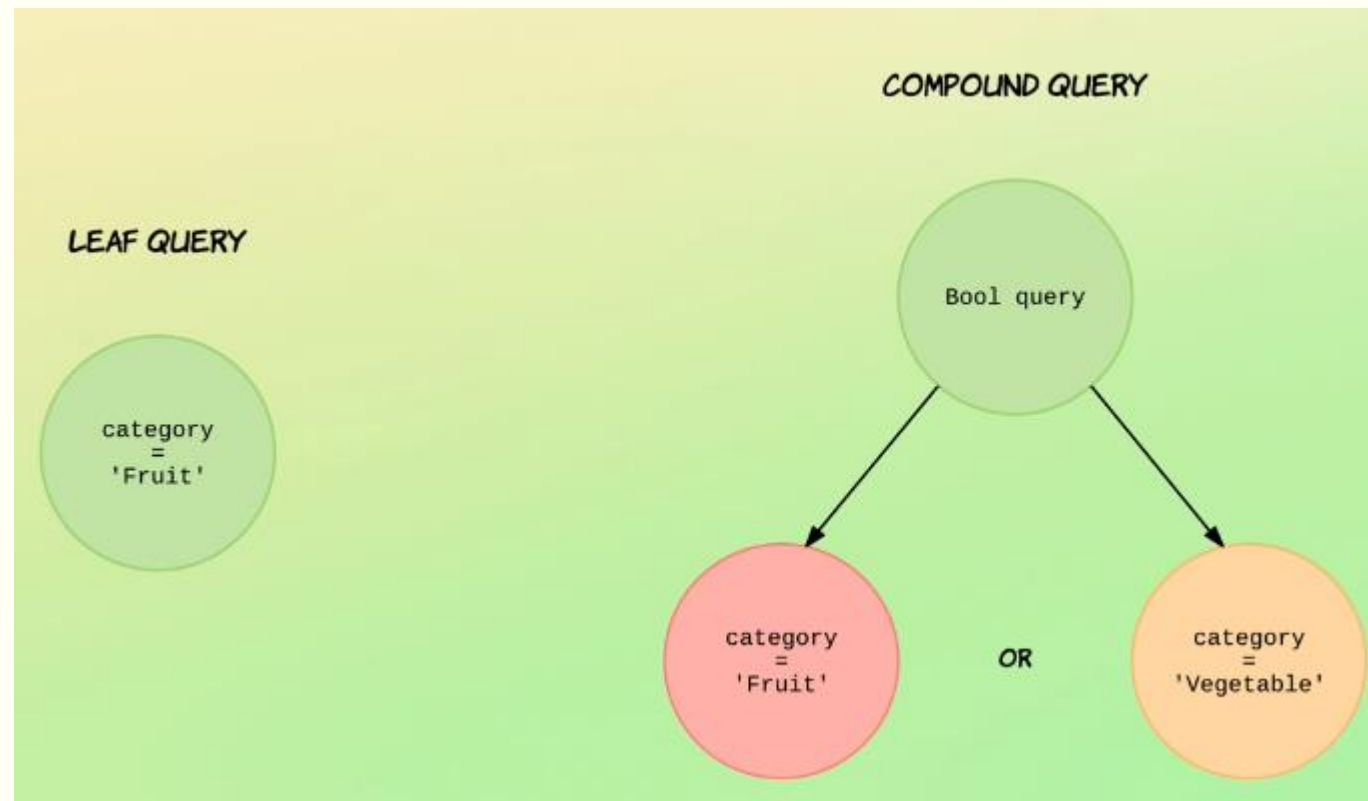
## Recherches avec une query DSL

- Requêtes :
  - Le résultat dépend d'un score attribué aux documents
  - Répond à la question : À quel point ce document correspond-il à cette clause de requête ? Score
- Filtres :
  - Pas de manipulation de score
  - Répond à la question : Ce document correspond-il à cette clause de requête ? Oui / Non

# Introduction à la recherche

---

## Requêtes





# Introduction à la recherche

---

## Exemple Leaf Query

```
GET /product/default/_search
{
  "query": {
    "match_all": {}
  }
}
```

# Introduction à la recherche

---

## Exemple Compound Query

```
POST /movies/default/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "auteur": "Stephen King" } },
        { "match": { "genre": "fantastique" } }
      ],
      "must_not": [
        { "match": { "titre": "Shining" } }
      ]
    }
  }
}
```

# TF-IDF : Déterminer un score de pertinence

---

- **TF-IDF** sont les acronymes de « **Terme Frequency** » et « **Inverse Document Frequency** ».
- On cherche à accorder une pertinence lexicale à un terme au sein d'un document.
- En ce qui concerne TF-IDF, on applique une relation entre un document, et un ensemble de documents partageant des similarités en matière de mots clés.
- On recherche en quelque sorte une relation de quantité / qualité lexicale à travers un ensemble de documents.
- Pour une requête avec un terme X, un document a plus de chances d'être pertinent comme réponse à la requête, si ce document possède une certaine occurrence de ce terme en son sein, et que ce terme possède une rareté dans d'autres documents reliés au premier.

# Formule mathématique

---

$$w_{x,y} = \text{tf}_{x,y} \times \log \left( \frac{N}{\text{df}_x} \right)$$

**TF-IDF**

Term  $x$  within document  $y$

$\text{tf}_{x,y}$  = frequency of  $x$  in  $y$

$\text{df}_x$  = number of documents containing  $x$

$N$  = total number of documents

# Explication de TF-IDF

---

- Score TF-IDF(que nous appelons par convention  $w$ )  $w = TF * IDF$ .
- **TF** = Nombre d'occurrences du terme au sein du document.
- Vous pouvez décomposer le document en lexie, et procéder cette opération :  
*Nombre d'occurrence du terme analysé / Nombre de termes total*
- **IDF** =  $\log(\text{Nombre total de documents} / \text{Nombre de documents contenant le terme analysé})$

## Exemple d'application concrète

---

- J'ai sur mon site un document de 100 lexies, avec une occurrence du mot **chat** de 3. On sait que  $TF=3/100$  donc **TF= 0.03**.
- Mon site a 10 millions de pages et le mot chat apparaît dans 1000 d'entre elles.
- On calcule donc  $IDF=\log(10\,000\,000 / 1000)$ . **IDF = 4**.
- Mon score **TF-IDF** est donc le résultat de la multiplication  **$0.03*4=0.12$** . Sur la requête chat, ce n'est pas la joie...
- Voir : Fichier Excel

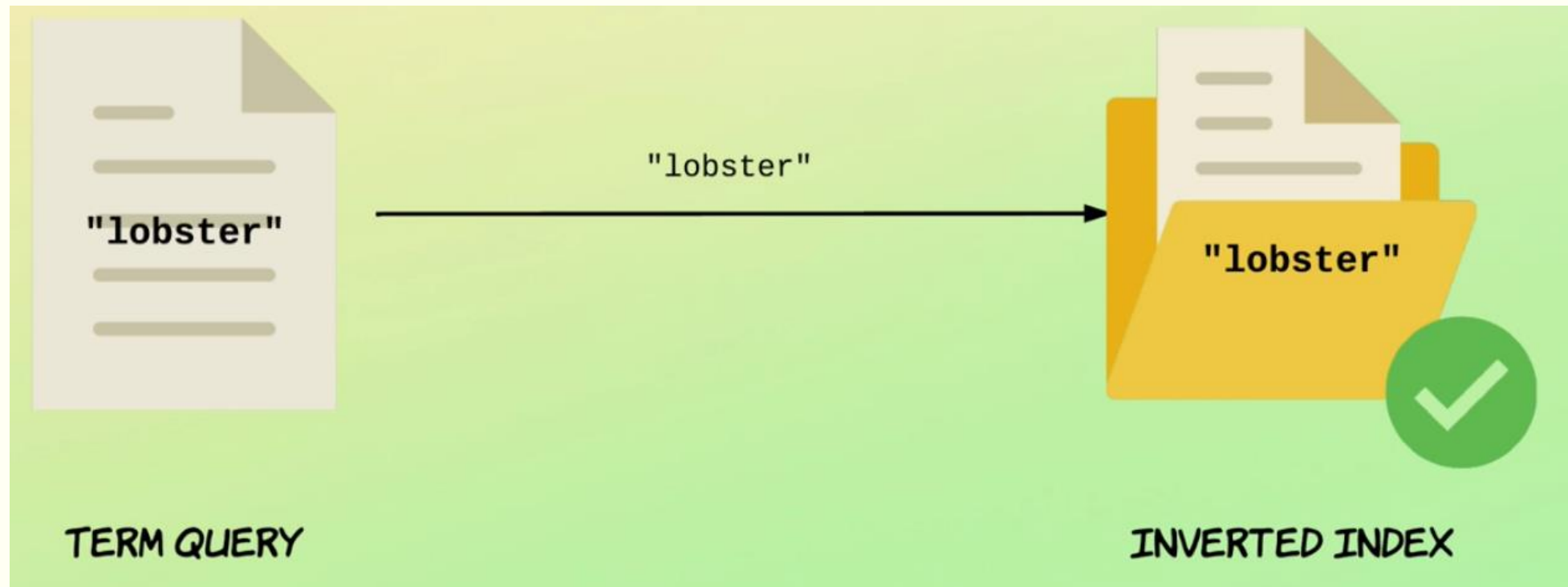
# Comment interpréter ce résultat

---

- Utiliser le fichier Excel fourni
- **Première expérience** : Si je passe à un nombre total de documents toujours plus grand (10 millions, 100 millions, 1000 millions...), mon score s'améliore à chaque augmentation. C'est bien évidemment l'inverse si je diminue le nombre de documents total.
  - La rareté d'un terme influe sur le score TF-IDF de manière non-négligeable, donc un terme plus rare, améliore la pertinence lexicale.
- **Deuxième expérience** : J'augmente l'occurrence du terme dans un document (TF). J'observe que le score final augmente également, tout comme dans la première expérience.
  - L'occurrence d'un terme influe donc grandement sur le score TF-IDF.

# Full text queries (term) vs term level queries (match)

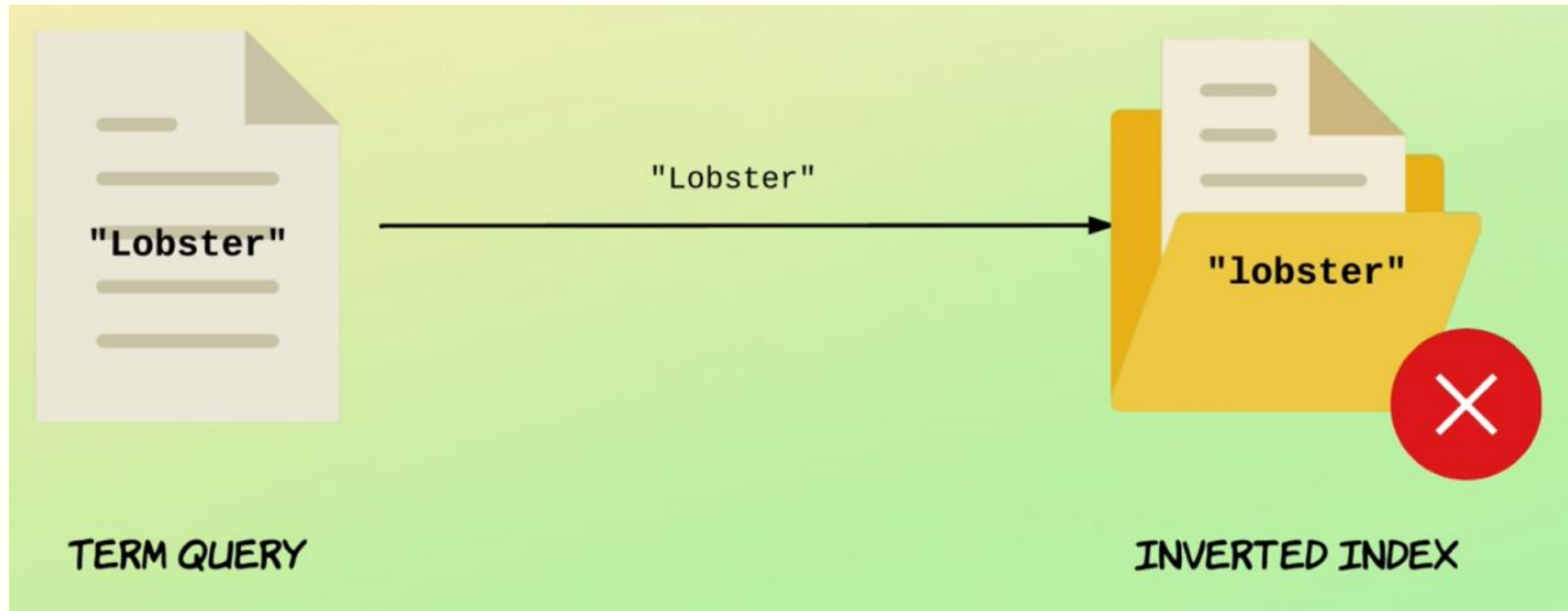
---





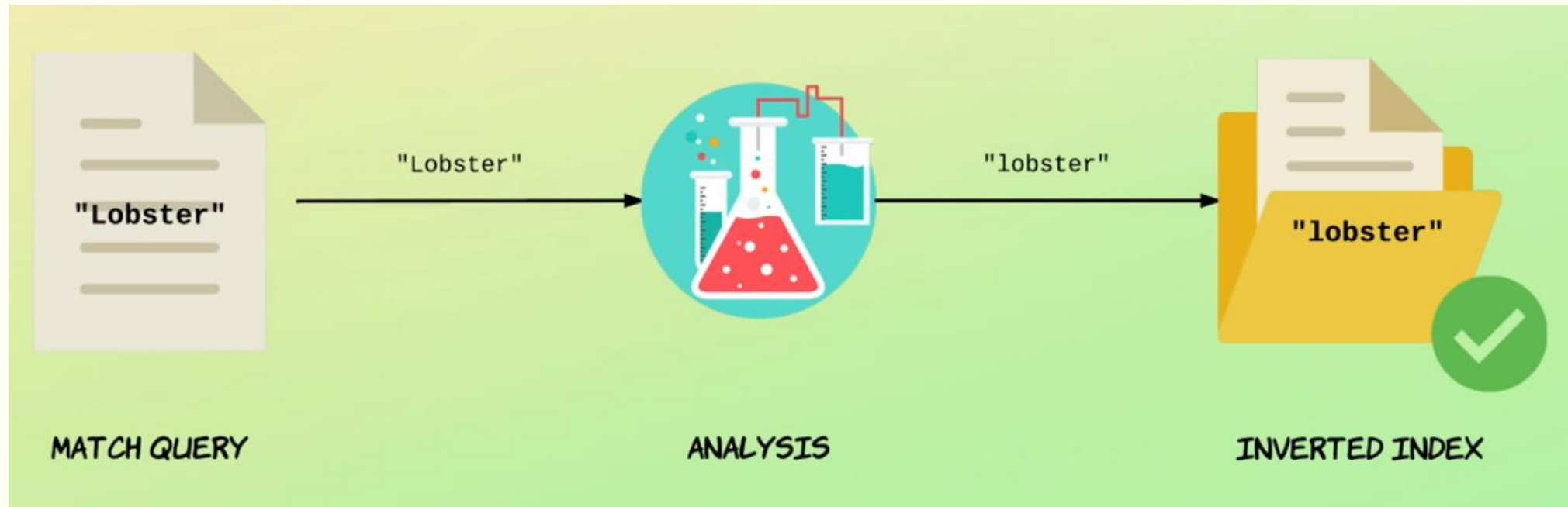
# Full text queries (term) vs term level queries (match)

---



# Full text queries (term) vs term level queries (match)

---



# Disposition de titre et de contenu avec liste

---

**Tuto 4**

**Intro-Recherche**



# MODULE 3

Manipulations de base

# Mapping et analyzers

---



# MODULE 4

Manipulations de base

# Term Level Queries

---

## Recherche par Term (Non analysé)

```
1 ## Recherche de document contenant le champ is_active=true
2 GET /product/default/_search
3 {
4   "query": {
5     "term": {
6       "is_active": true
7     }
8   }
9 }
10
```

## Recherche multiple de termes

```
1 GET /product/default/_search
2 {
3   "query": {
4     "terms": {
5       "tags.keyword": [ "Soup", "Cake" ]
6     }
7   }
8 }
```

# Term Level Queries

---

## Recherche basé sur les Ids

```
1 GET /product/default/_search
2 {
3   "query": {
4     "ids": {
5       "values": [ 1, 2, 3 ]
6     }
7   }
8 }
```

## Recherche de documents par intervalle

```
14 ## Match avec intervalle de temps (date)
15 GET /product/default/_search
16 {
17   "query": {
18     "range": {
19       "created": {
20         "gte": "2010/01/01",
21         "lte": "2010/12/31"
22       }
23     }
24   }
25 }
```



# Term Level Queries

---

## Traitement des valeurs non-null

```
1 GET /product/default/_search
2 {
3   "query": {
4     "exists": {
5       "field": "tags"
6     }
7   }
8 }
```

## Match avec préfixes

```
1 ## Matching documents containing a tag beginning with `Vege`
2 GET /product/default/_search
3 {
4   "query": {
5     "prefix": {
6       "tags.keyword": "Vege"
7     }
8   }
9 }
```

# Term Level Queries

---

## Recherche avec wildcards (\*, ?)

```
1 ## *(zero or more)
2 GET /product/default/_search
3 {
4   "query": {
5     "wildcard": {
6       "tags.keyword": "Veg*ble"
7     }
8   }
9 }
10
11 ## 1 single character
12 GET /product/default/_search
13 {
14   "query": {
15     "wildcard": {
16       "tags.keyword": "Veg?ble"
17     }
18   }
19 }
```

# Term Level Queries

---

## Reg-ex

```
1 GET /product/default/_search
2 {
3   "query": {
4     "regexp": {
5       "tags.keyword": "Veg[a-zA-Z]+ble"
6     }
7   }
8 }
```

# Disposition de titre et de contenu avec liste

---

## **Tuto 5**

### **Term Level Queries**

# Full Text Queries

---

## Match standard et opérateurs

```
1 ## Match standard
2 GET /recipe/default/_search
3 {
4   "query": {
5     "match": {
6       "title": "Recipes with pasta or spaghetti"
7     }
8   }
9 }
```

```
11 ## Match avec un opérateur booléen
12 GET /recipe/default/_search
13 {
14   "query": {
15     "match": {
16       "title": {
17         "query": "Recipes with pasta or spaghetti",
18         "operator": "and"
19       }
20     }
21   }
22 }
```

```
24 GET /recipe/default/_search
25 {
26   "query": {
27     "match": {
28       "title": {
29         "query": "pasta or spaghetti",
30         "operator": "and"
31       }
32     }
33   }
34 }
```

```
36 GET /recipe/default/_search
37 {
38   "query": {
39     "match": {
40       "title": {
41         "query": "pasta spaghetti",
42         "operator": "and"
43       }
44     }
45   }
46 }
```

# Match phrase

---

```
1 ## Ordres des mots importants
2 GET /recipe/default/_search
3 {
4   "query": {
5     "match_phrase": {
6       "title": "spaghetti puttanesca"
7     }
8   }
9 }
10
11 GET /recipe/default/_search
12 {
13   "query": {
14     "match_phrase": {
15       "title": "puttanesca spaghetti"
16     }
17   }
18 }
19
20
```

# Recherche champs multiple

---

```
1 GET /recipe/default/_search
2 {
3   "query": {
4     "multi_match": {
5       "query": "pasta",
6       "fields": [ "title", "description" ]
7     }
8   }
9 }
```

# Disposition de titre et de contenu avec liste

---

## **Tuto 6**

### **Full Text Queries**



# Logique booléenne et requête

---

# Logique booléenne et requête

---