



# 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)

# Lucene

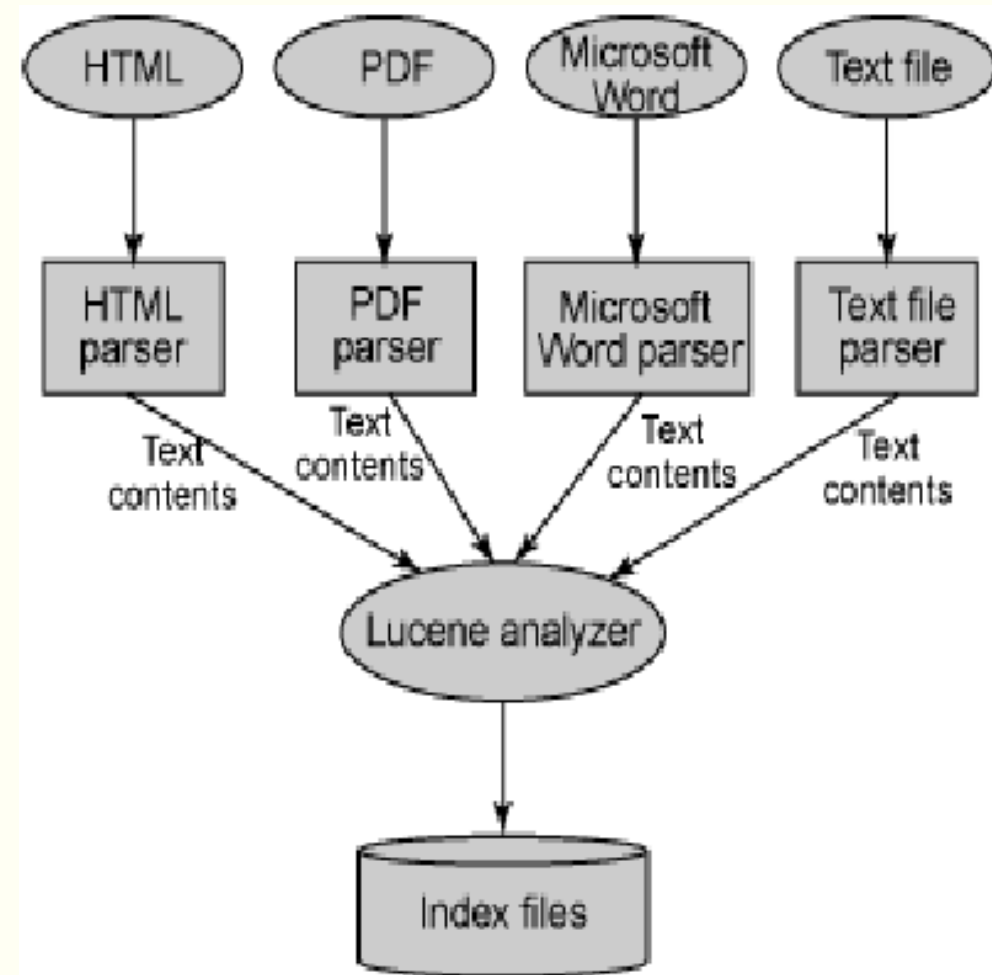
---

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



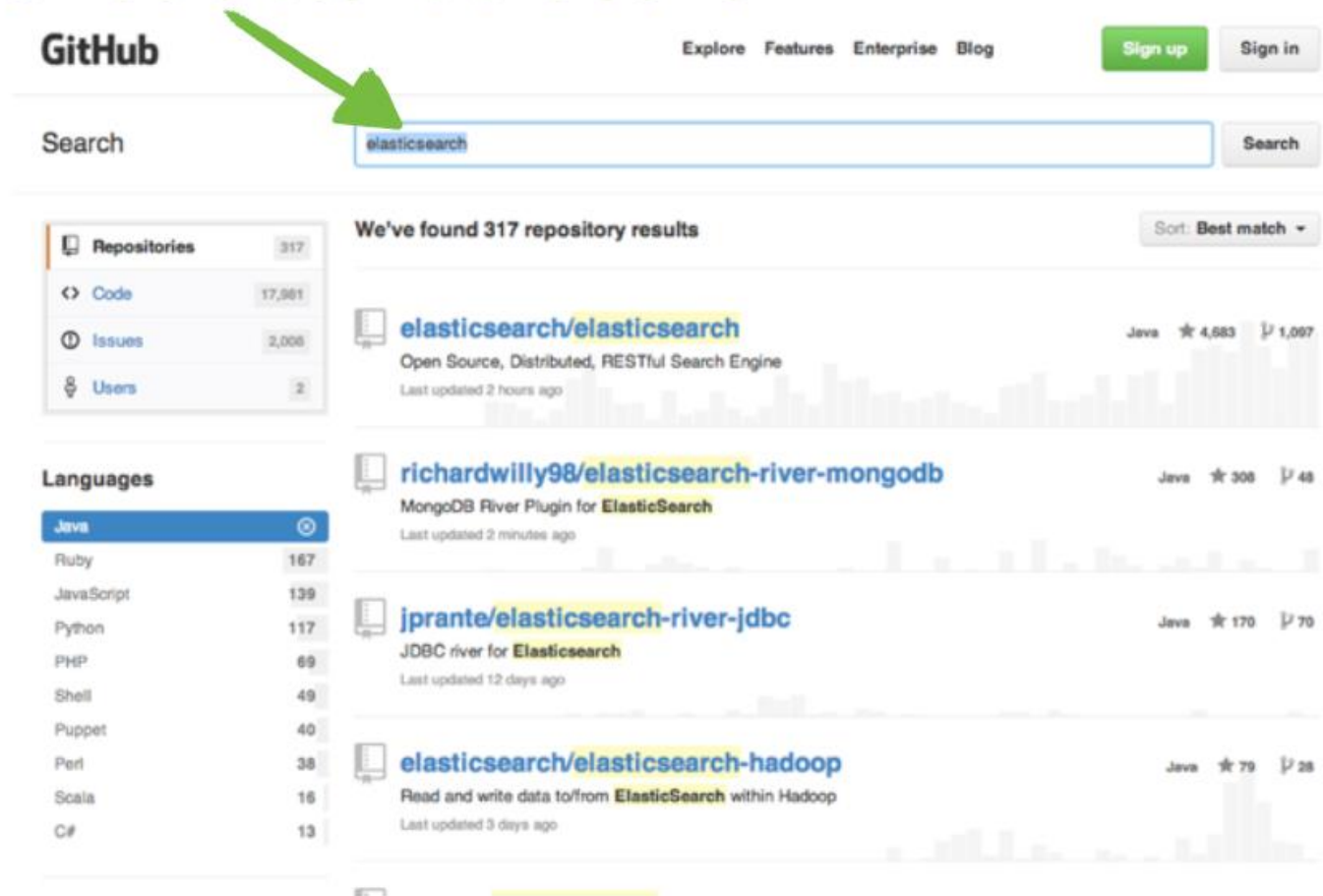
# Pour quels besoins ?

---

- 2 use cases principaux :
  - Recherche
  - Analyse
- 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

## Unstructured search



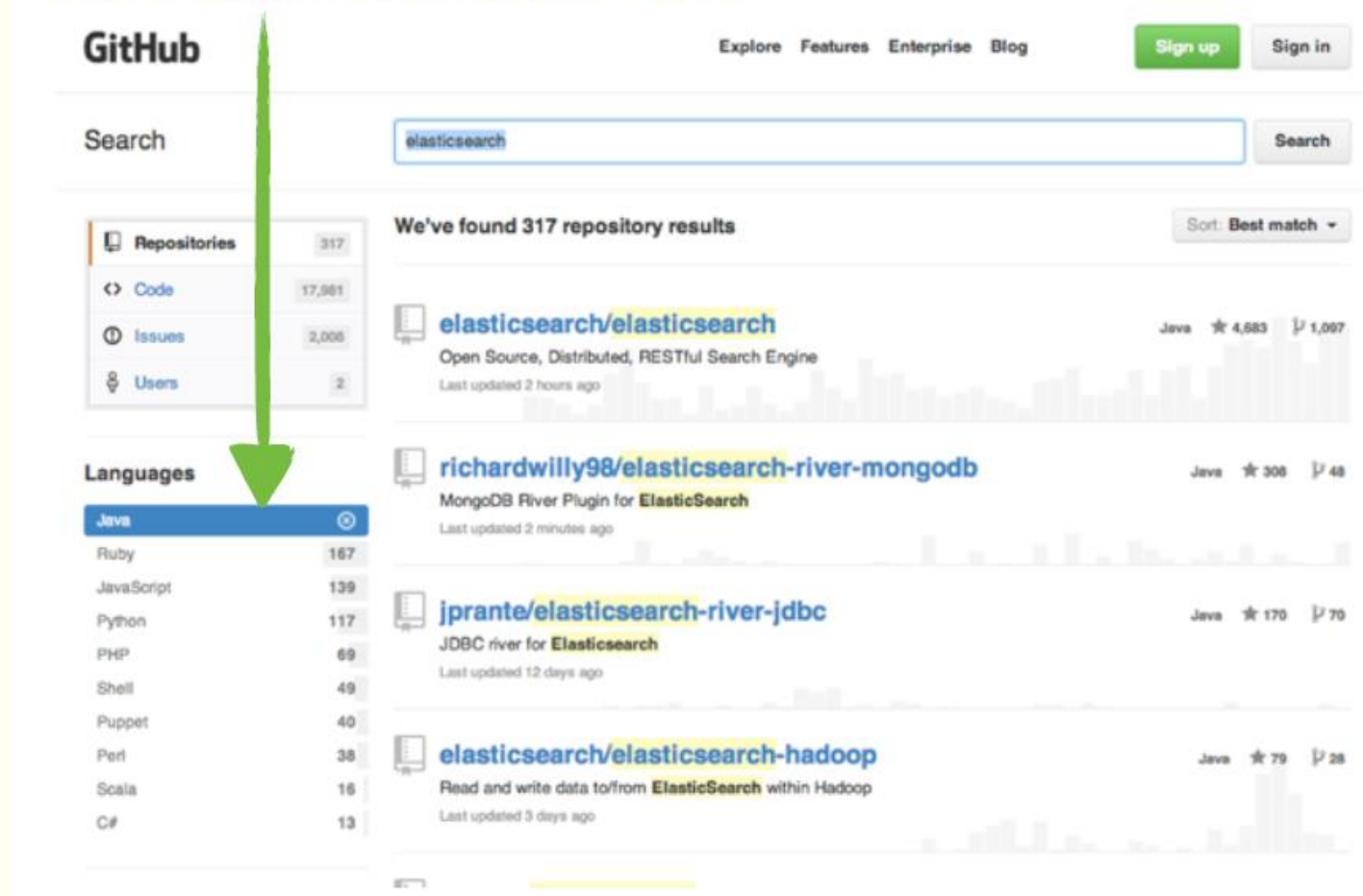
The screenshot shows the GitHub search interface. A green arrow points to the search bar where 'elasticsearch' is entered. The search results show 317 repository results. The top results are:

- elasticsearch/elasticsearch**: Open Source, Distributed, RESTful Search Engine. Last updated 2 hours ago. 4,583 stars, 1,097 forks.
- richardwilly98/elasticsearch-river-mongodb**: MongoDB River Plugin for ElasticSearch. Last updated 2 minutes ago. 308 stars, 48 forks.
- jprante/elasticsearch-river-jdbc**: JDBC river for Elasticsearch. Last updated 12 days ago. 170 stars, 70 forks.
- elasticsearch/elasticsearch-hadoop**: Read and write data to/from ElasticSearch within Hadoop. Last updated 3 days ago. 79 stars, 28 forks.

On the left sidebar, the 'Repositories' section shows 317 results, and the 'Languages' section shows a list of languages with counts: Java (167), Ruby (139), JavaScript (117), Python (69), PHP (49), Shell (40), Puppet (38), Perl (16), Scala (13), and C# (13).

# Exemple

## Structured search



The screenshot shows the GitHub search interface. A green arrow points from the 'Languages' filter section to the 'Java' language option. The search results are sorted by 'Best match' and show 317 repository results. The top results are:

- elasticsearch/elasticsearch**: Open Source, Distributed, RESTful Search Engine. Last updated 2 hours ago. 4,583 stars, 1,097 forks.
- richardwilly98/elasticsearch-river-mongodb**: MongoDB River Plugin for ElasticSearch. Last updated 2 minutes ago. 308 stars, 48 forks.
- jprante/elasticsearch-river-jdbc**: JDBC river for Elasticsearch. Last updated 12 days ago. 170 stars, 70 forks.
- elasticsearch/elasticsearch-hadoop**: Read and write data to/from ElasticSearch within Hadoop. Last updated 3 days ago. 79 stars, 28 forks.

The 'Languages' filter on the left shows the following counts:

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



# Exemple

## Enrichment

The screenshot shows the GitHub search interface. A green arrow points from the word 'Enrichment' to the search bar, which contains the text 'elasticsearch'. Another green arrow points from the search bar to the first search result, 'elasticsearch/elasticsearch'. The search results are sorted by 'Best match' and show four repositories. The first repository, 'elasticsearch/elasticsearch', is highlighted with a yellow background. The second repository, 'richardwilly98/elasticsearch-river-mongodb', is also highlighted with a yellow background. The third repository, 'jprante/elasticsearch-river-jdbc', is highlighted with a yellow background. The fourth repository, 'elasticsearch/elasticsearch-hadoop', is highlighted with a yellow background. The left sidebar shows the 'Repositories' section with 317 results, and the 'Languages' section with a list of languages and their counts.

GitHub

Explore Features Enterprise Blog Sign up Sign in

Search

elasticsearch Search

We've found 317 repository results Sort: Best match

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

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

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

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

**Repositories** 317

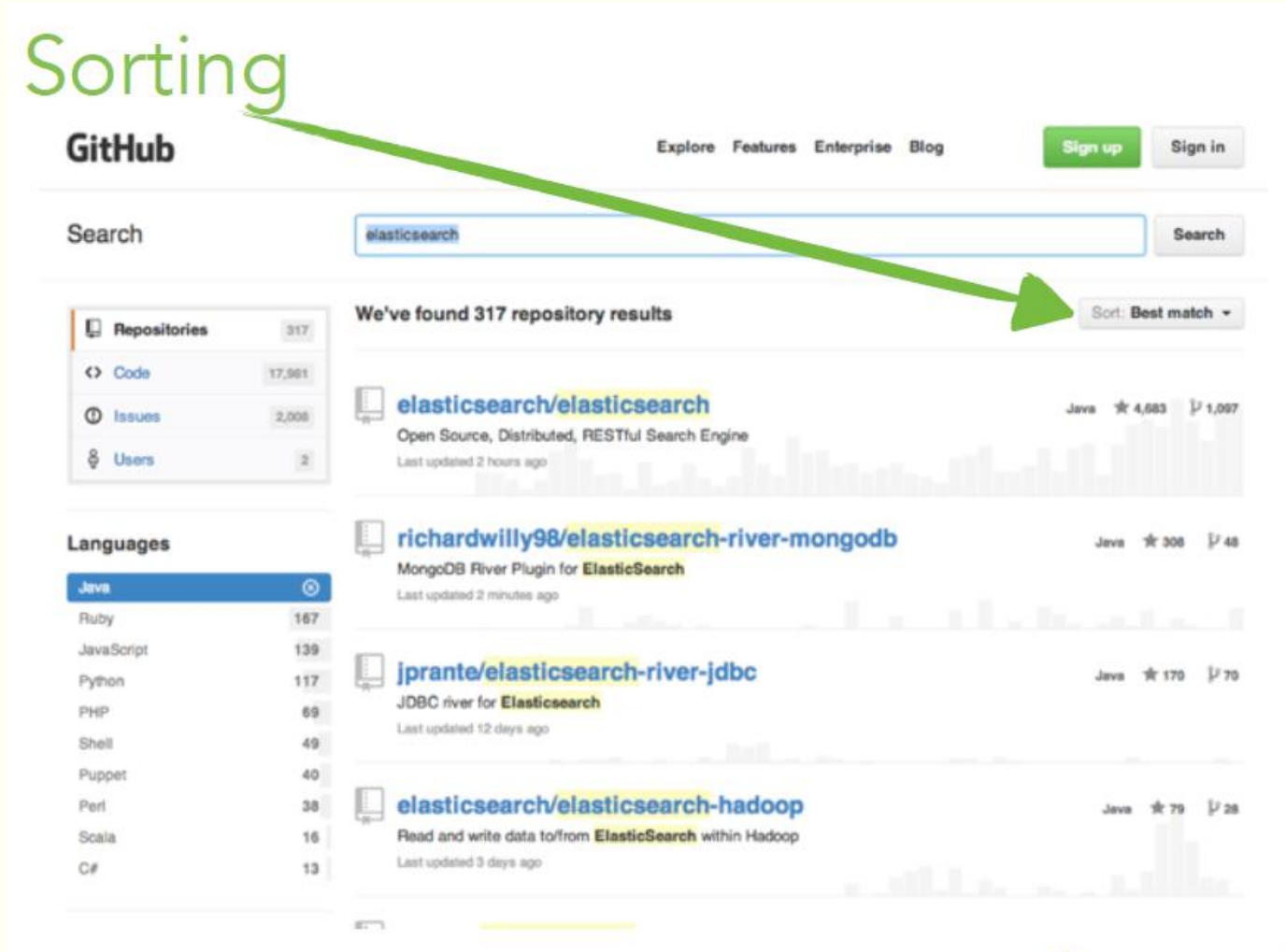
- Code 17,981
- Issues 2,008
- Users 2

**Languages**

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

# Exemple

## Sorting



The screenshot shows the GitHub search interface. A green arrow points from the word 'Sorting' to the 'Sort: Best match' dropdown menu. The search results are sorted by 'Best match'.

**Search**  **Search**

We've found 317 repository results

**Sort: Best match**

Repository	Stars	Forks
<a href="#">elasticsearch/elasticsearch</a> Open Source, Distributed, RESTful Search Engine Last updated 2 hours ago	4,683	1,097
<a href="#">richardwilly98/elasticsearch-river-mongodb</a> MongoDB River Plugin for ElasticSearch Last updated 2 minutes ago	308	48
<a href="#">jprante/elasticsearch-river-jdbc</a> JDBC river for Elasticsearch Last updated 12 days ago	170	70
<a href="#">elasticsearch/elasticsearch-hadoop</a> Read and write data to/from ElasticSearch within Hadoop Last updated 3 days ago	79	28

**Repositories** 317

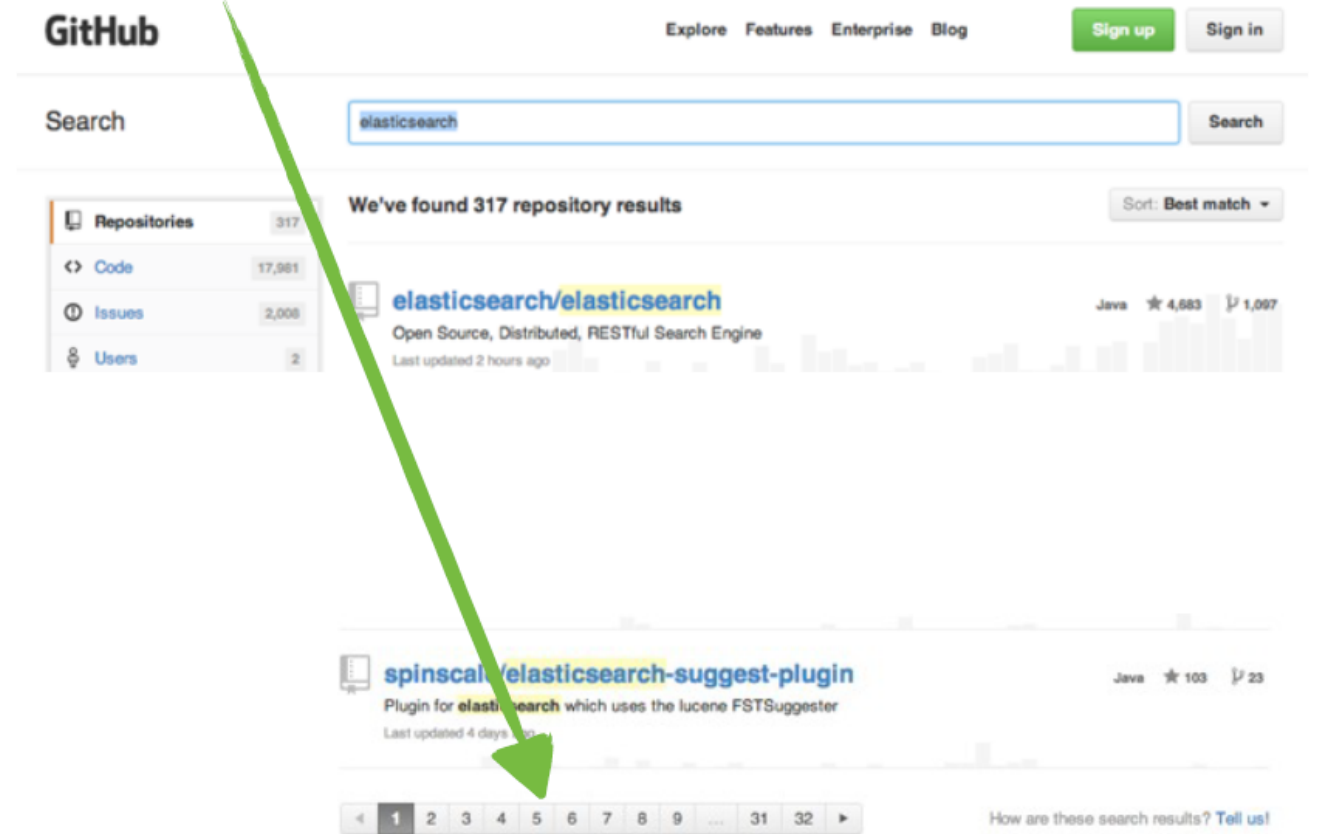
- [Code](#) 17,961
- [Issues](#) 2,008
- [Users](#) 2

**Languages**

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

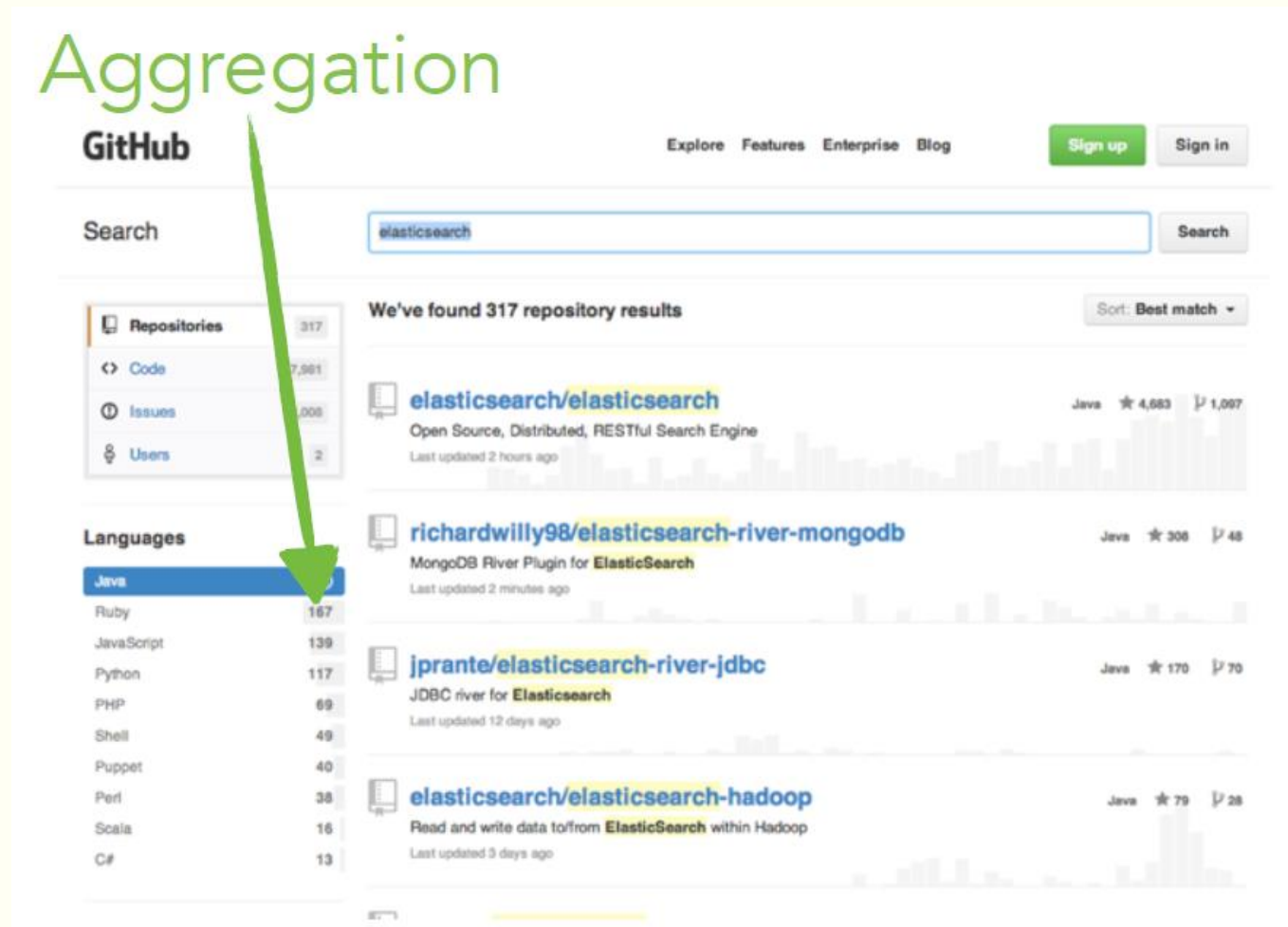
# Exemple

## Pagination

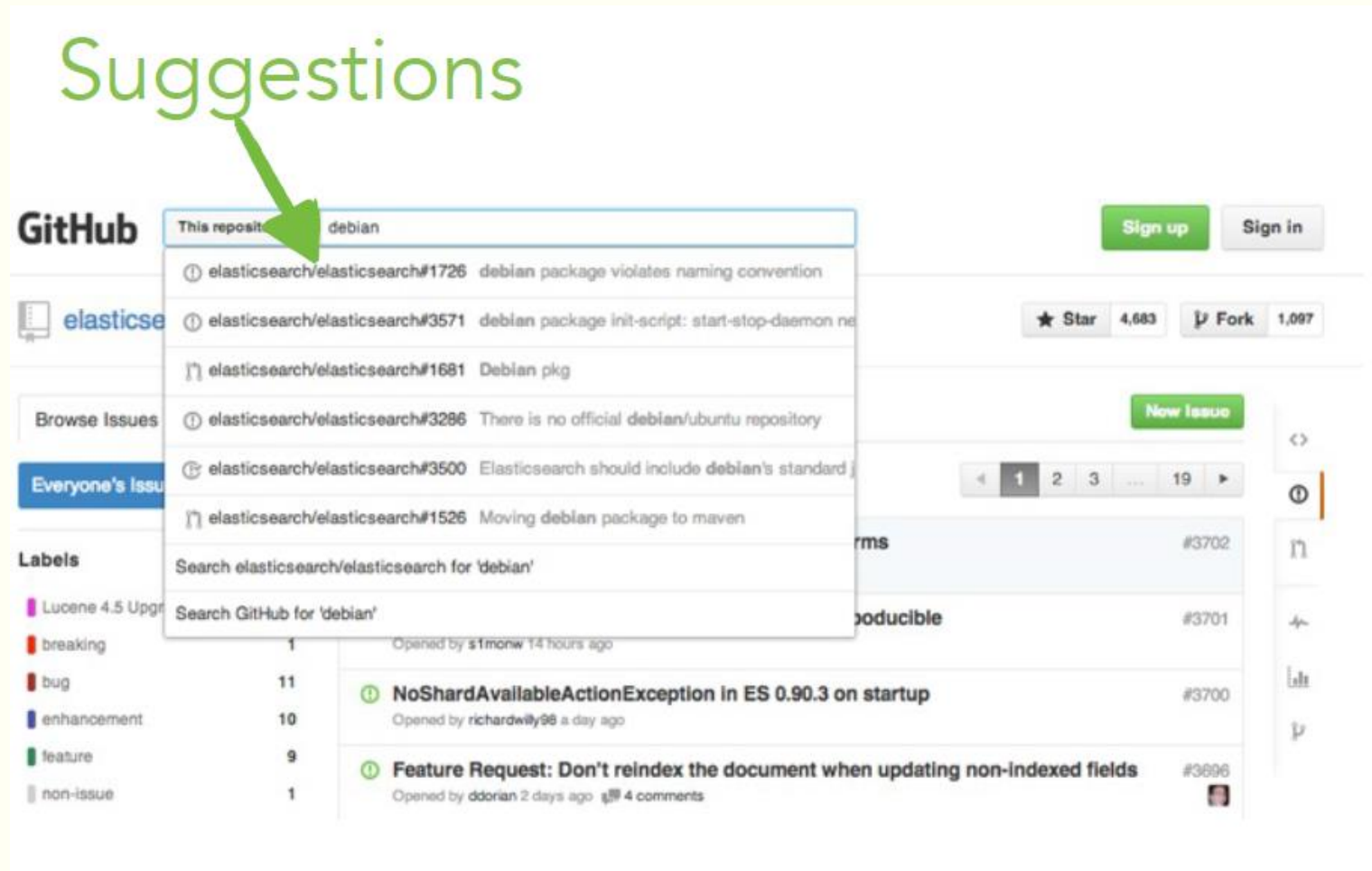


# Exemple

## Aggregation



# Exemple



# 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 ?

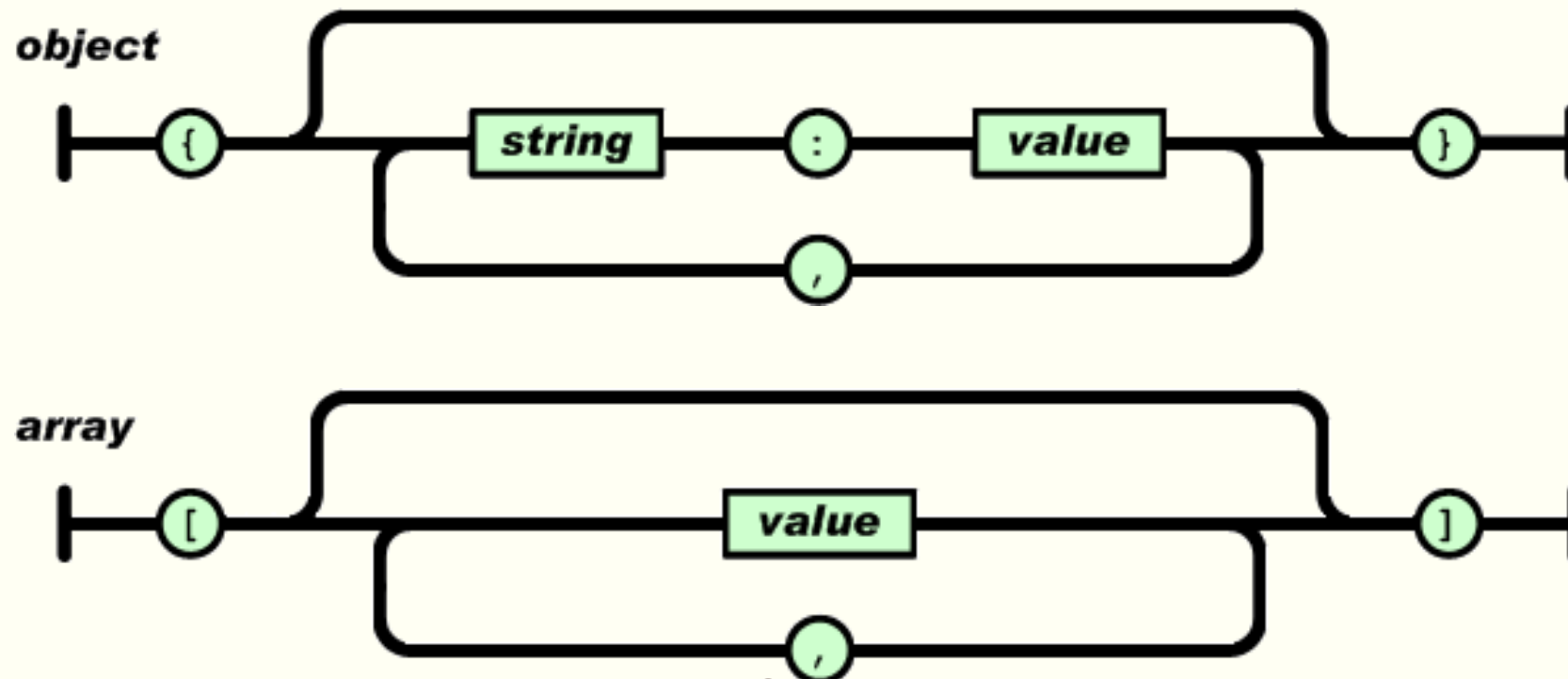
---

- 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.

# Format JSON

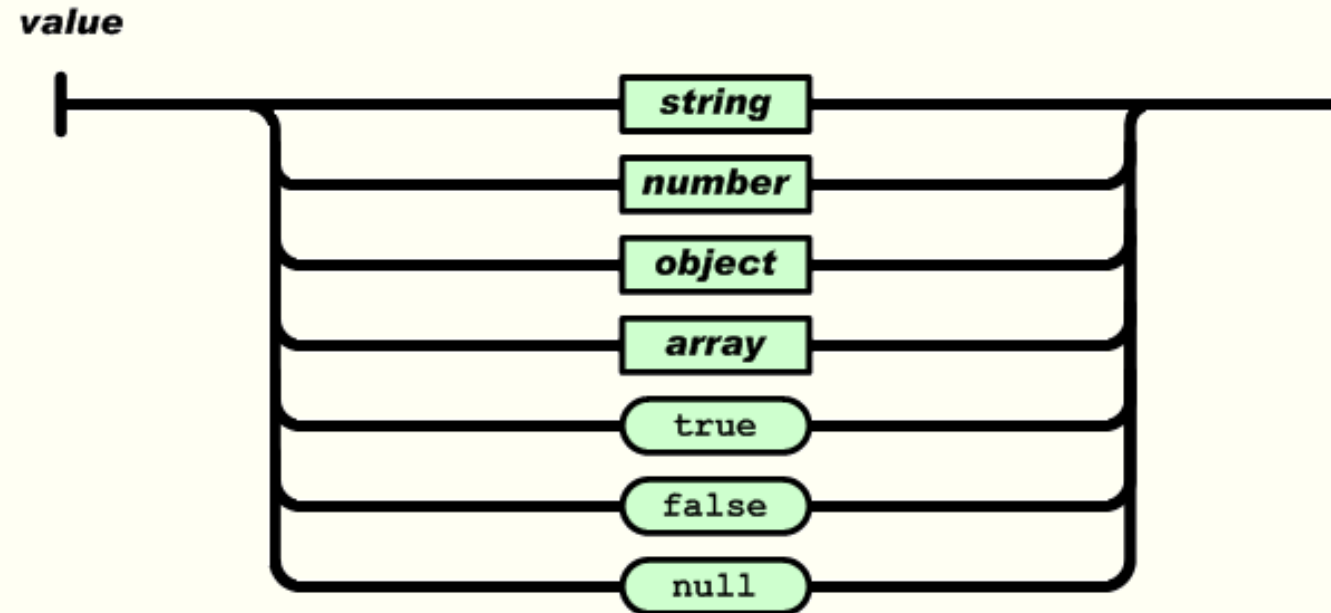
---

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



# Value JSON?

---



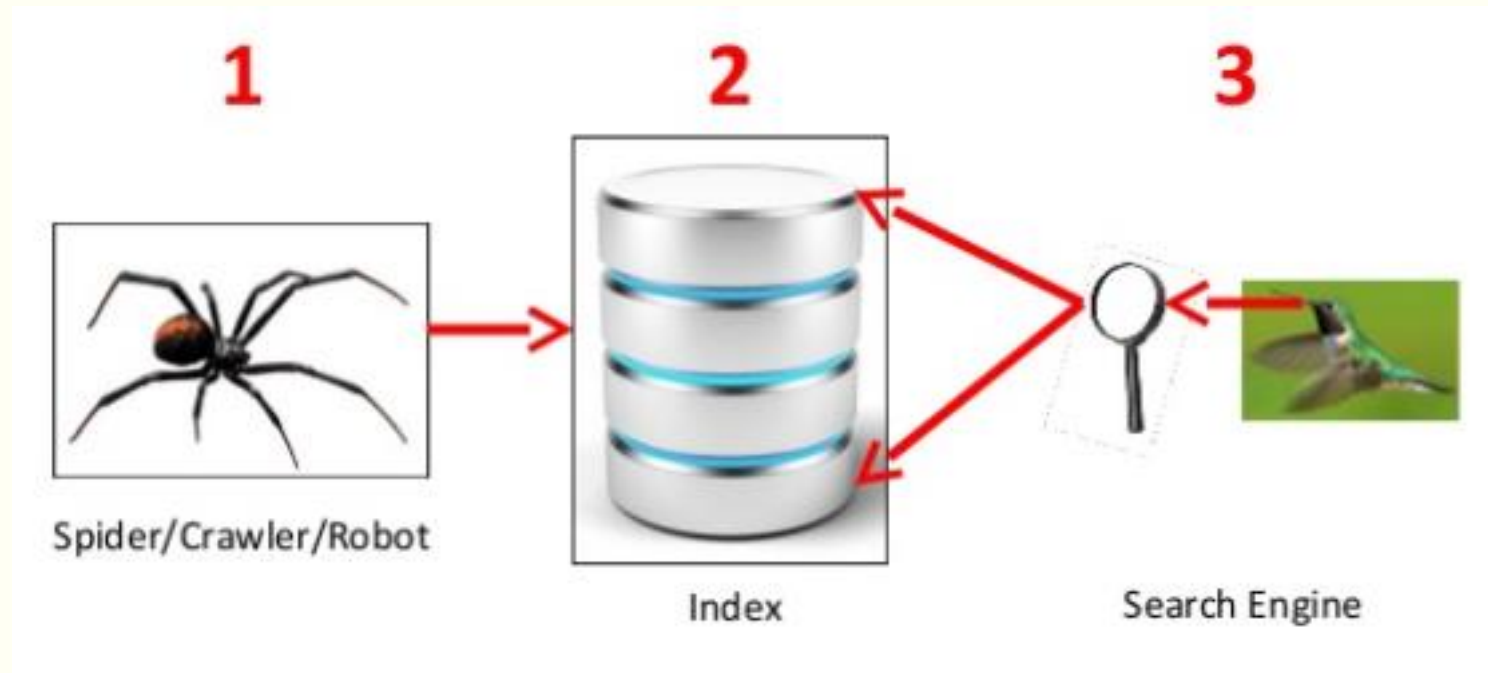
```
{ "firstName":"John" , "lastName":"Doe" }
```

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

# Composition d'un moteur de recherche?

---

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



# 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

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

# 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

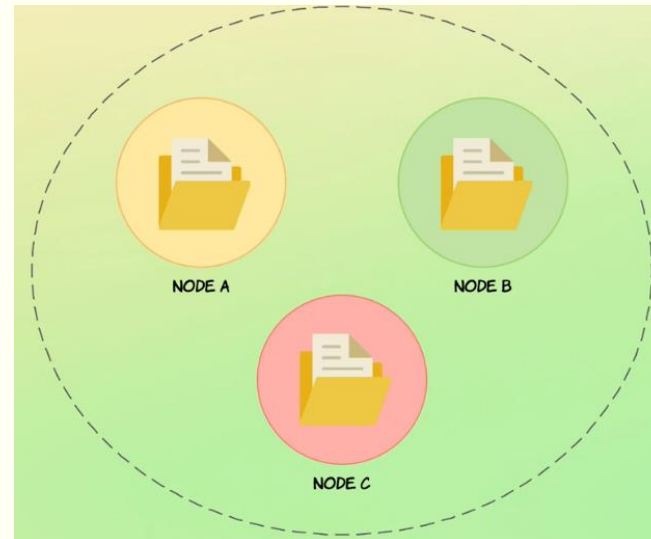
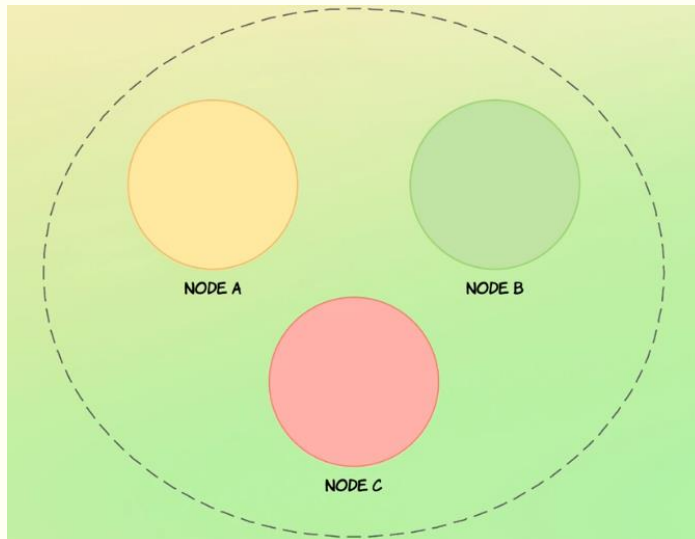
---

## 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.



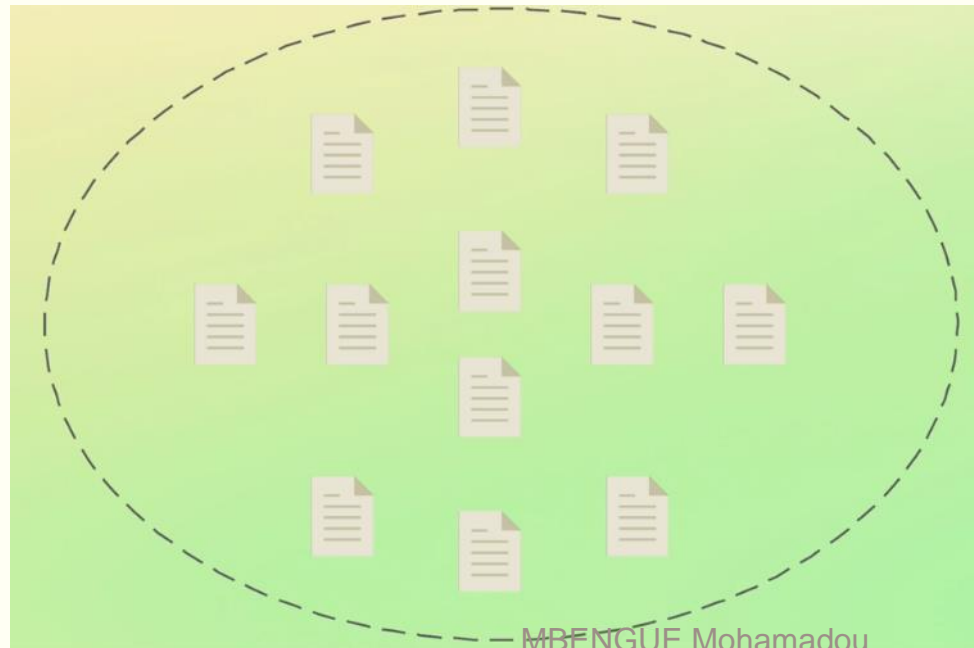
MBENGUE Mohamadou  
Chaque nœud contient une partie des données

# 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.

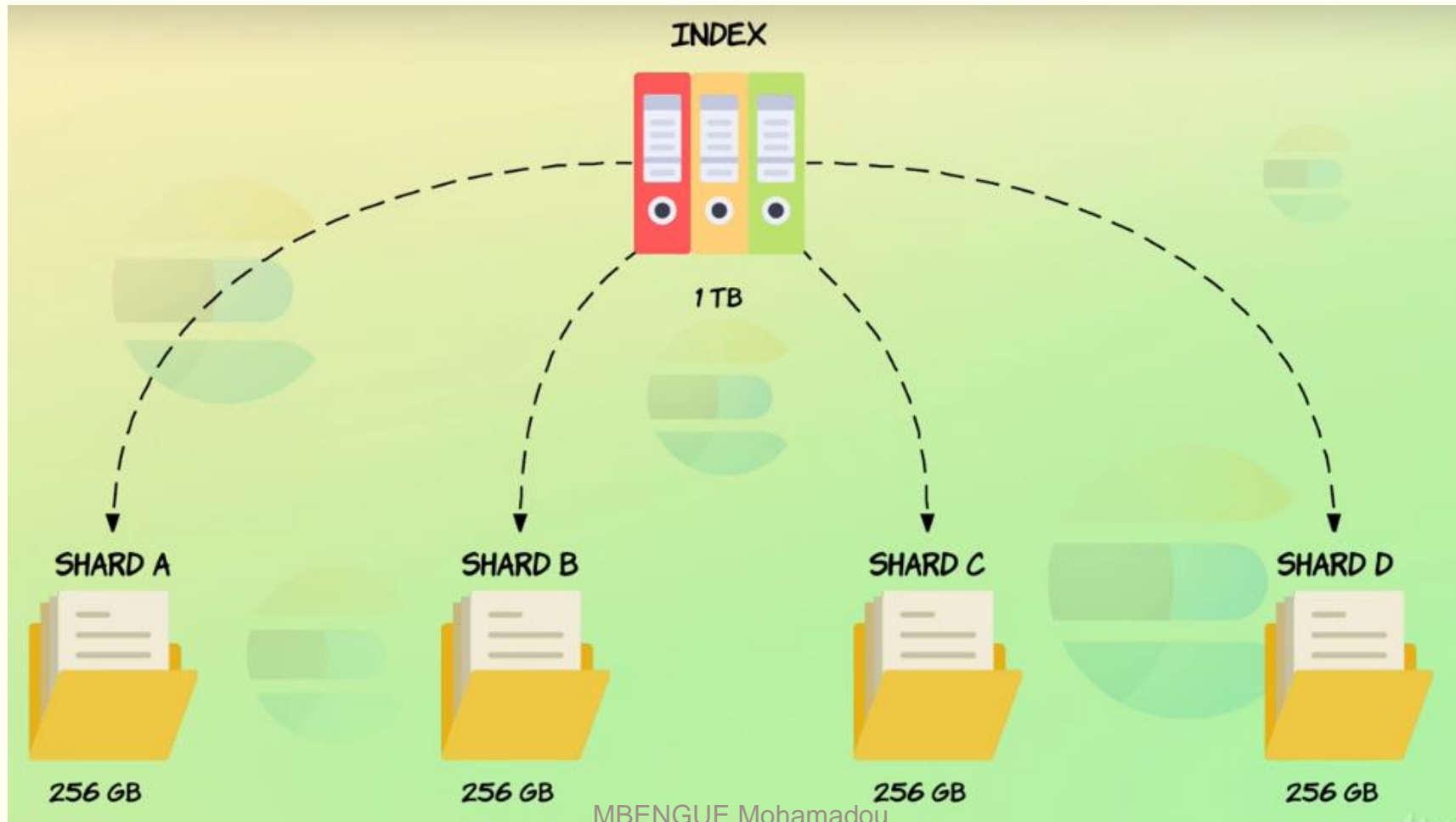
## Secondary Shards

- 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

---

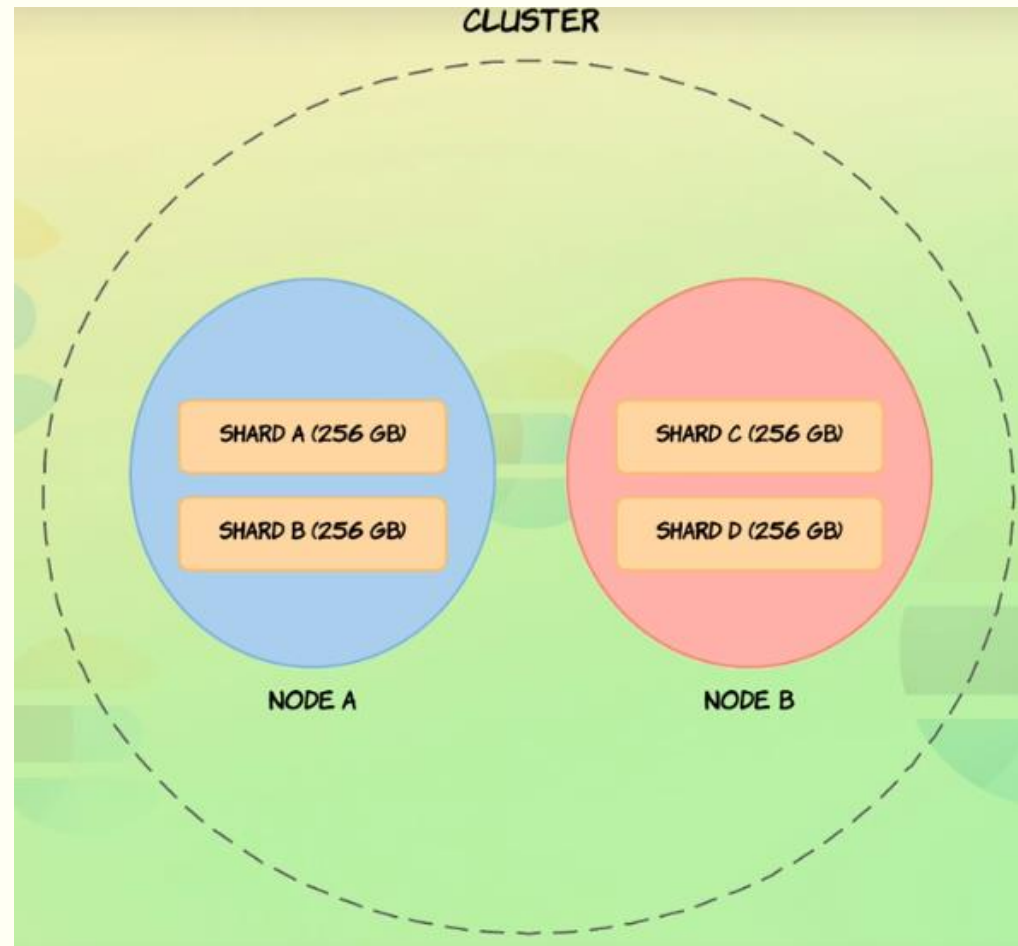
## Sharding



# Architecture générale

---

## Sharding vs distribution



# Mise en place d'ElasticSearch

---

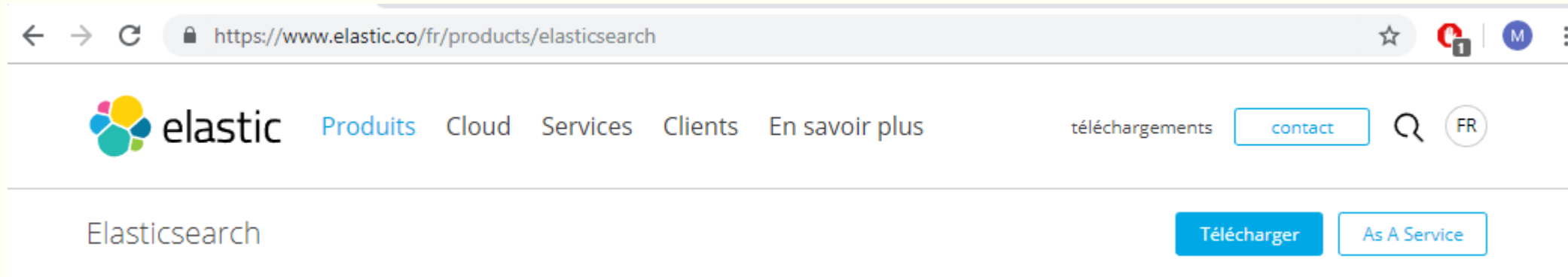
- Elasticsearch nécessite que Java soit installé sur votre machine
- Le jdk 1.8 est recommandé.
- Pour vérifier si Java 8 est installé, il suffit d'exécuter la commande suivante dans un shell \*nix ou une invite de commande Windows

```
C:\Users\mbeng>java -version
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13, mixed mode)
```

# Mise en place d'ElasticSearch

---

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



- 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
- **plugins**, où se trouveront tous les plugins que vous installerez
- **logs** : qui contient les fichiers de journalisation
- ...

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

- **data** : destiné à contenir les données indexées

Nom	Modifié le	Type	Taille
bin	29/04/2019 13:05	Dossier de fichiers	
config	29/04/2019 13:05	Dossier de fichiers	
jdk	29/04/2019 13:05	Dossier de fichiers	
lib	29/04/2019 13:05	Dossier de fichiers	
logs	29/04/2019 12:58	Dossier de fichiers	
modules	29/04/2019 13:05	Dossier de fichiers	
plugins	29/04/2019 12:58	Dossier de fichiers	
LICENSE.txt	29/04/2019 12:50	Document texte	14 Ko
NOTICE.txt	29/04/2019 12:58	Document texte	437 Ko
README.textile	29/04/2019 12:50	Fichier TEXTILE	9 Ko

# 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.



```
{
  "name" : "MM-KING",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "cDGX3-GlQQiGyCQwXxFrGg",
  "version" : {
    "number" : "7.0.1",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "e4efcb5",
    "build_date" : "2019-04-29T12:56:03.145736Z",
    "build_snapshot" : false,
    "lucene_version" : "8.0.0",
    "minimum_wire_compatibility_version" : "6.7.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

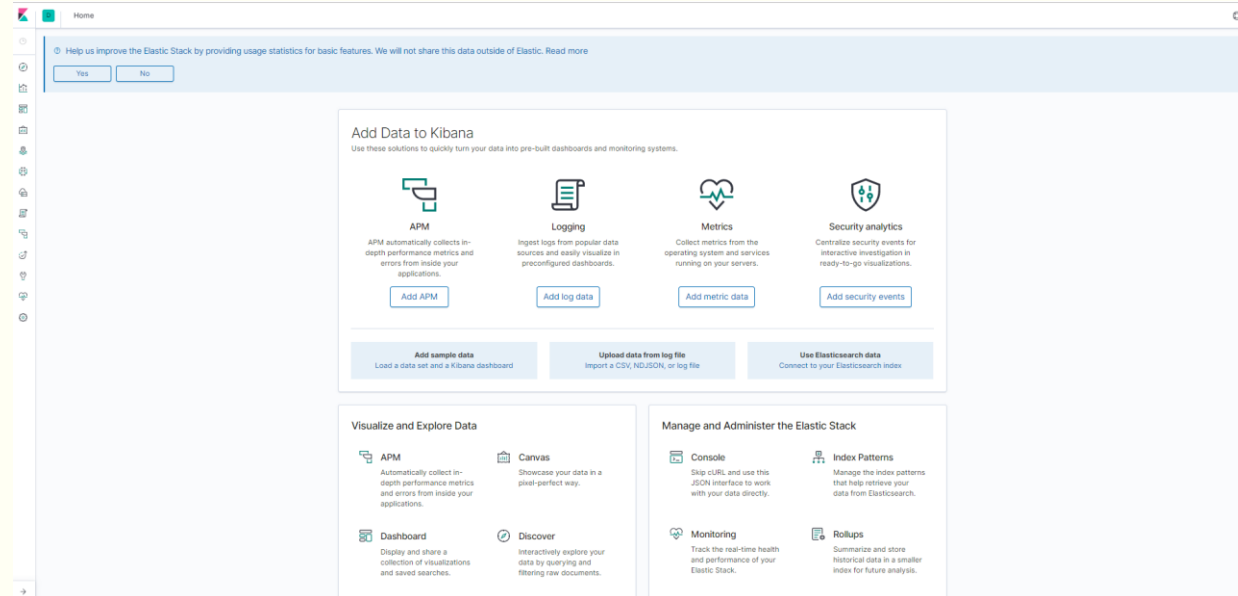
# Installation de Kibana

---

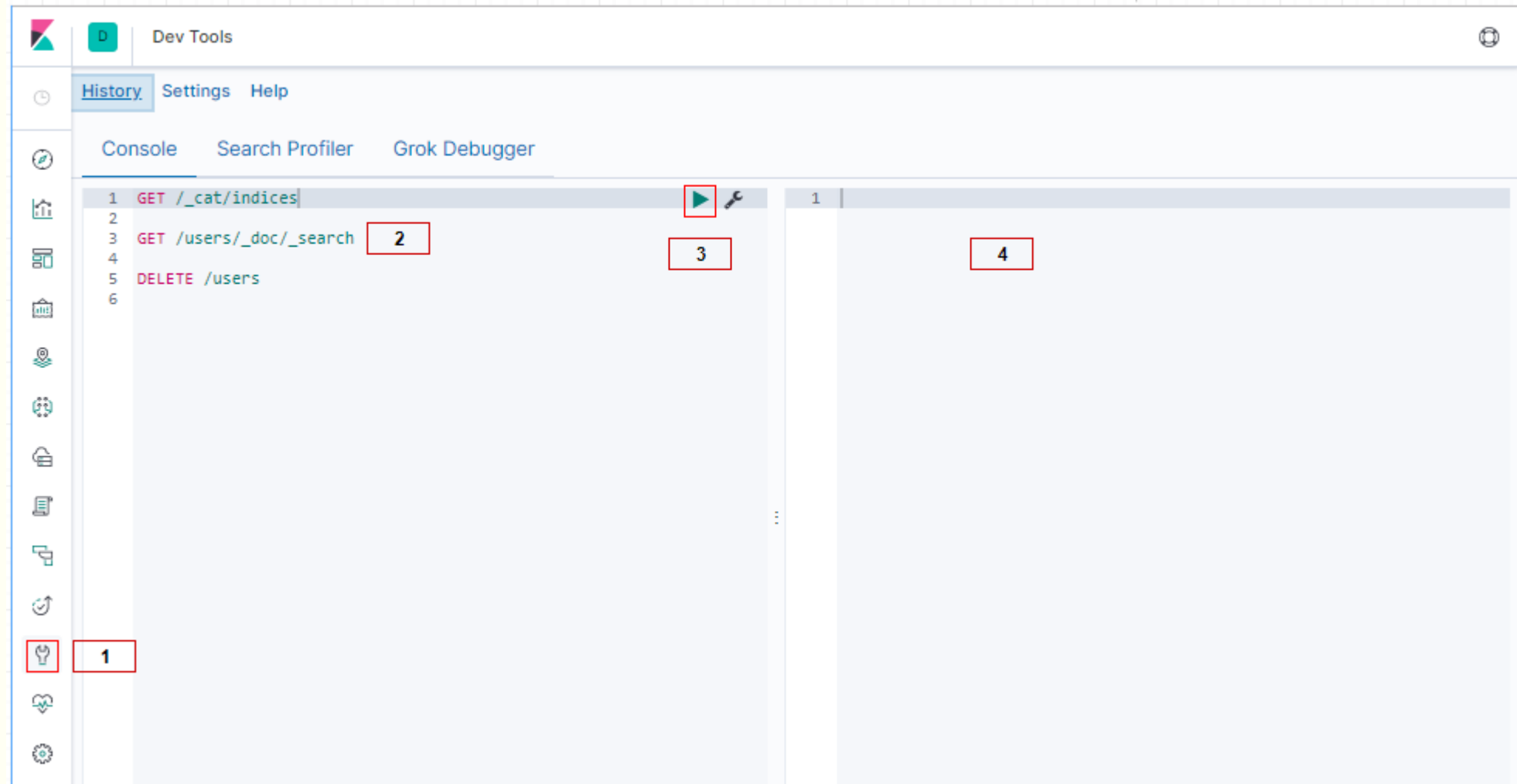
## ■ Installation de Kibana

- Lien : <https://www.elastic.co/fr/products/kibana>
- Dézippez l'archive
- Lancer : bin/[kibana.bat](#)

- Ouvrir l'URL <http://localhost:5601> dans un navigateur Web



# Mise en route - Kibana



# Disposition de titre et de contenu avec liste

---

**Question ?**

# Disposition de titre et de contenu avec liste

---

**Lab\_0**

**Installation**

# CURL

---

**curl -X <VERBE> '<PROTOCOL>: // <HÔTE>: <PORT>/<PATH>?<QUERY\_STRING>' -d '<CORPS>'**

Où:

- **VERBE**: méthode ou verbe HTTP approprié: GET, POST, PUT, HEAD ou DELETE
- **PROTOCOLE**: http ou https (si vous avez un proxy https devant Elasticsearch.)
- **HOST**: nom d'hôte de n'importe quel nœud de votre cluster Elasticsearch ou localhost pour un nœud sur votre ordinateur local.
- **PORT**: port exécutant le service HTTP Elasticsearch, dont la valeur par défaut est 9200.
- **PATH**: API Endpoint (par exemple, `_count` renverra le nombre de documents dans le cluster). Le chemin peut contenir plusieurs composants, tels que `_cluster / stats` ou `_nodes / stats / jvm`
- **QUERY\_STRING**: Tous les paramètres de chaîne de requête facultatifs (par exemple, "pretty" impriment la réponse JSON pour la rendre plus lisible).
- **BODY**: Un corps de requête encodé en JSON (si la requête en a besoin.)

# Créer un index

---

- Exemple

```
curl -XPUT http://localhost:9200/my-index?pretty
```

- Sortie

```
{  
  "acknowledged" : true,  
  "shards_acknowledged" : true,  
  "index" : "my-index"  
}
```



# Liste tous les documents d'un index

---

- Exemple

```
curl -XGET http://localhost:9200/my-index/_search?pretty=true
```

Cela utilise l'API de **Search** et renverra toutes les entrées sous l'index **my-index**

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 0,
      "relation" : "eq"
    },
    "max_score" : null,
    "hits" : [ ]
  }
}
```

# Liste tous les indices

---

- Exemple

```
curl -XGET http://localhost:9200/_cat/indices?v
```

- Sortie

health	status	index	uuid	pri	rep	docs.count	docs.deleted	store.size	pri.store.size
yellow	open	my-index	QdDohhc9QzySjjRK6upvBQ	1	1	0	0	283b	283b
green	open	.kibana_1	oJp-1fpCQUad3WgOYYo2wA	1	0	4	0	17.6kb	17.6kb
yellow	open	order	G7FpnUEdRWqvFQRk7ovA0g	1	1	1000	0	224kb	224kb
yellow	open	recipe	Nor-1Gpjr7CfvJDnhJ1zAw	1	1	21	0	80.2kb	80.2kb
yellow	open	product	nHEExoE2S4GaZ_4bKpqqBQ	1	1	1000	0	385.7kb	385.7kb
green	open	.kibana_task_manager	sBIc0SRDS3WtDCgGClq1gw	1	0	2	0	45.5kb	45.5kb

# Récupérer un document par identifiant

---

- Exemple

```
curl -XGET http://localhost:9200/product/_doc/1
```

```
curl -XGET http://localhost:9200/product/_doc/1?pretty
```

```
{
  "_index" : "product",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "name" : "Wine - Maipo Valle Cabernet",
    "price" : 152,
    "in_stock" : 38,
    "sold" : 47,
    "tags" : [
      "Alcohol",
      "Wine"
    ],
    "description" : "Aliquam augue quam, sollicitudin vitae, consectetuer eget, rutrum at, lorem. Integer tincidunt ante vel ipsum. Praesent blandit lacinia erat. Vestibulum sed magna at nunc commodo placerat. Praesent blandit. Nam nulla. Integer pede justo, lacinia eget, tincidunt eget, tempus vel, pede. Morbi porttitor lorem id ligula.",
    "is_active" : true,
    "created" : "2004/05/13"
  }
}
```

# Supprimer un index

---

- Exemple

```
curl -XDELETE http://localhost:9200/my-index
```

- Sortie

```
{"acknowledged":true}
```

# Disposition de titre et de contenu avec liste

---

**Question ?**

# Disposition de titre et de contenu avec liste

---

**Lab\_1**

**Curl**

**Lab\_2**

**Overview**



# 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 :

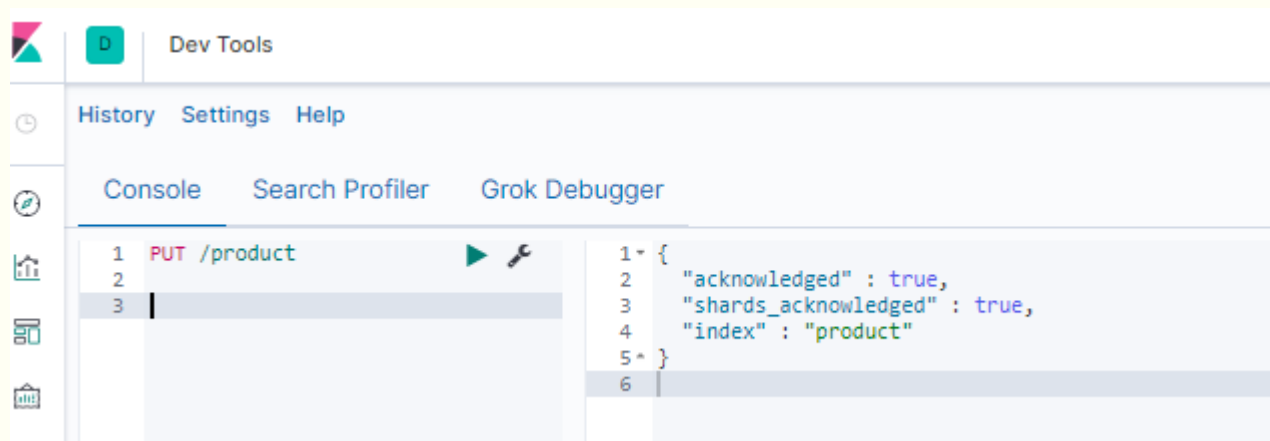
`http://host:port/[index]/_doc/[_action|id] -d document`

- **index** : nom de l'index sur lequel porte l'opération
- **type** : `_doc`
- **\_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 – 1

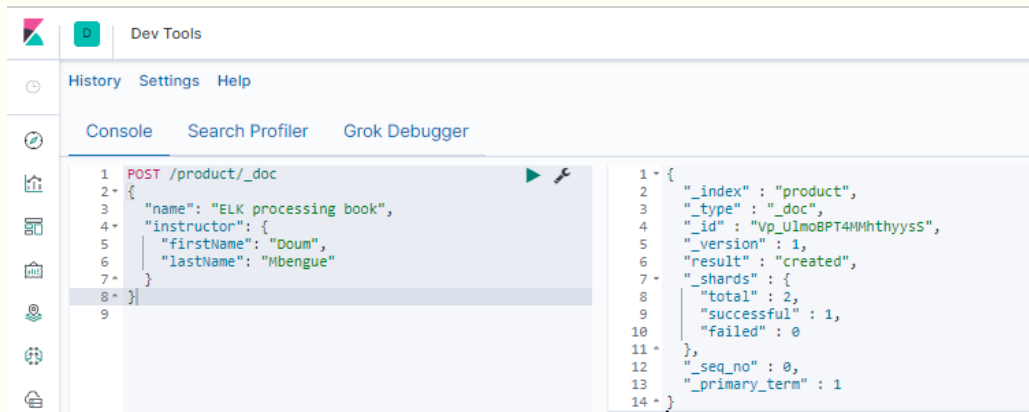
---

## ▪ Création d'un index



# Exemple - 2

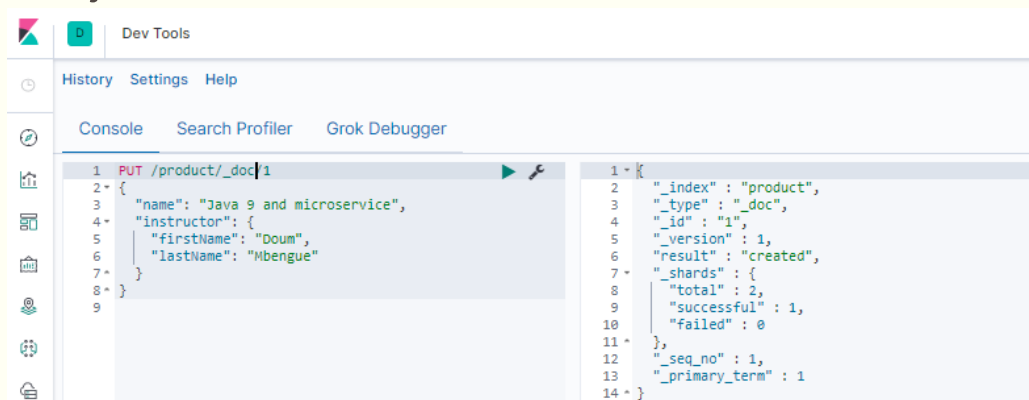
- Ajout de document avec ID auto-généré:



```
1 POST /product/_doc
2 {
3   "name": "ELK processing book",
4   "instructor": {
5     "firstName": "Doun",
6     "lastName": "Mbengue"
7   }
8 }
9
```

```
1 {
2   "_index": "product",
3   "_type": "_doc",
4   "_id": "Vp_UlmoBPT4MMhthyys",
5   "_version": 1,
6   "result": "created",
7   "_shards": {
8     "total": 2,
9     "successful": 1,
10    "failed": 0
11  },
12   "_seq_no": 0,
13   "_primary_term": 1
14 }
```

- Ajout de document avec ID



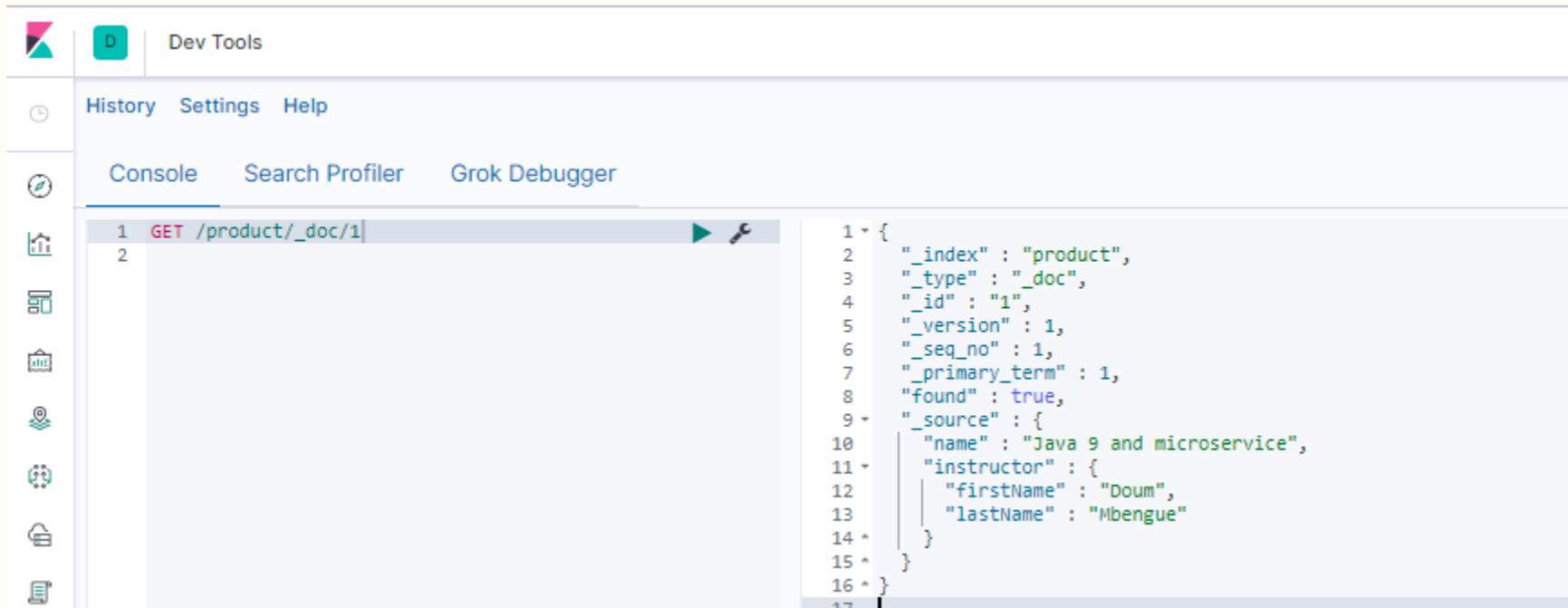
```
1 PUT /product/_doc/1
2 {
3   "name": "Java 9 and microservice",
4   "instructor": {
5     "firstName": "Doun",
6     "lastName": "Mbengue"
7   }
8 }
9
```

```
1 {
2   "_index": "product",
3   "_type": "doc",
4   "_id": "1",
5   "_version": 1,
6   "result": "created",
7   "_shards": {
8     "total": 2,
9     "successful": 1,
10    "failed": 0
11  },
12   "_seq_no": 1,
13   "_primary_term": 1
14 }
```

# Exemple - 3

---

- Récupérer un document



# Exemple - 4

---

- Remplacer un document existant



The screenshot shows a REST client interface with a 'Dev Tools' tab. The 'Console' tab is active, displaying a PUT request to `/product/_doc/1`. The request body is a JSON document with the following structure:

```
1 PUT /product/_doc/1
2 {
3   "name": "Java 9 and microservice",
4   "price": 213,
5   "instructor": {
6     "firstName": "Doun",
7     "lastName": "Mbengue"
8   }
9 }
```

The response body is also displayed, showing the document after the update:

```
1 {
2   "_index" : "product",
3   "_type" : "_doc",
4   "_id" : "1",
5   "_version" : 2,
6   "result" : "updated",
7   "_shards" : {
8     "total" : 2,
9     "successful" : 1,
10    "failed" : 0
11  },
12   "_seq_no" : 2,
13   "_primary_term" : 1
14 }
```

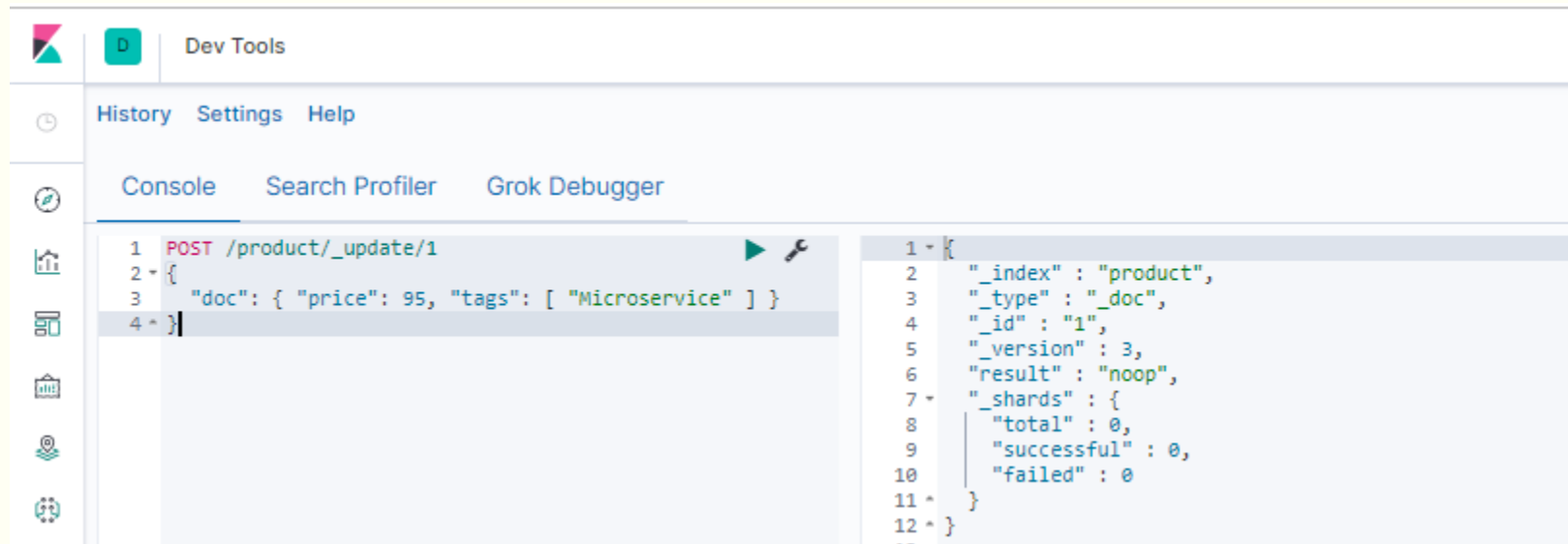
- Version d'un document

- 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.

# Exemple - 5

---

- Maj d'un document

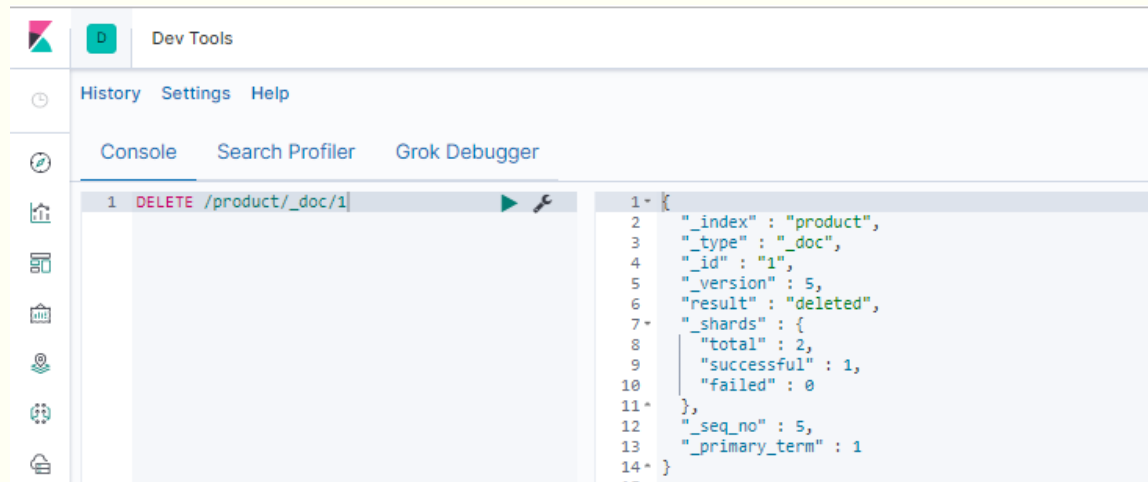


- Si le champ donné existe déjà, il est mis à jour, sinon il est ajouté au document
- Sans `_update` le document est entièrement remplacé

# Exemple - 6

---

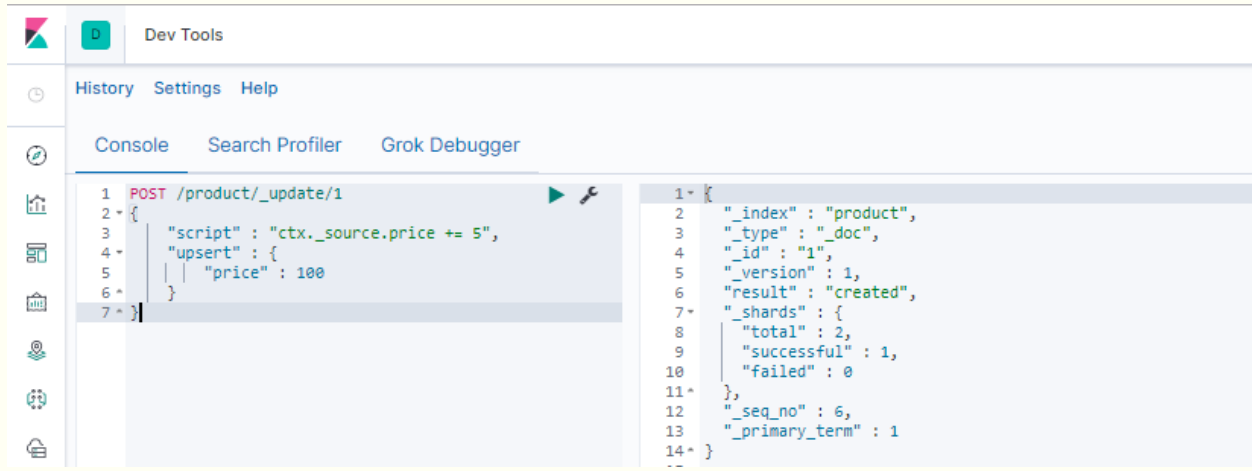
- Supprimer un document



# Exemple - 7

---

- Insertion ou Update d'un document existant



The screenshot shows the Chrome DevTools console with the 'Console' tab selected. On the left, the network log shows a POST request to `/product/_update/1` with a JSON body: `{ "script": "ctx._source.price += 5", "upsert": { "price": 100 } }`. On the right, the response is displayed as a JSON object: `{ "_index": "product", "_type": "_doc", "_id": "1", "_version": 1, "result": "created", "shards": { "total": 2, "successful": 1, "failed": 0 }, "_seq_no": 6, "_primary_term": 1 }`.

```
1 POST /product/_update/1
2 {
3   "script": "ctx._source.price += 5",
4   "upsert": {
5     | "price": 100
6   }
7 }
```

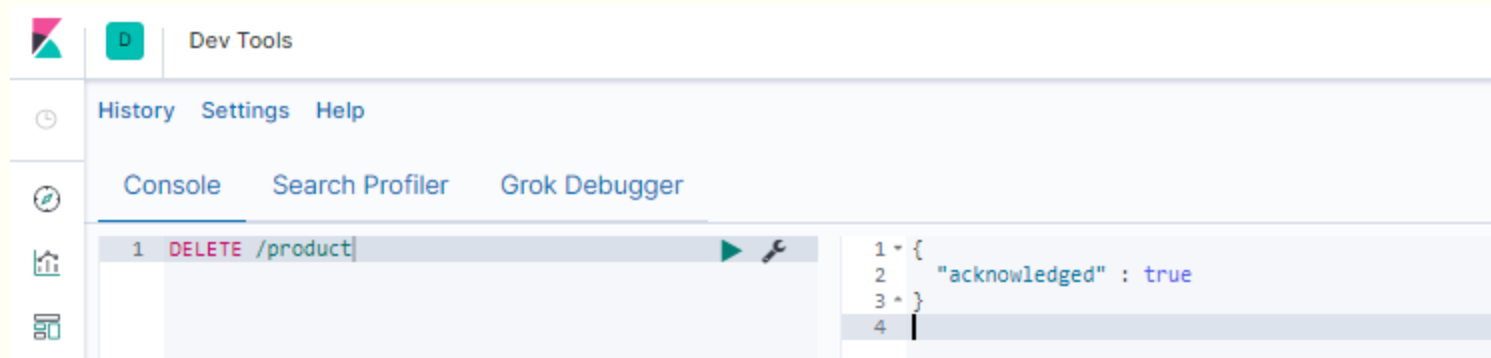
```
1 {
2   "_index": "product",
3   "_type": "_doc",
4   "_id": "1",
5   "_version": 1,
6   "result": "created",
7   "shards": {
8     "total": 2,
9     "successful": 1,
10    "failed": 0
11  },
12   "_seq_no": 6,
13   "_primary_term": 1
14 }
```



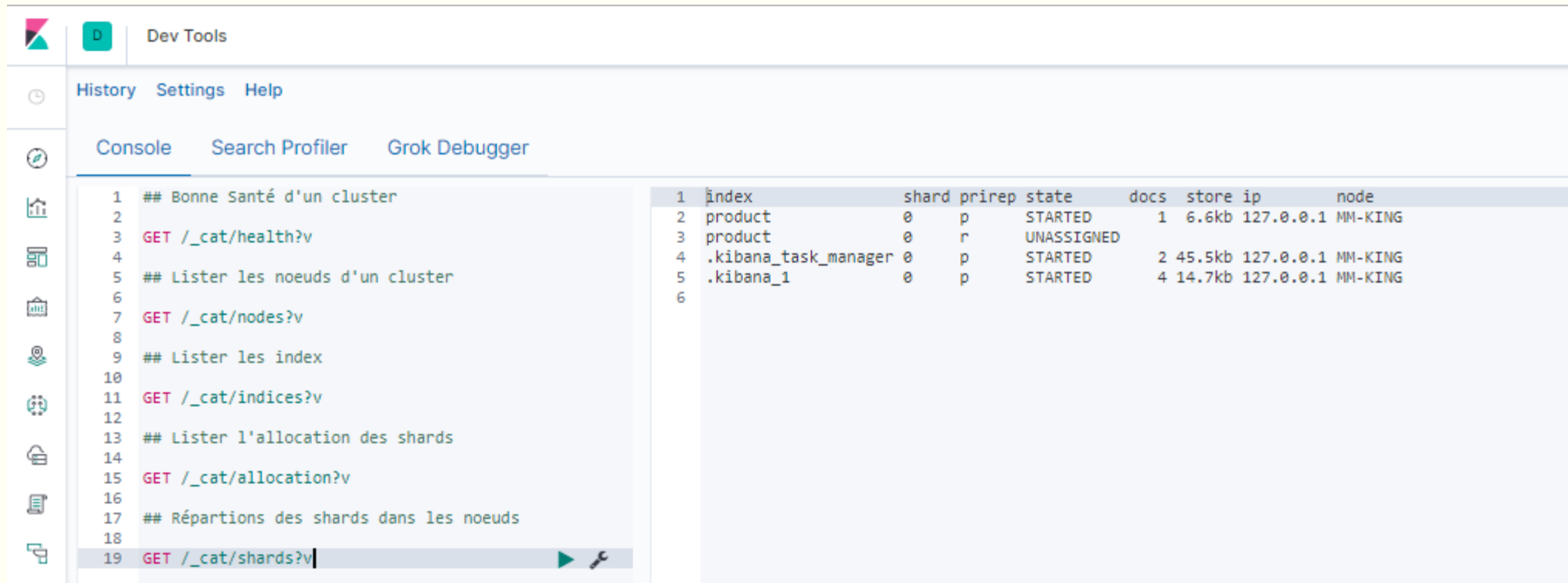
# Exemple - 8

---

- Suppression d'index



# Explorer un cluster



The screenshot shows the Dev Tools console with a series of REST client requests and their responses. The requests are as follows:

- 1. `## Bonne Santé d'un cluster`
- 2. `GET /_cat/health?v`
- 3. `## Lister les noeuds d'un cluster`
- 4. `GET /_cat/nodes?v`
- 5. `## Lister les index`
- 6. `GET /_cat/indices?v`
- 7. `## Lister l'allocation des shards`
- 8. `GET /_cat/allocation?v`
- 9. `## Répartitions des shards dans les noeuds`
- 10. `GET /_cat/shards?v`

The response for the last request (`GET /_cat/shards?v`) is a table showing the distribution of shards across nodes:

index	shard	prirep	state	docs	store	ip	node
product	0	p	STARTED	1	6.6kb	127.0.0.1	MM-KING
product	0	r	UNASSIGNED				
.kibana_task_manager	0	p	STARTED	2	45.5kb	127.0.0.1	MM-KING
.kibana_1	0	p	STARTED	4	14.7kb	127.0.0.1	MM-KING

# Importer des documents

---

- Télécharger et installer **Curl** : <https://curl.haxx.se/download.html>
- **cURL** est une interface en ligne de commande, destinée à récupérer le contenu d'une ressource accessible par http.
- Utilisation:

```
o>curl -H "Content-Type: application/json" -XPOST http://localhost:9200/product/_doc/_bulk?pretty --data-binary @test-data.json
```

# Disposition de titre et de contenu avec liste

---

**Question ?**

# Disposition de titre et de contenu avec liste

---

**TP**

**Lab-3-Crud**



# MODULE 3

Mapping et analyzers

# Introduction

---

- L'une des grandes forces d'un moteur de recherche Elasticsearch est son système d'analyse de textes, via le concept d'analyzer.
- Ce module à pour but d'expliquer :
  - comment fonctionne l'analyse des données contenues dans une base Elasticsearch,
  - et comment définir notre index pour nous permettre d'effectuer des recherches complexes.

# Mapping

---

***Mapping*** = Définir comment les documents sont analysés et indexés

Le **mapping** est similaire au *schema definition* dans une base de donnée relationnelle.

La mapping peut être défini :

- manuellement,
- généré automatiquement par ES quand les documents sont indexés.



# Mapping

---

Bien qu'Elasticsearch propose la détection automatique du type de champ, il est recommandé de définir comment ces documents doivent être manipulés :

- Les types de données des champs du document.
- Les relations entre les différents types de documents.
- La gestion des méta-données du document.
- La définition de la pertinence par champ/document (boosting)
- ...

# Mapping : Besoin

---

- Adapter l'analyse de données à un domaine métier spécifique
- Adapter la recherche aux utilisateurs :
  - Detected types might not be correct
- Certaines fonctionnalités l'exigent
  - Tri,
  - Agrégation
  - Hilighting

# Exemple

## Document

```
1 POST /directory/_doc
2 {
3   "name": "formationn_fr",
4   "adresse": {
5     "rue": "110 bld de belleville",
6     "zip": "75019",
7     "ville": "Paris"
8   }
9 }
```



## Mapping

```
1 {
2   "directory" : {
3     "mappings" : {
4       "properties" : {
5         "adresse" : {
6           "properties" : {
7             "rue" : {
8               "type" : "text",
9               "fields" : {
10                "keyword" : {
11                  "type" : "keyword",
12                  "ignore_above" : 256
13                }
14              }
15            },
16            "ville" : {
17              "type" : "text",
18              "fields" : {
19                "keyword" : {
20                  "type" : "keyword",
21                  "ignore_above" : 256
22                }
23              }
24            },
25            "zip" : {
26              "type" : "long"
27            }
28          }
29        },
30        "name" : {
31          "type" : "text",
32          "fields" : {
33            "keyword" : {
34              "type" : "keyword",
35              "ignore_above" : 256
36            }
37          }
38        }
39      }
40    }
41  }
42 }
```

# Le mapping automatique

---

Le document suivant  
est mappé comme suit  
dans ES, pour voir le mapping tapez:

```
--  
11 GET /users/_mapping  
--
```

```
requete.json  ✖  
1 {  
2   "name": "Paul",  
3   "age": 35  
4 }
```

```
{  
  "mytype":  
  {  
    "properties":  
    {  
      "age":  
      {  
        "type": "long"  
      },  
      "name":  
      {  
        "type": "string"  
      }  
    }  
  }  
}
```

# Le mapping automatique

---

## MAJ automatique

Il est possible d'ajouter des nouveaux champs, ou "upgrader" un champ en multi\_field.

**Exemple:** Si on rajoute un document suivant dans l'index précédent, le mapping change ...:

Le nouveau champ est  
automatiquement pris  
en compte

```
{  
  "name": "Paul",  
  "age": 45,  
  "prenom": "beta"  
}
```



```
{  
  "test" : {  
    "mappings" : {  
      "mytype" : {  
        "properties" : {  
          "age" : {  
            "type" : "long"  
          },  
          "name" : {  
            "type" : "string"  
          },  
          "prenom" : {  
            "type" : "string"  
          }  
        }  
      }  
    }  
  }  
}
```

# Le mapping automatique

---

## Arrays

Les tableaux sont automatiquement pris en compte par ES.

Si l'on insère le document suivant :

```
{
  "age": 55,
  "name": "Toto",
  "prenom": [ "Jean", "Henry", "Le", "Maire" ]
}
```

Par contre les éléments du tableau doivent être du même type:

```
{
  "age": "5p",
  "name": "Titi",
  "prenom": "Jean",
  "tag": [124, "abc"]
}
```

→ Exception

# Création d'un mapping

---

- Le mapping est une composante essentielle d'Elasticsearch pour profiter de l'ensemble des avantages du moteur de recherche.
- La définition d'un mapping est intimement liée aux requêtes et à l'analyse de texte.
- Elle va nous permettre de définir l'ensemble des champs de nos documents JSON, tel que leur type ou leur format.
- C'est dans le mapping que nous allons aussi définir quelle analyse sera appliqué au champ.

# Définir un mapping

---

```
PUT my_index ❶
{
  "mappings": {
    "properties": { ❷
      "title": { "type": "text" }, ❸
      "name": { "type": "text" }, ❹
      "age": { "type": "integer" }, ❺
      "created": {
        "type": "date", ❻
        "format": "strict_date_optional_time||epoch_millis"
      }
    }
  }
}
```

- ❶ Create an index called `my_index`.
- ❷ Specify the fields or *properties* in the mapping.
- ❸ Specify that the `title` field contains `text` values.
- ❹ Specify that the `name` field contains `text` values.
- ❺ Specify that the `age` field contains `integer` values.
- ❻ Specify that the `created` field contains `date` values in two possible formats.

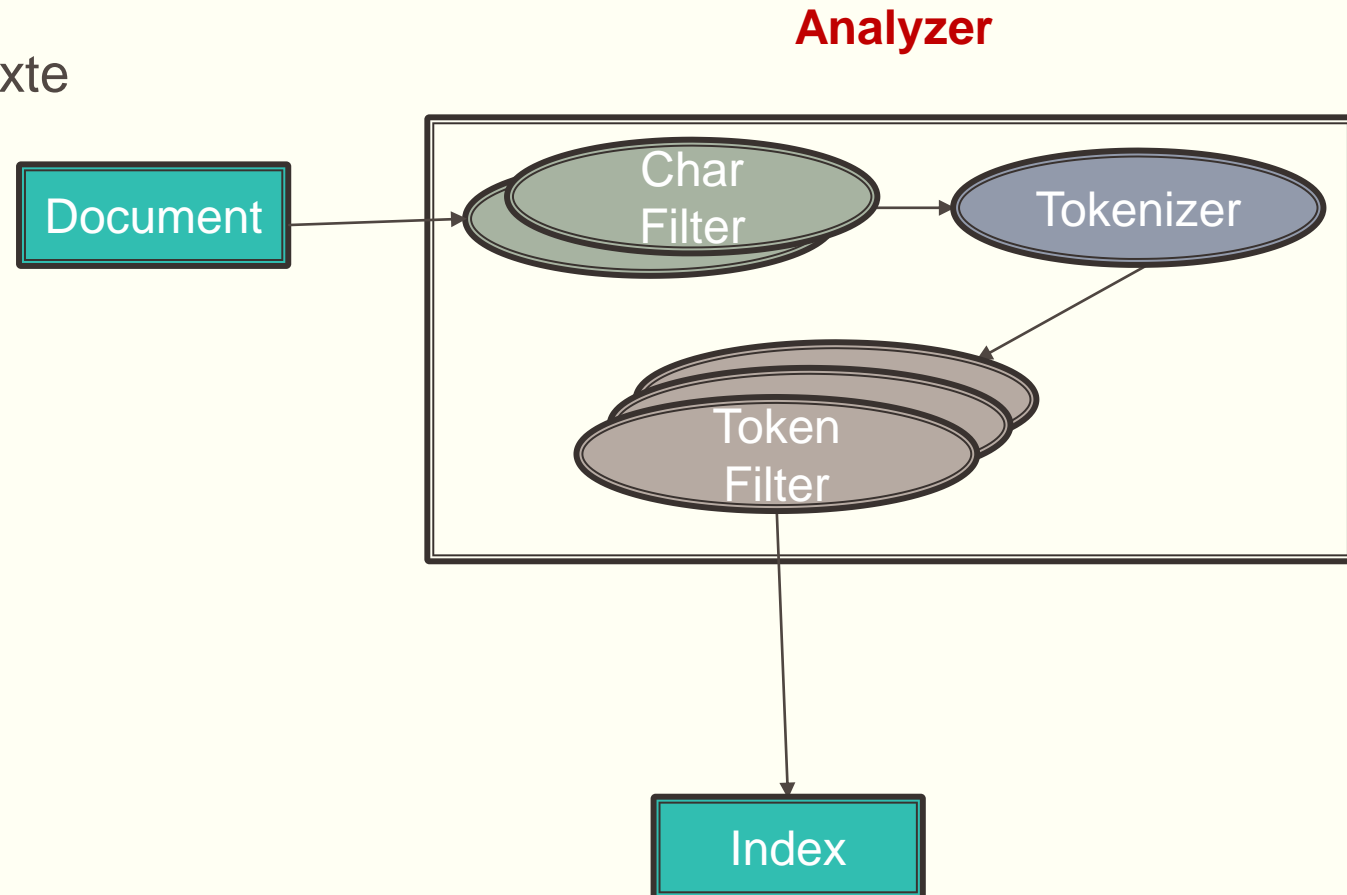


# Processus de création de l'index

---

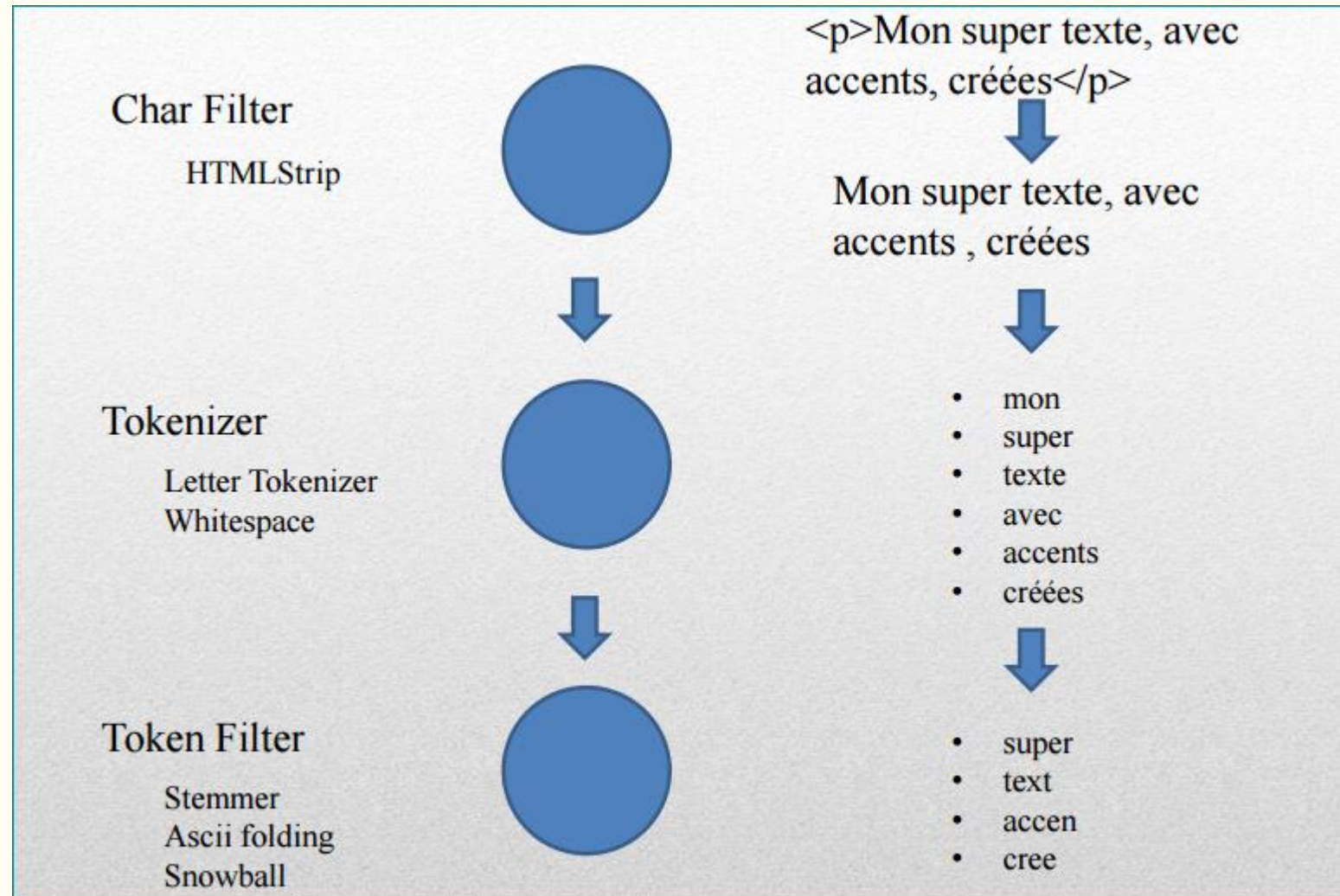
## Chaine de composants

- Extraction de termes du texte
- Normaliser les données
- A l'indexation/Requête



# Analyzer : Exemple

---



# Analyzer

---

Un analyzer, défini sur un attribut dans le mapping, permet d'analyser un texte et de le découper pour une recherche textuelle approfondie

Il est constitué de :

- **0 à plusieurs char filters** : Permettant de pré-traiter une chaîne de caractère avant son découpage.
- **1 tokenizer** : Permettant de découper la donnée.
- **0 à plusieurs token filters** : Permettant de filtrer ou transformer les données découpées.

# Caractere Filters

---

- Filtre optionnel
- Prétraitement du texte
- Exemple : `html_strip`
  - enlève les caractères html comme `<p>` ou `<div>`
  - Replace `&Aacute;` par le caractère Unicode correspondant.
- Un analyseur peut avoir 0 ou + char Filter

Exemple : `<div>Hello <b>World</b></div>` ➔ `Hello World`

# Des exemples de tokenizers

---

- **standard** : Découpe un texte par la plupart des délimiteurs possibles, intéressant pour la plupart des langues européennes
- **letter** : Découpe en texte par tous caractères qui n'est pas une lettre
- **thai** : Utilise la segmentation thaïlandaise.
- **keyword**,
- **whitespace**,
- **pattern**,
- ...
- Doc: <https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenizers.html>

# Des exemples de char/token filters

---

- **lowercase** : Normalise un texte en lower case
- **length** : Filtre les mots en fonction de leurs tailles
- **stemmer** : Transforme tous les mots d'un texte d'une langue en racines de mots
- Doc: <https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenfilters.html>

# Analyzers "built-in"

---

Pour les cas les plus courants, elasticsearch possède par défaut plusieurs analyzers :

- **standard** : Découpe le textes avec la plupart des délimiteurs possibles (espace, tirets, apostrophe...), supprime les ponctuations, et peut (désactivé par défaut) filtrer les stop words (mots communs tel que le, la, les ...)
- **simple** : Découpe par tous les caractères non littérales, et les passe en minuscule.
- **whitespace** : Comme le précédent, mais découpe par espace.
- Doc: <https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html>

# Test sur les analyseurs

---

## Format:

```
POST _analyze
{
  "analyzer": "whitespace",
  "text":     "The quick brown fox."
}
```

```
POST _analyze
{
  "tokenizer": "standard",
  "filter":    [ "lowercase", "asciifolding" ],
  "text":      "Is this déjà vu?"
}
```



# Test sur les analyseurs

---

C'est un joueur d'échecs, il réfléchit. bob@hotmail.com 123456

**standard Analyseur** : Découpage naturelle (anglais)

**c'est, un, joueur, d, checs, il, r, fl, chit, bob, hotmail.com, 123432**

**simple Analyseur**

**c, est, un, joueur, d, checs, il, r, fl, chit, bob, hotmail, com**

**whitespace Analyseur** : Découpe sur les espaces.

**C'est, un, joueur, d'échecs, il, réfléchit. , bob@hotmail.com, 123432**

**keyword Analyseur** : Ne découpe pas.

**C'est un joueur d'échecs, il réfléchit. bob@hotmail.com 12345**

# Custom analyzer

---

```
PUT my_index
{
  "settings": {
    "analysis": {
      "analyzer": {
        "std_folded": { ❶
          "type": "custom",
          "tokenizer": "standard",
          "filter": [
            "lowercase",
            "asciifolding"
          ]
        }
      }
    },
    "mappings": {
      "properties": {
        "my_text": {
          "type": "text",
          "analyzer": "std_folded" ❷
        }
      }
    }
  }
}
```

- ❶ Define a custom analyzer called `std_folded`.
- ❷ The field `my_text` uses the `std_folded` analyzer.

# Custom analyzer

---

```
GET my_index/_analyze ③
{
  "analyzer": "std_folded", ④
  "text": "Is this déjà vu?"
}

GET my_index/_analyze ⑤
{
  "field": "my_text", ⑥
  "text": "Is this déjà vu?"
}
```

③ To refer to this analyzer, the `analyze` API must specify the index name.

⑤

④ Refer to the analyzer by name.

⑥ Refer to the analyzer used by field `my_text`.

# Disposition de titre et de contenu avec liste

---

**Question ?**

# Disposition de titre et de contenu avec liste

---

**TP**

**Lab\_4\_Mapping**

**TP**

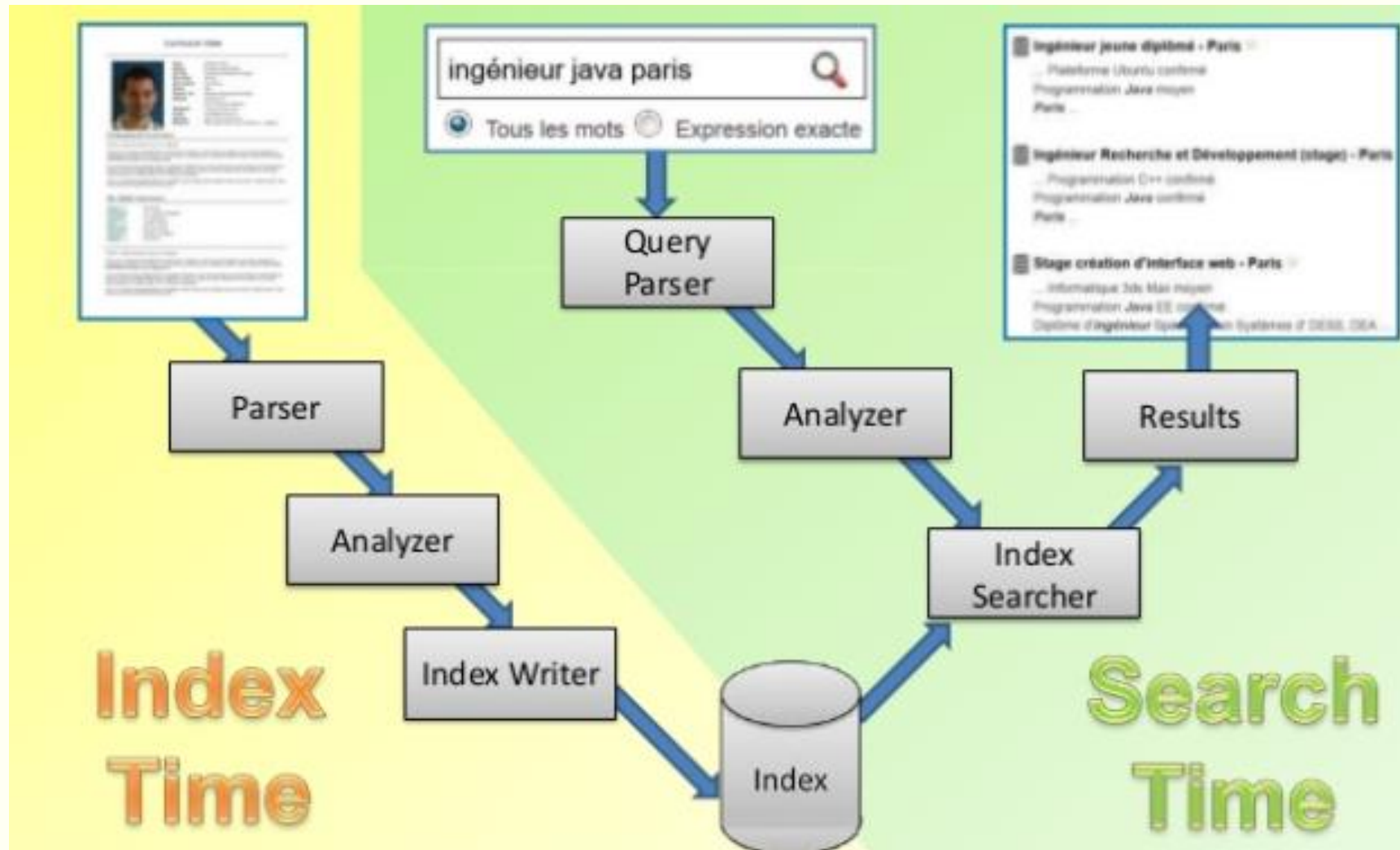
**Lab\_5\_Analyzers**



# MODULE 4

Introduction à la recherche

# Introduction



# Recherche

---

- 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



# 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.

# Disposition de titre et de contenu avec liste

---

**Question ?**

# Disposition de titre et de contenu avec liste

---

**TP**

**Lab\_6\_TF-IDF**

# Structure d'une recherche simple

---

Deux formes :

- Query-String

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

- Request-Body

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



# Query-String

---

- Les mots-clés sont passés dans le paramètre **q** (pour « **query** ») de l'URL.
- Pratique pour la recherche en ligne de commande.

```
## Trouver tous les documents
GET /product/_search?q=*

## Trouver tous les documents avec name=`Lobster`
GET /product/_search?q=name:Lobster

## Idem mais cette fois le champs de recherche est un tableau
## Trouver les documents dont le champ tags contient le mot `Meat`
GET /product/_search?q=tags:Meat

## Recherche de documents avec tags= `Meat` _et_ name = `Tuna`
GET /product/_search?q=tags:Meat AND name:Tuna
```

# Introduction à la recherche

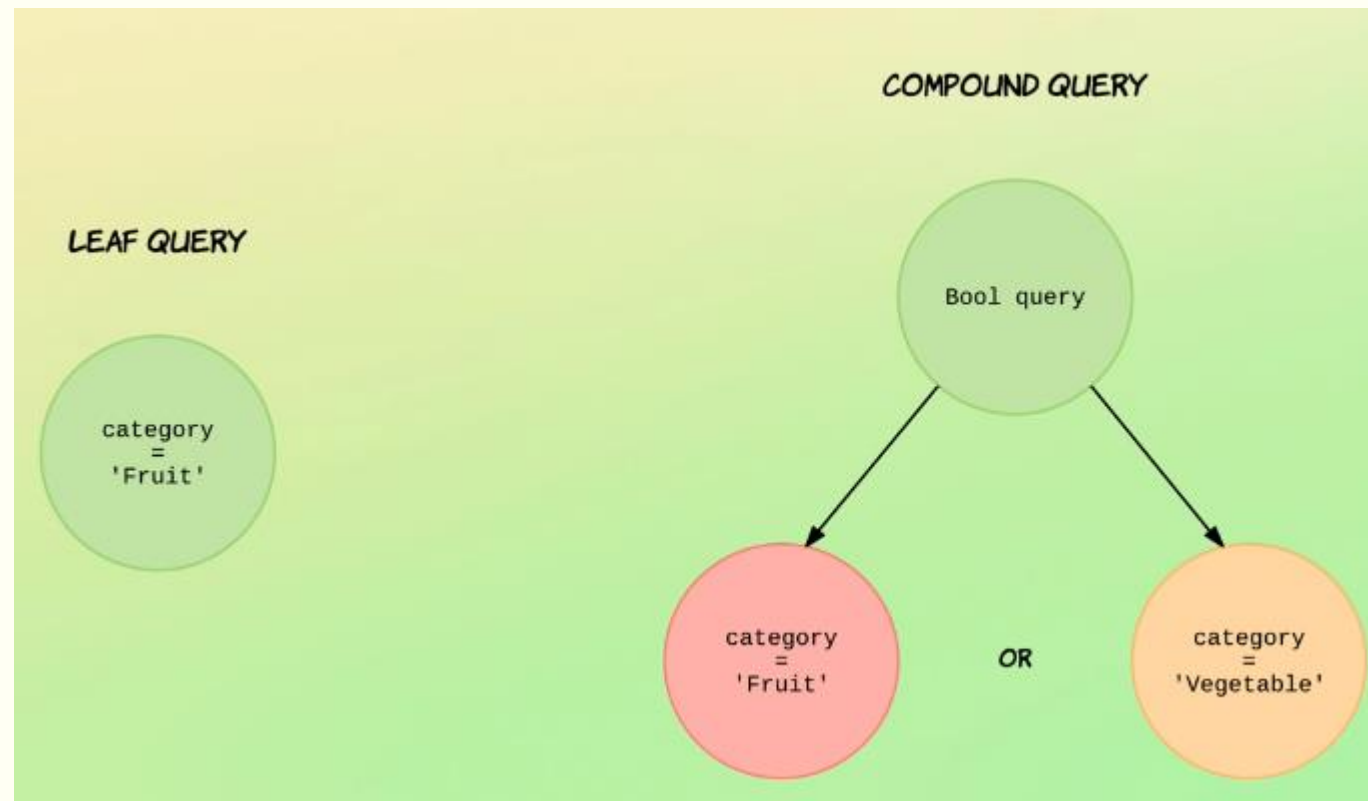
---

## 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

# 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" } }
      ]
    }
  }
}
```

# Paramètres de recherche

---

- En plus de la query, on peut passer d'autres paramètres dans le corps de la requête, notamment pour faire de la pagination.
- Par exemple, la requête suivante retournera uniquement le premier résultat. Le paramètre par défaut de **size** est 10.

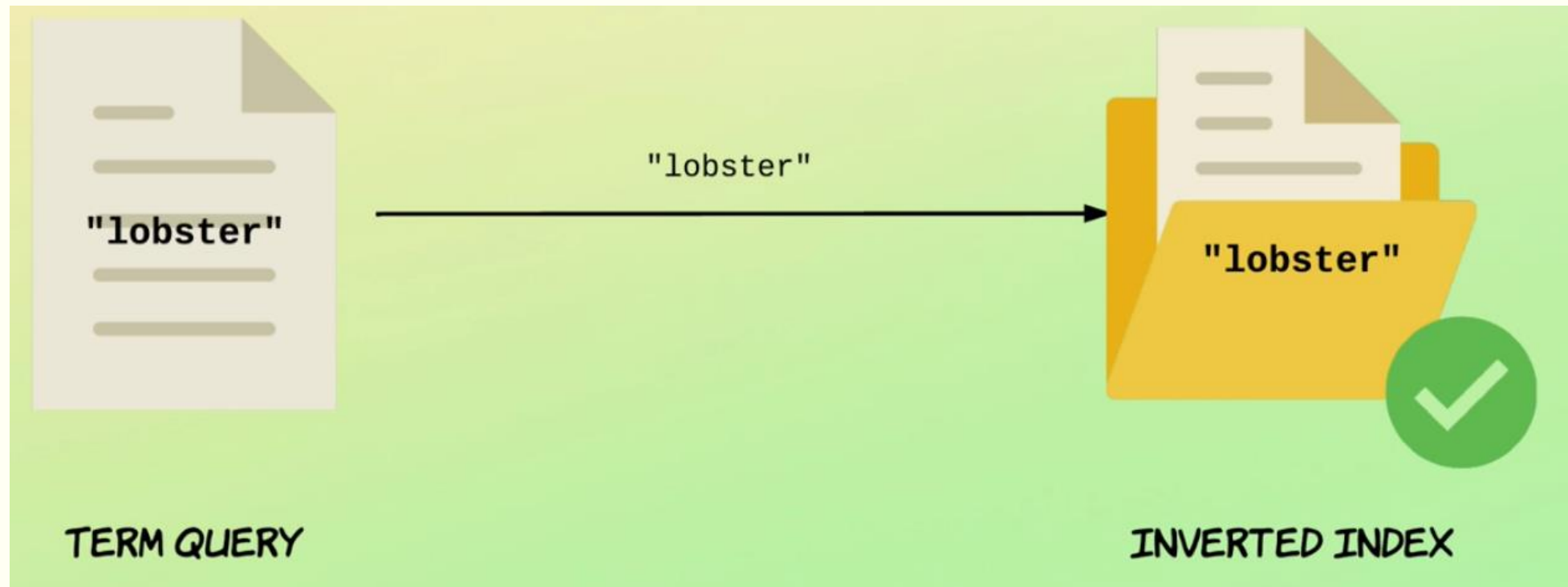
```
POST /biblio/_search
{
  "query": { "match_all": {} },
  "size": 1
}
```

- Celle-ci retournera les documents 11 à 20.
  - Par défaut, la valeur du paramètre **from** est 0.

```
1 POST /biblio/_search
2 {
3   "query": { "match_all": {} },
4   "from": 10,
5   "size": 10
6 }
```

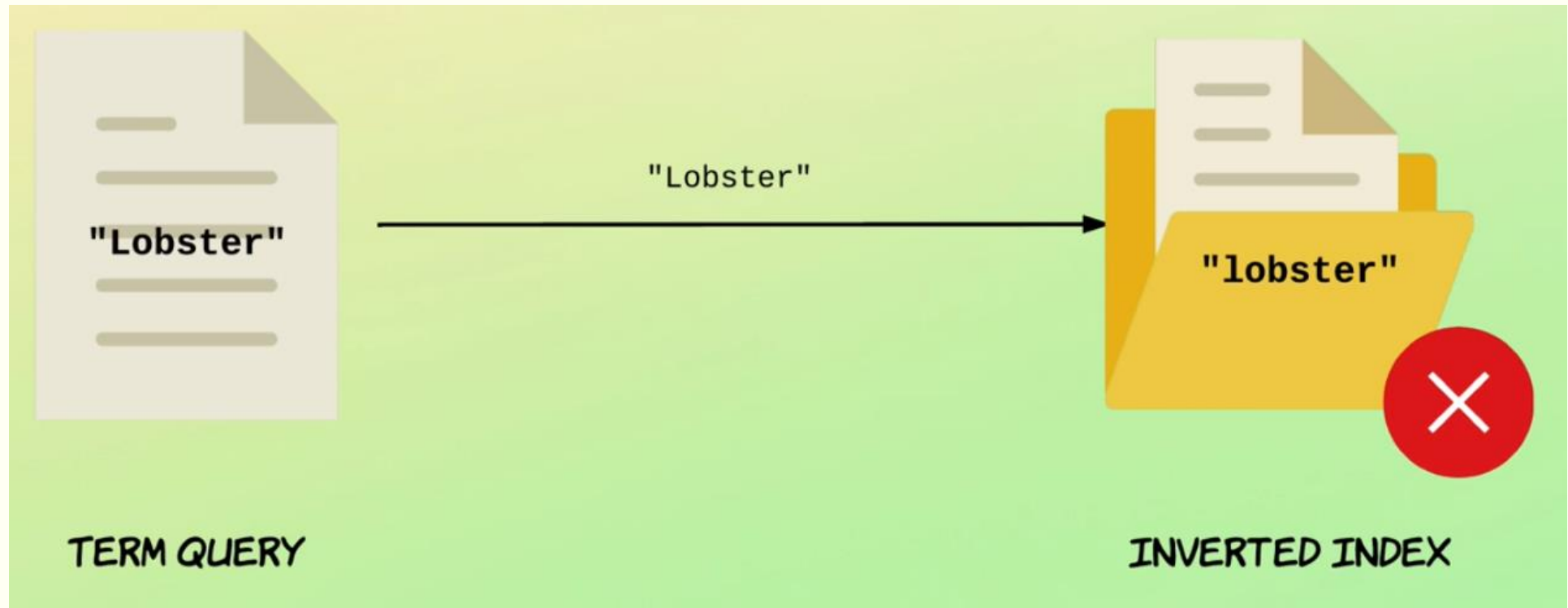
# Full text queries (term) vs term level queries (match)

---



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

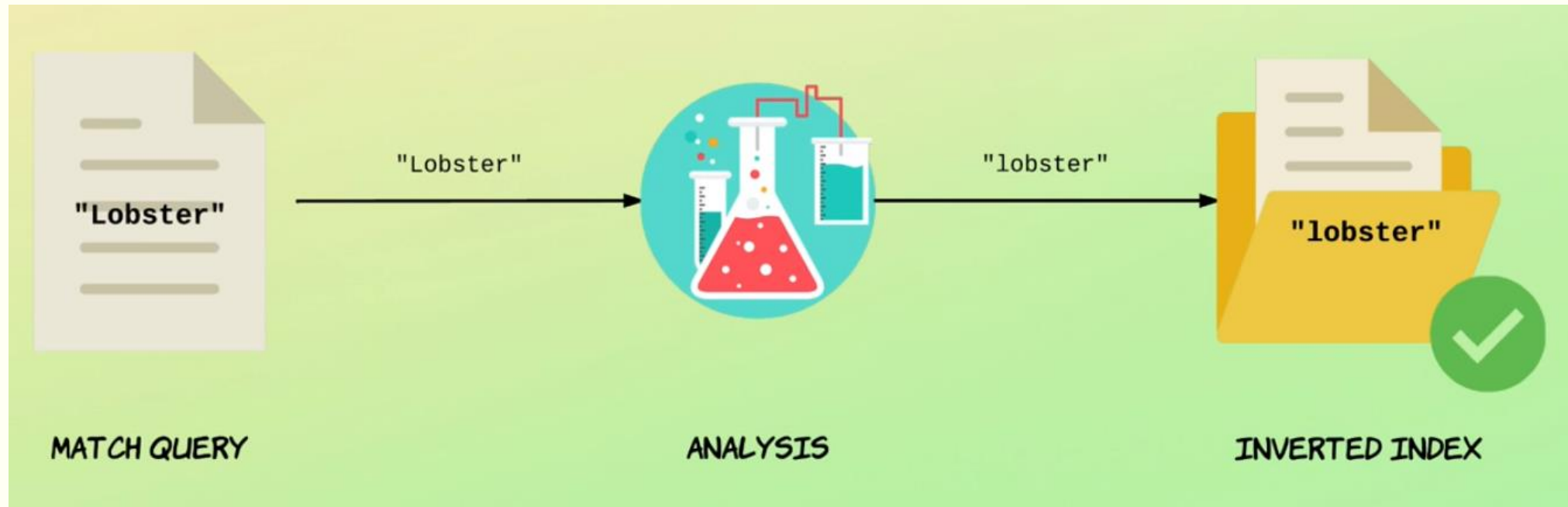
---





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

---



# Term Level Queries

---

```
1  ## Ne passe pas par l'analyseur (Term level n'p analysée)
2  GET /product/_search
3  {
4    "query": {
5      "term": {
6        "name": "lobster"
7      }
8    }
9  }
10
11 GET /product/_search
12 {
13   "query": {
14     "term": {
15       "name": "Lobster"
16     }
17   }
18 }
19
20 ## Full-text queries est analysée
21 GET /product/_search
22 {
23   "query": {
24     "match": {
25       "name": "Lobster"
26     }
27   }
28 }
```



# Disposition de titre et de contenu avec liste

---

**TP**

**Lab\_7\_IntroRecherche**

**TP**

**Lab\_8\_TermeLevel**

# Full Text Queries

---

- Introduisons maintenant une nouvelle requête appelée la requête de correspondance, qui peut être considérée comme une requête de recherche par champs de base (c'est-à-dire une recherche effectuée sur un champ ou un ensemble de champs spécifique).
- Cet exemple renvoie tous les recettes contenant le terme "pasta" ou "spaghetti" dans le titre:

```
## Match standard
GET /recipe/_search
{
  "query": {
    "match": {
      "title": "pasta spaghetti"
    }
  }
}
```

# Full Text Queries

---

- Cet exemple est une variante de match ( match\_phrase ) qui divise la requête en termes et ne renvoie que les documents contenant tous les termes du titre dans les mêmes positions les uns par rapport aux autres

```
## Ordres des mots importants
GET /recipe/_search
{
  "query": {
    "match_phrase": {
      "title": "spaghetti puttanesca"
    }
  }
}
```

# Disposition de titre et de contenu avec liste

---

**TP**

**Lab\_9\_FullTextQuery**

# Logique booléenne et requête

---

- La requête bool nous permet de composer des requêtes plus petites en requêtes plus grandes en utilisant la logique booléenne.
- Cet exemple compose deux requêtes de correspondance et renvoie tous les recettes contenant "parmesan" **et** "parmesan" dans "ingredients.name":

```
## renvoie tous les recettes contenant "parmesan" et "parmesan" dans "ingredients.name":  
GET /recipe/_search  
{  
  "query": {  
    "bool": {  
      "must": [  
        { "match": { "ingredients.name": "parmesan" } },  
        { "match": { "ingredients.name": "pasta" } }  
      ]  
    }  
  }  
}
```

- Dans l'exemple ci-dessus, la clause bool **must** spécifie toutes les requêtes devant être vraies pour qu'un document soit considéré comme une correspondance.

# Logique booléenne et requête

---

- En revanche, cet exemple compose deux requêtes de correspondance et renvoie tous les recipes contenant "parmesan" **ou** "pasta" dans "ingredients.name":

```
## renvoie tous les recipes contenant "parmesan" ou "parmesan" dans "ingredients.name":  
GET /recipe/_search  
{  
  "query": {  
    "bool": {  
      "should": [  
        { "match": { "ingredients.name": "parmesan" } },  
        { "match": { "ingredients.name": "pasta" } }  
      ]  
    }  
  }  
}
```

- Dans l'exemple ci-dessus, la clause bool **should** spécifie une liste de requêtes dont **l'une ou l'autre** doit être vraie pour qu'un document soit considéré comme une correspondance.



# Logique booléenne et requête

---

- Cet exemple compose deux requêtes de correspondance et renvoie tous les recipes qui ne contiennent **ni** "parmesan" **ni** "pasta" dans "ingredients.name":

```
## renvoie tous les recipes qui ne contiennent ni "parmesan" ni "pasta" dans "ingredients.name":  
GET /recipe/_search  
{  
  "query": {  
    "bool": {  
      "must_not": [  
        { "match": { "ingredients.name": "parmesan" } },  
        { "match": { "ingredients.name": "pasta" } }  
      ]  
    }  
  }  
}
```

- Dans l'exemple ci-dessus, la clause **bool must\_not** spécifie une liste de requêtes dont aucune ne doit être vraie pour qu'un document soit considéré comme une correspondance.

# Logique booléenne et requête

---

- Nous pouvons combiner des clauses `must`, `should` et `must_not` simultanément dans une requête bool.
- De plus, nous pouvons composer des requêtes bool dans chacune de ces clauses bool pour imiter toute logique booléenne complexe à plusieurs niveaux.

```
## Requete avec 'must_not'
GET /recipe/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "ingredients.name": "parmesan"
          }
        }
      ],
      "must_not": [
        {
          "match": {
            "ingredients.name": "tuna"
          }
        }
      ]
    }
  }
}
```

# Disposition de titre et de contenu avec liste

---

**TP**

**Lab\_10\_BooleanQuery**

# Spécifier le format des résultats

---

- Par défaut, le document indexé complet est renvoyé dans le cadre de toutes les recherches.
- C'est ce que l'on appelle la source (champ **\_source** dans les résultats de recherche).
- Si nous ne voulons pas que tout le document source soit retourné, nous avons la possibilité de ne demander que quelques champs de la source à retourner, ou nous pouvons définir `_source` sur `false` pour omettre complètement le champ.
- Cet exemple montre comment retourner deux champs, `account_number` et `balance` (à l'intérieur de `_source` ), à partir de la recherche:

```
GET /recipe/_search
{
  "query": { "match_all": { } },
  "_source": ["title", "preparation_time_minutes"]
}
```

# Disposition de titre et de contenu avec liste

---

**TP**

**Lab\_11\_ControlQuery**

# Agrégation

- Pour compter, grouper, ...

**Étoiles**  
☐ 1 étoile (2)  
☐ 2 étoiles (2)  
☐ 3 étoiles (4)  
☐ 4 étoiles (4)  
☐ non classé (3)

**Type d'établissement**  
☐ Hôtels (16)  
☐ B&B / Chambres d'hôtes (1)

**Note des commentaires**  
☐ Très bien : 8+ (7)  
☐ Bien : 7+ (15)  
☐ Agréable : 6+ (17)

**Équipement**  
☐ connexion Wi-Fi au réseau local (17)  
☐ parking (17)  
☐ navette aéroport (1)



# Agrégations

---

- Les agrégats fonctionnent avec deux concepts, les ***buckets*** (seaux, en français) qui sont les catégories que vous allez créer, et les ***metrics*** (indicateurs, en français), qui sont les statistiques que vous allez calculer sur les *buckets*.
- Si l'on compare à une requête SQL très simple :

```
SELECT COUNT(color)
FROM table
GROUP BY color
```

- COUNT(color) est la métrique, GROUP BY color crée les groupes (buckets).

# Agrégation types

---

## Buckets

Terms

Date Histogram

Range

Filter

IPv4 Range

## Metrics

Avg

Cardinality

Min / Max

Sum

Geo Bounds



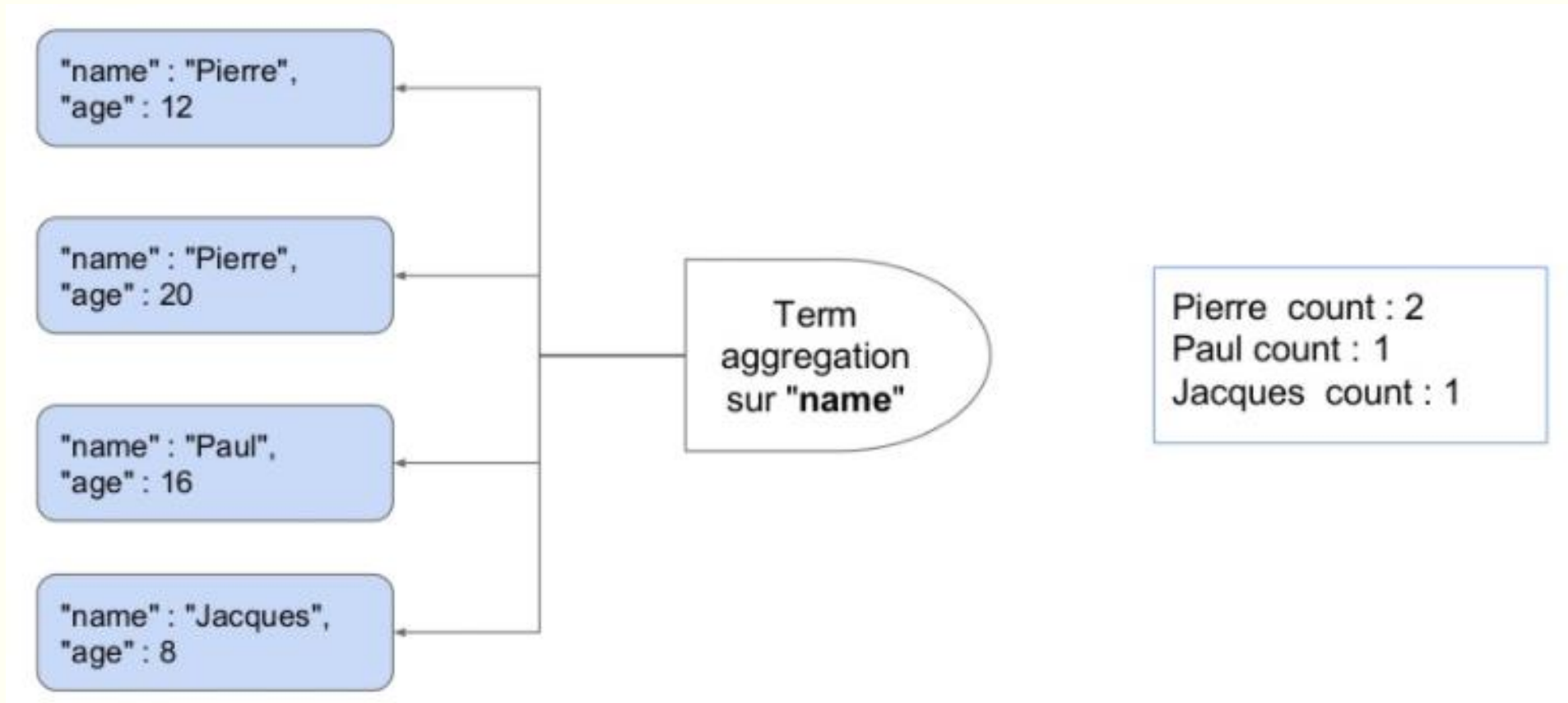
# Agrégations

---

- Une agrégation est la combinaison d'un *bucket* (au moins) et d'une *metric* (au moins).
- On peut, pour des requêtes complexes, imbriquer des *buckets* dans d'autres *buckets*.
- La syntaxe est, comme précédemment, très modulaire.

# Bucket agrégation

---



# Bucket agrégation

---

## Buckets

Buckets  $\approx$  GROUP BY

Buckets  $\Rightarrow$  doc\_count

Buckets inside Buckets

## Exemple

```
{
  [...],
  "aggregations": {
    "hashtags": {
      "buckets": [
        {
          "key": "IWD2016",
          "doc_count": 4
        },
        {
          "key": "heforshe",
          "doc_count": 2
        },
        {
          "key": "women",
          "doc_count": 2
        }
      ]
    }
  }
}
```

# Agrégations - Agrégation moyenne

---

- Il s'agit d'une agrégation de mesures à valeur unique qui calcule la moyenne des valeurs numériques extraites des documents agrégés.

```
POST /index/_search?
{
  "aggs" : {
    "avd_value" : { "avg" : { "field" : "name_of_field" } }
  }
}
```

- L'agrégation ci-dessus calcule la note moyenne sur tous les documents.
- Le type d'agrégation est moy et le paramètre de champ définit le champ numérique des documents sur lesquels la moyenne sera calculée.

# Agrégations - Agrégation moyenne

---

- Ce qui précède renverra ce qui suit:

```
{  
  ...  
  "aggregations": {  
    "avg_value": {  
      "value": 75.0  
    }  
  }  
}
```

- Le nom de l'agrégation (avg\_grade ci-dessus) sert également de clé permettant d'extraire le résultat de l'agrégation de la réponse renvoyée.

# Agrégations - Agrégation de cardinalité

---

- Une agrégation de mesures à valeur unique qui calcule un compte approximatif de valeurs distinctes.
- Les valeurs peuvent être extraites de champs spécifiques du document ou générées par un script.

```
POST /index/_search?size=0
{
  "aggs" : {
    "type_count" : {
      "cardinality" : {
        "field" : "type"
      }
    }
  }
}
```

## Réponse:

```
{
  ...
  "aggregations" : {
    "type_count" : {
      "value" : 3
    }
  }
}
```

# Agrégation étendue des statistiques

---

- Une agrégation de métriques à valeurs multiples qui calcule les statistiques sur les valeurs numériques extraites des documents agrégés.
- L'agrégation `extended_stats` est une version étendue de l'agrégation de statistiques, dans laquelle des mesures supplémentaires sont ajoutées, telles que `sum_of_squares`, `variance`, `std_deviation` et `std_deviation_bounds`.

```
{
  "aggs" : {
    "stats_values" : { "extended_stats" : { "field" : "field_name" } }
  }
}
```

# Agrégation étendue des statistiques

---

- Sortie de l'échantillon:

```
{
  ...
  "aggregations": {
    "stats_values": {
      "count": 9,
      "min": 72,
      "max": 99,
      "avg": 86,
      "sum": 774,
      "sum_of_squares": 67028,
      "variance": 51.55555555555556,
      "std_deviation": 7.180219742846005,
      "std_deviation_bounds": {
        "upper": 100.36043948569201,
        "lower": 71.63956051430799
      }
    }
  }
}
```



# Metrics agrégation

---

## Metrics

Metrics  $\approx$  SUM/AVG/MIN/MAX

Metrics inside Buckets

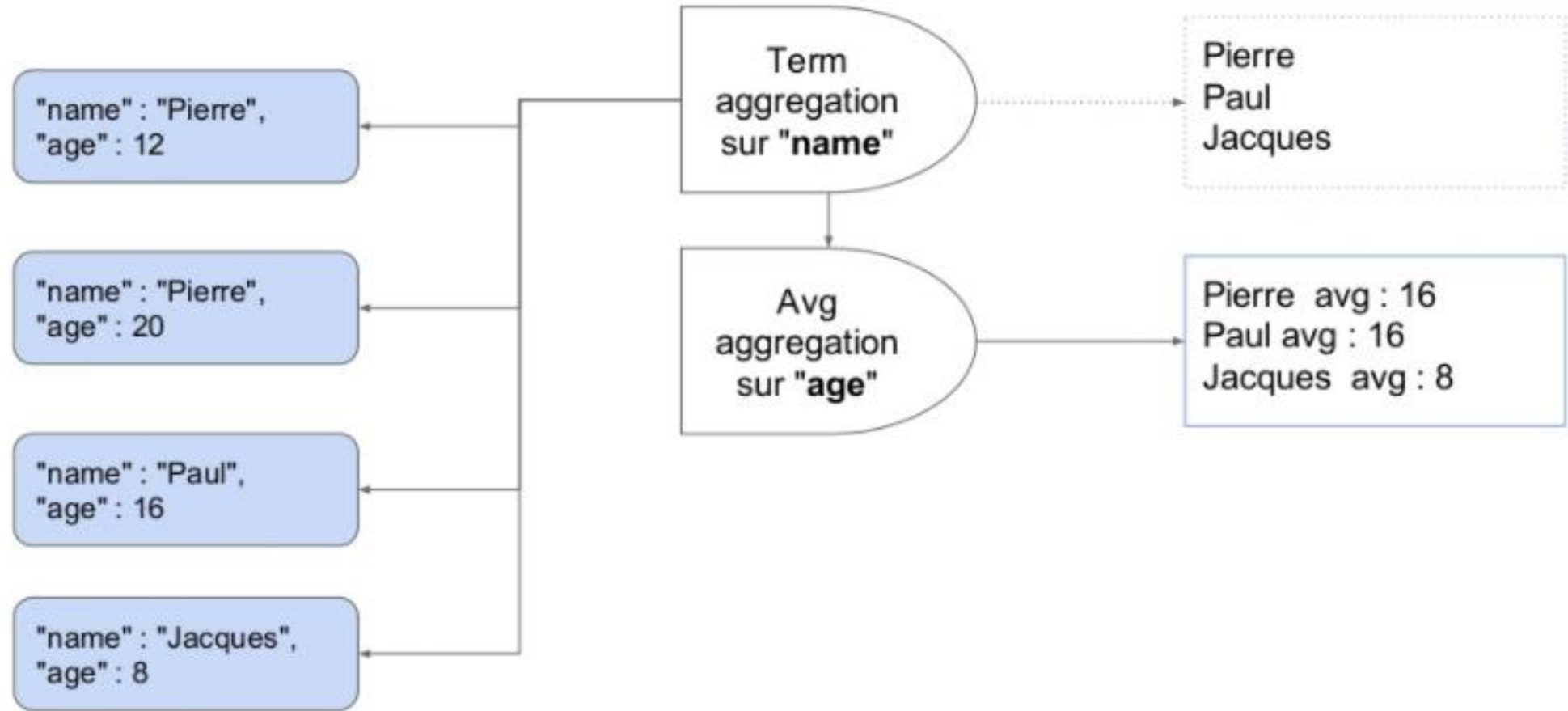
~~Metrics inside Metrics~~

## Exemple

```
{
  [...],
  "aggregations": {
    "user_follower_stats": {
      "count": 4871628,
      "min": 0,
      "max": 72529214,
      "avg": 5242.441252493007,
      "sum": 25539223594
    }
  }
}
```

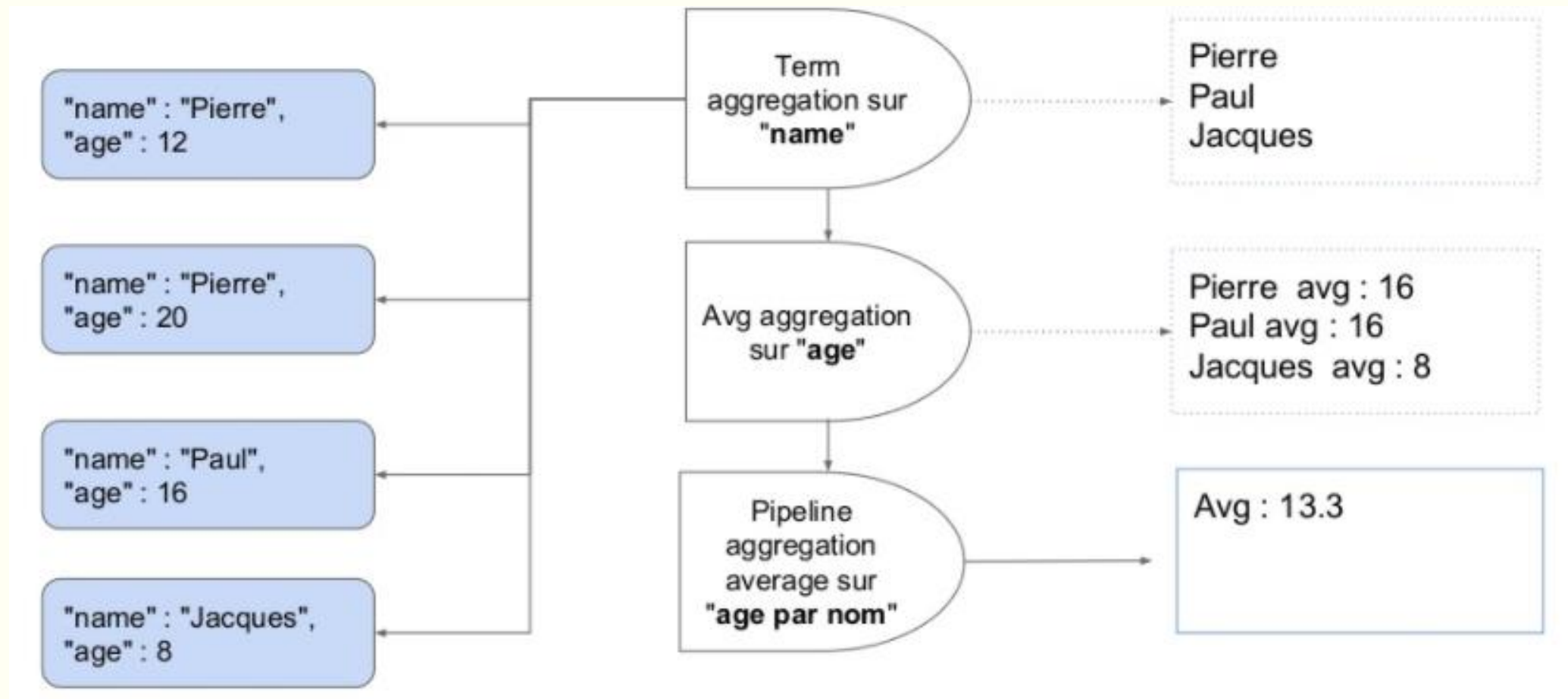
# Sub agrégation

---



# Pipeline agrégation

---



# Disposition de titre et de contenu avec liste

---

**Question ?**

# Disposition de titre et de contenu avec liste

---

**TP**

**Lab\_12\_Agregation**

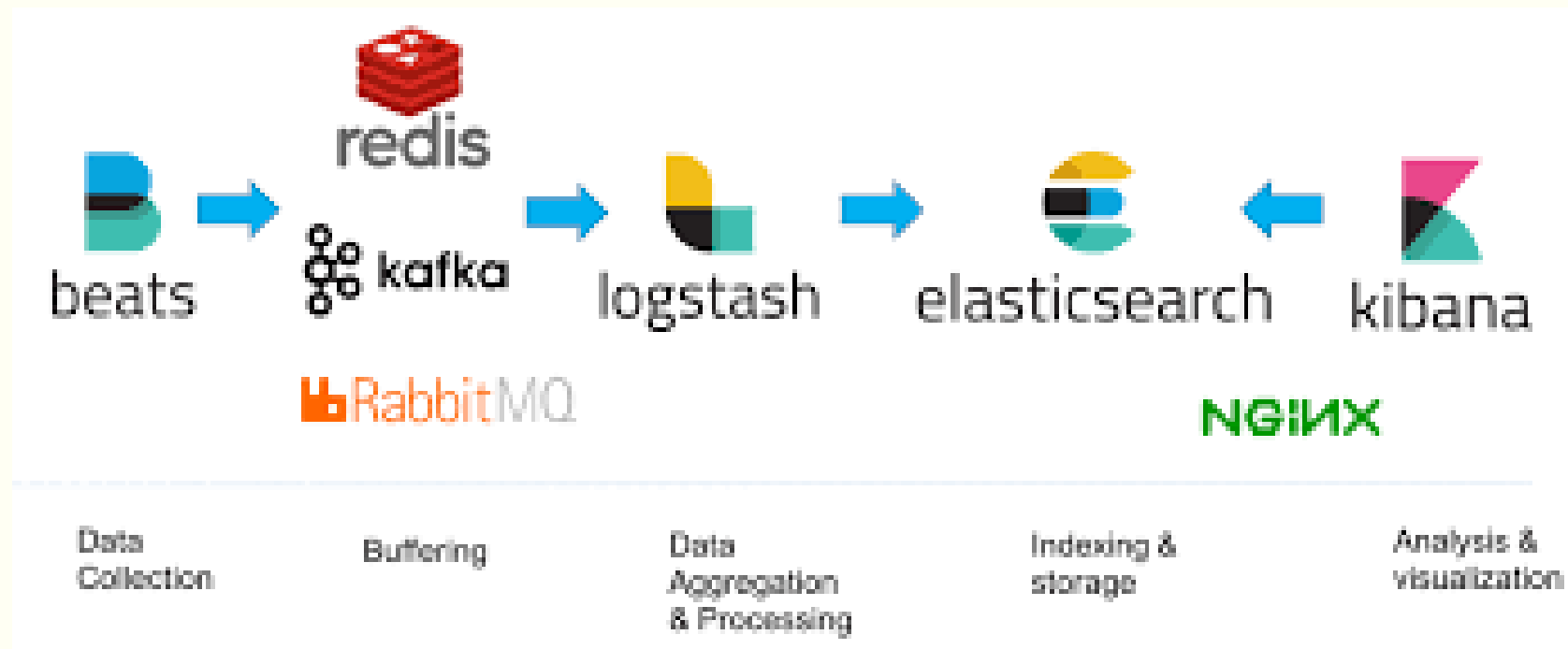


# MODULE 5

Logstash

# Présentation de la stack ELK

---

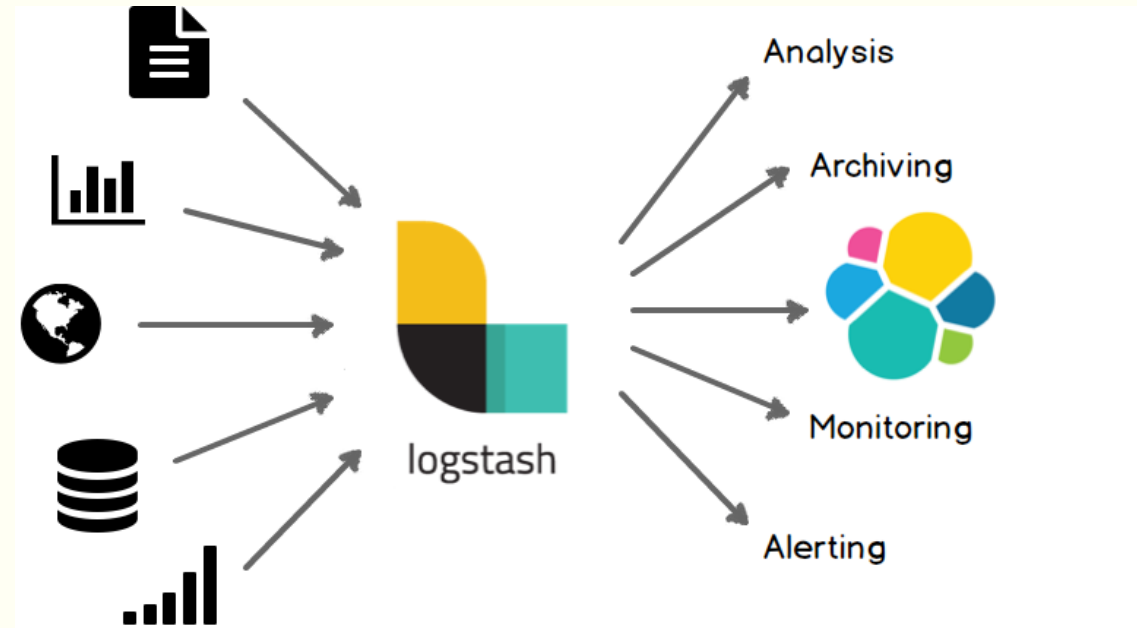


# Logstash

---

**Logstash** est un outil dédié à la transformation et à la transmission de données issues de sources diverses aux formats hétérogènes.

Ainsi, il extrait des données issues de sources variées, les formate vers une autre structure de données et les transmet selon plusieurs protocoles, tels que le chargement en base de données, le protocole http...



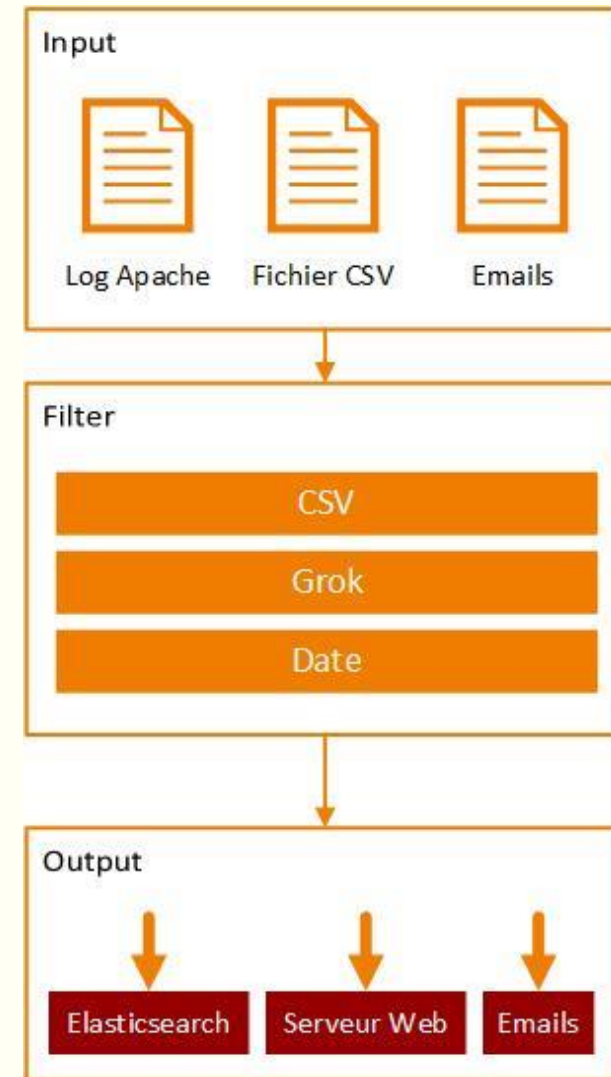


# Logstash

---

Logstash fonctionne en trois étapes séquentielles :

- l'extraction des données (Input)
- la normalisation (Filter)
- la transmission des données (Output).



# Étapes: Extraction – Transformation – Transmission (Load)

---

- L'étape d'extraction décrit les sources de données. Logstash s'appuie sur un écosystème fournit de plugins. Il est ainsi possible de récupérer des données depuis un fichier, une requête SQL, un flux twitter, un fichier csv... Certains plugins fournissent des options de configuration supplémentaires permettant d'affiner l'extraction des données.
- L'étape de transformation s'appuie sur les informations récoltées à l'étape précédente afin de construire la structure de données à retourner. Plusieurs sous-opérations de transformation peuvent se succéder afin de construire de manière itérative le résultat attendu.
- L'étape de transmission envoie l'information construite à l'étape précédente aux destinations mentionnées. Il est possible de réaliser des requêtes Http, SQL ou d'envoyer des e-mails depuis ce procédé.

# Logstash – Cycle de vie

---

- **Input → Filters → Output**
- **Filters** are processed in order of config file
- **Outputs** are processed in order of config file
- **Input:** Input stream
  - File input (tail)
  - Log4j
  - Redis
  - Syslog
  - and many more...

# Logstash – Cycle de vie

---

- **Codecs** : decoding log messages
  - Json
  - Multiline
  - Netflow
  - and many more...
- **Filters** : processing messages
  - Date – Date format
  - Grok – Regular expression based extraction
  - Mutate – Change data type
  - and many more...
- **Output** : storing the structured message
  - Elasticsearch
  - MongoDB
  - Email
  - • and many more...

## Exemple 1 : JDBC

---

```
1 # file: JDBC-In.conf
2 input {
3   jdbc {
4     # MySQL jdbc connection string to our database, elastic
5     jdbc_connection_string => "jdbc:mysql://localhost:3306/elastic"
6     # The user we wish to execute our statement as
7     jdbc_user => "root"
8     jdbc_password => ""
9     # The path to our downloaded jdbc driver
10    jdbc_driver_library => "C:\\prog\\serveurs\\logstash-2.3.1\\lib\\mysql-connector-java-5.1.31.jar"
11    # The name of the driver class for MYSQL
12    jdbc_driver_class => "com.mysql.jdbc.Driver"
13    # our query
14    statement => "SELECT * from contacts"
15  }
16 }
17 output {
18   stdout { codec => rubydebug }
19 }
```

## Exemple 2 : Twitter

---

```
1 input {
2   twitter {
3     consumer_key => "6wipccMnS0QqK7hSctSgarW0H"
4     consumer_secret => "AefAr0vBrQS5eJUHCHY3RFYniqEGrXA5i4CRV0pbKFeLSeNBvX"
5     oauth_token => "724270252239007744-EG9XtDBWx1b7f5Wmsb5bZ5KgJ5BxxIX"
6     oauth_token_secret => "djdAyER9CBHD5xnI6LmK02LTQYcQkdBOAjZzSljeCJXJl"
7     keywords => ["java","oracle","spring","hibernate"]
8     full_tweet => "true"
9   }
10 }
11 output {
12   elasticsearch {
13     index => "twitter"
14     document_type => "realtime"
15   }
16   stdout {}
17 }
```

## Exemple 3 : csv

---

```
1 input {
2   file {
3     path => "C:/prog/serveurs/logstash-2.3.1/data/quotesprice.csv"
4     start_position => "beginning"
5     ignore_older => 0
6   }
7 }
8 filter {
9   csv {
10    columns => ["date_of_record", "open", "high", "low", "close", "volume", "adj_close"]
11    separator => ","
12  }
13  date{
14    match => ["date_of_record", "yyyy-MM-dd"]
15    target => "@timestamp"
16  }
17  mutate {
18    convert => ["open", "float"]
19    convert => ["high ", "float"]
20    convert => ["low ", "float"]
21    convert => ["close ", "float"]
22    convert => ["volume", "integer"]
23    convert => ["adj_close", "float"]
24  }
25 }
26 output {
27   elasticsearch {
28     hosts => ["localhost"]
29     index => "tweeter"
30   }
31   stdout {}
32 }
```

# Disposition de titre et de contenu avec liste

---

**Question ?**



# Disposition de titre et de contenu avec liste

---

**TP**

**Lab\_13\_Logstash**



# MODULE 6

Kibana

# Qui utilise la stack ELK ?

---

## ▪ Exemples d'usage

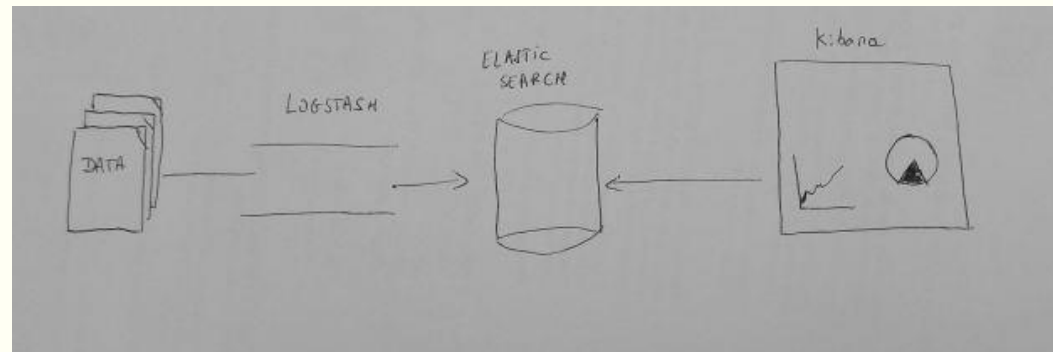
- Traitement de fichiers de logs applicatifs en temps réel
- Utilisation dans un processus d'ETL pour de l'informatique décisionnelle
- Supervision d'un parc informatique
- Configuration Management DataBase = Base de données regroupant les machines d'un parc informatique
- IOT Affichage des données issues des objets connectés
- ...



# Processus du traitement d'une donnée

---

- Récupération du fichier source (Rsyslog, fichier texte, base de données ...)
- Logstash (Processus ETL)
  - Filtrage des données
  - Ajout de champs
  - Redirection des données traitées
- Elasticsearch (Stockage des données)
- Kibana (Représentation des données)



# Kibana

---

- Kibana est une plateforme opensource de visualisation des données, conçue pour fonctionner conjointement avec Elasticsearch.
- **L'outil permet ainsi de construire graphiquement des requêtes sur les données, tout en proposant un affichage en temps réel de leurs résultats.**
- Puis, il est possible de construire des graphiques, tels que des histogrammes, des courbes, des secteurs... Chaque graphique est ensuite enregistrable, maintenable et chargeable.
- **Kibana permet également de construire des Dashboard personnalisés.** Ceux-ci décrivent graphiquement l'états de plusieurs données en temps réels. En effet, il est possible de configurer l'application pour que les données soit rafraîchies automatiquement.

# Kibana

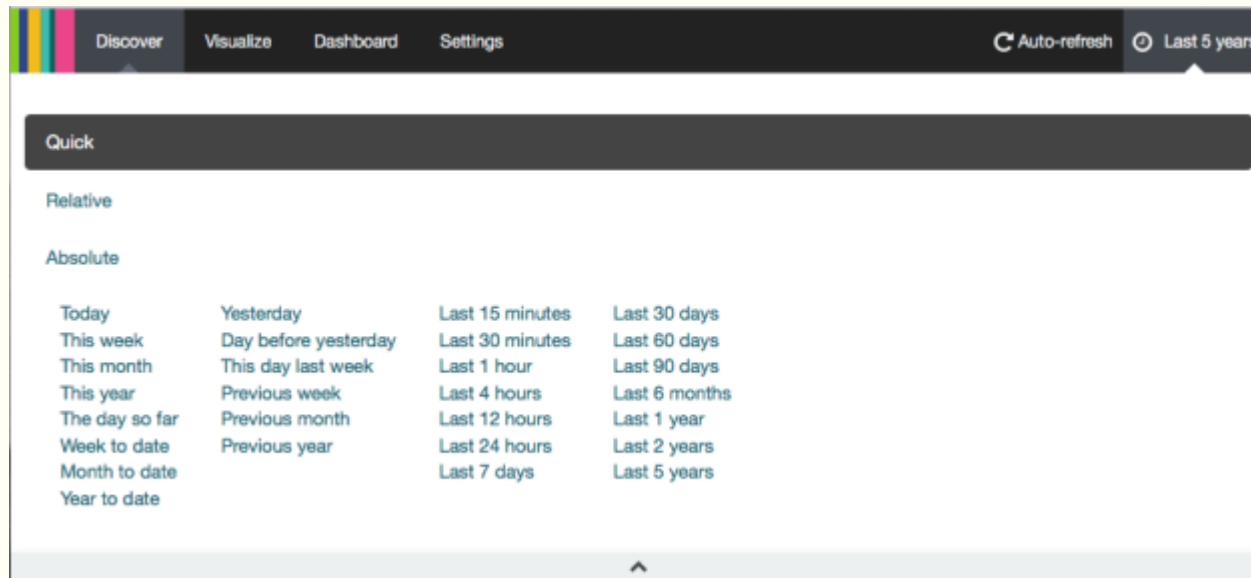
---

- Kibana est un outil graphique qui permet de requêter une base elasticsearch et de réaliser des graphes à partir des données qu'elle contient.
- Kibana se compose de plusieurs onglets :
  - **Discover**
  - **Visualize**
  - **Dashboard**
  - **Settings**

# Discover

---

- La page Discover permet:
  - Un accès à chaque document dans chaque index qui correspond au modèle d'index sélectionné.
  - De soumettre des requêtes de recherche, filtrer les résultats de recherche et afficher les données du document.
  - Enfin si un champ de temps est configuré pour le modèle sélectionné, la distribution des documents dans le temps est affichée dans un histogramme en haut de la page.



# Discover

---

- Il est possible de restreindre facilement la plage de recherche temporelle à l'aide du bouton à l'extrême droite de la barre de menu.
- De base, la valeur est configurée à 15 minutes



- Les recherches dans cette page se font via une syntaxe basée sur le langage Lucene.



# Visualize

---

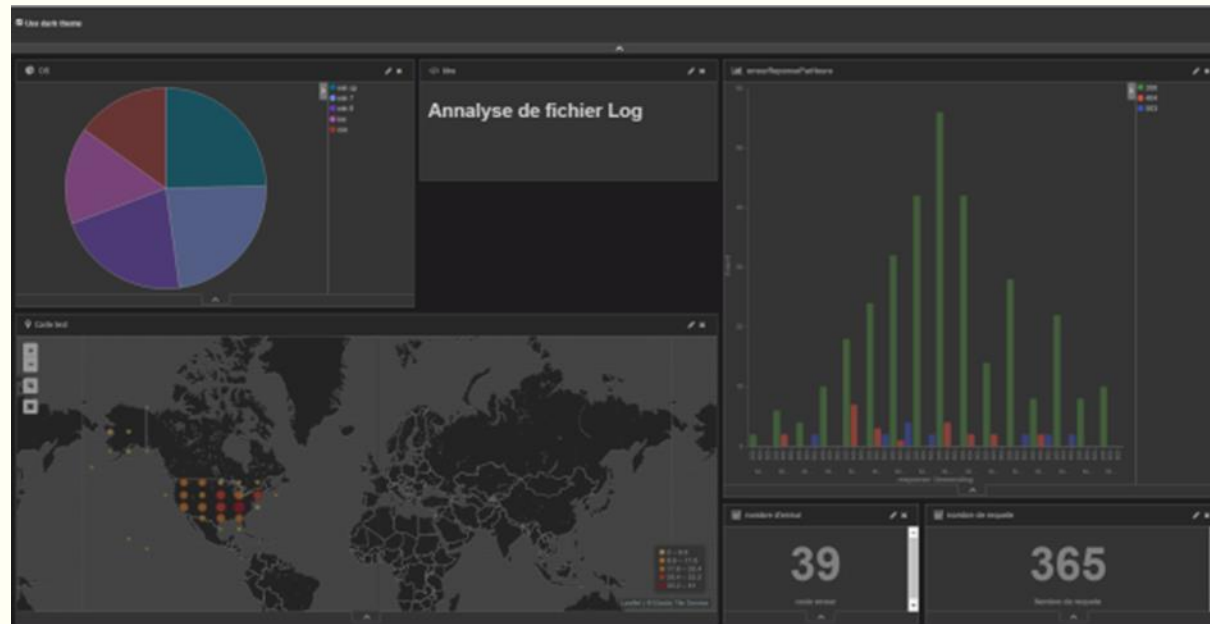
- La page Visualize permet:
  - De créer des graphes à partir des données contenues dans une base Elasticsearch
  - De créer rapidement un rendu visuel sur une série d'agrégations réalisée au préalable sur Elasticsearch
  - De réaliser des graphes à partir d'une recherche enregistrée dans Discover



# Dashboard

---

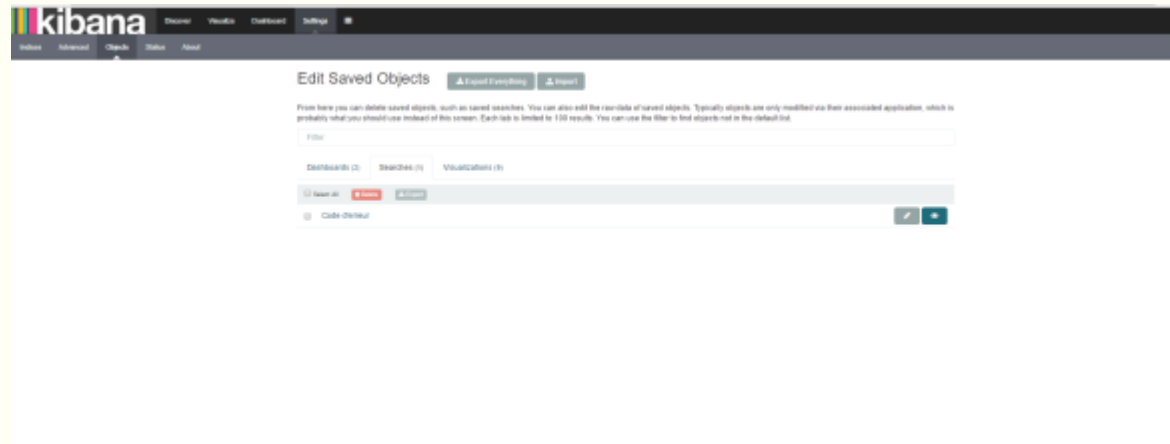
- La page Dashboard permet:
  - D'organiser et afficher une collection de graphe réalisée au préalable sur la page Visualize
  - D'obtenir une Url de partage de ce Dashboard.
  - d'enregistrer une mise à jour automatique et une période pour le dashboard.



# Settings

---

- La page Settings permet:
  - D'ajouter un index via l'onglet indices
  - D'accéder aux paramètres avancés de Kibana via l'onglet advanced
  - d'avoir une vue d'ensemble des différents éléments déjà enregistrés (dashboard, graphe, ...) via l'onglet objects



# Disposition de titre et de contenu avec liste

---

**Question ?**

# Disposition de titre et de contenu avec liste

---

**TP**

**Lab\_14\_Kibana**



# MODULE 7

Cluster

# Définition

---

## Un Nœud :

- est une instance d'Elasticsearch en cours d'exécution
- est dans un cluster
- communique avec les autres nœuds du cluster

Noeux 1



Noeux 2



Noeux 3



Figure: Cluster simple avec 3 noeuds vides

# Définition

---

## Un Noeud Maître :

- est un noeud élu
- gère les changements dans le cluster :
  - création ou suppression d'un index
  - ajout ou suppression d'un nœud du cluster

Noeux 1 - Master



Noeux 2



Noeux 3



Figure: Cluster simple avec 3 noeuds vides et 1 maître



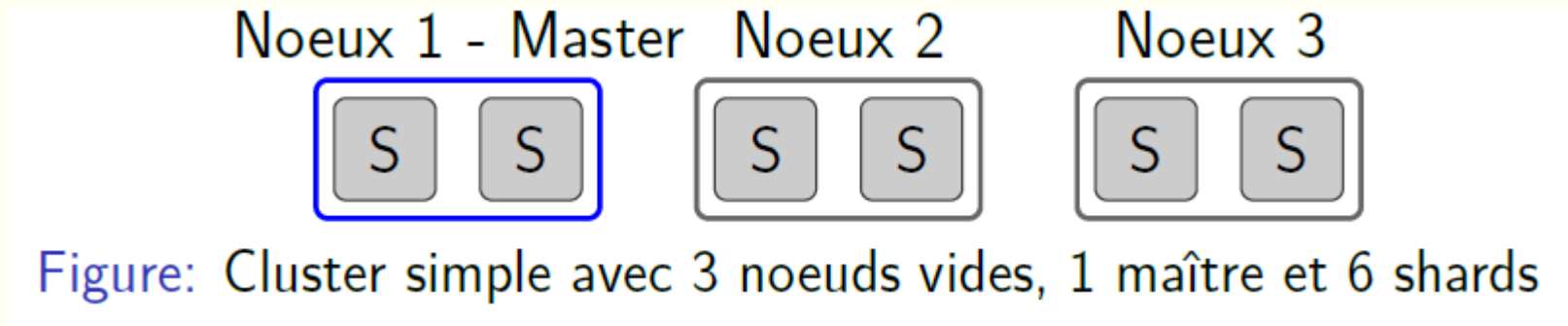
# Définition

---

## Un Shard:

- est une "unité de travail" bas niveau
- est une seule instance de Lucene
- est un moteur de recherche complet

Nos documents sont stockés et indexés dans les Shards, mais nous ne nous adressons pas directement à eux : nos applications s'adressent à un **index**.



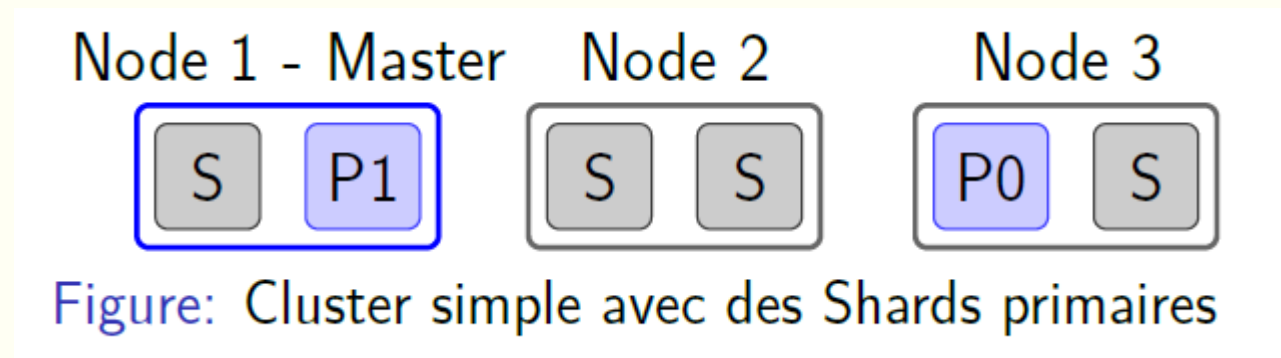
# Définition

---

## Un Shard primaire :

- contient tous les documents dans un index
- peut avoir d'autres Shards primaires pour séparer les données (similaire au RAID 0)

Le nombre de Shard primaire pour un index est fixé au moment de la création de l'index.



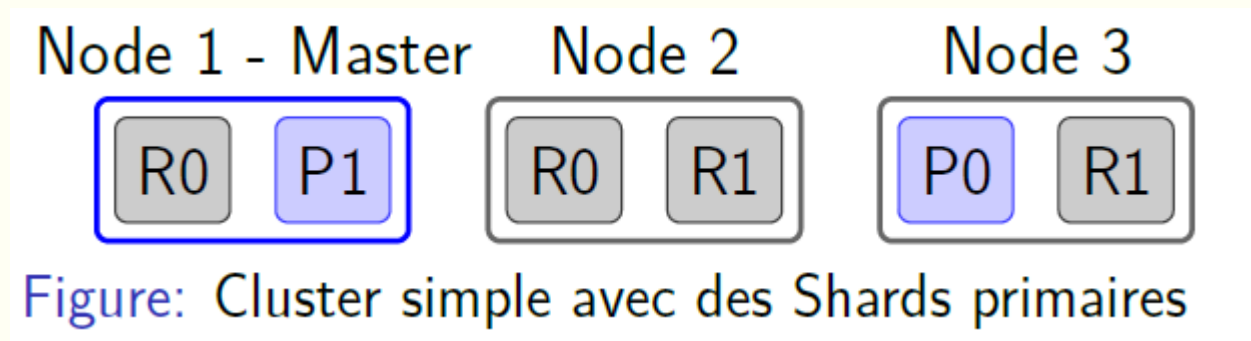
# Définition

---

Un Shard replica :

- est une copie d'un Shard primaire (similaire au RAID 1)
- est utilisé pour fournir des copies redondantes des données
- est utilisé pour répondre aux requêtes de lecture comme chercher un document

Le nombre de Shard replica peut être changé à n'importe quel moment.



# Statut du cluster

---

Pour savoir le statut du cluster :

```
curl -XGET http :// localhost :9200/ _cluster/health?pretty
```

Le champ status donne une indication global sur le fonctionnement du cluster :

- **vert** : Tous les Shards primaires et replicas sont actifs (Le cluster fonctionne et la tolérance aux pannes est assurée).
- **jaune** : Tous les Shards primaires sont actifs, mais des Shards replicas ne sont pas tous actifs (Le cluster fonctionne mais si un noeud tombe la tolérance aux pannes n'est pas assurée).
- **rouge** : Des Shards primaires sont inactifs (Le cluster n'est pas fonctionnel).

# Gestion des Shards

---

- Créons un index megacorp en spécifiant que nous voulons 3 Shards primaires et 1 Shard replica (pour chaque primaire) :

```
curl -XPUT 'http://localhost:9200/megacorp' -d '{
  "settings" : {
    "number_of_shards" : 3,
    "number_of_replicas" : 1
  }
}'
```

Node 1 - Master

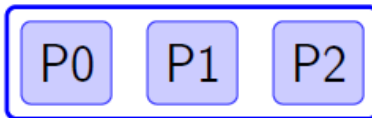


Figure: 1 noeuds avec 3 shards primaires

- Dans cet état le statut du cluster est "jaune" car les Shards replicas ne peuvent pas être lancés.

# Tolérance aux pannes

---

- 1 Nœud => Un point de défaillance
- La solution est simple : lancer un nouveau Nœud
- Le nouveau Nœud rejoindra automatiquement le cluster s'il a le même nom de cluster (cluster.name).

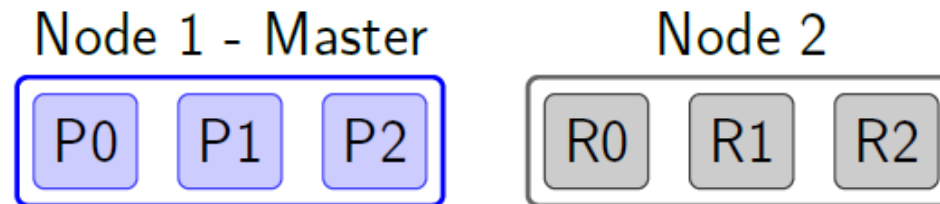


Figure: 2 noeuds avec 3 shards primaires et 1 shard replica pour chaque shard primaire

- Le statut cluster est maintenant "vert".

# Disposition de titre et de contenu avec liste

---

**Question ?**

# Disposition de titre et de contenu avec liste

---

**TP**

**Lab\_15\_Kibana**