ebiznext

ELASTICSEARCH

Mohamed Hedi Abidi @mh_abidi
David AZRIA @David_AZR

Introduction

ElasticSearch est un moteur de recherche Open Source (Apache 2). Il utilise la librairie Apache Lucene et indexe les données sous forme de documents. Sa mise en place est facile et rapide.

Il possède des avantages indéniables dont :

- recherche en quasi temps réel
- scalable, Haute disponibilité
- automatiquement sauvegardé et répliqué
- API REST

La dernière version en date est la 1.0.1.

http://www.elasticsearch.org/overview/elkdownloads/

Les bases (1)

La distribution contient :

- bin : script de lancement elasticsearch.bat (windows) et elasticsearch (linux)
- config : contient les fichiers elasticsearch.yml et logging.yml
- data : répertoire par défaut des données indexées
- lib : contient les librairies
- logs : fichiers logs

Les bases (2)

Démarrage d'ElasticSearch :

```
$./bin/elasticsearch
# mode daemon
$./bin/elasticsearch -d
```

Variables d'environnements:

- EZ_HEAP_SIZE : JMV Heap Size
- EZ_JAVA_OPTS : options de la JVM

Les bases (3)

Configuration:

config/elasticsearch.yml

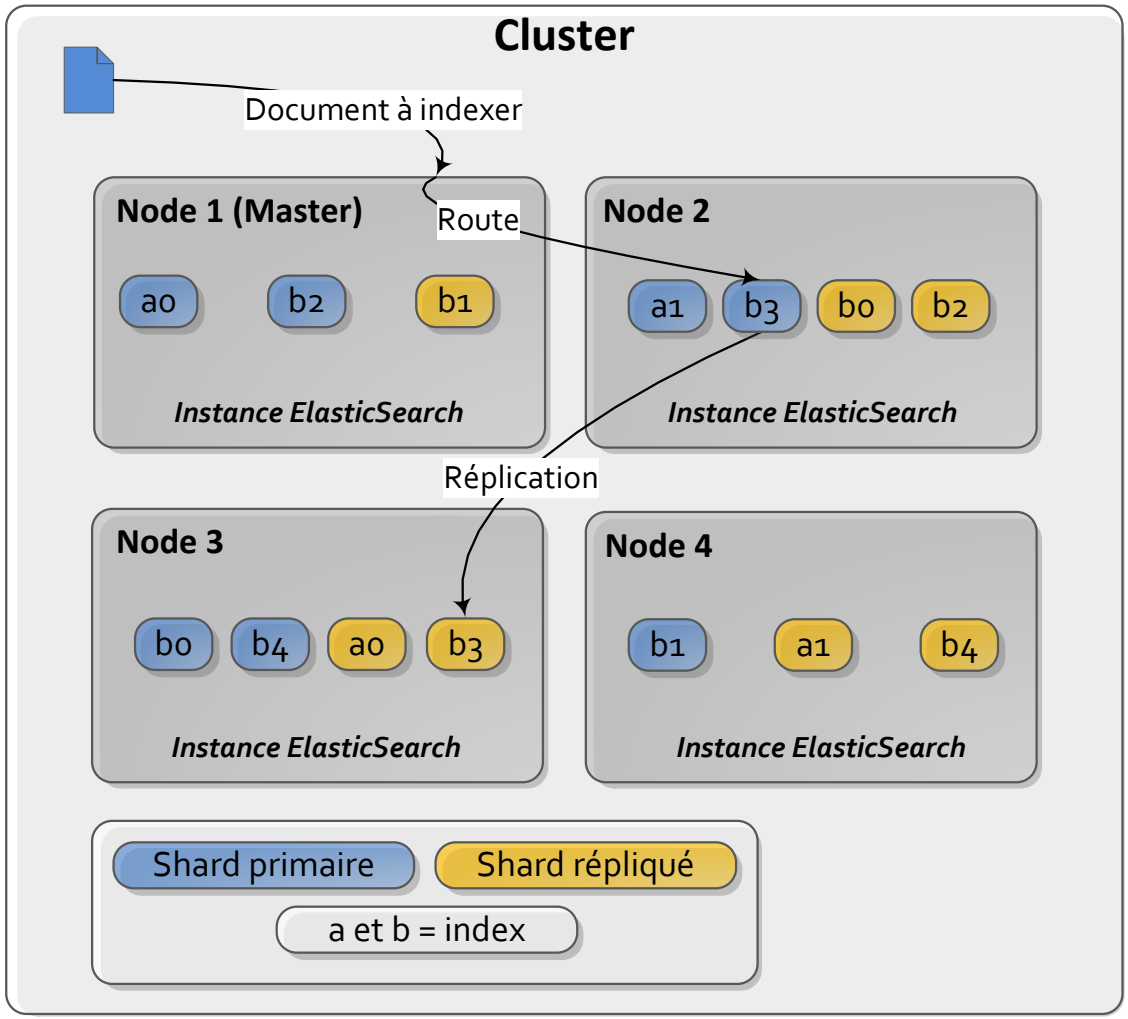
```
cluster.name: mycluster
http.port: 9200
transport.port: 9300
network.host: localhost
discovery.zen.ping.multicast.enabled: false
discovery.zen.ping.unicast.hosts: ["host1", "host2:port"]
```

Les bases (4)

Fonctionnement cluster:

- Nœud : Correspond à une instance ElasticSearch
- Cluster : Il est composé d'un à plusieurs Node. Un nœud maître est choisi, il sera remplacé en cas de défaillance. Par défaut, un nœud utilisera le multicast pour trouver un cluster du même nom. Mais il est préférable d'utiliser l'unicast dans des réseaux sécurisés.
- Index : Espace logique de stockage de documents de même type, découpé sur un à plusieurs Primary Shards et peut être répliqué sur zéro ou plusieurs Secondary Shards.
- Shard : Correspond à une instance Lucène.
- Primary Shard : 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 Shard : Il s'agit des Shards répliqués. Il peut en avoir zéro à plusieurs par Primary Shard.
- Routage : Permet de sélectionner un Primary Shard pour indexer le document. Il est choisi en hachant la valeur de l'ID du document ou de l'ID du document parent pour s'assurer que les documents parents et enfants soient stockés sur le même Primary Shard.

Les bases (4)


Cluster
Document à indexer
Route
Node 1 (Master)
a0
b2
b1
Instance ElasticSearch
Node 2
a1
b3
b0
b2
Instance ElasticSearch
Réplication
Node 3
b0
b4
a0
b3
Instance ElasticSearch
Node 4
b1
a1
b4
Instance ElasticSearch
Shard primaire
Shard répliqué
a et b = index

API REST (1)

EllasticSearch offre une API REST permettant d'effectuer tous types d'opération. Il supporte les méthodes HTTP (GET, PUT, POST et DELETE).

```
curl -XPUT 'http://localhost:9200/[index]/[type]/[id]/[action]'
```

- Index : Nom de l'index
- Type : Nom du type du document
- Id : ID du document
- Action : Action à effectuer

API REST (2) : Indexation de document

Ajout ou modification d'un document :

```
curl -XPUT 'http://localhost:9200/biblio/livres/1' -d '{
    "titre": "Shining",
    "auteur": "Stephen King",
    "genre": "Fantastique",
    "date_parution": "27/01/1977"
}'
```

Avec création de l'ID automatiquement :

```
curl -XPOST 'http://localhost:9200/biblio/livres' -d '{
    "titre": "Shining",
    "auteur": "Stephen King",
    "genre": "Fantastique",
    "date_parution": "27/01/1977"
}'
```

API REST (3) : Indexation de document

Réponse :

HTTP 200 : Document modifié

HTTP 201 : Document créé

HTTP 409 : Document existe déjà dans le cas d'une opération de création 'http://localhost:9200/biblio/livres/1/_create'


```
{
  "ok": "true",
  "_index": "biblio",
  "_type": "livres",
  "_id": "1",
  "_version": "1"
}
```

API REST (4) : Types de données

- string, integer, long, float, double, boolean, datetime, binary (base64)
- Array, Object
- geo_point, geo_shape, ip

API REST (4) : Méta data d'un document

Champ	Activé par défaut	Description
_id		Id du document
_type		Type du document
_source	Oui	Document original qui a été indexé
_all	Oui	Indexes de toutes les valeurs des champs du document
_timestamp	Non	Timestamp du document
_ttl	Non	Temps d'expiration du document
_size	Non	Taille du document original (_source)

API REST (5) : GET

```
curl -XGET 'http://localhost:9200/biblio/livres/1'
```

Réponse :

HTTP 200 : OK

HTTP 404 : NOT FOUND


```
{
  "_index": "biblio",
  "_type": "livres",
  "_id": "1",
  "_version": 1,
  "found" : true,
  "_source" :  {
              "titre": "Shining",
              "auteur": "Stephen King",
              "genre": "Fantastique",
              "date_parution": "28/01/1977"
   }
}
```

API REST (6) : UPDATE

```
curl -XPOST 'http://localhost:9200/biblio/livres/1/_update ' -d ' {
        "doc"  :  {
                "date_parution": "28/01/1977"
        }
```

Si le document existe, il est fusionné,

API REST (7) : Version

Un numéro de version est assigné à la création d'un document. Ce numéro de version est incrémenté pour chaque opération de réindexassions, modifications ou de suppressions. Une opération de modification est validé si le numéro de version de la requête correspond au numéro de version indexé.

API REST (8) : SEARCH

```
curl -XPOST 'http://localhost:9200/index1/type1/_search' –d { }
curl -XPOST 'http://localhost:9200/index1/type1,type2/_search' –d { }
curl -XPOST 'http://localhost:9200/index1,index2/type1/_search' –d { }
curl -XPOST 'http://localhost:9200/_all/type1/_search' –d { }
```

Requête :

```
curl -XGET 'http://localhost:9200/biblio/_search?q=Shining'
```

API REST (8) : SEARCH

Réponse :


```
{               "took":8,
                "timed_out":false,
                "_shards": {
                                "total":5,
                                "successful":5,
                                "failed":0
                },
                "hits": {
                                "total":1,
                                "max_score":0.09289607,
                                "hits":[
                                                {
                                                                "_index":“ebiznext",
                                                                "_type":"formation",
                                                                "_id":"3",
                                                                "_score":0.09289607,
                                                                "_source" : {
                                                                                "titre": "Shining",
                                                                                "auteur": "Stephen King",
                                                                                "genre": "Fantastique",
                                                                                "date_parution": "28/01/1977"
}]]]}
```

Analyse de texte (1)

```
curl -XPUT 'http://localhost:9200/twitter/tweet/1' -d '{
    "user": "david",
    "message": "C'est mon premier message de la journée !",
    "postDate": "2014-04-03T15:23:56",
    "priority":  2 ,
    "rank": 10.2
}'
```

Les deux requêtes ci-dessous ne retourne aucun document.

```
curl -XGET 'http://localhost:9200/twitter/tweet/_search?q=est&pretty=true'
```

```
curl -XGET 'http://localhost:9200/twitter/tweet/_search?q=journée&pretty=true'
```

Analyse de texte (2)

Comme on peut le voir ci-dessous, ElasticSearch type automatiquement les champs. Mais le document n'a pas été trouvé lorsque l'on effectue une recherche car il utilise l'Analyseur standard qui ne colle pas à notre besoin.

```
curl -XGET "http://localhost:9200/twitter/_mapping?pretty=true"
{
  "twitter" : {
    "mappings" : {
      "tweet" : {
        "properties" : {
          "message" : { "type" : "string" },
          "postDate" : { "type" : "date", "format" : "dateOptionalTime“},
          "priority" : {"type" : "long"},
          "rank" : {"type" : "double“ },
          "user" : {"type" : "string"}
}}}}}
```

Analyse de texte (3)

```
curl -XGET ‘http://localhost:9200/twitter/_analyze?pretty=true’ -d "C'est"
{
  "tokens" : [ {
    "token" : "c'est",
    "start_offset" : 0,
    "end_offset" : 5,
    "type" : "<ALPHANUM>",
    "position" : 1
  } ]
}
```

```
curl -XGET "http://localhost:9200/twitter/_analyze?pretty=true" -d "journée"
{
  "tokens" : [ {
    "token" : "journée",
    "start_offset" : 0,
    "end_offset" : 5,
    "type" : "<ALPHANUM>",
    "position" : 1
  } ]
}
```

Analyse de texte (4)

```
curl -XGET
"http://localhost:9200/twitter/_analyze?tokenizer=letter&filters=asciifolding,lowercase&pretty=true" -d
"c'est"
{
  "tokens" : [
                    { "token" : "c", "start_offset" : 0, "end_offset" : 1, "type" : "word", "position" : 1 },
                    { "token" : "est", "start_offset" : 2, "end_offset" : 5, "type" : "word","position" : 2 }
                ]
}
```

```
curl -XGET
"http://localhost:9200/twitter/_analyze?tokenizer=letter&filters=asciifolding,lowercase&pretty=true" -d
"journée"
{
  "tokens" : [ {
    "token" : "journee",
    "start_offset" : 1,
    "end_offset" : 8,
    "type" : "word",
    "position" : 1
  } ]
}
```

Analyse de texte (5)

Configuration global :

```
curl -XPUT 'http://localhost:9200/twitter' -d '{
  "settings" : {
                "analyzer" : {
                                "myAnalyzer" : {
                                                "tokenizer" : "letter",
                                                "filter" : ["asciifolding", "lowercase", "mySynonyms"]
                                }
                                "filter" : {
                                                "mySynonyms" : {
                                                                "type" : "synonym",
                                                                "synonyms" : ["premier, initial"]
                                                }
                                }
                }
  }
}'
```

Analyse de texte : Tokenizer et Filter

Un Analyser est composé d'un Tokenizer et de zéro à plusieurs Filters,

- Tokenizer : Il permet de scinder un texte en plusieurs éléments. (standard, keyword, whitespace, pattern, letter, custom, etc.)

- Filter : Les filtres peuvent modifier ou supprimer Tokens. (lowercase, synonym , asciifolding, nGrams, custom, etc.)

Mappings

- Elasticsearch est ‘sans schéma’
  - Configurer seulement au besoin
  - Mapping dynamique créé pour les types non configurés
  - Mapping dynamique : appliquer les mappings par défaut pour les informations extraites du document JSON.
- Mapping = Définir comment les documents sont analysés et indexés:
  - Les types des les champs/objets du document.
  - Les relations entre les documents.
  - La gestion des méta-données du document.
  - La définition de la pertinence par champ/document (boosting)

Mappings : Pourquoi?

- Adapter l'analyse de données à un domaine métier spécifique
- Adapter la recherche aux utilisateurs :
  - terminologies spécifiques, multi-langages.
- Certaines fonctionnalités l'exigent
  - Tri,
  - Agrégation
  - Hilighting

Mappings API

- Création de mappings lors de la c… …ndex :

Index

```
curl -XPOST localhost:9200/crunchbase -d '{
   "settings" : {
      "number_of_shards" : 1
   },
   "mappings" : {
      "company" : {
         "_source" : { "enabled" : false },
         "properties" : {
            "name" : { "type" : "string", "analyzer" : "standard" }
         }
      }
   }
}'
```

type du document

Mapping des champs

- Récupération du mapping :

```
Curl –XGET 'localhost:9200/crunchbase/_mapping/company'
```

Mappings : Types

Type	Attribut	Description
String	Index	analyzed : indexé, analysé. not_analyzed : indexé mais pas analysé. no : non indexé.
	Analyzer	définit l'analyseur qui sera utilisé lors de l'indexation ou de la recherche.
Nombres	ignore_malformed	ignorer un nombre mal formaté. valeur par défaut : false
	Coerce	convertit les strings en nombres. valeur par défaut : false
Date	Format	format attendu. exemple : yyy/MM/dd
	ignore_malformed	ignorer les valeurs mal formatées

Multi-fields

- Le multi-field permet d'indexer un champ du document sous plusieurs formes
  - cas d'utilisation : utiliser plusieurs analyseurs pour le même champ

```
"tweet" : {
  "properties" : {
    "name" : {
      "type" : "multi_field",
      "fields" : {
        "name":{"type":"string","index":"analyzed"},
        "untouched":{"type":"string","index":"not_analyzed"}
      }
    }
  }
}
```

Relations (1) : Les types 'nested' vs parent/fils

- Nested : Les objets de la relations sont stockés dans le même document

```
{
    "company" : {
      "properties" : {
        "product" : {
          "type" : "nested",
          "properties": {
            "name" : {"type": "string", "index": "not_analyzed"},
            "count" : {"type": "integer"}
          }
        }
      }
    }
}
```

- Parent/fils : le pères et ses fils sont stockés dans des documents séparés

```
{
    "employee" : {
      "_parent" : { "type" : "blog " }
    }
}
```

Relations (2) : Les types 'nested' vs parent/fils

- **Les types 'nested'**
  - Lecture rapide : les documents sont stockés dans le même block Lucene.
  - Mise à jour est couteuse : ES réindexe tout le document si le document 'nested' est mis à jour
  - Moins flexible : mieux adapté pour les données qui ne changent pas souvent
- **Parent/fils**
  - Moins performant : Père et fils sont stockés dans des documents séparés (mais dans le même shard)
  - Plus de mémoire : Les IDs sont chargés en mémoire.
  - Flexible : Parent et fils sont modifiés séparément, pas de re-indexation

Recherche : Query DSL

- Le query DSL (Domain Specific Language) permet d'exprimer des définitions complexes de requêtes.
- Il existes deux types de query DSL :
  - Requêtes :
    - Utilisé pour la recherche full text
    - Le résultat dépend d'un score attribué aux documents
    - Pas d'utilisation de cache
  - Filtres :
    - Pas de manipulation de scores.
    - Utilisation de cache.

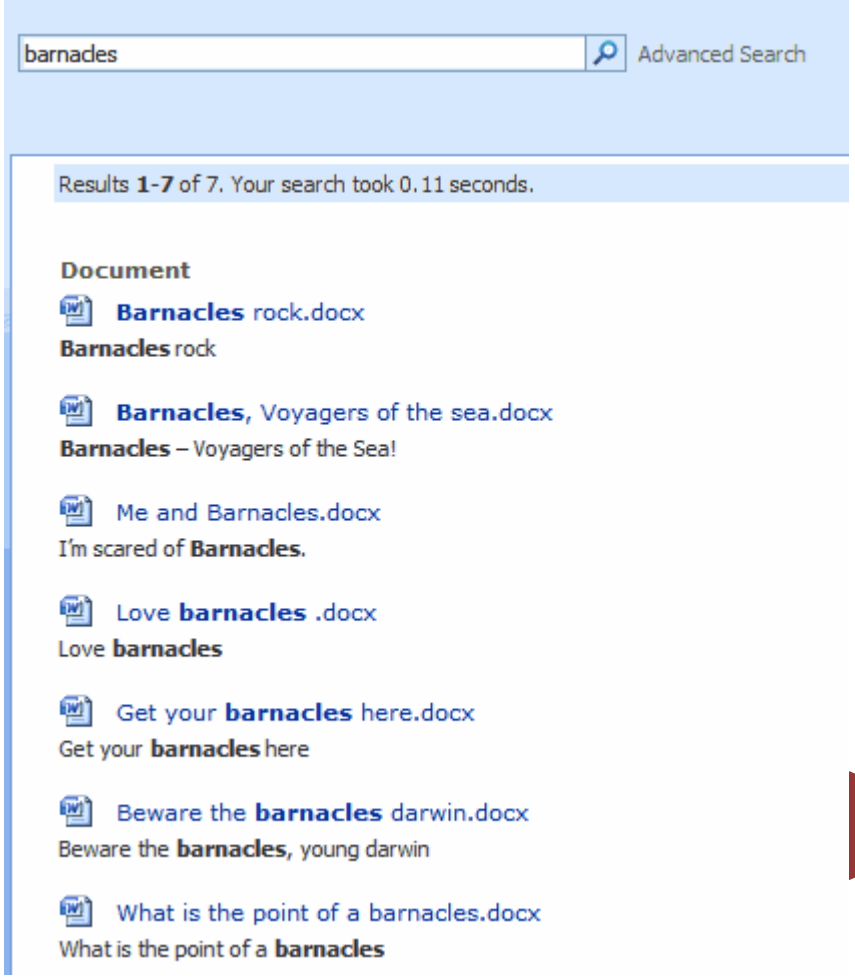
Recherche : Exemples

Requêtes	Filtres
Boosting Query Constant Score Query	And Filter Bool Filter Not Filter Or Filter
Match Query Multi Match Query	Term Filter Terms Filter
Match All Query	GeoShape Filter
Has Child Query Has Parent Query Nested Query	Has Child Filter Has Parent Filter Nested Filter
	Regexp Filter Script Filter

HIGHLIGHT DES RÉSULTAT DE LA RECHERCHE

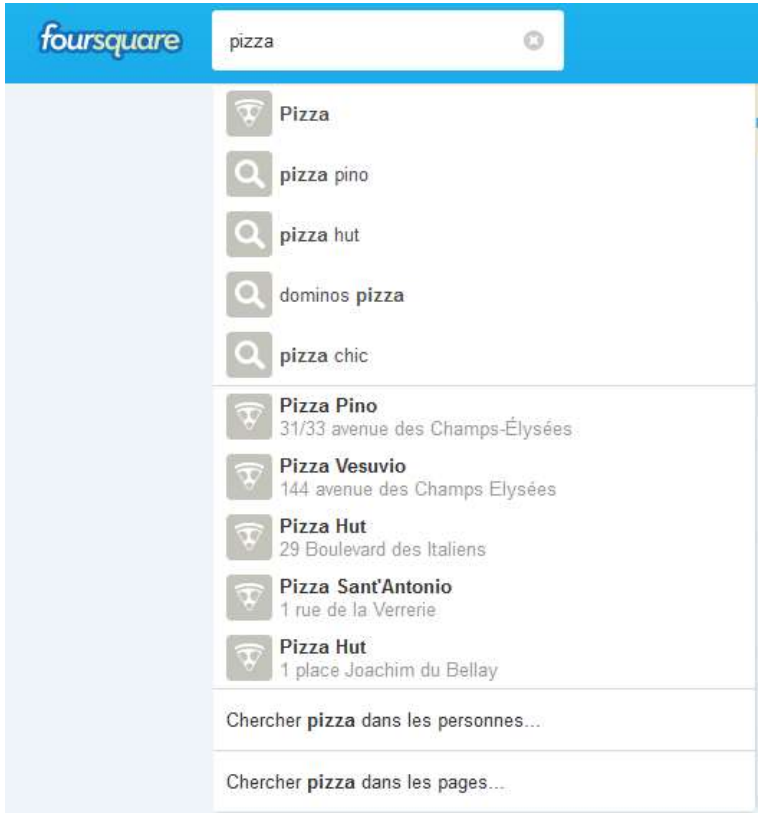
- Highlight : montrer des extraits du texte correspondant au résultat de la requête

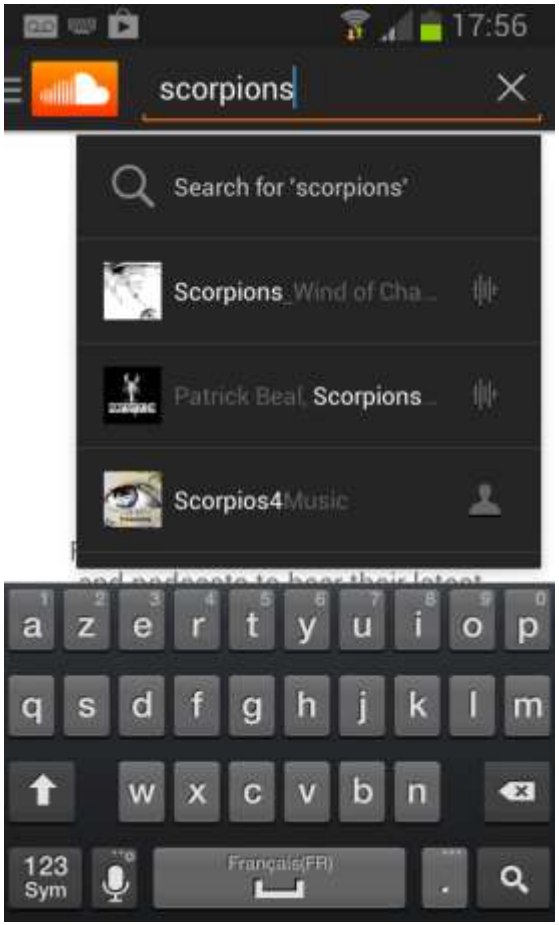
```
"query" : {
  "match" : {
      "title" :   "branches"
  }
},
  "highlight" : {
    "fields" : {
      "content" : {}
    }
  }
```



Suggestion

- Suggestion : suggérer des termes similaires à la recherche.





Percolator

- Le percolateur permet de savoir quelle requête correspond à un document donné
  - Fonctionnement inversé de la recherche
- Cas d'utilisation :
  - Alertes et monitoring
- Enregistrement d'une requête

```
curl -XPUT 'localhost:9200/my-index/.percolator/1' -d '{
   "query" : {
      "match" : {
         "message" : "bonsai tree"
      }
   }
}'
```

PERCOLATOR

- Commande permettant de savoir quelle requête correspond au document donné

```
curl -XGET 'localhost:9200/my-index/message/_percolate' -d '{
   "doc" : {
      "message" : "A new bonsai tree in the office"
   }
}'
```

- Réponse : l'id de la requête qui 'matche' le document

```
{"took" : 19,
   "_shards" : {
      ...
   },
   "total" : 1,
   "matches" : [{
         "_index" : "my-index",
         "_id" : "1"
      }]}
```

Plugins

- ElasticSearch dispose de son propre système de plugins qui permet d'enrichir les fonctionnalités de base. Plusieurs plugins sont développés par Elasticsearch ou par la communauté. On distingue :
  - Les analyseurs : Ensemble de filtres et séparateurs pour supporter les langues.
  - Les rivières : Les rivers (rivières) ont pour rôle d'injecter les données provenant de divers sources de données dans ElasticSearch. Il existe 4 rivières standards : CouchDB, RabbitMQ, Twitter et Wikipedia.
  - Il y a également plusieurs autres rivières développées par la communauté qui permettent d'injecter les données provenant de MongoDB, Neo4J, flux RSS, Git…
  - Support des scripts : Clojure, Groovy, Javascript…

Q&R