# NFE204

## Recherche d'Information : l'indexation

## S2: indexation dans ElasticSearch

Auteurs : Raphaël Fournier-S'niehotta, Philippe Rigaux
(fournier@cnam.fr, philippe.rigaux@cnam.fr)

EPN Informatique
Conservatoire National des Arts & Métiers, Paris, France

# Plan du cours

# Indexation : motivation

- Par défaut, ElasticSearch essaie d'analyser les documents qu'on lui fournit.
- Pour chaque champ, il tente de trouver le type des données dont il s'agit (entier, date, IP, texte en français, en anglais, etc.)
- Si cela suffit pour des données simples (texte brut), on souhaite généralement faire mieux
- "on connaît toujours mieux ses données qu'ElasticSearch"...
- Il faut donc entrer dans le détail de la configuration et dans approfondir notre connaissance des techniques de RI

# Schéma de l'index

- Le schéma donne la liste de tous les champs d'un document
- Nombreuses options :
  - type (numérique, entier)
  - possibilité de calcul de la valeur du champ à partir d'un autre
  - traitements divers sur les valeurs du champ

# Notre premier document


```
{
  "title": "Vertigo",
  "year": 1958,
  "genre": "drama",
  "summary": "Scottie Ferguson, ancien inspecteur de police, est sujet au vertige depuis qu'il a
              vu mourir son collègue. Elster, son ami, le charge de surveiller sa femme,
              Madeleine, ayant des tendances suicidaires. Amoureux de la jeune femme Scottie ne
              remarque pas le piège qui se trame autour de lui et dont il va être la victime... ",
  "country": "DE",
  "director":         {
    "_id": "artist:3",
    "last_name": "Hitchcock",
    "first_name": "Alfred",
    "birth_date": "1899"
  },
  "actors": [
  {
    "_id": "artist:15",
    "first_name": "James",
    "last_name": "Stewart",
    "birth_date": "1908",
    "role": "John Ferguson"
  },
  {
    "_id": "artist:282",
    "first_name": "Arthur",
    "last_name": "Pierre",
    "birth_date": null,
    "role": null
  }
  ]
}
```

# Schéma

```
{
  "mappings": {
    "movie": {
      "_all":       { "enabled": true  },
      "properties": {
        "title":    { "type": "string"  },
        "year":     { "type":   "date", "format": "yyyy" },
        "genre":      { "type": "string" },
        "summary": { "type": "string"},
        "country": {"type": "string"},
        "director":     {
          "properties": {
            "_id": { "type":"string" },
            "last_name": { "type":"string"},
            "first_name": { "type":"string"},
            "birth_date": { "type": "date", "format":"yyyy"}
          }},
        "actors": {
          "type": "nested",
          "properties": {
            "_id": { "type":"string"}
            "first_name": { "type":"string"}
            "last_name": { "type":"string"}
            "birth_date": { "type": "date", "format":"yyyy"}
            "role": { "type":"string"}
          }
        }
      }
    }
```

# Détaillons le schéma

- la liste des champs, dans l'élement "mappings"
- le champ `_all` est spécial, tout y est par défaut concaténé, de façon à permettre une recherche sur tous les champs
- la liste des **types de champ**, dans l'élément "properties".

# Définition des types et de la clé

- Chaque champ utilisé dans un document doit apparaître dans un des élements
- Sinon, heureusement, ElasticSearch essaie de deviner et propose un "Dynamic Mapping"
- ElasticSearch fournit tout un ensemble de types pré-définis qui suffisent pour les besoins courants; on peut associer des options à un type
- Les options indiquent d'éventuels traitements à appliquer à chaque valeur du type avant son insertion dans l'index
- ex: type **string** (remplacé par **text** depuis 5.0)
  - on lui définit un "analyzer" "standard"
  - se charge de découper le texte en **tokens** pour une recherche plein-texte (détails plus tard)
  - retenir : cela permet d'indexer chacun des mots, et donc de faire des recherches sur toutes les combinaisons de mots

# Définition des champs

```
{
        "genre": {
                "type": "string",
                "index": "no",
                "store": "true",
                "index_options": "offsets"
},
```

- Les attributs de l'élement JSON caractérisent le champ
- Le **nom** et le **type** sont les informations de base
- Ensuite, divers attributs (souvent optionnels) :
  - **index** indique simplement si le champ peut être utilisé dans une recherche;
  - **store** indique que la **valeur** du champ est **stockée** dans l'index, et qu'il est donc possible de récupérer cette valeur comme résultat d'une recherche, **sans avoir besoin de retourner à la base principale**; en d'autres termes, "store" permet de traiter l'index **aussi** comme une base de données;
  - remarquez la structure pour le réalisateur (**directors**)
  - enfin, **nested** pour les champs ayant plusieurs valeurs, soit, concrètement, un **tableau** en JSON; c'est le cas par exemple pour le nom des acteurs.

# Définition des champs

Les champs **index** et **store** sont très importants

Toutes les combinaisons de valeur sont possibles :

- **index=true**, **store=false**: on pourra interroger le champ, mais il faudra accéder au document principal dans la base documentaire si on veut sa valeur;
- **index=true**, **store=true**: on pourra interroger le champ, **et** accéder à sa valeur dans l'index;
- **index=false**, **store=true**: on ne peut pas interroger le champ, mais on peut récupérer sa valeur dans l'index;
- **index=false**, **store=false**: n'a pas de sens à priori; le seul intérêt est d'ignorer le champ s'il est fourni dans le document Solr.

# Définition des champs (suite)

Comment peut-on indexer un champ sans le stocker ?

- c'est notamment le cas pour les textes qui sont décomposés en **termes**: chaque terme est indexé indépendamment
- très difficile pour l'index de reconstituer le texte
- d'où l'intérêt de conserver ce dernier dans son intégralité, à part

C'est une question de compromis:

- **stocker** une valeur prend plus d'espace que **l'indexer**
- Dans la situation la plus extrême, on dupliquerait la base documentaire en stockant chaque document **aussi** dans l'index
- un stockage plus important dégrade les performances

# Champ par défaut, document original

Le champ `_source` contient le document, mais n'est pas indexé. Il est possible ainsi de récupérer la totalité du document passé à l'indexation. Ce comportement par défaut peut être désactivé, totalement ou partiellement.

Le squelette de schéma comprend un champ *par défaut*, le champ `_all`.

Par défaut, tous les champs sont copiés dans ce champ, analysé avec un découpage simple (espaces), puis indexé mais pas stocké. Cela permet de retrouver les documents contenant une valeur, sans savoir dans quel champ elle se trouve.

# Champ calculé

Il est possible de créer des champs `all` spécifiques, correspondants à des besoins précis, à l'aide d'une instruction `copy_to`

- Ainsi, le nom et le prénom peuvent être regroupés au sein d'un champ "nom complet".
- Cela permet aussi d'indexer certains champs plusieurs fois, de différentes façons.
- par exemple le contenu d'un titre est indexé comme une chaîne de caractères dans le champ **title**, et comme un texte "tokenisé" quand on le copie dans le champ **text**;

# Exemple de champ calculé

```
{
  "mappings": {
    "my_type": {
      "properties": {
        "first_name": {
          "type": "text",
          "copy_to": "full_name"
        },
        "last_name": {
          "type": "text",
          "copy_to": "full_name"
        },
        "full_name": {
          "type": "text"
        }
      }
    }
  }
}
```

# Schéma

On peut stocker le schéma dans un fichier json et créer l'index dans ElasticSearch comme ceci :

```
curl -XPUT 'localhost:9200/monindex' -H 'Content-Type: application/json' -d movie-schema-2.4.json
```

# Mise à jour de l'index

- Avec ElasticSearch, il est (plus) difficile de faire évoluer un schéma (qu'avec BDD classique)
- Sauf rares exceptions, il faut en général créer un nouvel index et réindexer les données
- ElasticSearch permet la copie d'un index vers l'autre avec l'API `reindex`

```
 curl -XPOST 'localhost:9200/_reindex?pretty' -H 'Content-Type: application/json' -d'
{
  "source": {
    "index": "nfe204"
  },
  "dest": {
    "index": "new_nfe204"
  }
}'
```

# Définir une chaîne d'analyse avec Elasticsearch

- Les analyseurs sont composés d'un tokenizer, et TokenFilters (0 ou plus)
- Le tokenizer peut être précédé de CharFilters
- les filtres (filter) examinent les tokens un par un et décident de les conserver, de les remplacer par un ou plusieurs autres;
- l'analyseur est une chaîne de traitement (pipeline) constituée de tokeniseurs et de filtres.

# Définir une chaîne d'analyse avec Elasticsearch

```
{
  "settings": {
    "analysis": {
      "filter": {
        "custom_english_stemmer": {
          "type": "stemmer",
          "name": "english"
        }
      },
      "analyzer": {
        "custom_lowercase_stemmed": {
          "tokenizer": "standard",
          "filter": [
            "lowercase",
            "custom_english_stemmer"
          ]
        }
      }
    }
  },
  "mappings": {
    "test": {
      "properties": {
        "text": {
          "type": "string",
          "analyzer": "custom_lowercase_stemmed"
        }
      }
    }
  }
}'
```

# tester les analyseurs

```
{
  "analyzer": "english",
  "text": "j'aime les couleurs de l'arbre durant l'été"
}
{
  "analyzer": "french",
  "text": "j'aime les couleurs de l'arbre durant l'été"
}
{
  "analyzer": "standard",
  "filter":  [ "lowercase", "asciifolding" ],
  "text": "j'aime les COULEURS de l'arbre durant l'été"
}
```