Structure des données

API : parler avec Elasticsearch

- API = *Application Programming Interface* : "Une **liste de fonctions** qu'on peut appeler de l'**extérieur** d'un logiciel"
- Elasticsearch a une **API REST JSON**
  - **REST** = basé sur le protocole HTTP et une hiérarchie cohérente de routes
  - **JSON** = les données de requêtes et de réponse (le BODY de la requête) sont formé en JSON pour êtr lues par la machine.

Connaitre la version de Elasticsearch

Dans la vue **Devtools** tapez:

```
GET /
```

Réponse:

```
{
  "name": "ZEWiZLN",
  "cluster_name": "elk_formation",
  "cluster_uuid": "rGzTBgbXRyev62Ku4vTWFw",
  "version": {
    "number": "6.4.3",
    ...
  },
  "tagline": "You Know, for Search"
}
```

Version de Elasticsearch

- C'est important car entre chaque version majeure (3, 4, 5, 6) il y a des changements dans les fonctions (L'API)
- La référence c'est la documentation: https://www.elastic.co/guide/en/elasticsearch/reference/6.4/index.html
- Toutes les fonctions de elasticsearch y sont décrites et on peut choisir la version selon celle installée.

Syntaxe d'un appel de fonction

```
<METHODE> <URI>
<DATA>
```

```
PUT /bibliotheque/livre/1
{
    "title": "La Promesse de l'aube",
    "description": "[...] J'entendis une fois de plus la formule intolérable
    "author": "Romain Gary",
}
```

Le BODY

- est *facultatif* ...
- est en **JSON** (JavaScript Objet Notation)
  - décrire des données complexes avec du texte
  - très répendu
  - pas trop dur à lire pour un humain

Syntaxe du JSON

```
{
    "champ1": "valeur1",
    "champ2_nombre": 3, // pas de guillemets
    "champ3_liste": [
        "item1",
        "item2",
        "item3",
    ],
    "champ4_objet": { // on ouvre un "nouveau json" imbriqué
        "souschamp1": "valeur1.1";
        ...
    },
    "champ5": "Pour échapper des \"guillemets\" et des \\n" // échappement pou
}
```

Index, Mapping et Document

Elasticsearch stocke des `documents` dans des `index`. Les documents correspondent à un `mapping` qui définit les champs du document et leurs types.

Analogie avec SQL:

- l'`index` est comme une `base de donnée` SQL.
- le `mapping` est comme une `table` SQL.
- le document est une `entrée`/`ligne` de table SQL.

Types de données des mappings

Quelques types de données:

- `text` : texte indexé en mode full text
- `keyword` : texte non indexé en full text
- `int`, `long`: des entiers
- `float`:
- `geo_point` : coordonnées LAT:LONG
- `date` : un timestamp type `2019-07-27T19:30:00.000Z`
- `ip` : stocke une adresse ip et permet de faire des opération et des filtres spécifiques dessus.
- etc.

Exemple de mapping

- Naviguer dans `Management` > `Index management` > `kibana_sample_data_logs` > `Mapping`
- Observons la structure:
  - `_doc` le mapping par défaut.
  - une liste de `properties` avec un type.

Différences avec les bases SQL

- Les documents ne sont pas obligés de correspondre exactement au mapping et les champs supplémentaires sont indexés comme des champs `text`.
- Tous les champs sont indexés dans elasticsearch, on a régulièrement besoin de réindexer les document si leur mapping évolue.
- Le mode fulltext est le mode par défaut et très performant dans Elasticsearch alors qu'il est facultatif et moins performant dans les base de données SQL.
- Elasticsearch reconnaît les mots du langage naturel -> il est adapté aux moteurs de recherche.
- `Elasticsearch` est massivement scalable (horizontalement) contrairement aux bases SQL classiques.