Indexation de documents

Création d'un index

- Création d'un index avec l'API REST « Create Index » :

```
PUT  /bibliotheque/
```

- Création d'un index avec des settings :

```
PUT  /bibliotheque/
{  "settings"  :  {"index"  :  {
        "number_of_shards"  :  8,
        "number_of_replicas":  2
    }}}
```

- Les index sont automatiquement créés s'ils n'existent pas au moment de l'indexation des documents

Modification d'un index

- Mise à jour des **settings** d'un index avec l'API REST «Update Settings » :

```
PUT  /bibliotheque/_settings
{  "index"  :  {
        "number_of_replicas":  6,
        "refresh_interval":  "30s"
}}
```

- Principaux settings :
  - `number_of_replicas` : Nombre de réplicas
  - `refresh_interval` : Intervalle de temps (en secondes) de rafraîchissement de l'index

Suppression d'un index

- Suppression d'un index avec l'API REST :

```
DELETE  /bibliotheque/
```

- Suppression de plusieurs index

```
DELETE  /index1,index2,index3/
```

- Suppression de tous les index

```
DELETE  /_all
DELETE  /*
```

- Il est possible d'empêcher la suppression de tous les index en configurant dans le fichier `elasticsearch.yml`:

```
action.destructive_requires_name  :  true
```

Indexation de documents avec l'API Rest

- L'API REST permet d'indexer un document au format JSON
  - En utilisant la méthode HTTP `PUT` et en précisant un identifiant de document :

```
PUT /bibliotheque/livre/1
{"titre": "7 ans après...", "auteurs": "Guillaume Musso", "prix": 20.80}
```

```
{ "created":true, "_index":"bibliotheque", "_type":"books", "_id":"1",
"_version":1,
"_shards": { ... }}
```

- En utilisant la méthode HTTP `POST` pour qu'un identifiant de document soit automatiquement généré :

```
POST /bibliotheque/livre
{"titre": "7 ans après...", "auteurs": "Guillaume Musso", "prix": 20.80}
```

```
{ "created":true, "_index":"bibliotheque", "_type":"books",
"_id":"AVGRV_usnjDD0CQltR_N", "_version":1,
  "_shards": { ... }}
```

Mise à jour d'un document 1/3

- Mettre à jour un document consiste à **réindexer le document**
  - Nécessite de connaître son identifiant
- Incrémente le numéro de version automatiquement
- Annule et **remplace** la version précédente du document
  - Nécessite de réindexer la totalité du document

```
PUT  /bibliotheque/livre/1
{  "titre":  "7  ans  après.....",  "prix":  20.80}
```

Mise à jour d'un document 2/3

- L'API REST « Update » permet de mettre à jour un ou plusieurs champs d'un document via script
- Permet :
  - d'ajouter de nouveaux champs,
  - d'ajouter de nouvelles valeurs à un champ existant,
  - de supprimer un champ existant,
  - supprimer le document si une condition est vérifiée...
- Comme indiqué, réindexe totalement le document et nécessite donc que la `_source` du document soit activée.
- Utilise en interne la `version` du document pour détecter les accès concurrents (voir plus loin pour plus de détails sur le fontionnement du champ version)

Mise à jour d'un document 3/3

Deux syntaxes :

- A base de `script` :

```
POST  /bibliotheque/livre/1/_update
   {  "script"  :  "ctx._source.prix  +=  addition",
      "params"  :  {"addition"  :  4}}
```

  - map `ctx` permet d'accéder à plusieurs infos (`_index`,`_type`,`_id` ...)
  - nécessite d'autoriser l'exécution de scripts: `script.inline: true`
  - suppression de champs possible:

```
"script"  :  "ctx._source.remove(\"prix\")"
```

- A base de document partiel `doc` :

```
POST  /bibliotheque/livre/1/_update
{  "doc"  :  {"prix"  :  25  }}
```

  - mergé avec le document actuel
  - pas de suppression possible

Suppression d'un document

- L'API REST permet de supprimer un document

```
DELETE  /bibliotheque/livre/1
```

API Bulk 1/3

- Endpoint `_bulk` dédié aux opérations en lot

```
POST  /_bulk
{  ...  }
```

- Beaucoup plus performante pour un grand nombre d'opérations
  - Exemple: Batch
- Format spécifique des données :

```
action  +  meta_donnees\n
source_optionelle\n
action  +  meta_donnees\n
source_optionelle\n
...
```

- En sortie : JSON contenant le résultat de chaque action
  - code réponse HTTP
  - éventuellement, message d'erreur

API Bulk 2/3

- Opérations possibles : `index` / `create` / `update` / `delete`
- Source inutile pour `delete`
- Exemple de données en entrée :

```
{ "create" : { "_index" : "bibliotheque", "_type" : "livre", "_id" : "43" } }
{ "auteur" : "Victor Hugo", "titre":"Les Misérables" }
{ "delete" : { "_index" : "bibliotheque", "_type" : "livre", "_id" : "2" } }
{ "update" : {"_index" : "bibliotheque", "_type" : "dvd", "_id" : "1"} }
{ "doc" : {"lang" : "fr"} }
...
```

API Bulk 3/3

- Réponse :


```
[{ "create": {
       "_index": "bibliotheque",
       "_type": "livre",
       "_id": "43",
       "_version": 1,
       "status": 201 }
},{ "delete": {
       "_index": "bibliotheque",
       "_type": "livre",
       "_id": "2",
       "_version": 1,
       "status": 404, "found": false }
},{ "update": {
       "_index": "bibliotheque",
       "_type": "dvd",
       "_id": "43",
       "status": 404, "error": {
              "type": "document_missing_exception",
              "reason": "[dvd][1]: document missing",
              "index": "bibliotheque",
              "shard": "-1" }}
},...]
```

Identifiant de document

- Tous les documents ont un identifiant
  - Lors de l'indexation du document, l'identifiant peut être spécifié (`PUT`) ou généré automatiquement (`POST`)
  - L'**identifiant et le type de document** permettent d'identifier un document dans un index
  - L'id est retourné dans le champ `_id` lors d'une recherche
  - L'id est utilisé pour récupérer un document dans une requête `GET`
  - Le champ `_uid` est l'identifiant unique dans l'index `_uid = _type + _id`
- Utiliser autant que possible un id métier
  - Toutes les opérations sur un document passent par l'identifiant
  - Sans l'identifiant, il faut préalablement faire une recherche pour retrouver un document

Version

- Les documents sont automatiquement **versionnés** lors de l'indexation
  - Le numéro de version commence à 1
  - Il est incrémenté à chaque nouvelle indexation du document
  - Le numéro de version est retourné lors de l'indexation mais pas lors des recherches (par défaut)
- Il est possible de gérer manuellement les numéros de version
  - En précisant les paramètres version et `version_type=external`

Version | Optimistic Concurrency Control 1/4

- **Optimistic Concurrency Control**
  - Elasticsearch n'a **pas de support de transactions**
  - Lors d'une indexation, il est possible de fournir un numéro de version via le paramètre `version`
  - Elasticsearch vérifiera si le document est dans la version indiquée avant d'effectuer l'opération demandée
  - Utile pour simuler une transaction de type readthenupdate

Version | Optimistic Concurrency Control 2/4

Exemples de requêtes illustrant l'Optimistic Concurrency Control

- Indexation sans version (v2 → v3;)

```
PUT /bibliotheque/livre/1 {...}

{"_index": "bibliotheque", "_type": "livre", "_id":"1", "_version":3,
"created":false}
```

- Indexation v2 (erreur de type `version_conflict_engine_exception`)

```
PUT /bibliotheque/livre/1?version=2 {...}

{"error": {
   "type": "version_conflict_engine_exception",
   "reason": "[livre][1]: version conflict, current [3], provided [2]",
   "index": "bibliotheque", "shard": "3"
}, "status": 409}
```

- Indexation v3 (v3 → v4)

```
PUT /bibliotheque/livre/1?version=3 {...}

{"_index": "bibliotheque", "_type": "livre", "_id": "1", "_version":4,
"created": false}
```

Version | Optimistic Concurrency Control 3/4

La version actuelle est 4...

- Suppression v3 (erreur de type `version_conflict_engine_exception`)

```
DELETE  /bibliotheque/livre/1?version=3

{"error":{"reason":  "[livre][1]:  version  conflict,  current  [4],  provided  [3]"
...  }},  "status":409}
```

Version | Optimistic Concurrency Control 4/4

- Exemple de requête illustrant le `version_type=external`
  - Impose `version` > version stockée dans l'index

```
PUT /bibliotheque/livre/1?version_type=external&version=4
{"titre": "Spring Batch in Action"}
{"_index": "bibliotheque", "_type": "livre", "_id": "1", "_version": 4,
"created": false}
```

Refresh

- **Near Real Time**
  - Il existe un léger décalage entre le moment où un document est indexé et le moment où il est disponible et remonte en résultat de recherche
  - Ce laps de temps est configuré à 1 seconde par défaut pour les index
  - Configurable au niveau de chaque index avec le setting `refresh_interval`
- Avant une indexation en masse (`_bulk`), il est conseillé d'augmenter le `refresh_interval` pour éviter trop de rafraîchissements de l'index

Refresh et segments Lucene

- Avant d'être transmis à Lucene, les documents indéxés sont **analysés** et **stockés** dans un buffer mémoire:
  - à ce stade, ces documents **ne peuvent** pas être **recherchés**.
- Ce buffer mémoire est accompagné d'un fichier **TransLog** qui a le même cycle de vie et qui est écrit dans le **cache disque système**. Il sert à persister les commandes ayant rempli le buffer.
- Ce buffer mémoire est **flushé** dans un segment Lucene lorsque:
  - le buffer est **plein**
  - lors d'un **reopen** Lucene
  - sur un **refresh** ElasticSearch:
    - **sans commit** Lucene,
    - déclenché selon valeur du `index.refresh_interval` ou via l'API `_refresh`
  - sur un **flush** ElasticSearch:
    - provoque un **commit** Lucene,
    - **déclenché** automatiquement par Elasticsearch ou via l'API `_flush`

Segments et merges Lucene 1/2

- Un segment Lucene est un **inverted vector index** autosuffisant et **immuable** (jamais de suppression)
- Les segments augmentent en nombre et **consomment** de plus en plus de **ressources** (pointeurs sur fichiers, caches, memoire...)
- Pour pallier ce problème, Elasticsearch choisit de **merger** régulièrement des segments Lucene
- Ces opérations de merge répondent à une **merge policy** choisie optionnellement à la création de l'index. (réglage **expert**)
- Un **merge** consiste à:
  - Supprimer plusieurs segments
  - Supprimer les caches associés
  - Construire un segment **immuable** plus grand pour contenir les segments précédents sans reprendre les documents **marqués** détruits

Segments et merges Lucene 2/2

- L'opération de merge peut être coûteuse, jouer sur les paramètres suivants au niveau du `noeud` pour maîtriser ce coût:
  - positionner `indices.store.throttle.type` to merge
  - positionner `indices.store.throttle.max_bytes_per_sec` à `x`mb

Voir cette simulation des merges Lucene pour comprendre:

https://www.youtube.com/watch?v=YW0bOvLp72E

Refresh forcé

- Certaines opérations acceptent un paramètre `refresh=true` pour rafraîchir directement l'index après l'opération
  - Utile pour les tests d'intégrations
  - A utiliser judicieusement, car très coûteux

```
PUT  /bibliotheque/livre/1?refresh=true
{"titre":"Spring  Batch  in  Action"}
```

Operation Type

- Lors d'une indexation, il est possible de préciser le paramètre `op_type=create`
- Permet d'indexer un document uniquement s'il n'existe pas déjà

```
PUT /bibliotheque/livre/1?op_type=create
{"titre":"Spring Batch in Action"}
{ "error": {
   "type": "document_already_exists_exception",
   "reason": "[livre][1]: document already exists",
   "index": "bibliotheque", "shard": "2"
 }, "status": 409 }
```

- Peut aussi s'écrire avec la notation `_create`

```
PUT /bibliotheque/livre/1/_create
{"titre":"Spring Batch in Action"}
```

Autres fonctionnalités

- Lors d'une indexation de document, il est aussi possible de préciser
  - Le `routing` à utiliser, pour que le document soit ajouté à un shard précis de l'index
  - Le document `parent` du document qu'on s'apprête à indexer, et ainsi permettent les recherches parent/child
  - Le `consistency` à utiliser, pour avoir la main sur le nombre minimum de shards/replicas dans lequel le document doit être écrit: `one`, `quorum` (par defaut), `all`.
  - Un `timeout`, au delà duquel une erreur sera renvoyée si l'opération prend plus de temps à s'exécuter

Alias d'index

- Il est possible de créer des **alias** sur les index
  - Un alias pointe vers un ou plusieurs autres **index**
  - La création et la suppression des alias se fait via une API Rest

```
POST  /_aliases
{"actions"  :  [{  "add"  :  {  "index"  :  "bibliotheque",  "alias"  :  "znk"  }  }]}
PUT  bibliotheque/_aliases/znk
GET  bibliotheque/_aliases
```

  - Il existe un alias par défaut : `_all`
- Permet différents cas d'utilisations
  - Alias « current » sur le dernier index de logs
  - Réunir plusieurs index fréquemment utilisés ensemble
- Fonctionnalités avancées
  - Filtre : alias sur un index filtré sur une requête
  - Route : spécifie le routage sur les shards de l'index

TP2