le cnam

Guide pratique pour Elasticsearch

Travaux Pratiques

CNAM Paris
traversn / fournier (at) cnam.fr

Chapitre

Table des matières

Chapitre 1

Mise en place

Le logiciel `elasticsearch`, développé en Java, est très facile d'installation et de déploiement dans un environnement distribué. Son but est d'intégrer du contenu semi-structuré (JSon) orienté texte et de permettre son interrogation.

L'API est implémentée sous forme de service REST que l'on peut interroger via le port 9200.

1.1 Téléchargement et installation

Vous pouvez télécharger la dernière version sur : `http://elastic.co`

Dans le répertoire de *elasticsearch*, dénommé ici par $ELASTIC, vous pourrez retrouver les répertoires suivants après décompression :

- `$ELASTIC/bin` : exécutables,
- `$ELASTIC/config` : fichiers de configuration,
- `$ELASTIC/plugins` : extensions,
- `$ELASTIC/logs` : fichiers de journalisation en cas de problèmes.

1.2 Configuration

Pour configurer un serveur, ouvrez le fichier suivant : `$ELASTIC/config/elasticsearch.yml`

- `cluster.name` : nom du cluster pour l'ensemble des noeuds *elastic*,
- `node.name` : nom du noeud que vous souhaitez démarrer (doit être unique pour un cluster),
- `index.number_of_shards` : nombre de serveurs (défaut 1),
- `index.number_of_replicas` : nombre de serveurs de réplication pour la tolérance aux pannes (défaut 0).

Pour faciliter l'administration, installez l'interface Web '*head*' dans une console.

```
$ELASTIC/bin/plugin -install mobz/elasticsearch-head
```

1.3 Lancer le serveur

1.3.1 Dans une console, lancez le serveur : `$ELASTIC/bin/elasticsearch`
Ne pas éteindre ce serveur, ni fermer la fenêtre !

1.3.2 Pour vérifier l'état, ouvrir dans un navigateur : `http://localhost:9200/?pretty` (ou utilisez *curl*[1])

1.3.3 Pour éteindre le serveur : `curl -XPOST 'http://localhost:9200/_shutdown'`

1.3.4 Pour ouvrir l'interface 'head' d'administration : `http://localhost:9200/_plugin/head`

1.4 Fonctionnement d'une base

Une base 'elastic' se nomme un **index**, dans lequel on peut créer des collections appelées '**type**'.

Ainsi, pour importer des données dans *elasticsearch*, il vous faut préciser cet index et ce type en prefixant chaque document importé par :

```
{"index":{"_index": "MA BASE","_type":"MA COLLECTION","_id":1}}
```

L'identifiant "_id" doit être unique dans le type et sera associé au document suivant.

Pour importer les données, il est possible d'utiliser l'importation massive de documents avec le service "_bulk". Pour cela, il faut avoir mis les documents dans un fichier avec le format ci-dessus, et prefixer le fichier par un "@" comme ci-dessous :

```
curl -XPUT localhost:9200/_bulk --data-binary @nom_du_fichier.json
```

1. `https://curl.haxx.se/dlwiz/?type=bin`

Chapitre 2

Interrogation

Toute requête se fait sur l'API REST, elle doit contenir :

— le nom de l'index
— le nom du type
— Un opérateur de recherche : _count, _search
— Optionnel un identifiant

2.1 Principe

`curl -XGET 'http://localhost:9200/tests/test/1'`
Index 'tests', de type 'test', pour le premier document (_id=1)
`curl -XGET 'http://localhost:9200/tests/test/_count'`
Compte le nombre de documents de 'tests/test'

2.2 Requête de 'Matching'

Le service `_search` permet de faire des requêtes simples

— Recherche de mots dans l'ensemble de chaque document

```
curl -XGET 'http://localhost:9200/tests/test/_search?q=alien&pretty=true'
```

— Dans le titre de chaque document

```
curl -XGET 'http://localhost:9200/tests/test/_search?q=title:alien&pretty=true'
```

— Combinaison de critères avec l'espace (%20 pour une URL). Ici, la négation est utilisée ("-")

```
curl -XGET 'http://localhost:9200/tests/test/_search?q=title:alien%20-year:1992&pretty=true'
```

— Pour avoir une requête complexe en utilisant le DSL (Domain Specific Language), il faut envoyer un document JSon "requête"

```
curl -XGET http://localhost:9200/tests/test/_search?pretty -d '
{
  "query" : ...
}'
```

— Requêtes de simple correspondance (matching)

```
{
  "query" : {
    "match" : { "nom_de_l'attribut" : "la\_valeur\_a\_chercher"}
  }
}
```

— Requêtes d'intervalles (range)

```
{
  "query" : {
    "range" : { "nom_de_l'attribut" : "la\_borne"}
  }
}
```

— Notions de critère optionnel 'should' (avec calcul de score de pertinence) et d'obligatoire 'must' (sans score). Il faut utiliser le critère en imbriquant dans un opérateur 'bool'

```
{"query":{
  "bool": {
    "should": { "match": { "nom_de_l'attribut" : "valeur" }},
    "must": { "range": { "nom_de_l'attribut" : "valeur"}},
    "must_not":[
        {"match":{"nom_de_l'attribut" : "valeur"}},
        {"match":{"nom_de_l'attribut" : "valeur"}},
        ...
    ]
}}}
```

— Une requête avec l'opérateur '*filter*' ne calcule aucun score pour la requête correspondante (même avec 'should')

```
{
  "query": {
    "filtered": {
      "query": {
        "match": {"nom_de_l'attribut" : "valeur"}
      },
      "filter": {
        "range": {"nom_de_l'attribut" : "valeur"}
}}}}
```

Vous pourrez trouver la syntaxe complète ici :
`https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html`

2.3 Requêtes d'agrégats

Il est possible de regrouper les documents pour pouvoir appliquer une agrégation à des fins de statistiques. Pour cela, il faut utiliser la clé "aggs" dans les documents requêtes, donner la clé où prendre les valeurs "terms : field : XXX" (identique à un GROUP BY) et donner la clé où mettre la valeur d'agrégation. Par défaut, cela va compter le nombre d'occurences.

```
{"aggs" : {
        "nom_de_la_cle_en_sortie" : {
                "terms" : {"field" : "nom_de_la_cle_a_grouper"}
}}}
```

Attention, cette méthode va faire décomposer les valeurs de la clé "nom_de_la_cle_a_grouper". Ainsi, un texte se verra décomposer en ensemble de mots pour être ensuite regroupés (par mot).

On peut également changer "`terms`" en "`avg`", "`min`", "`max`", "`cardinality`" (distinct)... Voire même ressortir les mots importants "`significant_terms`" (grâce à leurs occurences dans un ensemble filtrés)

Il est possible de filtrer les documents avant l'agrégat. Il suffit de faire une requête classique "`query`".

```
{"query" :{
                "match" : {"cle_a_filtrer" : "valeur"}
        }
 ,"aggs" : {
                "cle_de_sortie" : {
                        "avg" : {"field" : "cle_a_grouper"}
        }}}
```

Il est possible d'imbriquer les agrégats pour grouper par type :

```
{"aggs" : {
        "nom_groupe" : {
                "terms" : {
                        "field" : "cle_du_group_by"
                },
                "aggs" : {
                        "nom_sous_groupe" : {
                                "avg" : {"field" : "cle_pour_la_moyenne"}
                        }}
                }
        }}
```

Le tri fonctionne de la même manière avec la clé "`order`" en utilisant le nom de la clé à trier.
`"order" : { "cle_pour_la_moyenne" : "desc" }`

Il est également possible de grouper par plages de valeur plutot que les valeurs du document :

```
{"aggs" : {
        "nom_cle_sortie" : {
                "range" : {
                        "field" : "cle_ou_prendre_la_valeur",
                        "ranges" : [
                                {"to" : "valeur_max"},
                                {"from" : "valeur_min", "to" : "valeur_max"},
                                {"from" : "valeur_min"}
                        ]
                }
                }}}
```

2.4 INSERT

L'index et le type sont obligatoires. Chaque document inséré DOIT avoir la même structure que le premier document inséré (celui qui détermine le schéma du 'type').

```
curl -XPOST 'http://localhost:9200/tests/test2/' -d "{'test':1, 'title' : 'my new elastic document'}"
```