Tutorial Elasticsearch

Installation de elasticsearch

Les différents outils peuvent être downloadés ici.

Tool	Description
Elasticsearch	Base de données
Kibana	UI → console pour exécuter les requêtes
X-Pack	Single extension that integrates handy features: • Security • Alerting • Monitoring • Reporting • Graph

Une fois X-Pack installé, un mot de passe est demandé :

```
Username : elastic
Password : changeme
```

Pour désactiver la sécurité :

- Éditer le fichier elasticsearch.yml
- Ajouter l'entrée :

```
xpack.security.enabled: false
```

Démarrer le serveur ElasticSearch :

```
/Users/Laurent/elasticsearch/5.1.2/elasticsearch/bin/elasticsearch
```

Le port par défaut est le :9200

Ensuite seulement, on peut démarrer le serveur ElasticSearch :

```
/Users/Laurent/elasticsearch/5.1.2/kibana/bin/kibana
```

Le port par défaut est le :5601

Kibana

Afficher les infos de la version de ElasticSearch

```
GET /
```

Création d'un index

```
PUT /library
{
     "settings" : {
          "index" : {
               "number_of_shards" : 1,
               "number_of_replicas" : 0
          }
     }
}
```

Liste des index (indique également le nombre d'éléments)

```
GET /_cat/indices?v
```

Suppression d'un index

```
DELETE library
```

Ajouter un objet dans l'index = *my-index*, avec le type = *my-type*, et l'identifiant = *1* :

```
POST /my-index/my-type/1
{
     "body": "foo"
}
```

Récupérer l'objet sauvegardé :

```
GET /my-index/my-type/1
```

Compter le nombre d'éléments d'un index

```
GET /my-index/my-type/_count
```

Rechercher l'objet :

```
GET /my-index/_search
{
     "query":{
          "match": {
               "body": "foo"
          }
     }
}
```

Supprimer un objet :

```
DELETE /my-index/my-type/1
```

Ajout d'objets : Le bulk fonctionne comme un batch et permet d'indexer un groupe d'éléments → plus rapide.

```
POST _bulk
{"index" : { "_index" : "test", "_type" : "type1", "_id" : "1" }}
{"field1" : "value1"}
```

Afficher tous les éléments

```
GET /my-index/my-type/_search
```

Afficher les 5 premiers éléments

```
GET /my-index/my-type/_search?size=5
```

Afficher les éléments 10 à 20

```
GET /my-index/my-type/_search?size=10&from=10
```

Rechercher tous les éléments contenant *Ghosts* dans le titre

```
GET /amazonreader/item/_search
{
     "query": {
          "match": {
               "title": "ghosts"
          }
     }
}
```

→ 1'177 éléments

Rechercher tous les éléments contenant *American* **ou** *Ghosts* dans le titre

```
GET /amazonreader/item/_search
{
     "query": {
          "match": {
               "title": "american ghosts"
          }
     }
}
```

→ 36'840 éléments

Rechercher tous les éléments contenant la phrase *American Ghosts* dans le titre

```
GET /amazonreader/item/_search
{
     "query": {
          "match_phrase": {
               "title": "american ghosts"
          }
     }
}
```

→ 4 éléments

La recherche sur un mot peut également s'écrire comme ceci (low-level) :

```
GET /amazonreader/item/_search
{
  "query": {
    "term" : { "title" : "switzerland" }
  },
}
```

Récupérer les n premiers éléments :

```
GET /amazonreader/item/_search
{
  "query": {
    "term" : { "title" : "switzerland" }
  },
  "size": 2
}
```

Récupérer les n éléments à partir de l'offset :

```
GET /amazonreader/item/_search
{
  "query": {
    "term" : { "title" : "switzerland" }
  },
  "size": 2,
  "from": 10
}
```

Tri ascendant – afficher les n titres contenant *Switzerland* à partir de l'offset du plus cher au moins cher :

```
GET /amazonreader/item/_search
{
  "query": {
    "term" : { "title" : "switzerland" }
  },
   "sort" : [
      {"price" : {"order" : "desc"}}
   ],
  "size": 2,
  "from": 10
}
```

Du moins cher au plus cher :

```
"sort" : [
      {"price" : {"order" : "asc"}}
   ],
```

Sort mode option

Si le prix est une chaine de nombres, on peut indiquer (avec mode) la façon d'exécuter le tri :

```
GET /amazonreader/item/_search
{
  "query": {
    "term" : { "title" : "switzerland" }
  },
   "sort" : [
      {"price" : {"order" : "desc", "mode" : "avg"}}
   ],
  "size": 2,
  "from": 10
}
```

Elasticsearch supports sorting by array or multi-valued fields. The `mode` option controls what array value is picked for sorting the document it belongs to. The `mode` option can have the following values:

`min`	Pick the lowest value.
`max`	Pick the highest value.
`sum`	Use the sum of all values as sort value. Only applicable for number based array fields.
`avg`	Use the average of all values as sort value. Only applicable for number based array fields.
`median`	Use the median of all values as sort value. Only applicable for number based array fields.

Missing values

The `missing` parameter specifies how docs which are missing the field should be treated:
The `missing` value can be set to `_last`, `_first`, or a custom value (that will be used for missing docs as the sort value). The default is `_last`.

```
GET /_search
{
    "sort" : [
        { "price" : {"missing" : "_last"} }
    ],
    "query" : {
        "term" : { "product" : "chocolate" }
    }
}
```

Une ***bool query*** est une requête contenant des sous-requêtes.

La requête suivante permet de récupérer les éléments contenant dans leur titre *quick* **et** *lazy dog.*

```
GET /amazonreader/item/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "title": "quick"
          }
        },
        {
          "match_phrase": {
            "title": "lazy dog"
          }
        }
      ]
    }
  }
}
```

Must correspond à AND.

Il est également possible d'afficher les éléments n'ayant dans leur titre ni *quick* ni *lazy dog*:

```
GET /amazonreader/item/_search
{
  "query": {
    "bool": {
      "must_not": [
        {
          "match": {
            "title": "quick"
          }
        },
        {
          "match_phrase": {
            "title": "lazy dog"
          }
        }
      ]
    }
  }
}
```

La requête suivante permet de récupérer les éléments contenant dans leur titre *quick dog* **ou** *lazy dog:*

```
GET /amazonreader/item/_search
{
  "query": {
    "bool": {
      "should": [
        {
          "match_phrase": {
            "title": "quick dog"
          }
        },
        {
          "match_phrase": {
            "title": "lazy dog"
          }
        }
      ]
    }
  }
}
```

On peut donner plus de poids à l'un des critères :

```
GET /amazonreader/item/_search
{
  "query": {
    "bool": {
      "should": [
        {
          "match_phrase": {
            "title": {
              "query": "quick dog",
              "boost": 3
            }
          }
        },
        {
          "match_phrase": {
            "title": {
              "query": "lazy dog",
              "boost": 5
            }
          }
        }
      ]
    }
  }
}
```

Ajouter un filtre :

```
GET /amazonreader/item/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match_phrase": {
            "title": {
              "query": "lazy dog"
            }
          }
        }
      ],
      "filter": {
        "range": {
          "price": {
            "gte": 10,
            "lte": 20
          }
        }
      }
    }
  }
}
```

Une requête peut aussi ne comporter qu'un filtre :

```
GET /amazonreader/item/_search
{
     "query": {
          "bool": {
               "filter": {
                    "range": {
                         "price": {
                              "gt": 998
                         }
                    }
               }
          }
     }
}
```

Récupérer la liste des éléments ayant books dans leur liste de catégories associées :

```
GET /amazonreader/item/_search
{
    "query": {
        "constant_score" : {
            "filter" : {
                 "bool" : {
                    "must" : [
                        { "term" : { "categories" : "books" } }
                    ]
                }
            }
        }
    }
}
```

Récupérer le mapping des champs d'un index :

```
GET /amazonreader/_mapping
GET /amazonreader/item/_mapping?pretty=true
```

```
PUT /amazonreader/_mapping/item
{
    "item" : {
        "properties" : {
            "brand": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          },
          "categories": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          },
          "id": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          },
          "imUrl": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          },
          "price": {
            "type": "float"
          },
          "related": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          },
          "title": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          }
        }
    }
}
```

La bibliothèque Go "*gopkg.in/olivere/elastic.v5*" pour Elasticsearch ne fonctionne pas très bien et la documentation est obsolète.

Pour pallier à ce problème, le transfert des commandes entre Go et Elasticsearch peut se faire par l'intermédiaire des commandes :

- **httpClient.Get** pour les commandes simples

```
resp, err :=
httpClient.Get("http://localhost:9200/amazonreader/item/_search?q=title:Switzerland")
if err != nil {
   // handle err
   log.Fatalln(err.Error())
}

defer resp.Body.Close()

responseData, err := ioutil.ReadAll(resp.Body)
```

- **http.DefaultClient.Do** pour les commandes complexes

```
req, err := http.NewRequest("GET",
"http://localhost:9200/amazonreader/item/_search?pretty", body)
if err != nil {
   // handle err
}
req.Header.Set("Content-Type", "application/x-www-form-urlencoded")

resp, err := http.DefaultClient.Do(req)
if err != nil {
   // handle err
}

defer resp.Body.Close()

responseData, err := ioutil.ReadAll(resp.Body)
```

La traduction des requêtes Elasticsearch en Go peut se faire au travers de ce processus :
Requête ElasticSearch → Copy as Curl → curl-to-go

Exemples

Affiche tous les index de manière verbeuse en incluant l'**authentification** :

```
curl --user elastic:changeme -XGET 'http://localhost:9200/_cat/indices?v'
```

Recherche id=*0590334107* dans l'index est *amazonreader* et le type est *item* :

```
curl -XGET 'localhost:9200/amazonreader/item/_search?q=_id:0590334107'
```

Recherche id=*0590334107* dans l'index *amazonreader* et le type est *item* ou *user* :

```
curl -XGET 'localhost:9200/amazonreader/item,user/_search?q=_id:0590334107'
```

Recherche les articles contenant *Switzerland* dans le champ title :

```
curl -XGET 'localhost:9200/amazonreader/item/_search?q=title:Switzerland'
```

Cas général de requête :

```
GET /amazonreader/item/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "title": "quick brown"
          }
        },
        {
          "match_phrase": {
            "title": "lazy dog"
          }
        },
        {
          "match_phrase": {
            "title": "brown"
          }
        }
      ],
      "filter": {
        "range": {
          "price": {
            "gte": 0,
            "lte": 1000
          }
        }
      }
    }
  },
  "sort": [
    {
      "price": {
        "order": "asc"
      }
    }
  ],
  "size": 10,
  "from": 0
}
```