Hands On Elasticsearch

Malloum Laya / David Pilato

Qui ?

$ curl http://localhost:9200/talk/speaker/TheMalloum


```
{
 "nom" : "Malloum Laya",
 "jobs" : [
  { "boite" : "Logica", "mission" : "développeur", "duree" : 3 },
  { "boite" : "DGDDI (douane)", "mission" : "chef de projets", "duree" : 4 } ],
 "passions" : [ "famille", "informatique", "rap" ],
 "blog" : "http://malloum.fr/",
 "twitter" : [ "@TheMalloum", "@scrutmydocs" ],
 "email" : "malloum.laya@gmail.com"
}
```


$ curl http://localhost:9200/talk/speaker/dpilato


```
{
 "nom" : "David Pilato",
 "jobs" : [
  { "boite" : "SRA Europe (SSII)", "mission" : "bon à tout faire", "duree" : 3 },
  { "boite" : "SFR", "mission" : "touche à tout", "duree" : 3 },
  { "boite" : "e-Brands / Vivendi", "mission" : "chef de projets", "duree" : 4 },
  { "boite" : "DGDDI (douane)", "mission" : "mouton à 5 pattes", "duree" : 8 },
  { "boite" : "IDEO Technologies", "mission" : "directeur technique", "duree" : 0 } ],
 "passions" : [ "famille", "job", "deejay" ],
 "blog" : "http://dev.david.pilato.fr/",
 "twitter" : [ "@dadoonet", "@elasticsearchfr", "@scrutmydocs" ],
 "email" : "david@pilato.fr"
}
```

Agenda

- Aperçu d'Elasticsearch
- Atelier 1 : indexons
- Atelier 2 : cherchons
- Atelier 3 : analysons

Elasticsearch

Les fondamentaux

Un moteur de recherche

- un moteur d'indexation de "documents"
- un moteur de recherche sur les index

Elasticsearch

- Moteur de recherche pour la génération NoSQL
- Basé sur le standard Apache Lucene
- Masque la complexité Java/Lucene à l'aide de services standards HTTP / RESTful / JSON
- Utilisable à partir de n'importe quelle technologie
- Ajoute la couche cloud manquante à Lucene avec distribution, réplication, fail over
- Très performant : distribution des calculs sur plusieurs nœuds
- C'est un moteur, pas une interface graphique !

Penser document !

- Changement de paradigme :
  - On ne pense plus SQL !
  - On indexe ce qu'on veut trouver !
- Un document, c'est :
  - Un objet JSON
  - Des propriétés simples (String, boolean, number)
  - Des propriétés complexes (array, object)
  - Des propriétés évoluées (geoloc, binary, attachments)

Ranger ses documents

- Elasticsearch met à notre disposition
  - Des types de document
  - Des index
- Le document 1 de type Tweet dans l'index Twitter :


```
{
  "text": "Bienvenue au Hands On Lab #elasticsearch pour #jduchess",
  "created_at": "2012-04-06T20:45:36.000Z",
  "source": "Twitter for iPad",
  "truncated": false,
  "retweet_count": 0,
  "hashtag": [ { "text": "elasticsearch", "start": 27, "end": 40 },
               { "text": "jduchess", "start": 47, "end": 55 } ],
  "user": { "id": 51172224,"name": "David Pilato",
            "screen_name": "dadoonet", "location": "France",
            "description": "Soft Architect, Project Manager, Senior Developper.\r\nAt this time, enjoying NoSQL
world : CouchDB, ElasticSearch.\r\nDeeJay 4 times a year, just for fun !" }
}
```

Agir avec Elasticsearch

- **API REST : http://host:port/[index]/[type]/[_action/id]**
  Méthodes HTTP : GET, POST, PUT, DELETE

- **Documents**

  - **curl -XPUT http://localhost:9200/twitter/tweet/1**
  - **curl -XGET http://localhost:9200/twitter/tweet/1**
  - **curl -XDELETE http://localhost:9200/twitter/tweet/1**

- **Recherche**

  - **curl -XPOST http://localhost:9200/twitter/tweet/_search**
  - **curl -XPOST http://localhost:9200/twitter/_search**
  - **curl -XPOST http://localhost:9200/_search**

- **Meta données Elasticsearch**

  - **curl -XGET http://localhost:9200/twitter/_status**
  - **curl -XPOST http://localhost:9200/_shutdown**

Quelques notions

- **Nœud** (**node**) : Une instance d'Elasticsearch (~ machine ?)
- **Cluster** : Un ensemble de nœuds
- **Partition** (**shard**) : permet de découper un index en plusieurs parties pour y distribuer les documents
- **Réplication** (**replica**) : recopie d'une partition en une ou plusieurs copies dans l'ensemble du cluster
- **Partition primaire** (**primary shard**) : partition élue "principale" dans l'ensemble du cluster. C'est là que se fait l'indexation par Lucene. Il n'y en a qu'une seule par shard dans l'ensemble du cluster.
- **Partition secondaire** (**secondary shard**) : partitions secondaires stockant les replicas des partitions primaires.

Atelier 0

On prépare le terrain

Agir en Java

```
// Création du noeud. Il s'agit d'une instance d'Elasticsearch complète !
Node node = NodeBuilder.nodeBuilder().node();

// Les opérations courantes (put, get, search, delete) se font à l'aide d'un client !
Client client = node.client();

// On peut aussi faire des opérations d'administration (état du cluster, suppression index, ...) !
AdminClient client = node.client().admin();
```

Atelier 0

- Récupérer la distribution Elasticsearch

  ES 0.19.8, elasticsearch.yml modifié, plugins : MOBZ Head, BigDesk, Paramedic

  curl -OL -k https://github.com/downloads/elasticsearchfr/hands-on/elasticsearch-0.19.9-handson.zip

- Cloner la repository

  git clone https://github.com/elasticsearchfr/hands-on.git

- Regarder les sources du test 0

  src/test/java/org/elasticsearchfr/handson/ex0/NodeTest.java

Atelier 1

On indexe des bières ;-)

Indexer en Java

```java
// On peut fabriquer du json à la main
String json = "{ \"field\":\"value\" }";

// En utilisant le XContent d'Elasticsearch
XContentBuilder xcb = XContentFactory.jsonBuilder()
.startObject()
.field("field", "value")
.endObject();
String json = xcb.toString();

// Sérialiser JavaBean en JSon
// On peut utiliser Jackson : http://wiki.fasterxml.com/JacksonHome
Beer beer = new Beer("Heineken", Colour.PALE, 0.33, 3);
ObjectMapper mapper = new ObjectMapper();
String json = mapper.writeValueAsString(beer);

// On peut indexer !
client.prepareIndex("index", "type").setSource(json).execute().actionGet();
client.prepareIndex("index", "type", "id").setSource(xcb).execute().actionGet();

// Insérer en mode Bulk
BulkRequestBuilder brb = node.client().prepareBulk();

// Ajouter des actions à réaliser
brb.add(new IndexRequest("index", "type").source(json));

// Exécuter le bulk
brb.execute().actionGet();
```

Récupérer en Java

```java
// Récupérer
GetResponse gr = client.prepareGet("index", "type", "id").execute().actionGet();

// Désérialiser de JSon vers JavaBean
ObjectMapper mapper = new ObjectMapper();
Beer beer = mapper.readValue(gr.getSourceAsBytes(), Beer.class);
```

Effacer en Java

```java
// Effacer
client.prepareDelete("index", "type", "id").execute().get();
```

Atelier 1

- Regarder les sources du test 1

  src/test/java/org/elasticsearchfr/handson/ex1/IndexTest.java

Atelier 2

On cherche des bières ;-)

Différentes recherches

Type	Description
Match All	Recherche tout le contenu (pratique avec des filtres)
QueryString	Recherche avec analyse, jokers (syntaxe Lucene possible* +, -, FROM, TO, ^)
Term	Recherche d'un terme sans analyse préalable
Text	Recherche d'un texte avec analyse (par défaut OR sur chaque token)
Wildcard	Recherche avec joker (*, ?)
Bool	Recherche multi-critères (MUST, MUST NOT, SHOULD)
Range	Recherche intervalle (>, >=, <, <=)
Prefix	Commence par (plus efficace que wilcard*)
Filtered	Filtrage (couplage de filtres et de queries)
Fuzzy like this	Permet des recherches par vraisemblance de termes
More like this	Permet de trouver des documents avec un minimum de termes

Chercher en Java

```java
// Lancer une recherche
SearchResponse sr = client.prepareSearch().setQuery(qb).execute().actionGet();

// Fabriquer un QueryBuilder (qb)
// MatchAll
QueryBuilder qb = QueryBuilders.matchAllQuery();

// TermQuery
QueryBuilder qb = QueryBuilders.termQuery("field", "value");

// TextQuery
QueryBuilder qb = QueryBuilders.textQuery("field", "you can enter a text here");

// QueryString
QueryBuilder qb = QueryBuilders.queryString("you can enter a text here");

// RangeQuery
QueryBuilder qb = QueryBuilders.rangeQuery("field").from(5).to(10);

// BoolQuery
QueryBuilder qb = QueryBuilders.boolQuery()
    .must(QueryBuilders.textQuery("field1", "value"))
    .mustNot(QueryBuilders.textQuery("field2", "value"))
    .should(QueryBuilders.rangeQuery("field3").from(0).to(5));
```

Les résultats

```
// Lancer une recherche
SearchResponse sr = client.prepareSearch().setQuery(qb).execute().actionGet();

// Récupérer le temps d'exécution
long nbHits = sr.getTookInMillis();

// Récupérer le nombre de hits
long nbHits = sr.getHits().getTotalHits();

// Récupérer les hits
SearchHit[] hits = sr.getHits().getHits();


// Lire un hit
SearchHit hit = hits[0];

// Récupérer les coordonnées du document
hit.getIndex();
hit.getType();
hit.getId();

// Récupérer la pertinence du document
hit.getScore();

// Récupérer le document source au format Json
hit.getSourceAsString();
```

Atelier 2

- Regarder les sources du test 2

  src/test/java/org/elasticsearchfr/handson/ex2/SearchTest.java

Atelier 3

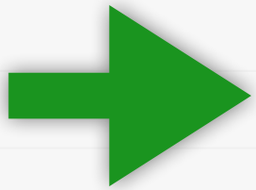
La bière vue sous un autre angle
Les facettes

Les facettes

ID	Marque	Taille	Prix
1	Heineken	0.25	3.50
2	Grimbergen	0.25	4.00
3	Kriek	0.25	3.80
4	Heineken	0.5	6.80
5	Grimbergen	0.5	7.80
6	Kriek	0.5	7.50
7	Heineken	1	12.00
8	Grimbergen	1	14.00
9	Kriek	1	14.00

Term Facet

ID	Marque	Taille	Prix
1	Heineken	0.25	3.50
2	Grimbergen	0.25	4.00
3	Kriek		
4	Heineken		
5	Grimbergen		
6	Kriek		
7	Heineken	1	12.00
8	Grimbergen	1	14.00
9	Kriek	1	14.00



Marque	Count
Heineken	3
Grimbergen	3
Kriek	3

TermFacet en Java

```
// Fabriquer une TermFacet (facet)
AbstractFacetBuilder facet = FacetBuilders.termsFacet("bybrand").field("marque");

// Avec la recherche
SearchResponse sr = client.prepareSearch().setQuery(qb).addFacet(facet).execute().actionGet();

// Facette en retour
TermsFacet bybrand = sr.getFacets().facet("bybrand");
for (TermsFacet.Entry entry : bybrand) {
String marque = entry.getTerm();
int nb = entry.count();
}
```

Range Facet

ID	Marque	Taille	Prix
1	Heineken	0.25	3.50
2	Grimbergen	0.25	4.00
3	Kriek	0.25	3.80
4	Heineken	0.5	6.80
5	Grimbergen	0.5	7.80
6	Kriek	0.5	7.50
7	Heineken	1	12.00
8	Grimbergen	1	14.00
9	Kriek	1	14.00

Ranges	Count	Min	Max	Moy	Total
x < 5	3	3.50	4.00	3.77	11.30
5 <= x < 10	3	6.80	7.80	7.37	22.10
10 <= x	3	12.00	14.00	13.33	40.00

RangeFacet en Java

```
// Fabriquer une TermFacet (facet)
AbstractFacetBuilder facet = FacetBuilders.rangeFacet("byprice")
.field("price")
.addUnboundedFrom(5)
.addRange(5, 10)
.addUnboundedTo(10);

// Avec la recherche
SearchResponse sr = client.prepareSearch().setQuery(qb).addFacet(facet).execute().actionGet();

// Facette en retour
RangeFacet byprice = sr.getFacets().facet("byprice");
for (RangeFacet.Entry entry : byprice) {
entry.to();
entry.from();
entry.min();
entry.max();
entry.mean();
entry.count();
}
```

Atelier 3

- Regarder les sources du test 3

  src/test/java/org/elasticsearchfr/handson/ex3/FacetTest.java

Mailing List ES France

elasticsearch-fr@googlegroups.com