

Überlog: ElasticSearch

Rapport et documentation

Auteurs : Wassim Ajjali
Professeur : Ghorbel Hatem

27 mai 2016

1	Introduction	3
2	Objectif	3
3	Les moteurs de recherche	3
3.1	ElasticSearch vs Solr	3
3.2	ElasticSearch	4
4	Base de données	6
5	Indexation et recherche de documents MongoDB avec ElasticSearch	6
6	Procédure d'installation et de configuration	7
6.1	Installation et configuration d'Elasticsearch	7
6.1.1	Premier démarrage	8
6.1.2	Modification du cluster et du nœud	8
6.2	Installation et configuration de MongoDB	8
6.2.1	Créez le répertoire de données	8
6.2.2	Définir des autorisations pour le répertoire de données.	9
6.2.3	Restaurer la base de données	9
6.2.4	Exécution	9
6.3	Indexation et recherche de documents MongoDB avec ElasticSearch	10
6.3.1	Installation du plugin elasticsearch-river-mongodb	10
6.3.2	Démarrage de MongoDB en mode replicaSet	11
6.3.3	Configurer le plugin river pour indexer les villes	11
6.3.4	Recherche par wildcard ou regex	13
6.4	Résultats	14
6.5	Mapping	15
6.5.1	Analysers	16
7	Conclusion	17

Tables des figures

<i>Figure 1 indexation et recherche avec MongoDB et Elasticsearch</i>	<i>6</i>
<i>Figure 2 Premier démarrage d'Elasticsearch</i>	<i>8</i>
<i>Figure 3 Affichage des base de données</i>	<i>9</i>
<i>Figure 4 Choix d'uberlog</i>	<i>10</i>
<i>Figure 5 Affichage de collection de uberlog</i>	<i>10</i>
<i>Figure 6 Parcours de la collection account</i>	<i>10</i>
<i>Figure 7 Log de création de l'index Logbook</i>	<i>12</i>
<i>Figure 8 Console Mongo river</i>	<i>12</i>
<i>Figure 9 Réponse console</i>	<i>13</i>
<i>Figure 10 Exemple de requête avec le plugin elastic-head</i>	<i>13</i>
<i>Figure 11 Cluster packet diagram</i>	<i>15</i>
<i>Figure 12 Résultat du mapping dans la console</i>	<i>16</i>
<i>Figure 13 Réponse de l'anaysers</i>	<i>16</i>

1 Introduction

Solenix fournit à ses clients un journal de bord électronique, nommé Überlog. Cette solution est utilisée principalement dans le domaine spatial et permet de gérer plusieurs journaux contenant un ensemble structuré d'événements. Un événement est constitué de texte et des métadonnées habituelles (auteur, date de création/modification, gravité, commentaires, fichiers joints, ...).

Les données d'Überlog sont sauvegardées dans MongoDB et chaque entrée est un document. Avec ce projet, Solenix voudrait analyser et démontrer la possibilité d'enregistrer les entrées d'Überlog dans Elasticsearch. En effet, certains clients de Solenix souhaitent réduire le nombre de bases de données utilisées pour stocker leurs données de mission. Elasticsearch est un candidat pour toutes les données avec une composante temporelle comme les entrées d'Überlog.

2 Objectif

Le projet consiste à :

- Analyser exactement quelles données d'Überlog peuvent être enregistrées efficacement dans Elasticsearch ;
- Étendre la couche d'accès aux données d'Überlog pour enregistrer ses données clés dans Elasticsearch et/ou MongoDB selon la configuration ;
- Définir les indexes dans Elasticsearch nécessaire à de bonnes performances
- Mettre en place une démonstration et comparer les performances de MongoDB et d'ElasticSearch pour un grand nombre d'entrées (> 1'000'000).

3 Les moteurs de recherche

Pour commencer nous avons commencé par regarder les moteurs de recherches et d'indexation disponible afin de pouvoir indexer correctement les données d'Überlog dans MongoDB. Très vite nous avons pu garder deux principaux moteurs à savoir :

- Solr
- Elasticsearch

3.1 Elasticsearch vs Solr

Dans cette partie on va s'intéresser à la comparaison des deux moteurs de recherche Elasticsearch et Solr le tableau ci-dessous montre les avantages et inconvénients de chacun.

À l'heure actuelle, aucune entreprise n'offre de distribution commerciale. Son point fort est la facilité avec laquelle il est possible de déployer une grappe de serveurs pour gérer un index distribué de très grande taille. C'est à ce niveau qu'il a fait sa marque et plusieurs développeurs ayant essayé de déployer une telle infrastructure sous Solr ont finalement adopté Elasticsearch, créant un véritable engouement autour du projet .

Lorsqu'exécuté sur un index de taille raisonnable (< 1M de documents), Solr offre des performances supérieures à Elasticsearch. C'est toutefois le contraire qui se produit avec une taille d'index plus importante ou lorsqu'un très grand volume de données est indexé en temps réel. Le tout est dû au fait que le traitement est distribué sur des index plus petits, hébergés sur plusieurs serveurs. Si 100 nouveaux documents sont ajoutés en même temps, ceux-ci peuvent être répartis sur 10 « shards » différents effectuant le traitement en parallèle. L'absence de transactions réduit aussi le temps de traitement pour l'ajout de documents.

Une répartition de la charge et une distribution de l'index peuvent aussi être réalisées avec Solr, mais le tout exige beaucoup de configuration et la mise en place de règles définies manuellement. Le projet SolrCloud 4.0 vise à répondre à ce problème en offrant des fonctions pour le découpage automatique de l'index en plusieurs « shards ». La gestion de la grappe de serveurs et de sa configuration ne sont toutefois pas incorporées dans le produit, mais reposent plutôt sur le produit Apache ZooKeeper. À l'heure actuelle, cette solution n'est pas encore comparable à Elastic Search en termes de convivialité et de facilité de déploiement.

Ce qu'on retiendra principalement, c'est que Elasticsearch a été conçu dès le départ pour résoudre la problématique Big data: distribution, réplication, gros volume, temps-réel. A cela s'ajoutent de réelles nouvelles fonctionnalités: la percolation, le schema-less (et autres).

De plus Elasticsearch permet d'indexer des données sous forme de documents, contrairement à Solr (autre moteur de recherche basé sur Lucene) qui gère des structures « à plat ». Un fichier CSV a une structure à plat

Exemple de fichier « à plat » :

nom;code postal;

Neuchatel;2000http://www.he-arc.ch

Lausanne;1000http://www.hes-so.ch/

Les fichiers XML ou JSON ont une structure documentaire, ci-dessous un exemple :

```
1 <ville>
2   <nom>paris</nom>
3   <codePostal>75001</codePostal>
4   <url>http://www.paris.fr</url>
5 </ville>
6 <ville>
7   <nom>boulogne-billancourt</nom>
8   <codePostal>92100</codePostal>
9   <url>http://www.boulognebillancourt.com/</url>
10 </ville>
```

3.2 Elasticsearch

ElasticSearch est un moteur de recherche libre basé sur Lucene qui permet notamment des recherches poussées au sein de texte. Il ne peut pas être utilisé comme votre moteur de base de données principal (On ne pourra jamais remplacer mysql ou mongoDB). Voici quelques points spécifiques :

- Elasticsearch peut être distribué et plusieurs instances (nodes) peuvent communiquer entre-elles dans un même cluster.
- Les données sont sauvegardées sous forme de documents JSON.
- Schema free, lorsque l'on index un nouveau document, Elastic Search détectera la structure et les types tout seul et construira les indexs pour rendre les données recherchables (on n'est pas obligé de créer le schéma avant d'enregistrer des données).
- Facile à utiliser, pour communiquer avec Elasticsearch on pourra utiliser une API RESTful.
- Elasticsearch permet des recherches full text puissantes, très puissantes.
- Mettre à jour partiellement des données est très pénible. Par défaut, il faudra lui envoyer tout le document pour le mettre à jour.
- On ne peut pas joindre des données. On peut cependant rechercher sur plusieurs types de données en même temps.

Terminologie :

- **Un index** est un peu comme une base de données sur un SGBD relationnel. Chaque index dispose d'un mapping, qui permet de définir la structure des types.
- **Le mapping** est similaire au schéma définition dans une base de donnée relationnelle. Le mapping peut être défini manuellement, mais aussi généré automatiquement quand les documents sont indexés.
- Les types sont comme des tables dans un système relationnel. Chaque type contient une liste des champs disponibles pour les documents.
- Les documents sont comme les lignes dans une base de données relationnelle. Ces documents sont stockés au format JSON et ont un index, un type et un id en plus des données.
- **Un node** est une instance d'ElasticSearch qui appartient à un cluster. Plusieurs noeuds peuvent être lancés sur un même serveur mais il est conseillé d'avoir un noeud par serveur
- **Un shard** est une instance de Lucene. Les shards sont gérés de manière automatique par Elasticsearch, il peut être primaire ou être un duplicat.
- **Les replica shards** permettent d'améliorer les performances et d'éviter les erreurs en remplaçant une shard primaire en cas d'erreur.

4 Base de données

Depuis quelques années on ne cesse d'entendre parler du NoSQL et des SGBD comme Cossandra utilisé auparavant par Facebook.

Le NoSQL (*Not only SQL*) désigne une catégorie de systèmes de gestion de bases de données (SGBD) qui n'est plus fondée sur l'architecture classique des bases relationnelles. Le NoSql repose sur une unité logique qui n'est plus la table, et les données ne sont en général pas manipulées avec SQL.

Parmi les nombreux SGBD faisant partie de la mouvance NoSQL se trouve MongoDB. MongoDB (de l'anglais *humongous* qui peut être traduit par « énorme ») est un système de gestion de bases de données orientées documents, répartitionnables sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données. Il est écrit en C++ et distribué sous licence AGPL.

De plus MongoDB est une base de données open source orientée documents qui fournit de hautes performances, une haute disponibilité, et mise à l'échelle automatique. Les enregistrements y sont sous forme de document, qui est une structure de données champ-valeur. Ils sont similaires à des objets JSON. Les valeurs d'un champ peuvent inclure d'autres documents, des tableaux, ou même des tableaux de documents.

5 Indexation et recherche de documents MongoDB avec ElasticSearch

Afin de bénéficier des avantages d'ElasticSearch et MongoDB et minimiser les inconvénients de chacun nous avons décidé de les lier les deux puisque les deux bases sont orientées documents et utilisent le même format de données (JSON).

Nous allons donc écrire dans MongoDB, lire et rechercher dans Elasticsearch.

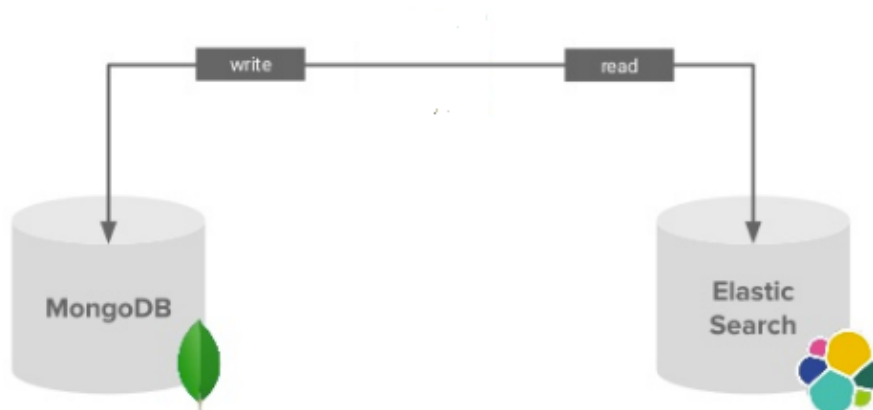


Figure 1 indexation et recherche avec Mongoddb et ElasticSearch

6 Procédure d'installation et de configuration

6.1 Installation et configuration d'Elasticsearch

L'installation d'ElasticSearch est assez simple :

- Récupérer ElasticSearch <http://www.elasticsearch.org/overview/elkdownloads/> (version 1.4.1 utilisée)
- Dézipper
- Lancer bin/elasticsearch.bat (sous windows) ou bin/elasticsearch (sous linux ou macOS)

Si vous avez l'erreur suivant **Unsupported major.minor version 51.0**, vérifiez bien que vous possédez une des dernières version de Java 7 ou Java 8. Vérifiez aussi que la variable d'environnement JAVA_HOME pointe sur une de ces versions. Pour tester qu'ElasticSearch est bien up, il suffit de taper dans un navigateur : <http://localhost:9200/>

Et d'avoir la réponse :

```
11  {
12    status: 200,
13    name: "Dmitri Smerdyakov",
14    cluster_name: "elasticsearch",
15    version:
16    {
17      number: "1.4.1",
18      build_hash: "89d3241d670db65f994242c8e8383b169779e2d4",
19      build_timestamp: "2014-11-26T15:49:29Z",
20      build_snapshot: false,
21      lucene_version: "4.10.2"
22    },
23    tagline: "You Know, for Search"
24  }
```

Avant d'aller plus loin, nous allons installer des consoles d'administration pour suivre ce que fait ElasticSearch. Il existe plusieurs plugins « console » et j'en ai choisi 3 :

elastichQ http://www.elastichq.org/support_plugin.html Aller dans le répertoire d'installation d'ElasticSearch et exécuter :

```
25 bin/plugin.bat -install royrusso/elasticsearch-HQ (sous windows)
```

```
26 bin/plugin -install royrusso/elasticsearch-HQ (sous linux)
```

La console sera accessible à l'adresse : http://localhost:9200/_plugin/HQ/ (une fois ElasticSearch démarré) **elasticsearch-head** <https://github.com/mobz/elasticsearch-head> Aller dans le répertoire d'installation d'ElasticSearch et exécuter

```
27 bin/plugin.bat -install mobz/elasticsearch-head (sous windows)
```

```
28 bin/plugin -install mobz/elasticsearch-head (sous linux)
```


La console sera accessible à l'adresse : http://localhost:9200/_plugin/head/ (une fois Elasticsearch démarré) **bigDesk** <http://bigdesk.org/> Aller dans le répertoire d'installation d'ElasticSearch et exécuter

```
29 bin/plugin.bat -install lukas-vlcek/bigdesk (sous windows)
```

```
30 bin/plugin -install lukas-vlcek/bigdesk (sous linux)
```

La console sera accessible à l'adresse : http://localhost:9200/_plugin/bigdesk/ (une fois Elasticsearch démarré) Cette console est particulièrement utile pour configurer les clusters/nœuds/shards ...

6.1.1 Premier démarrage

Au premier démarrage d'une instance d'ElasticSearch, un nœud est créé avec un nom aléatoirement choisi parmi 3000 noms des personnages Marvel. Dans mon exemple Dmitri Smerdyakov. Ce nœud cherche sur le réseau s'il existe un cluster appelé « elasticsearch » (original). Ce n'est pas le cas, alors il crée un cluster et se déclare **master** à l'intérieur de ce cluster. Par défaut le nœud est partitionné en 5 shards.

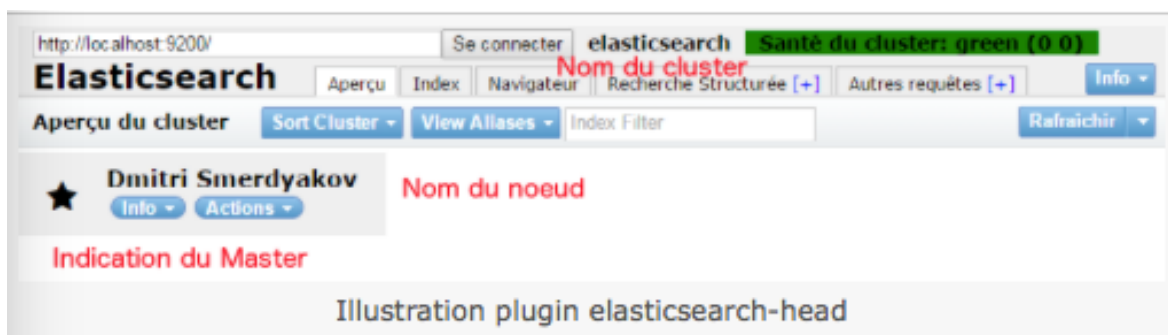


Figure 2 Premier démarrage d'Elasticsearch

6.1.2 Modification du cluster et du nœud

Nous allons faire une première modification dans le fichier de configuration d'ElasticSearch. Aller dans config/**elasticsearch.yml** modifier et dé-commenter les lignes suivantes :

```
31 cluster.name: cluster-blog  
32 node.name: "blog-01"
```

Redémarrer ElasticSearch et constater la modification dans une des consoles.

6.2 Installation et configuration de MongoDB

Dans cette section nous allons installer et configurer MongoDB

- Récupérer MongoDB <https://docs.mongodb.com/> (Version 2.6.4 utilisée)
- Dézipper
- Lancer bin/mongod.bat (sous windows) ou bin/ mongod (sous linux ou macOs) pour démarrer le serveur.
- Lancer bin/mongo.bat (sous windows) ou bin/ mongo (sous linux ou macOs) pour démarrer le client.

6.2.1 Créez le répertoire de données

Avant de commencer à MongoDB pour la première fois, créer le répertoire dans lequel le mongod processus va écrire les données. Par défaut, le mongod processus utilise le répertoire **/data/db**. Si

vous créez un répertoire autre que celui - ci, vous devez spécifier ce répertoire dans le **dbpath** Option lors du démarrage du mongod processus plus tard dans cette procédure.

L'exemple de commande suivant crée la valeur par défaut des données **/data/db** répertoire:

```
33  mkdir -p / data / db
```

6.2.2 Définir des autorisations pour le répertoire de données.

Avant d'exécuter mongod pour la première fois, assurez-vous que le compte d'utilisateur exécutant **mongod** a les autorisations de lecture et d'écriture pour le répertoire.

6.2.3 Restaurer la base de données

Uberlog nous a fourni un dump de sa base de données afin de pouvoir extraire ces données nous allons utiliser mongorestore. En effet ce package permet de créer une nouvelle base de données ou ajouter des données à une base de données existante. Cependant, mongorestore effectue des insertions seulement et ne procède pas à des mises à jour. Autrement dit, si la restauration du documents à une base de données et collection existante et documents existants ont la même valeur **_id** domaine que les documents à être restaurées, mongorestore ne vas pas écraser ces documents.

Afin de restaurer notre base de données nous allons donc utiliser la commande suivante :

```
34  /bin/mongorestore <path du dump>
```

6.2.4 Exécution

Pour exécuter MongoDB, exécutez le mongod processus à l'invite du système. Si nécessaire, indiquez le chemin de mongod ou le répertoire de données. Voir les exemples suivants.

Exécuter sans spécifier les chemins

```
35  mongod
```

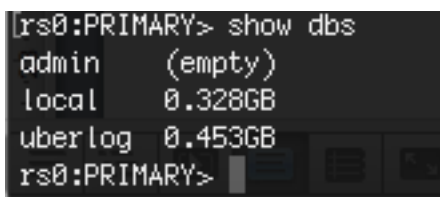
Indiquer le chemin du répertoire de données

```
36  mongod --dbpath <chemin au répertoire de données>
```

Lancer maintenant :

```
37  mongo
```

Nous allons indiquer maintenant de shell de l'instance de mongo que nous allons utiliser la base de donnée d'uberlog, ci dessous la procédure :



```
rs0:PRIMARY> show dbs
admin      (empty)
local      0.328GB
uberlog    0.453GB
rs0:PRIMARY>
```

Figure 3 Affichage des base de données

- Plugin : 2.0.4

L'installation est à faire à travers la commande

```
38 /bin/plugin.bat --install  
com.github.richardwilly98.elasticsearch/elasticsearch-river-mongodb/2.0.4
```

La console qui va bien après redémarrage : http://localhost:9200/_plugin/river-mongodb/

6.3.2 Démarrage de MongoDB en mode replicaSet

Le plugin `elasticsearch-river-mongodb` ne fonctionnera que si Mongo est en mode Replica Set (même s'il n'y a qu'une seule instance en cours). En ligne de commande :

```
39 mongod --port 27017 --dbpath <répertoire de données> --replSet rs0
```

Maintenant nous allons indiquer à MongoDB qu'il n'y a qu'une seule instance (celle qui tourne sur localhost:27017). Dans un shell mongo (commande mongo) on tape :

```
40 cfg = {  
41   "_id": "rs0",  
42   "version": 1,  
43   "members":  
44     [ { "_id": 0,  
45         "host": "localhost:27017"  
46     } ]  
47 }  
48 rs.initiate(cfg)
```

On attend que MongoDB ait fini l'initialisation en mode ReplicaSet :

6.3.3 Configurer le plugin river pour indexer les villes

Nous allons créer l'indexe **uberlog** avec des documents de type **Logbook2**, à partir de la base de données **uberlog** et de la collection **Logbook**. On utilise pour cela l'API REST :

```
49 curl -XPUT "localhost:9200/_river/uberlog/_meta" -d '  
50 {  
51   "type": "mongodb",  
52   "mongodb": {  
53     "servers": [  
54       { "host": "127.0.0.1", "port": 27017 }  
55     ],  
56     "options": { "secondary_read_preference": true},  
57     "db": "uberlog",  
58     "collection": "Logbook"  
59   },  
60   "index": {  
61     "name": "uberlog",  
62     "type": "Logbook2"  
63   }  
64 }'
```

Dans les logs d'ElasticSearch on peut voir :

```
[2016-05-26 21:19:59,507][INFO ][river.mongodb.util ] setRiverStatus called with uberlog - RUNNING
[2016-05-26 21:19:59,512][INFO ][cluster.metadata ] [uberlog] [_river] update_mapping [uberlog] (dynamic)
[2016-05-26 21:19:59,536][INFO ][cluster.metadata ] [uberlog] [uberlog] creating index, cause [api], shards [5]/[1], mappings []
[2016-05-26 21:19:59,607][INFO ][river.mongodb ] [uberlog] Creating MongoClient for [[127.0.0.1:27017]]
[2016-05-26 21:20:00,146][INFO ][org.elasticsearch.river.mongodb.MongoConfigProvider] MongoDB version - 2.6.5
[2016-05-26 21:20:00,163][INFO ][org.elasticsearch.river.mongodb.CollectionSlurper] MongoDBRiver is beginning initial import of uberlog.Logbook
[2016-05-26 21:20:00,170][INFO ][org.elasticsearch.river.mongodb.CollectionSlurper] Number of documents indexed in initial import of uberlog.Logbook: 8
[2016-05-26 21:20:00,217][INFO ][cluster.metadata ] [uberlog] [_river] update_mapping [uberlog] (dynamic)
[2016-05-26 21:20:00,224][INFO ][cluster.metadata ] [uberlog] [uberlog] update_mapping [Logbook2] (dynamic)
[2016-05-26 21:20:00,682][INFO ][cluster.metadata ] [uberlog] [river] update_mapping [uberlog] (dynamic)
```

Figure 7 Log de création de l'index Logbook

Et dans la console MongoDB river (http://localhost:9200/_plugin/river-mongodb/) nous avons :

The screenshot shows the 'MongoDB River Administration' interface. At the top, there are buttons for 'Home', 'Previous Page', 'Next Page', and 'Auto-refresh disabled'. Below this, the 'uberlogLogbook' is shown as 'RUNNING'. Underneath, it says 'Last Replicated - Documents Indexed - 0'. The main section displays the 'Settings' for the river, which is a JSON configuration. At the bottom left, there is a red 'Stop' button.

```
Settings:
{
  "index": {
    "name": "uberlogLogbook",
    "type": "Logbook2"
  },
  "type": "mongodb",
  "mongodb": {
    "servers": [
      {
        "port": 27017,
        "host": "127.0.0.1"
      }
    ],
    "options": {
      "secondary_read_preference": true
    },
    "collection": "Logbook",
    "db": "uberlog"
  }
}
```

Figure 8 Console Mongo river

6.3.3.1 Recherche full

```
65 curl -XGET
'http://localhost:9200/uberlog/Logbook2/_search?q=Anomalies&pretty=true'
```

La réponse est la suivante :

```

MacBook-Pro-de-wassim:elasticsearch-1.4.1 wassimajjali$ curl -XGET 'http://localhost:9200/uberlog/Logbook2/_search?q=Anomalies&pretty=true'
{
  "took": 2,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 1,
    "max_score": 0.17568314,
    "hits": [
      {
        "_index": "uberlog",
        "_type": "Logbook2",
        "_id": "51dfc85d27ffb8f1e5bac21a",
        "_score": 0.17568314,
        "_source": {
          "parent": {
            "ref": "Folder",
            "id": "51dfc7bd27ffb8f1e5bac217"
          },
          "templateGroups": [
            {
              "name": "Anomalies",
              "templates": [
                {
                  "name": "Link Missing TC",
                  "severity": "Warning",
                  "text": "Oops... TC",
                  "background_color": "#E3D7FF",
                  "class_name": "ch.solenix.uberlog.Link closed",
                  "icon": "info",
                  "soundLoop": false,
                  "sound": "NO_SOUND"
                },
                {
                  "name": "Link opened",
                  "severity": "Info",
                  "text": "Link opened",
                  "background_color": "#E3D7FF",
                  "class_name": "ch.solenix.uberlog.Link opened",
                  "icon": "info",
                  "soundLoop": false,
                  "sound": "NO_SOUND"
                },
                {
                  "name": "Warning",
                  "severity": "Warning",
                  "text": "Warning",
                  "background_color": "#E3D7FF",
                  "class_name": "ch.solenix.uberlog.Warning",
                  "icon": "warning",
                  "soundLoop": false,
                  "sound": "NO_SOUND"
                }
              ]
            },
            {
              "name": "Station ASD",
              "severity": "Info",
              "text": "Station ASD",
                  "background_color": "#E3D7FF",
                  "class_name": "ch.solenix.uberlog.Station ASD",
                  "icon": "info",
                  "soundLoop": false,
                  "sound": "NO_SOUND"
            }
          ]
        }
      }
    ]
  }
}

```

Figure 9 Réponse console

6.3.3.2 Recherche approximative

Si on ne connaît que le début du nom du Logbook:

```
66 curl -XGET 'http://localhost:9200/uberlog/Logbook2/_search?q=An*&pretty=true'
```

6.3.3.3 Recherche avancée

Il est possible de faire des requêtes beaucoup plus complexes.

Le plugin Elasticsearch Head permet de construire facilement ce genre de requête. On passe alors par une requête REST en POST :

Elasticsearch

http://localhost:9200/

Se connecter

cluster-uberlog

Santé du cluster: yellow (6 12)

Aperçu

Index

Navigateur

Recherche Structurée [+]

Autres requêtes [+]

Autres requêtes 1 [-]

Chercher dans

uberlog (10 docs)

les documents correspondant à

must

Logbook2.templateGroups.name

prefix

+

-

must

match_all

+

-

Recherche

Format d'affichage des résultats

Tableau

Nombre de Résultats: 10

Voir la requête source

Recherche sur 5 des 5 shards. 2 résultats. 0.002 secondes

_index	_type	_id	_score	parent.ref	parent.id	templateGroups.name	templateGroups.templates.name
uberlog	Logbook2	51dfc85d27ffb8f1e5bac21a	1	Folder	51dfc7bd27ffb8f1e5bac217	Anomalies	Link lost unexpectedly
uberlog	Logbook2	51dfc88e27ffb8f1e5bac220	1	Folder	51dfc7b827ffb8f1e5bac216	01 Support Activities	00 Login to console

Figure 10 Exemple de requête avec le plugin elastic-head

6.3.4 Recherche par wildcard ou regex

On peut vouloir chercher les logooks par wildcard en utilisant les caractères :

- : correspond à zéro ou plusieurs caractères, y compris le caractère espace
- ? : correspond à un seul caractère

Exemple :

```
67  {
68    "query":{
69      "wildcard":{
70        "name":"Co?l de *"
71      }
72    }
73  }
```

La recherche par regex permet d'être encore plus précis. On peut rechercher par exemple toutes les logbook dont le featureCode commence par LCT et finit par une lettre entre A et F :

```
74  {
75    "query":{
76      "regex":{
77        "feature.featureCode":"LCT[A-F]"
78      }
79    }
80  }
```

6.4 Résultats

Le plugin BigDesk, montre que les documents ont été uniformément indexés et répliqués dans l'indexe uberlog :

Cluster: cluster-uberlog
Number of nodes: 1
Status: **yellow**

Experimental cluster Pack diagram:



Figure 11 Cluster packet diagram

6.5 Mapping

Par défaut lorsqu'on indexe un document, Elasticsearch définit un mapping par défaut :

Les types de champs : String, date, ...

- Si ces champs doivent servir à la recherche ou juste être stockés
- Vous pouvez accéder au mapping de vos données avec la requête :

```
81 curl -XGET  
'http://localhost:9200/uberlog/Logbook2/_mapping?pretty=true'
```



```
MacBook-Pro-de-wassim:elasticsearch-1.4.1 wassimajjali$ curl -XGET 'http://localhost:9200/uberlog/Logbook2/_mapping?pretty=true'
{
  "uberlog" : {
    "mappings" : {
      "Logbook2" : {
        "properties" : {
          "backgroundColor" : {
            "type" : "string"
          },
          "className" : {
            "type" : "string"
          },
          "creationDate" : {
            "type" : "date",
            "format" : "dateOptionalTime"
          },
          "inheritMessageTypes" : {
            "type" : "boolean"
          },
          "inheritPermissions" : {
            "type" : "boolean"
          },
          "inheritSeverityLevels" : {
            "type" : "boolean"
          },
          "inheritTemplateGroups" : {
            "type" : "boolean"
          },
          "justificationPolicy" : {
            "type" : "string"
          },
          "messageTypes" : {
            "type" : "string"
          },
          "name" : {
            "type" : "string"
          }
        }
      }
    }
  }
}
```

Figure 12 Résultat du mapping dans la console

6.5.1 Analysers

La meilleure manière d'avoir une recherche qualitative est d'utiliser les bons « analysers » lors de l'indexation et lors de la recherche.

Si mes documents avait contenu beaucoup de texte en français, il aurait été conseillé d'utiliser le « french analyser » :

- Les articles français ne sont pas indexés (le, la, ...) car ils ne servent à rien
- La recherche avec ou sans accent fonctionne correctement

```
MacBook-Pro-de-wassim:elasticsearch-1.4.1 wassimajjali$ curl 'http://localhost:9200/uberlog/_analyze?tokenizer=letter&filters=asciifolding,lowercase&pretty=true' -d "an ies"
{
  "tokens" : [ {
    "token" : "an",
    "start_offset" : 0,
    "end_offset" : 2,
    "type" : "word",
    "position" : 1
  }, {
    "token" : "ies",
    "start_offset" : 3,
    "end_offset" : 6,
    "type" : "word",
    "position" : 2
  } ]
}
```

Figure 13 Réponse de l'analysers

7 Conclusion

Elasticsearch s'avère très performant dans le domaine de l'indexation et de la recherche. Que ce soit au niveau des temps de réponse ou de la qualité de réponse.

Il faut aussi mettre en avant la facilité de mise en place pour une indexation standard.

Un niveau plus avancé sera requis pour calibrer l'indexeur aux documents à structure plus complexes tels que des logs de serveurs, des textes, etc...

Maintenant il ne reste plus qu'à créer définir les indexes dans ElasticSearch nécessaire à de bonnes performances et créer une application web en AngularJs et jee afin de pouvoir indexer correctement notre base de données.