Introduction à ElasticSearch

Présentée par :

Romain Pignolet

LINAGORA

Lundi 7 Juillet 2014

RMLL MONTPELLIER 2014

Le libre et vous !
15èmes Rencontres Mondiales du Logiciel Libre
Du 5 au 11 juillet 2014

Sommaire

Présentation de Elasticsearch

- Elasticsearch est un outil de recherche distribué en temps réel et un outil d'analyse.
- Il est utilisé pour
  - recherche full text
  - recherche structurée
  - analyse
  - et les trois combinés

Qui utilise Elasticsearch ?

- Wikipedia (http://fr.wikipedia.org)
- The Guardian (http://www.theguardian.com)
- StackOverflow (http://stackoverflow.com/)
- GitHub (https://github.com/)
- Goldman Sachs (http://www.goldmansachs.com/)

Dépendances et fonctionnalités

Elasticsearch a besoin de :

- Apache Lucene™, un moteur de recherche full-text.
- Java donc la JVM est requise.

Elasticsearch est:

- un stockage de document temps réel distribué où **tous les champs** sont indéxés et consultable
- un moteur de recherche distribué avec de l'analyse temps réel
- capable de supporter la monté en charge avec une centaine de servers et des peta-octets de données structurées ou non

Sommaire

Installation et lancement d'Elasticsearch

- Simplement télécharger l'archive sur le site officiel (http://www.elasticsearch.org/overview/elkdownloads/) et décompresser.
- Pour le lancer, allez dans le répertoire créé par la décompression et lancer cette commande :

```
./bin/elasticsearch
```


Installation et lancement d'Elasticsearch

Testez le en lançant cette commande :

```
curl 'http://localhost:9200/?pretty'
```

Vous devriez voir une réponse comme cela :

```
{
  "status" : 200,
  "name" : "Brother Nature",
  "version" : {
    "number" : "1.1.0",
    "build_hash" : "2181
     e113dea80b4a9e31e58e9686658a2d46e363",
    "build_timestamp" : "2014-03-25T15:59:51Z",
    "build_snapshot" : false,
    "lucene_version" : "4.7"
  },
  "tagline" : "You Know, for Search"
}
```


Communication avec Elasticsearch

Il y a deux manières de communiquer avec Elasticsearch:

- Java API sur le port 9300
- Restful API sur le port 9200

Dans cette présentation nous ne parlerons que de l'API Rest.

Comment sont stockés les documents ?

- Orienté Document
- Le contenu de chaque Document est indexé
- Un Document possède un Type (qui défini son mapping)
- Les Types sont contenus dans un Index

Quelques comparaisons avec une base de donnée relationnelle :

Relational DB	Base de données	Tables	Lignes	Colonnes
Mongo DB	Base de données	Collections	Documents	Champs
Elasticsearch	Index	Types	Documents	Champs

Sommaire

API Rest - Type de requête

- PUT : création ou modification d'un document
- GET : récupération d'un document
- HEAD : test si un document existe
- DELETE : suppression d'un document

Retourne

- un code de retour HTTP (200, 404, etc.)
- une réponse encodé en JSON (sauf pour les requêtes HEAD)

Requête PUT - Exemple

La commande suivante sauvegarde un document dans l'index "megacorp" avec comme type "employee" et avec l'id "1" :

```
curl -XPUT 'localhost:9200/megacorp/employee/1' -d '
{
  "first_name" : "John",
  "last_name"  : "Smith",
  "age"        : 25,
  "about"      : "I love to go rock climbing",
  "interests"  : [ "sports", "music" ]
}'
```


Requête GET - Exemple

```
curl -XGET 'localhost:9200/megacorp/employee/1?pretty'
```

```
{
  "_index" : "megacorp",
  "_type" : "employee",
  "_id" : "2",
  "_version" : 1,
  "found": true,
  "_source" : {
    "first_name" : "John",
    "last_name"  : "Smith",
    "age"        : 25,
    "about"      : "I love to go rock climbing",
    "interests"  : [ "sports", "music" ]
  }
}
```


Requête GET - Endpoint _search

Nous allons rechercher tous les employés avec cette requête :

```
curl -XGET 'localhost:9200/megacorp/employee/_search?
  pretty'
```

Par défaut, la recherche retourne 10 résultats dans le tableau `hits`.

```
{
   "took" : 3,
   "timed_out" : false,
   "_shards" : { ... },
   "hits" : {
      "total" : 1,
      "max_score" : 1.0,
      "hits" : [ { ... } ]
   }
}
```

Note : la recherche inclue l'intégralité du document dans le champ `_source`.

Requête GET - Endpoint _search avec l'option query et Query DSL

Vous pouvez utiliser l'option de *query* (q) pour spécifier une simple demande comme :

```
curl -XGET 'localhost:9200/megacorp/employee/_search?q=
  last_name:Smith&pretty'
```

Cette requête demande tous les employés dont le `last_name` est égal à "Smith".

Ci-dessous la même requête mais en utilisant le *Query DSL* de Elasticsearch :

```
curl -XGET 'localhost:9200/megacorp/employee/_search?
  pretty' -d '
{
  "query" : {
    "match" : {
      "last_name" : "smith"
    }
  }
}'
```


Requête GET - Endpoint _search avec l'option query et Query DSL

Un autre exemple avec l'utilisation d'un filtre pour trouver tous les employés dont le nom de famille est "Smith" et agés de plus de 30 ans :

```
curl -XGET 'localhost:9200/megacorp/employee/_search?
  pretty' -d '
{
   "query" : {
      "filtered" : {
         "filter" : {
            "range" : { "age" : { "gt" : 30 } }
         },
         "query" : {
            "match" : { "last_name" : "smith" }
         }
      }
   }
}'
```


Sommaire

Indexation (champ _all) et metadata

- Toutes les données de chaque champ sont indexées
- Quand un document est indexé :
  1. Récupération de tous les champs
  2. Concaténation de ces champs dans une grosse chaine de caractères
  3. Sauvegarde cette chaine dans le champ spécial `_all`

- _index : Où le document est stocké.
- _type : Représente le mapping entre les champs et leurs types.
- _id : L'identificant unique du document.

Types and Mappings

Pour connaître le mapping pour un type vous pouvez faire une requête GET :

```
curl -XGET 'localhost:9200/megacorp/_mapping/employee?
  pretty'
```

```
{
  "megacorp" : {
    "mappings" : {
      "employee" : {
        "properties" : {
          "about"      : { "type" : "string" },
          "age"        : { "type" : "long" },
          "first_name" : { "type" : "string" },
          "interests"  : { "type" : "string" },
          "last_name"  : { "type" : "string" }
        } } } }
}
```


Sommaire

Definitions

Definition

Un Noeux :

- est une instance d'Elasticsearch en cours d'exécution
- est dans un cluster
- communique avec les autres noeuds du cluster

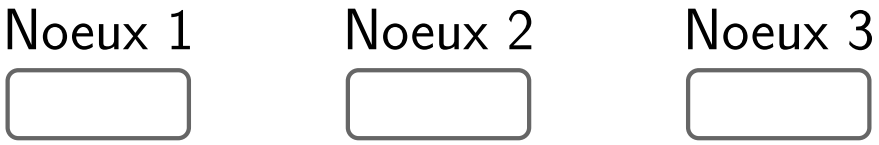

Figure: Cluster simple avec 3 noeuds vides

Definitions

Definition

Un Noeud Maître :

- est un noeud élu
- gère les changements dans le cluster :
  - creation ou suppression d'un index
  - ajout ou suppression d'un noeud du cluster

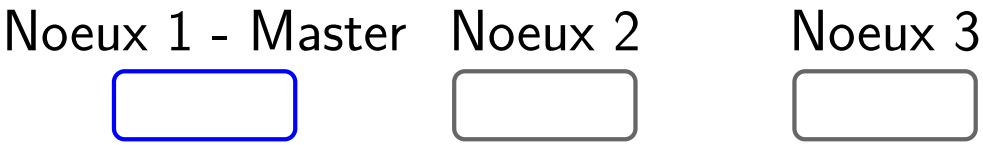

Figure: Cluster simple avec 3 noeuds vides et 1 maître

Definitions

Definition

Un Shard:

- est une "unité de travail" bas niveau
- est une seule instance de Lucene
- est un moteur de recherche complet

Nos documents sont stockés et indexés dans les Shards, mais nous ne nous adressons pas directement à eux : nos applications s'adressent à un index.

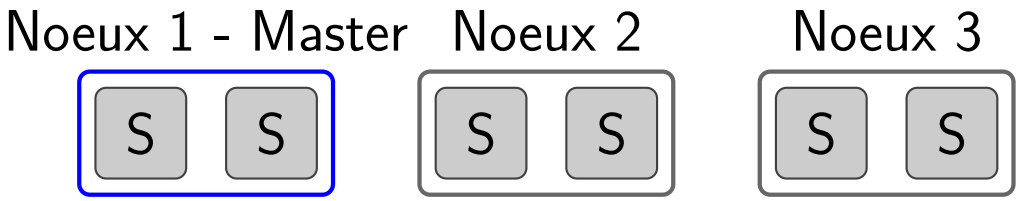

Figure: Cluster simple avec 3 noeuds vides, 1 maître et 6 shards

Definitions

Definition

Un Shard primaire :

- contient tous les documents dans un index
- peut avoir d'autres Shards primaires pour séparer les données (similaire au `RAID 0`)

Le nombre de Shard primaire pour un index est fixé au moment de la création de l'index.

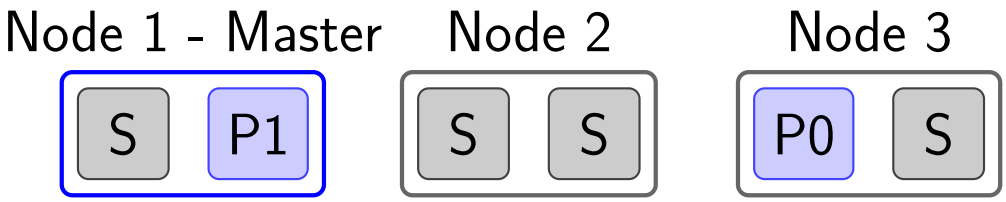

Figure: Cluster simple avec des Shards primaires

Definitions

Definition

Un Shard replica :

- est une copie d'un Shard primaire (similaire au `RAID 1`)
- est utilisé pour fournir des copies redondantes des données
- est utilisé pour répondre au requête de lecture comme chercher un document

Le nombre de Shard replica peut être changé à n'importe quel moment.

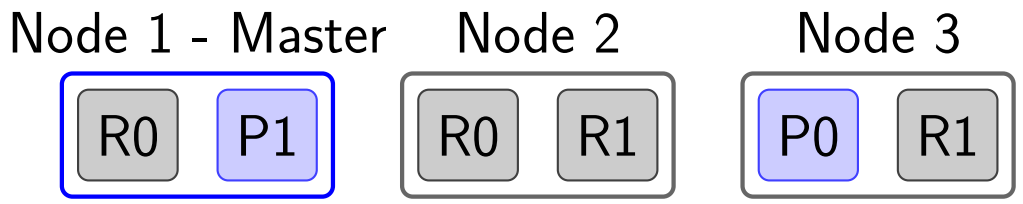

Figure: Cluster simple avec des Shards primaires

Statut du cluster

Pour savoir le statut du cluster :

```
curl -XGET 'http://localhost:9200/_cluster/health?pretty'
```

Le champ status donne une indication global sur le fonctionnement du cluster :

- vert : Tous les Shards primaires et replicas sont actifs (Le cluster fonctionne et la tolérance aux pannes est assurée).
- jaune : Tous les Shards primaires sont actifs, mais des Shards replicas ne sont pas tous actifs (Le cluster fonctionne mais si un noeud tombe la tolérance aux pannes n'est pas assurée).
- rouge : Des Shards primaires sont inactifs (Le cluster n'est pas fonctionnel).

Gestion des Shards

Créons un index `megacorp` en spécifiant que nous voulons 3 Shards primaires et 1 Shard replica (pour chaque primaire) :

```
curl -XPUT 'http://localhost:9200/megacorp' -d '
{
  "settings" : {
    "number_of_shards" : 3,
    "number_of_replicas" : 1
  }
}'
```

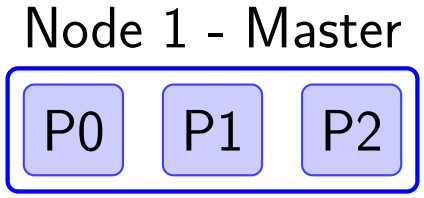

Figure: 1 noeuds avec 3 shards primaires

Dans cet état le statut du cluster est "jaune" car les Shards replicas ne peuvent pas être lancés.

Tolérance aux pannes

- 1 Noeud ⇒ Un point de défaillance
- La solution est simple : lancer un nouveau Noeud
- Le nouveau Noeud rejoindra automatiquement le cluster s'il a le même nom de cluster (`cluster.name`).

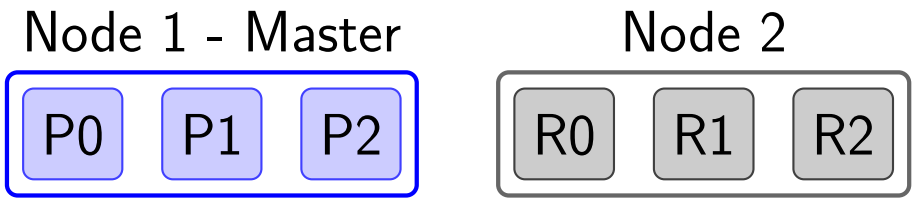

Figure: 2 noeuds avec 3 shards primaires et 1 shard replica pour chaque shard primaire

Le statut cluster est maintenant "vert".

Sommaire

Prérequis

Le plugin a une dépendance avec `elasticsearch-mapper-attachment` :

```
./bin/plugin --install elasticsearch/elasticsearch-mapper-
  attachments/2.0.0
```

Ensuite on install le plugin river pour MongoDB :

```
./bin/plugin --install com.github.richardwilly98.
  elasticsearch/elasticsearch-river-mongodb/2.0.0
```


Configuration de la river - Exemple

```
curl -XPUT "localhost:9200/_river/test/_meta" -d '
{
  "type": "mongodb",
  "mongodb": {
    "servers": [
      { "host": "10.75.9.193", "port": 27017 }
    ],
    "db": "test",
    "collection": "users"
  },
  "index": {
    "name": "users.idx",
    "type": "users"
  }
}'
```


Fin

Merci pour votre attention :)