
eBook Gratuit
APPRENEZ
Elasticsearch
eBook gratuit non affilié créé à partir des
contributeurs de Stack Overflow.
#elasticsear
ch

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: elasticsearch

It is an unofficial and free Elasticsearch ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Elasticsearch.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec Elasticsearch

Remarques

Elasticsearch est un serveur de recherche open source avancé basé sur Lucene et écrit en Java.

Il fournit des fonctionnalités de recherche distribuées, en texte intégral ou partiel, basées sur la requête et la géolocalisation, accessibles via une API HTTP REST.

Versions

Version	Date de sortie
5.2.1	2017-02-14
5.2.0	2017-01-31
5.1.2	2017-01-12
5.1.1	2016-12-08
5.0.2	2016-11-29
5.0.1	2016-11-15
5.0.0	2016-10-26
2.4.0	2016-08-31
2.3.0	2016-03-30
2.2.0	2016-02-02
2.1.0	2015-11-24
2.0.0	2015-10-28
1.7.0	2015-07-16
1.6.0	2015-06-09
1.5.0	2015-03-06
1.4.0	2014-11-05
1.3.0	2014-07-23
1.2.0	2014-05-22

Version	Date de sortie
1.1.0	2014-03-25
1.0.0	2014-02-14

Examples

Installer Elasticsearch sur Ubuntu 14.04

Conditions préalables

Pour exécuter Elasticsearch, un environnement d'exécution Java (JRE) est requis sur la machine. Elasticsearch nécessite Java 7 ou supérieur et recommande `Oracle JDK version 1.8.0_73` .

Installez Oracle Java 8

```
sudo add-apt-repository -y ppa:webupd8team/java
sudo apt-get update
echo "oracle-java8-installer shared/accepted-oracle-license-v1-1 select true" | sudo debconf-set-selections
sudo apt-get install -y oracle-java8-installer
```

Vérifier la version de Java

```
java -version
```

Télécharger et installer le paquet

Utiliser des binaires

1. Téléchargez la dernière version stable d'Elasticsearch ici .
2. Décompressez le fichier et exécutez

Linux:

```
$ bin/elasticsearch
```

Utiliser apt-get

Une alternative à télécharger elasticsearch à partir du site Web est de l'installer, en utilisant `apt-get` .

```
wget -qO - https://packages.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
echo "deb https://packages.elastic.co/elasticsearch/2.x/debian stable main" | sudo tee -a /etc/apt/sources.list.d/elasticsearch-2.x.list
```

```
sudo apt-get update && sudo apt-get install elasticsearch
sudo /etc/init.d/elasticsearch start
```

Installation d'elasticsearch version 5.x

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key add -
sudo apt-get install apt-transport-https
echo "deb https://artifacts.elastic.co/packages/5.x/apt stable main" | sudo tee -a
/etc/apt/sources.list.d/elastic-5.x.list
sudo apt-get update && sudo apt-get install elasticsearch
```

Exécuter en tant que service sous Linux:

Après l'installation, ce qui précède ne se lance pas. nous devons donc le démarrer en tant que service. Comment démarrer ou arrêter Elasticsearch selon que votre système utilise SysV init ou systemd. vous pouvez le vérifier avec la commande suivante.

```
ps -p 1
```

Si votre distribution utilise SysV init, vous devrez exécuter:

```
sudo update-rc.d elasticsearch defaults 95 10
sudo /etc/init.d/elasticsearch start
```

Sinon, si votre distribution utilise systemd:

```
sudo /bin/systemctl daemon-reload
sudo /bin/systemctl enable elasticsearch.service
```

Exécutez la commande `CURL` partir de votre navigateur ou d'un client REST pour vérifier si Elasticsearch a été correctement installé.

```
curl -X GET http://localhost:9200/
```

Installation d'Elasticsearch sous Windows

Conditions préalables

La version Windows d'Elasticsearch peut être obtenue à partir de ce lien: https://www.elastic.co/downloads/elasticsearch . La dernière version stable est toujours au top.

Comme nous installons sous Windows, nous avons besoin de l'archive `.ZIP` . Cliquez sur le lien dans la section `Downloads:` enregistrez le fichier sur votre ordinateur.

Cette version d'Elastic est "portable", ce qui signifie que vous n'avez pas besoin de lancer un

programme d'installation pour utiliser le programme. Décompressez le contenu du fichier dans un endroit facile à mémoriser. Pour démonstration, nous supposerons que tout est décompressé en `C:\elasticsearch` .

Notez que l'archive contient un dossier nommé `elasticsearch-<version>` par défaut, vous pouvez soit extraire ce dossier dans `C:\` et le renommer en `elasticsearch` soit créer `C:\elasticsearch` vous-même, puis décompresser uniquement le *contenu* du dossier dans l'archive. à là

Elasticsearch étant écrit en Java, il a besoin de Java Runtime Environment pour fonctionner. Donc, avant d'exécuter le serveur, vérifiez si Java est disponible en ouvrant une invite de commande et en tapant:

```
java -version
```

Vous devriez avoir une réponse qui ressemble à ceci:

```
java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b14)
Java HotSpot(TM) Client VM (build 25.91-b14, mixed mode)
```

Si vous voyez ce qui suit à la place

> "java" n'est pas reconnu comme une commande interne ou externe, un programme utilisable ou un fichier de commandes.

Java n'est pas installé sur votre système ou n'est pas configuré correctement. Vous pouvez suivre ce tutoriel pour (ré) installer Java. Assurez-vous également que ces variables d'environnement sont définies sur des valeurs similaires:

Variable	Valeur
JAVA_HOME	C: \ Program Files \ Java \ jre
CHEMIN	…; C: \ Program Files \ Java \ jre

Si vous ne savez pas encore comment inspecter ces variables, consultez ce tutoriel .

Exécuter à partir du fichier de commandes

Avec Java installé, ouvrez le dossier `bin` . Il peut être trouvé directement dans le dossier vers lequel vous avez décompressé le tout, il devrait donc être sous `c:\elasticsearch\bin` . Dans ce dossier se trouve un fichier appelé `elasticsearch.bat` qui peut être utilisé pour démarrer Elasticsearch dans une fenêtre de commande. Cela signifie que les informations consignées par le processus seront visibles dans la fenêtre d'invite de commandes. Pour arrêter le serveur, appuyez sur `CTRL C` ou fermez simplement la fenêtre.

Exécuter en tant que service Windows

Idéalement, vous ne voulez pas avoir une fenêtre supplémentaire dont vous ne pouvez vous débarrasser pendant le développement, et pour cette raison, Elasticsearch peut être configuré pour s'exécuter en tant que service.

Avant de pouvoir installer Elasticsearch en tant que service, nous devons ajouter une ligne au fichier `C:\elasticsearch\config\jvm.options` :

> Le programme d'installation du service nécessite que le paramètre de taille de la pile de threads soit configuré dans `jvm.options` avant d'installer le service. Sur Windows 32 bits, vous devez ajouter `-Xss320k` […] et sur Windows 64 bits, vous devez ajouter `-Xss1m` au fichier `jvm.options` . [la source]

Une fois cette modification effectuée, ouvrez une invite de commande et accédez au répertoire `bin` en exécutant la commande suivante:

```
C:\Users\user> cd c:\elasticsearch\bin
```

La gestion des services est gérée par `elasticsearch-service.bat` . Dans les anciennes versions, ce fichier pourrait simplement s'appeler `service.bat` . Pour voir tous les arguments disponibles, exécutez-le sans aucun:

```
C:\elasticsearch\bin> elasticsearch-service.bat

Usage: elasticsearch-service.bat install|remove|start|stop|manager [SERVICE_ID]
```

La sortie nous indique également qu'il existe un argument `SERVICE_ID` facultatif, mais nous pouvons l'ignorer pour l'instant. Pour installer le service, exécutez simplement:

```
C:\elasticsearch\bin> elasticsearch-service.bat install
```

Après avoir installé le service, vous pouvez le démarrer et l'arrêter avec les arguments respectifs. Pour démarrer le service, exécutez

```
C:\elasticsearch\bin> elasticsearch-service.bat start
```

et pour l'arrêter, courez

```
C:\elasticsearch\bin> elasticsearch-service.bat stop
```

Si vous préférez une interface graphique pour gérer le service à la place, vous pouvez utiliser la commande suivante:

```
C:\elasticsearch\bin> elasticsearch-service.bat manager
```

Cela ouvrira Elastic Service Manager, qui vous permet de personnaliser certains paramètres liés au service et d'arrêter / démarrer le service à l'aide des boutons situés en bas du premier onglet.

Indexer et récupérer un document

Elasticsearch est accessible via une API HTTP REST, généralement via la bibliothèque cURL. Les messages entre le serveur de recherche et le client (votre ou votre application) sont envoyés sous la forme de chaînes JSON. Par défaut, Elasticsearch s'exécute sur le port 9200.

Dans les exemples ci-dessous, `?pretty` est ajouté pour indiquer à Elasticsearch de préciser la réponse JSON. Lorsque vous utilisez ces points de terminaison dans une application, vous n'avez pas besoin d'ajouter ce paramètre de requête.

Documents d'indexation

Si nous avons l'intention de mettre à jour les informations dans un index ultérieurement, nous vous conseillons d'attribuer des identifiants uniques aux documents que vous indexez. Pour ajouter un document à l'index nommé `megacorp` , avec type `employee` et ID `1` exécutez:

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

Réponse:

```
{
  "_index": "megacorp",
  "_type": "employee",
  "_id": "1",
  "_version": 1,
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "created": true
}
```

L'index est créé s'il n'existe pas lorsque nous envoyons l'appel PUT.

Indexation sans identifiant

```
POST /megacorp/employee?pretty
{
```

```
    "first_name" :  "Jane",
    "last_name" :   "Smith",
    "age" :         32,
    "about" :       "I like to collect rock albums",
    "interests":  [ "music" ]
}
```

Réponse:

```
{
  "_index": "megacorp",
  "_type": "employee",
  "_id": "AVYg2mBJYy9ijdngfeGa",
  "_version": 1,
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "created": true
}
```

Récupération de documents

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

Réponse:

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

Récupérez 10 documents à partir de l'index `megacorp` avec le type `employee` :

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

Réponse:

```
{
  "took": 2,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 2,
    "max_score": 1,
    "hits": [
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "1",
        "_score": 1,
        "_source": {
          "first_name": "John",
          "last_name": "Smith",
          "age": 25,
          "about": "I love to go rock climbing",
          "interests": [
            "sports",
            "music"
          ]
        }
      },
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "AVYg2mBJYy9ijdngfeGa",
        "_score": 1,
        "_source": {
          "first_name": "Jane",
          "last_name": "Smith",
          "age": 32,
          "about": "I like to collect rock albums",
          "interests": [
            "music"
          ]
        }
      }
    ]
  }
}
```

Recherche simple à l'aide de la requête de `match` , qui recherche des correspondances exactes dans le champ fourni:

```
curl -XGET "http://localhost:9200/megacorp/employee/_search" -d'
{
    "query" : {
        "match" : {
            "last_name" : "Smith"
        }
    }
}'
```

Réponse:

```
{
  "took": 2,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "failed": 0
  },
  "hits": {
    "total": 1,
    "max_score": 0.6931472,
    "hits": [
      {
        "_index": "megacorp",
        "_type": "employee",
        "_id": "1",
        "_score": 0.6931472,
        "_source": {
          "first_name": "John",
          "last_name": "Smith",
          "age": 25,
          "about": "I love to go rock climbing",
          "interests": [
            "sports",
            "music"
          ]
        }
      }
    ]
  }
}
```

Paramètres de recherche de base avec des exemples:

Par défaut, le document indexé complet est renvoyé dans le cadre de toutes les recherches. C'est ce que l'on appelle la source (champ `_source` dans les résultats de recherche). Si nous ne voulons pas que tout le document source soit retourné, nous avons la possibilité de ne demander que quelques champs de la source à retourner, ou nous pouvons définir `_source` sur false pour omettre complètement le champ.

Cet exemple montre comment retourner deux champs, `account_number` et `balance` (à l'intérieur de `_source`), à partir de la recherche:

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": { "match_all": {} },
  "_source": ["account_number", "balance"]
}'
```

Notez que l'exemple ci-dessus réduit simplement les informations renvoyées dans le champ `_source` . Il ne renverra toujours qu'un seul champ nommé `_source` mais seuls les champs `account_number` et `balance` seront inclus.

Si vous venez d'un arrière-plan SQL, le concept ci-dessus est quelque peu similaire à la requête SQL

```
SELECT account_number, balance FROM bank;
```

Passons maintenant à la partie requête. Auparavant, nous avons vu comment la requête `match_all` est utilisée pour faire correspondre tous les documents. Introduisons maintenant une nouvelle requête appelée la requête de correspondance, qui peut être considérée comme une requête de recherche par champs de base (c'est-à-dire une recherche effectuée sur un champ ou un ensemble de champs spécifique).

Cet exemple renvoie le compte avec le `account_number` défini à `20` :

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": { "match": { "account_number": 20 } }
}'
```

Cet exemple renvoie tous les comptes contenant le terme "moulin" dans l' `address` :

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": { "match": { "address": "mill" } }
}'
```

Cet exemple renvoie tous les comptes contenant le terme "moulin" ou "voie" dans l' `address` :

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": { "match": { "address": "mill lane" } }
}'
```

Cet exemple est une variante de `match` (`match_phrase`) qui divise la requête en termes et ne renvoie que les documents contenant tous les termes de l' `address` dans les mêmes positions les uns par rapport aux autres [1] .

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": { "match_phrase": { "address": "mill lane" } }
}'
```

Introduisons maintenant la requête bool (ean). La requête bool nous permet de composer des requêtes plus petites en requêtes plus grandes en utilisant la logique booléenne.

Cet exemple compose deux requêtes de correspondance et renvoie tous les comptes contenant "mill" et "lane" dans l'adresse:

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": {
```

```
    "bool": {
      "must": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}'
```

Dans l'exemple ci-dessus, la clause bool `must` spécifie toutes les requêtes devant être vraies pour qu'un document soit considéré comme une correspondance.

En revanche, cet exemple compose deux requêtes de correspondance et renvoie tous les comptes contenant "mill" ou "lane" dans l' `address` :

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": {
    "bool": {
      "should": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}'
```

Dans l'exemple ci-dessus, la clause bool `should` spécifie une liste de requêtes dont l'une ou l'autre doit être vraie pour qu'un document soit considéré comme une correspondance.

Cet exemple compose deux requêtes de correspondance et renvoie tous les comptes qui ne contiennent ni "mill" ni "lane" dans l' `address` :

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": {
    "bool": {
      "must_not": [
        { "match": { "address": "mill" } },
        { "match": { "address": "lane" } }
      ]
    }
  }
}'
```

Dans l'exemple ci-dessus, la clause bool must_not spécifie une liste de requêtes dont aucune ne doit être vraie pour qu'un document soit considéré comme une correspondance.

Nous pouvons combiner des clauses must, should et must_not simultanément dans une requête bool. De plus, nous pouvons composer des requêtes bool dans chacune de ces clauses bool pour imiter toute logique booléenne complexe à plusieurs niveaux.

Cet exemple renvoie tous les comptes appartenant à des personnes qui ont exactement 40 ans et ne vivent pas à Washington (`WA` en abrégé):

```
curl -XPOST 'localhost:9200/bank/_search?pretty' -d '
{
  "query": {
    "bool": {
      "must": [
        { "match": { "age": "40" } }
      ],
      "must_not": [
        { "match": { "state": "WA" } }
      ]
    }
  }
}'
```

Installer Elasticsearch et Kibana sur CentOS 7

Pour exécuter Elasticsearch, un environnement d'exécution Java (JRE) est requis sur la machine. Elasticsearch nécessite Java 7 ou supérieur et recommande `Oracle JDK version 1.8.0_73` .

Alors, assurez-vous d'avoir Java dans votre système. Sinon, suivez la procédure:

```
# Install wget with yum
yum -y install wget

# Download the rpm jre-8u60-linux-x64.rpm for 64 bit
wget --no-cookies --no-check-certificate --header "Cookie:
gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-cookie"
"http://download.oracle.com/otn-pub/java/jdk/8u60-b27/jre-8u60-linux-x64.rpm"

# Download the rpm jre-8u101-linux-i586.rpm for 32 bit
wget --no-cookies --no-check-certificate --header "Cookie:
gpw_e24=http%3A%2F%2Fwww.oracle.com%2F; oraclelicense=accept-securebackup-cookie"
"http://download.oracle.com/otn-pub/java/jdk/8u101-b13/jre-8u101-linux-i586.rpm"

# Install jre-.*.rpm
rpm -ivh jre-.*.rpm
```

Java devrait maintenant être installé dans votre système centOS. Vous pouvez le vérifier avec:

```
java -version
```

Téléchargez et installez elasticsearch

```
# Download elasticsearch-2.3.5.rpm
wget
https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/rpm/elasticsearch/2.3.
2.3.5.rpm

# Install elasticsearch-.*.rpm
rpm -ivh elasticsearch-.*.rpm
```

Exécuter elasticsearch en tant que service systemd au démarrage

```
sudo systemctl daemon-reload
```

```
sudo systemctl enable elasticsearch
sudo systemctl start elasticsearch

# check the current status to ensure everything is okay.
systemctl status elasticsearch
```

Installer Kibana

Importer d'abord la clé GPG sur le rpm

```
sudo rpm --import http://packages.elastic.co/GPG-KEY-elasticsearch
```

Ensuite, créez un dépôt local `kibana.repo`

```
sudo vi /etc/yum.repos.d/kibana.repo
```

Et ajoutez le contenu suivant:

```
[kibana-4.4]
name=Kibana repository for 4.4.x packages
baseurl=http://packages.elastic.co/kibana/4.4/centos
gpgcheck=1
gpgkey=http://packages.elastic.co/GPG-KEY-elasticsearch
enabled=1
```

Installez maintenant le kibana en suivant la commande suivante:

```
yum -y install kibana
```

Commencez avec:

```
systemctl start kibana
```

Vérifier l'état avec:

```
systemctl status kibana
```

Vous pouvez l'exécuter en tant que service de démarrage.

```
systemctl enable kibana
```

Lire Démarrer avec Elasticsearch en ligne:
https://riptutorial.com/fr/elasticsearch/topic/941/demarrer-avec-elasticsearch

Chapitre 2: Agrégations

Syntaxe

- ". - [, "agrégations": {[<sub_aggregation>] +}]? -} - [, "<nom_agrégation_2>": {...}] * -}

Examples

Agrégation moyenne

Il s'agit d'une agrégation de mesures à valeur unique qui calcule la moyenne des valeurs numériques extraites des documents agrégés.

```
POST /index/_search?
{
    "aggs" : {
        "avd_value" : { "avg" : { "field" : "name_of_field" } }
    }
}
```

L'agrégation ci-dessus calcule la note moyenne sur tous les documents. Le type d'agrégation est moy et le paramètre de champ définit le champ numérique des documents sur lesquels la moyenne sera calculée. Ce qui précède renverra ce qui suit:

```
{
    ...
    "aggregations": {
        "avg_value": {
            "value": 75.0
        }
    }
}
```

Le nom de l'agrégation (avg_grade ci-dessus) sert également de clé permettant d'extraire le résultat de l'agrégation de la réponse renvoyée.

Agrégation de cardinalité

Une agrégation de mesures à valeur unique qui calcule un compte approximatif de valeurs distinctes. Les valeurs peuvent être extraites de champs spécifiques du document ou générées par un script.

```
POST /index/_search?size=0
{
    "aggs" : {
        "type_count" : {
            "cardinality" : {
                "field" : "type"
            }
```

```
        }
    }
}
```

Réponse:

```
{
    ...
    "aggregations" : {
        "type_count" : {
            "value" : 3
        }
    }
}
```

Agrégation étendue des statistiques

Une agrégation de métriques à valeurs multiples qui calcule les statistiques sur les valeurs numériques extraites des documents agrégés. Ces valeurs peuvent être extraites de champs numériques spécifiques dans les documents ou générées par un script fourni.

L'agrégation extended_stats est une version étendue de l'agrégation de statistiques, dans laquelle des mesures supplémentaires sont ajoutées, telles que sum_of_squares, variance, std_deviation et std_deviation_bounds.

```
{
    "aggs" : {
        "stats_values" : { "extended_stats" : { "field" : "field_name" } }
    }
}
```

Sortie de l'échantillon:

```
{
    ...

    "aggregations": {
        "stats_values": {
           "count": 9,
           "min": 72,
           "max": 99,
           "avg": 86,
           "sum": 774,
           "sum_of_squares": 67028,
           "variance": 51.55555555555556,
           "std_deviation": 7.180219742846005,
           "std_deviation_bounds": {
            "upper": 100.36043948569201,
            "lower": 71.63956051430799
           }
        }
    }
}
```

Lire Agrégations en ligne: https://riptutorial.com/fr/elasticsearch/topic/10745/agregations

Chapitre 3: Analyseurs

Remarques

Les analyseurs prennent le texte d'un champ de chaîne et génèrent des jetons qui seront utilisés lors de l'interrogation.

Un analyseur fonctionne dans une séquence:

- `CharFilters` (zéro ou plus)
- `Tokenizer` (One)
- `TokenFilters` (zéro ou plus)

L'analyseur peut être appliqué aux mappages de sorte que, lorsque les champs sont indexés, ils soient effectués sur une base individuelle plutôt que sur la chaîne dans son ensemble. Lors de l'interrogation, la chaîne d'entrée sera également exécutée via l'analyseur. Par conséquent, si vous normalisez le texte dans l'analyseur, il correspondra toujours même si la requête contient une chaîne non normalisée.

Examples

Cartographie

Un analyseur peut être appliqué à un mappage en utilisant "analyseur", par défaut l'analyseur "standard" est utilisé. Alternativement, si vous ne souhaitez pas utiliser d'analyseur (parce que la segmentation ou la normalisation ne serait pas utile), vous pouvez spécifier "index": "not_analyzed"

```
PUT my_index
{
  "mappings": {
    "user": {
      "properties": {
        "name": {
          "type": "string"
          "analyzer":  "my_user_name_analyzer"
        },
        "id": {
          "type": "string",
          "index": "not_analyzed"
        }
      }
    }
  }
}
```

Multi-champs

Parfois, il peut être utile d'avoir plusieurs index distincts d'un champ avec différents analyseurs.

Vous pouvez utiliser la fonctionnalité multi-champs pour le faire.

```
PUT my_index
{
  "mappings": {
    "user": {
      "properties": {
        "name": {
          "type": "string"
          "analyzer": "standard",
          "fields": {
            "special": {
               "type": "string",
               "analyzer": "my_user_name_analyzer"
            },
            "unanalyzed": {
              "type": "string",
              "index": "not_analyzed"
            }
          }
        }
      }
    }
  }
}
```

Lors de l'interrogation, au lieu d'utiliser simplement "user.name" (qui dans ce cas utiliserait toujours le standard Analyzer), vous pouvez utiliser "user.name.special" ou "user.name.unanalyzed". Notez que le document restera inchangé, cela ne concerne que l'indexation.

Analyseurs

L'analyse dans elasticsearch entre en contexte lorsque vous souhaitez analyser les données de votre index.

Les analyseurs nous permettent d'effectuer les opérations suivantes:

- Abréviations
- Enracinement
- Manipulation de typo

Nous allons regarder chacun d'eux maintenant.

1. **Abréviations** :

   En utilisant des analyseurs, nous pouvons dire à elasticsearch comment traiter les abréviations dans nos données, c'est-à-dire dr => Doctor.

2. **Stemming** :

   L’utilisation des méthodes d’analyse nous permet d’utiliser des mots de base pour des verbes modifiés comme

Mot	Modifications
exiger	exigence requise

3. **Typo Handling** :

   Les analyseurs fournissent également une gestion des fautes de frappe en interrogeant si nous recherchons des mots particuliers, disons «résurrection», alors elasticsearch retournera les résultats dans lesquels les fautes de frappe sont présentes.

Mot	Modifications
résurrection	la résurrection, la résurrection

Analyseurs dans Elasticsearch

1. la norme
2. Simple
3. Espace blanc
4. Arrêtez
5. Mot-clé
6. Modèle
7. La langue
8. Boule de neige

Ignorer l'analyseur de cas

Parfois, il peut être nécessaire d'ignorer la casse de notre requête par rapport à la correspondance dans le document. Un analyseur peut être utilisé dans ce cas pour ignorer le cas lors de la recherche. Chaque champ devra contenir cet analyseur dans sa propriété pour pouvoir fonctionner:

```
"settings": {
        "analysis": {
            "analyzer": {
                "case_insensitive": {
               "tokenizer": "keyword",
               "filter": ["lowercase"]
            }
            }
        }
    }
```

Lire Analyseurs en ligne: https://riptutorial.com/fr/elasticsearch/topic/6232/analyseurs

Chapitre 4: API de recherche

Introduction

L'API de recherche vous permet d'exécuter une requête de recherche et de récupérer des résultats de recherche correspondant à la requête. La requête peut être fournie en utilisant une simple chaîne de requête en tant que paramètre ou en utilisant un corps de requête.

Examples

Le routage

Lors de l'exécution d'une recherche, celle-ci sera diffusée vers tous les fragments d'index / indices (round robin entre les répliques). Les fragments qui seront recherchés peuvent être contrôlés en fournissant le paramètre de routage. Par exemple, lors de l'indexation de tweets, la valeur de routage peut être le nom d'utilisateur:

```
curl -XPOST 'localhost:9200/twitter/tweet?routing=kimchy&pretty' -d'
{
    "user" : "kimchy",
    "postDate" : "2009-11-15T14:12:12",
    "message" : "trying out Elasticsearch"
}'
```

Rechercher en utilisant le corps de la requête

Des recherches peuvent également être effectuées sur elasticsearch à l'aide d'une recherche DSL. L'élément de requête dans le corps de la requête de recherche permet de définir une requête à l'aide de la requête DSL.

```
GET /my_index/type/_search
{
    "query" : {
        "term" : { "field_to_search" : "search_item" }
    }
}
```

Multi recherche

L'option multi_search nous permet de rechercher une requête dans plusieurs champs à la fois.

```
GET /_search
{
  "query": {
    "multi_match" : {
      "query":    "text to search",
      "fields": [ "field_1", "field_2" ]
    }
```

```
  }
}
```

Nous pouvons également augmenter le score de certains champs en utilisant l'opérateur boost (^), et utiliser des caractères génériques dans le nom du champ (*)

```
GET /_search
    {
      "query": {
        "multi_match" : {
          "query":    "text to search",
          "fields": [ "field_1^2", "field_2*" ]
        }
      }
    }
```

Recherche d'URI et mise en évidence

Une demande de recherche peut être exécutée uniquement en utilisant un URI en fournissant des paramètres de requête. Toutes les options de recherche ne sont pas exposées lors de l'exécution d'une recherche à l'aide de ce mode, mais cela peut s'avérer utile pour des "tests de boucle" rapides.

```
GET Index/type/_search?q=field:value
```

Une autre fonctionnalité utile est la mise en évidence des correspondances dans les documents.

```
GET /_search
{
    "query" : {
        "match": { "field": "value" }
    },
    "highlight" : {
        "fields" : {
            "content" : {}
        }
    }
}
```

Dans le cas ci-dessus, le champ particulier sera mis en évidence pour chaque accès à la recherche.

Lire API de recherche en ligne: https://riptutorial.com/fr/elasticsearch/topic/8625/api-de-recherche

Chapitre 5: Apprendre Elasticsearch avec kibana

Introduction

Kibana est un outil de visualisation de données front-end pour elasticsearch. pour installer kibana, reportez-vous à la documentation de kibana. Pour lancer kibana sur localhost, accédez à https: // localhost: 5601 et accédez à la console kibana.

Examples

Explorez votre cluster à l'aide de Kibana

La syntaxe de la commande sera du type suivant:

```
<REST Verb> /<Index>/<Type>/<ID>
```

Exécutez la commande suivante pour explorer le cluster elasticsearch via la console Kibana.

- Pour vérifier la santé du cluster

```
GET /_cat/health?v
```

- Pour lister tous les indices

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

- Pour créer un index avec nom car

```
 PUT /car?pretty
```

- Pour indexer le document avec nom car de type externe en utilisant l'id 1

```
PUT /car/external/1?pretty
{
  "name": "Tata Nexon"
}
```

la réponse de la requête ci-dessus sera:

```
{
  "_index": "car",
  "_type": "external",
  "_id": "1",
  "_version": 1,
```

```
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "created": true
}
```

- récupérer le document ci-dessus peut être fait en utilisant:

```
GET /car/external/1?pretty
```

- Pour supprimer un index

```
DELETE /car?pretty
```

Modifier vos données elasticsearch

Elasticsearch fournit des fonctionnalités de manipulation de données et de recherche de données en temps quasi réel. Dans cet exemple, nous avons des opérations de mise à jour, de suppression et de traitement par lots.

- Mettre à jour le même document. Supposons que nous ayons déjà indexé un document sur / car / external / 1. Ensuite, l'exécution de la commande d'indexation des données remplace le document précédent.

```
PUT /car/external/1?pretty
{
  "name": "Tata Nexa"
}
```

Le document de voiture précédent à l'id 1 avec le nom "Tata Nexon" sera mis à jour avec le nouveau nom "Tata Nexa"

- indexer les données avec un identifiant explicite

```
POST /car/external?pretty
{
  "name": "Jane Doe"
}
```

  pour indexer le document sans un identifiant, nous utilisons le verbe **POST** au lieu du verbe **PUT** . Si nous ne fournissons pas d'identifiant, elasticsearch générera un identifiant aléatoire et l'utilisera ensuite pour indexer le document.

- Mettre à jour le document précédent à un identifiant partiellement.

```
POST /car/external/1/_update?pretty
{
```

```
  "doc": { "name": "Tata Nex" }
}
```

- mettre à jour le document avec des informations supplémentaires

```
POST /car/external/1/_update?pretty
{
  "doc": { "name": "Tata Nexon", "price": 1000000 }
}
```

- mettre à jour le document en utilisant des scripts simples.

```
POST /car/external/1/_update?pretty
{
  "script" : "ctx._source.price += 50000"
}
```

ctx._source fait référence au document source en cours de mise à jour. Le script ci-dessus fournit un seul script à mettre à jour en même temps.

- Supprimer le document

```
DELETE /car/external/1?pretty
```

Remarque: la suppression d'un index complet est plus efficace que la suppression de tous les documents à l'aide de l'API Supprimer par requête

Le traitement par lots

Outre l'indexation de la mise à jour et de la suppression du document, elasticsearch fournit également la possibilité d'effectuer l'une des opérations ci-dessus par lots à l'aide de l'API **_bulk** .

- pour mettre à jour plusieurs documents en utilisant l'API **_bulk**

```
POST /car/external/_bulk?pretty
{"index":{"_id":"1"}}
{"name": "Tata Nexon" }
{"index":{"_id":"2"}}
{"name": "Tata Nano" }
```

- pour mettre à jour et supprimer les documents en utilisant l'API **_bulk**

```
POST /car/external/_bulk?pretty
{"update":{"_id":"1"}}
{"doc": { "name": "Tata Nano" } }
{"delete":{"_id":"2"}}
```

Si une opération échoue, l'API en vrac ne s'arrête pas. Il exécute toutes les opérations et retourne enfin le rapport pour toutes les opérations.

Lire Apprendre Elasticsearch avec kibana en ligne:

https://riptutorial.com/fr/elasticsearch/topic/10058/apprendre-elasticsearch-avec-kibana

Chapitre 6: Commandes Curl

Syntaxe

- curl -X <VERBE> '<PROTOCOL>: // <HÔTE>: <PORT> / <PATH>? <QUERY_STRING>' -d '<CORPS>'
- Où:
- VERBE: méthode ou verbe HTTP approprié: GET, POST, PUT, HEAD ou DELETE
- PROTOCOLE: http ou https (si vous avez un proxy https devant Elasticsearch.)
- HOST: nom d'hôte de n'importe quel nœud de votre cluster Elasticsearch ou localhost pour un nœud sur votre ordinateur local.
- PORT: port exécutant le service HTTP Elasticsearch, dont la valeur par défaut est 9200.
- PATH: API Endpoint (par exemple, _count renverra le nombre de documents dans le cluster). Le chemin peut contenir plusieurs composants, tels que _cluster / stats ou _nodes / stats / jvm
- QUERY_STRING: Tous les paramètres de chaîne de requête facultatifs (par exemple, "pretty" impriment la réponse JSON pour la rendre plus lisible).
- BODY: Un corps de requête encodé en JSON (si la requête en a besoin.)
- Référence: Parler à Elasticsearch: Elasticsearch Docs

Examples

Commande Curl pour compter le nombre de documents dans le cluster

```
curl -XGET 'http://www.example.com:9200/myIndexName/_count?pretty'
```

Sortie:

```
{
  "count" : 90,
  "_shards" : {
    "total" : 6,
    "successful" : 6,
    "failed" : 0
  }
}
```

L'index contient 90 documents.

Lien de référence: ici

Récupérer un document par identifiant

```
curl -XGET 'http://www.example.com:9200/myIndexName/myTypeName/1'
```

Sortie:

```
{
    "_index" : "myIndexName",
    "_type" : "myTypeName",
    "_id" : "1",
    "_version" : 1,
    "found": true,
    "_source" : {
        "user" : "mrunal",
        "postDate" : "2016-07-25T15:48:12",
        "message" : "This is test document!"
    }
}
```

Lien de référence: ici

Créer un index

```
curl -XPUT 'www.example.com:9200/myIndexName?pretty'
```

Sortie:

```
{
  "acknowledged" : true
}
```

Lien de référence: ici

Liste tous les indices

```
curl 'www.example.com:9200/_cat/indices?v'
```

sortie:

```
health status index               pri rep docs.count docs.deleted store.size pri.store.size
green  open   logstash-2016.07.21   5   1       4760            0      4.8mb          2.4mb
green  open   logstash-2016.07.20   5   1       7232            0      7.5mb          3.7mb
green  open   logstash-2016.07.22   5   1      93528            0    103.6mb           52mb
green  open   logstash-2016.07.25   5   1      20683            0     41.5mb         21.1mb
```

Lien de référence: ici

Supprimer un index

```
curl -XDELETE 'http://www.example.com:9200/myIndexName?pretty'
```

sortie:

```
{
  "acknowledged" : true
}
```

Lien de référence: ici

Liste tous les documents d'un index

```
curl -XGET http://www.example.com:9200/myIndexName/_search?pretty=true&q=*:*
```

Cela utilise l'API de `Search` et renverra toutes les entrées sous l'index `myIndexName` .

Lien de référence: ici

Lire Commandes Curl en ligne: https://riptutorial.com/fr/elasticsearch/topic/3703/commandes-curl

Chapitre 7: Configuration Elasticsearch

Remarques

Elasticsearch est livré avec un ensemble de valeurs par défaut qui offrent une expérience de développement optimale. L'affirmation implicite est qu'il n'est pas nécessairement bon pour la production, qui doit être adaptée à vos propres besoins et ne peut donc pas être prédite.

Les paramètres par défaut facilitent le téléchargement et l'exécution de plusieurs nœuds *sur le même ordinateur* sans modification de la configuration.

Où sont les paramètres?

A l'intérieur de chaque installation d'Elasticsearch se trouve un `config/elasticsearch.yml` . C'est là que vivent les paramètres suivants:

- `cluster.name`
  - Le nom du cluster auquel le nœud se joint. Tous les nœuds du même cluster **doivent** partager le même nom.
  - Actuellement, par défaut, `elasticsearch` .
- `node.*`
  - `node.name`
    - S'il n'est pas fourni, un nom aléatoire sera généré à *chaque démarrage du nœud* . Cela peut être amusant, mais ce n'est pas bon pour les environnements de production.
    - Les noms ne doivent *pas* nécessairement être uniques, mais ils **doivent** être uniques.
  - `node.master`
    - Un réglage booléen. Lorsque la valeur est `true` , cela signifie que le nœud est un nœud maître éligible et qu'il peut s'agir *du* nœud maître élu.
    - La valeur par défaut est `true` , ce qui signifie que chaque nœud est un nœud maître éligible.
  - `node.data`
    - Un réglage booléen. Lorsque la valeur est `true` , cela signifie que le nœud stocke des données et gère l'activité de recherche.
    - La valeur par défaut est `true` .
- `path.*`
  - `path.data`
    - L'emplacement des fichiers écrits pour le noeud. *Tous les nœuds utilisent ce répertoire* pour stocker les métadonnées, mais les nœuds de données l'utilisent également pour stocker / indexer les documents.
    - Par défaut à `./data` .
      - Cela signifie que des `data` seront créées pour vous en tant que répertoire homologue à `config` *dans* le répertoire Elasticsearch.
  - `path.logs`
    - L'emplacement des fichiers journaux est écrit.
    - La valeur par défaut est `./logs` .

- `network.*`
    - `network.host`
        - Par défaut, `_local_` , qui est effectivement `localhost` .
            - Cela signifie que, par défaut, les nœuds ne peuvent pas être communiqués depuis l'extérieur de la machine actuelle!
    - `network.bind_host`
        - Potentiellement un tableau, cela indique à Elasticsearch quelles adresses de la machine en cours pour lier les sockets également.
            - C'est cette liste qui permet aux machines extérieures à la machine (par exemple, d'autres nœuds du cluster) de communiquer avec ce nœud.
        - Par défaut, `network.host` .
    - `network.publish_host`
        - Un hôte singulier utilisé pour annoncer aux autres nœuds comment communiquer au mieux avec ce nœud.
            - Lors de la fourniture d' un tableau à `network.bind_host` , cela devrait être le *seul* hôte qui est destiné à être utilisé pour la communication inter-noeuds.
        - Par défaut, network.host`.
- `discovery.zen.*`
    - `discovery.zen.minimum_master_nodes`
        - Définit le quorum pour l'élection principale. Cela **doit** être défini en utilisant cette équation: `(M / 2) + 1` où `M` est le nombre de nœuds maîtres *éligibles* (nœuds utilisant `node.master: true` implicitement ou explicitement).
        - La valeur par défaut est `1` , ce qui est uniquement valable pour un cluster à nœud unique!
    - `discovery.zen.ping.unicast.hosts`
        - Le mécanisme permettant de joindre ce noeud au reste d'un cluster.
        - Cela *devrait* répertorier les nœuds principaux éligibles afin qu'un nœud puisse trouver le reste du cluster.
        - La valeur à utiliser ici est le `network.publish_host` de ces autres nœuds.
        - La valeur par défaut est `localhost` , ce qui signifie qu'elle ne regarde que la machine locale pour qu'un cluster puisse se joindre.

Quels types de paramètres existent?

Elasticsearch propose trois types de paramètres différents:

- Paramètres à l'échelle du cluster
    - Ces paramètres s'appliquent à tous les éléments du cluster, tels que tous les nœuds ou tous les index.
- Paramètres du noeud
    - Ce sont des paramètres qui s'appliquent uniquement au nœud actuel.
- Paramètres d'index
    - Ce sont des paramètres qui s'appliquent uniquement à l'index.

Selon le réglage, il peut être:

- Changé dynamiquement à l'exécution
- Changé suite à un redémarrage (fermeture / ouverture) de l'index
    - Certains paramètres de niveau index ne nécessitent pas la fermeture et la réouverture de l'index, mais peuvent nécessiter une nouvelle fusion forcée de l'index pour que le paramètre s'applique.
        - Le niveau de compression d'un index est un exemple de ce type de paramètre. Il peut être changé dynamiquement, mais seuls les nouveaux *segments* profitent du changement. Donc, si un index ne change pas, il ne profite jamais de la modification, sauf si vous forcez l'index à recréer ses segments.
- Changé suite à un redémarrage du noeud
- Changé suite à un redémarrage du cluster
- Jamais changé

Vérifiez toujours la documentation de votre version d'Elasticsearch pour savoir ce que vous pouvez ou ne pouvez pas faire avec un paramètre.

Comment puis-je appliquer les paramètres?

Vous pouvez définir les paramètres de plusieurs manières, dont certaines ne sont pas suggérées:

- Arguments de ligne de commande

Dans Elasticsearch 1.x et 2.x, vous pouvez soumettre la plupart des paramètres en tant que propriétés système Java avec le préfixe `es.` :

```
$ bin/elasticsearch -Des.cluster.name=my_cluster -Des.node.name=`hostname`
```

Dans Elasticsearch 5.x, cela change pour éviter d'utiliser les propriétés du système Java, en utilisant plutôt un type d'argument personnalisé avec `-E` à la place de `-Des.` :

```
$ bin/elasticsearch -Ecluster.name=my_cluster -Enode.name=`hostname`
```

Cette approche de l'application des paramètres fonctionne parfaitement lorsque vous utilisez des outils tels que Puppet, Chef ou Ansible pour démarrer et arrêter le cluster. Cependant, cela fonctionne très mal lorsque vous le faites manuellement.

- Paramètres YAML
    - Illustré dans des exemples
- Paramètres dynamiques
    - Illustré dans des exemples

L'ordre dans lequel les paramètres sont appliqués est dans l'ordre le plus dynamique:

1. Paramètres transitoires
2. Paramètres persistants
3. Paramètres de ligne de commande

4. Paramètres YAML (statique)

Si le réglage est défini deux fois, une fois à l'un de ces niveaux, le niveau le plus élevé prend effet.

Examples

Paramètres statiques d'Elasticsearch

Elasticsearch utilise un fichier de configuration YAML (Yet Another Markup Language) qui se trouve dans le répertoire Elasticsearch par défaut (les installations RPM et DEB changent cet emplacement, entre autres).

Vous pouvez définir les paramètres de base dans `config/elasticsearch.yml` :

```
# Change the cluster name. All nodes in the same cluster must use the same name!
cluster.name: my_cluster_name

# Set the node's name using the hostname, which is an environment variable!
# This is a convenient way to uniquely set it per machine without having to make
#  a unique configuration file per node.
node.name: ${HOSTNAME}

# ALL nodes should set this setting, regardless of node type
path.data: /path/to/store/data

# This is a both a master and data node (defaults)
node.master: true
node.data: true

# This tells Elasticsearch to bind all sockets to only be available
#  at localhost (default)
network.host: _local_
```

Paramètres de cluster dynamique persistants

Si vous devez appliquer un paramètre de manière dynamique après le démarrage du cluster et qu'il peut effectivement être défini dynamiquement, vous pouvez le définir à l'aide de l'API `_cluster/settings` .

Les paramètres persistants sont l'un des deux types de paramètres à l'échelle du cluster pouvant être appliqués. Un paramètre persistant survivra un redémarrage du cluster complet.

Remarque: Tous les paramètres ne peuvent pas être appliqués dynamiquement. Par exemple, le nom du cluster ne peut pas être renommé dynamiquement. La plupart des paramètres au niveau du nœud ne peuvent pas non plus être définis dynamiquement (car ils ne peuvent pas être ciblés individuellement).

Ce n'est **pas** l'API à utiliser pour définir les paramètres de niveau index. Vous pouvez dire que ce paramètre est un paramètre de niveau d'index car il doit commencer par `index.` . Paramètres dont le nom est sous forme d' `indices.` *sont* des paramètres à l'échelle du cluster car ils s'appliquent à tous les index.

```
POST /_cluster/settings
{
  "persistent": {
    "cluster.routing.allocation.enable": "none"
  }
}
```

Attention : Dans Elasticsearch 1.x et 2.x, vous ne pouvez pas *désactiver* un paramètre persistant.

Heureusement, cela a été amélioré dans Elasticsearch 5.x et vous pouvez maintenant supprimer un paramètre en le définissant sur `null` :

```
POST /_cluster/settings
{
  "persistent": {
    "cluster.routing.allocation.enable": null
  }
}
```

Un paramètre non défini reprendra sa valeur par défaut ou toute valeur définie à un niveau de priorité inférieur (par exemple, paramètres de ligne de commande).

Paramètres de cluster dynamique transitoire

Si vous devez appliquer un paramètre de manière dynamique après le démarrage du cluster et qu'il peut effectivement être défini dynamiquement, vous pouvez le définir à l'aide de l'API `_cluster/settings` .

Les paramètres transitoires sont l'un des deux types de paramètres à l'échelle du cluster pouvant être appliqués. Un paramètre transitoire **ne** survivra **pas** à un redémarrage complet du cluster.

Remarque: Tous les paramètres ne peuvent pas être appliqués dynamiquement. Par exemple, le nom du cluster ne peut pas être renommé dynamiquement. La plupart des paramètres au niveau du nœud ne peuvent pas non plus être définis dynamiquement (car ils ne peuvent pas être ciblés individuellement).

Ce n'est **pas** l'API à utiliser pour définir les paramètres de niveau index. Vous pouvez dire que ce paramètre est un paramètre de niveau d'index car il doit commencer par `index.` . Paramètres dont le nom est sous forme d' `indices.` *sont* des paramètres à l'échelle du cluster car ils s'appliquent à tous les index.

```
POST /_cluster/settings
{
  "transient": {
    "cluster.routing.allocation.enable": "none"
  }
}
```

Avertissement : Dans Elasticsearch 1.x et 2.x, vous ne pouvez pas désactiver les paramètres transitoires sans un redémarrage complet du cluster.

Heureusement, cela a été amélioré dans Elasticsearch 5.x et vous pouvez maintenant supprimer un paramètre en le définissant sur null:

```
POST /_cluster/settings
{
  "transient": {
    "cluster.routing.allocation.enable": null
  }
}
```

Un paramètre non défini reprendra sa valeur par défaut ou toute valeur définie à un niveau de priorité inférieur (par exemple, paramètres `persistent`).

Paramètres d'index

Les paramètres d'index sont les paramètres qui s'appliquent à un seul index. Ces paramètres commenceront par `index.` . L'exception à cette règle est `number_of_shards` et `number_of_replicas` , qui existent également sous la forme `index.number_of_shards` et `index.number_of_replicas` .

Comme son nom l'indique, les paramètres de niveau index s'appliquent à un seul index. Certains paramètres doivent être appliqués au moment de la création, car ils ne peuvent pas être modifiés dynamiquement, tels que le paramètre `index.number_of_shards` , qui contrôle le nombre de fragments primaires pour l'index.

```
PUT /my_index
{
  "settings": {
    "index.number_of_shards": 1,
    "index.number_of_replicas": 1
  }
}
```

ou, dans un format plus concis, vous pouvez combiner des préfixes clés à chacun `.` :

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

Les exemples ci-dessus créeront un index avec les paramètres fournis. Vous pouvez modifier dynamiquement les paramètres par index à l'aide du `_settings` final index `_settings` . Par exemple, ici, nous modifions dynamiquement les paramètres de ralentissement *uniquement* pour le niveau d'avertissement:

```
PUT /my_index/_settings
{
  "index": {
```

```
    "indexing.slowlog.threshold.index.warn": "1s",
    "search.slowlog.threshold": {
      "fetch.warn": "500ms",
      "query.warn": "2s"
    }
  }
}
```

Attention : Elasticsearch 1.x et 2.x n'ont pas validé très strictement les noms de paramètres au niveau de l'index. Si vous aviez une faute de frappe, ou simplement créé un paramètre, alors il l'accepterait aveuglément, mais l'ignorerait sinon. Elasticsearch 5.x valide strictement les noms de paramètres et rejette toute tentative d'application de paramètres d'index avec un ou des paramètres inconnus (en raison d'une erreur de frappe ou d'un plug-in manquant). Les deux instructions s'appliquent à la modification dynamique des paramètres d'index et au moment de la création.

Paramètres d'index dynamique pour plusieurs indices en même temps

Vous pouvez appliquer la même modification indiquée dans l'exemple des `Index Settings` à *tous* les index existants avec une seule demande, voire un sous-ensemble:

```
PUT /*/_settings
{
  "index": {
    "indexing.slowlog.threshold.index.warn": "1s",
    "search.slowlog.threshold": {
      "fetch.warn": "500ms",
      "query.warn": "2s"
    }
  }
}
```

ou

```
PUT /_all/_settings
{
  "index": {
    "indexing.slowlog.threshold.index.warn": "1s",
    "search.slowlog.threshold": {
      "fetch.warn": "500ms",
      "query.warn": "2s"
    }
  }
}
```

ou

```
PUT /_settings
{
  "index": {
    "indexing.slowlog.threshold.index.warn": "1s",
    "search.slowlog.threshold": {
      "fetch.warn": "500ms",
      "query.warn": "2s"
```

```
    }
  }
}
```

Si vous préférez le faire de manière plus sélective, vous pouvez sélectionner plusieurs options sans tout fournir:

```
PUT /logstash-*,my_other_index,some-other-*/_settings
{
  "index": {
    "indexing.slowlog.threshold.index.warn": "1s",
    "search.slowlog.threshold": {
      "fetch.warn": "500ms",
      "query.warn": "2s"
    }
  }
}
```

Lire Configuration Elasticsearch en ligne:
https://riptutorial.com/fr/elasticsearch/topic/3411/configuration-elasticsearch

Chapitre 8: Différence entre les bases de données relationnelles et Elasticsearch

Introduction

Ceci est pour les lecteurs qui viennent de fond relationnel et veulent apprendre elasticsearch. Cette rubrique présente les cas d'utilisation pour lesquels les bases de données relationnelles ne sont pas une option appropriée.

Examples

Différence terminologique

Base de données relationnelle	Elasticsearch
Base de données	Indice
Table	Type
Rangée / Record	Document
Nom de colonne	champ

Le tableau ci-dessus établit une analogie entre les éléments de base de la base de données relationnelle et elasticsearch.

Installer

Considérant la structure suivante dans une base de données relationnelle:

```
create databse test;

use test;

create table product;

create table product (name varchar, id int PRIMARY KEY);

insert into product  (id,name) VALUES (1,'Shirt');

insert into product  (id,name) VALUES (2,'Red Shirt');

select * from product;

name      | id
----------+----
Shirt     |  1
Red Shirt |  2
```

Elasticsearch Equivalent:

```
POST test/product
{
    "id" : 1,
    "name" : "Shirt"
}

POST test/product
{
    "id" : 2,
    "name" : "Red Shirt"
}

GET test/product/_search


"hits": [
     {                                          ==============
        "_index": "test",                       ===> index    |
        "_type": "product",                     ===>type      |
        "_id": "AVzglFomaus3G2tXc6sB",                        |
        "_score": 1,                                          |
        "_source": {                                          |===> document
           "id": 2,                            ===>field      |
           "name": "Red Shirt"                 ===>field      |
        }                                                     |
     },                                         ==============
     {
        "_index": "test",
        "_type": "product",
        "_id": "AVzglD12aus3G2tXc6sA",
        "_score": 1,
        "_source": {
           "id": 1,
           "name": "Shirt"
        }
     }
  ]
```

Cas d'utilisation où les bases de données relationnelles ne conviennent pas

- L'essence de la recherche réside dans son ordre. Tout le monde veut que les résultats de recherche soient affichés de manière à ce que les meilleurs résultats soient affichés en haut. La base de données relationnelle n'a pas cette capacité. Elasticsearch, par contre, affiche les résultats sur la base de la pertinence par défaut.

  Installer

  Identique à celui utilisé dans l'exemple précédent.

  Énoncé de problème

  Supposons que l'utilisateur veuille chercher des `shirts` mais qu'il s'intéresse aux chemises de couleur `red`. Dans ce cas, les résultats contenant `red` mots-clés `red` et des `shirts` doivent figurer en tête de liste. Ensuite, les résultats des autres chemises doivent être affichés après

eux.

Solution utilisant une requête de base de données relationnelle

```
select * from product where name like '%Red%' or name like '%Shirt%';
```

Sortie

```
name       | id
-----------+----
Shirt      |  1
Red Shirt  |  2
```

Solution Elasticsearch

```
POST test/product/_search
{
     "query": {
          "match": {
            "name": "Red Shirt"
          }
     }
}
```

Sortie

```
"hits": [
  {
     "_index": "test",
     "_type": "product",
     "_id": "AVzglFomaus3G2tXc6sB",
     "_score": 1.2422675,              ===> Notice this
     "_source": {
        "id": 2,
        "name": "Red Shirt"
     }
  },
  {
     "_index": "test",
     "_type": "product",
     "_id": "AVzglD12aus3G2tXc6sA",
     "_score": 0.25427115,             ===> Notice this
     "_source": {
        "id": 1,
        "name": "Shirt"
     }
  }
 ]
```

Conclusion

Comme nous pouvons le voir ci-dessus, la base de données relationnelle a renvoyé des résultats dans un ordre aléatoire, tandis que Elasticsearch renvoie des résultats en ordre décroissant de `_score` calculé sur la base de la pertinence.

- Nous avons tendance à faire des erreurs en entrant la chaîne de recherche. Il y a des cas où l'utilisateur entre un paramètre de recherche incorrect. Les bases de données relationnelles ne gèrent pas de tels cas. Elasticsearch à la rescousse.

  Installer

  Identique à celui utilisé dans l'exemple précédent.

  Énoncé de problème

  L' *utilisateur veut rechercher On suppose* que *pour* des `shirts` , *mais il entre un mot incorrect* `shrt` *par erreur. L'utilisateur s'attend toujours à voir les résultats de la chemise* .

  Solution utilisant une requête de base de données relationnelle

  ```
  select * from product where name like '%shrt%' ;
  ```

  Sortie

  ```
  No results found
  ```

  Solution Elasticsearch

  ```
  POST /test/product/_search

   {
      "query": {
        "match": {
          "name": {
            "query": "shrt",
            "fuzziness": 2,
            "prefix_length": 0
           }
        }
      }
   }
  ```

  Sortie

  ```
   "hits": [
    {
       "_index": "test",
       "_type": "product",
       "_id": "AVzglD12aus3G2tXc6sA",
       "_score": 1,
       "_source": {
          "id": 1,
          "name": "Shirt"
       }
    },
    {
       "_index": "test",
       "_type": "product",
       "_id": "AVzglFomaus3G2tXc6sB",
  ```

```
      "_score": 0.8784157,
      "_source": {
         "id": 2,
         "name": "Red Shirt"
      }
   }
]
```

Conclusion

Comme nous pouvons le voir ci-dessus, la base de données relationnelle n'a renvoyé aucun résultat pour un mot incorrect recherché, tandis qu'Elasticsearch utilisant sa requête `fuzzy` spéciale renvoie des résultats.

Lire Différence entre les bases de données relationnelles et Elasticsearch en ligne: https://riptutorial.com/fr/elasticsearch/topic/10632/difference-entre-les-bases-de-donnees-relationnelles-et-elasticsearch

Chapitre 9: Différence entre les indices et les types

Remarques

Il est facile de voir les `type` comme une table dans une base de données SQL, où l' `index` est la base de données SQL. Cependant, ce n'est pas un bon moyen d'aborder le `type` s.

Tout sur les types

En fait, les types sont *littéralement* un champ de métadonnées ajouté à chaque document par Elasticsearch: `_type` . Les exemples ci-dessus ont créé deux types: `my_type` et `my_other_type` . Cela signifie que chaque document associé aux types a un champ supplémentaire défini automatiquement comme `"_type": "my_type"` ; Ceci est indexé avec le document, ce qui en fait un *champ de recherche ou de filtrage* , mais cela n'a pas d'impact sur le document brut lui-même, votre application n'a donc pas besoin de s'en préoccuper.

Tous les types vivent dans le même index, et donc dans les mêmes fragments collectifs de l'index. Même au niveau du disque, ils vivent dans les mêmes fichiers. La seule séparation fournie par la création d'un second type est une séparation logique. Chaque type, qu'il soit unique ou non, doit exister dans les mappages et tous ces mappages doivent exister dans votre état de cluster. Cela consomme de la mémoire et, si chaque type est mis à jour de manière dynamique, il réduit les performances à mesure que les mappages changent.

En tant que tel, il est recommandé de ne définir qu'un seul type, sauf si vous avez réellement besoin d'autres types. Il est courant de voir des scénarios où plusieurs types sont souhaitables. Par exemple, imaginez que vous avez un index de voiture. Il peut vous être utile de le décomposer en plusieurs types:

- BMW
- chevy
- honda
- Mazda
- mercedes
- nissan
- rangerover
- toyota
- ...

De cette façon, vous pouvez rechercher toutes les voitures ou les limiter par fabricant à la demande. La différence entre ces deux recherches est aussi simple que:

```
GET /cars/_search
```

et

```
GET /cars/bmw/_search
```

Ce qui n'est pas évident pour les nouveaux utilisateurs d'Elasticsearch, c'est que la seconde forme est une spécialisation de la première forme. Il est littéralement réécrit pour:

```
GET /cars/_search
{
  "query": {
    "bool": {
      "filter": [
        {
          "term" : {
            "_type": "bmw"
          }
        }
      ]
    }
  }
}
```

Il filtre simplement tout document non indexé avec un champ `_type` dont la valeur était `bmw` . Comme chaque document est indexé avec son type en tant `_type` champ `_type` , cela sert de filtre assez simple. Si une recherche réelle avait été fournie dans l'un ou l'autre exemple, le filtre serait alors ajouté à la recherche complète, le cas échéant.

Ainsi, si les types sont identiques, il est préférable de fournir un seul type (par exemple, le `manufacturer` dans cet exemple) et de l'ignorer efficacement. Ensuite, dans chaque document, fournissez explicitement un champ appelé `make` *ou quel que soit le nom que vous préférez* et filtrez-le manuellement chaque fois que vous souhaitez vous y limiter. Cela réduira la taille de vos mappages à `1/n` où `n` est le nombre de types séparés. Il ajoute un autre champ à chaque document, au profit d'une cartographie autrement simplifiée.

Dans Elasticsearch 1.x et 2.x, un tel champ doit être défini comme

```
PUT /cars
{
  "manufacturer": { <1>
    "properties": {
      "make": { <2>
        "type": "string",
        "index": "not_analyzed"
      }
    }
  }
}
```

1. Le nom est arbitraire.
2. Le nom est arbitraire *et* il pourrait correspondre au nom du type si vous le vouliez également.

Dans Elasticsearch 5.x, ce qui précède fonctionnera toujours (il est obsolète), mais le meilleur moyen est d'utiliser:

```
PUT /cars
{
  "manufacturer": { <1>
    "properties": {
      "make": { <2>
        "type": "keyword"
      }
    }
  }
}
```

1. Le nom est arbitraire.
2. Le nom est arbitraire *et* il pourrait correspondre au nom du type si vous le vouliez également.

Les types doivent être utilisés avec parcimonie dans vos index, car ils gonflent les mappages d'index, généralement sans grand bénéfice. Vous devez en avoir au moins un, mais rien ne dit que vous devez en avoir plus d'un.

Questions courantes

- Que se passe-t-il si j'ai deux types (ou plus) qui sont pour la plupart identiques, mais qui ont quelques champs uniques par type?

Au niveau de l'index, il n'y a pas de différence entre un type utilisé avec quelques champs peu utilisés *et* entre plusieurs types partageant un groupe de champs non fragmentés avec quelques champs non partagés (ce qui signifie que l'autre type n'utilise même pas le champ (s)).

Dit différemment: un champ peu utilisé est rare à travers l'index *indépendamment des types* . La rareté ne profite pas - ou ne fait vraiment pas mal - l'indice simplement parce qu'il est défini dans un type distinct.

Vous devez simplement combiner ces types et ajouter un champ de type distinct.

- Pourquoi des types distincts doivent-ils définir des champs exactement de la même manière?

Parce que chaque champ n'est réellement défini qu'une seule fois au niveau Lucene, quel que soit le nombre de types. Le fait que les types existent du tout est une caractéristique d'Elasticsearch et ne constitue *qu'une* séparation logique.

- Puis-je définir des types séparés avec le même champ défini différemment?

Non. Si vous parvenez à trouver un moyen de le faire dans ES 2.x ou version ultérieure, vous devez alors ouvrir un rapport de bogue . Comme indiqué dans la question précédente, Lucene les considère tous comme un seul champ, il n'y a donc aucun moyen de faire en sorte que cela fonctionne correctement.

ES 1.x en a fait une exigence implicite, ce qui permettait aux utilisateurs de créer des conditions dans lesquelles les correspondances d'une partition dans un index différaient réellement d'une autre partition du même index. C'était effectivement une condition de course et cela *pouvait*

entraîner des problèmes imprévus.

Exceptions à la règle

- Les documents parent / enfant **nécessitent** des types distincts à utiliser dans le même index.
  - Le parent vit dans un type.
  - L'enfant vit dans un type distinct (mais chaque enfant vit dans le même *fragment* que son parent).
- Des cas d'utilisation extrêmement complexes où la création de tonnes d'indices est indésirable et l'impact de champs rares est préférable à l'alternative.
  - Par exemple, le plug-in de surveillance Elasticsearch, Marvel (1.x et 2.x) ou X-Pack Monitoring (5.x +) surveille Elasticsearch lui-même pour détecter les modifications dans le cluster, les nœuds, les index, les index spécifiques (le niveau d'index), et même des éclats. Il pourrait créer plus de 5 index chaque jour pour isoler les documents ayant des correspondances uniques *ou* aller à l'encontre des meilleures pratiques pour réduire la charge du cluster en partageant un index (note: le nombre de mappages définis est le même, mais le nombre d'index créés est réduit de `n` à 1).
  - Ceci est un scénario avancé, mais vous devez tenir compte des définitions de champs partagés entre les types!

Examples

Créer explicitement un index avec un type

L'exemple utilise le protocole HTTP de base, qui se traduit facilement par cURL et d'autres applications HTTP. Ils correspondent également à la syntaxe Sense , qui sera renommée en console dans Kibana 5.0.

Remarque: L'exemple insère `<#>` pour attirer l'attention sur les pièces. Ceux-ci doivent être supprimés si vous le copiez!

```
PUT /my_index <1>
{
  "mappings": {
    "my_type": { <2>
      "properties": {
        "field1": {
          "type": "long"
        },
        "field2": {
          "type": "integer"
        },
        "object1": {
          "type": "object",
          "properties": {
            "field1" : {
              "type": "float"
            }
          }
```

```
        }
      }
    }
  },
  "my_other_type": {
    "properties": {
      "field1": {
        "type": "long" <3>
      },
      "field3": { <4>
        "type": "double"
      }
    }
  }
}
```

1. Cela crée l' `index` utilisant le noeud final create index.
2. Cela crée le `type` .
3. Les champs partagés dans le `type` s dans le même `index` **doivent** partager la même définition! ES 1.x n'a pas strictement appliqué ce comportement, mais c'était une exigence implicite. ES 2.x et ci-dessus appliquent strictement ce comportement.
4. Les champs uniques à travers le `type` s sont corrects.

Les index (ou index) *contiennent des* types. Les types sont un mécanisme pratique pour séparer des documents, mais ils nécessitent que vous définissiez - dynamiquement / automatiquement ou explicitement - un mappage pour chaque type que vous utilisez. Si vous définissez 15 types dans un index, vous avez 15 mappages uniques.

Voir les remarques pour plus de détails sur ce concept et pourquoi vous pouvez ou non vouloir utiliser des types.

Création dynamique d'un index avec un type

L'exemple utilise le protocole HTTP de base, qui se traduit facilement par cURL et d'autres applications HTTP. Ils correspondent également à la syntaxe Sense , qui sera renommée en console dans Kibana 5.0.

Remarque: L'exemple insère `<#>` pour attirer l'attention sur les pièces. Ceux-ci doivent être supprimés si vous le copiez!

```
DELETE /my_index <1>

PUT /my_index/my_type/abc123 <2>
{
  "field1" : 1234, <3>
  "field2" : 456,
  "object1" : {
    "field1" : 7.8 <4>
  }
}
```

1. S'il existe déjà (à cause d'un exemple précédent), supprimez l'index.
2. `my_index` un document dans l'index, `my_index` , avec le type, `my_type` et l'ID `abc123` (peut être

numérique, mais c'est toujours une chaîne).
    - Par défaut, la création d'index dynamique est activée en indexant simplement un document. Ceci est idéal pour les environnements de développement, mais ce n'est pas nécessairement bon pour les environnements de production.
3. Ce champ est un nombre entier, donc la première fois qu'il est vu, il doit être mappé. Elasticsearch prend toujours le type le *plus large* pour tout type entrant, de sorte qu'il serait mappé comme un `long` plutôt qu'un `integer` ou un `short` (les deux pouvant contenir `1234` et `456`).
4. La même chose est vraie pour ce domaine également. Il sera mappé comme un `double` plutôt que comme un `float` comme vous le souhaitez.

Cet index et ce type créés dynamiquement correspondent approximativement au mappage défini dans le premier exemple. Cependant, il est essentiel de comprendre comment `<3>` et `<4>` affectent les mappages définis automatiquement.

Vous pouvez suivre cela en ajoutant un autre type dynamiquement au même index:

```
PUT /my_index/my_other_type/abc123 <1>
{
  "field1": 91, <2>
  "field3": 4.567
}
```

1. Le type est la seule différence avec le document ci-dessus. L'identifiant est le même et ça va! Il n'a pas de relation avec l'autre `abc123` n'est qu'il *se trouve* dans le même index.
2. `field1` existe déjà dans l'index, il *doit donc* être du même type que celui défini dans les autres types. Soumettre une valeur qui était une chaîne ou non un entier échouerait (par exemple, `"field1": "this is some text"` ou `"field1": 123.0`).

Cela créerait dynamiquement les mappages pour `my_other_type` dans le même index, `my_index` .

Remarque: Il est *toujours* plus rapide de définir les mappages au lieu de demander à Elasticsearch de l'exécuter dynamiquement au moment de l'index.

Le résultat final de l'indexation des deux documents serait similaire au premier exemple, mais les types de champs seraient différents et par conséquent légèrement inutiles:

```
GET /my_index/_mappings <1>
{
  "mappings": {
    "my_type": { <2>
      "properties": {
        "field1": {
          "type": "long"
        },
        "field2": {
          "type": "long" <3>
        },
        "object1": {
          "type": "object",
          "properties": {
            "field1" : {
```

```
              "type": "double" <4>
            }
          }
        }
      }
    }
  },
  "my_other_type": { <5>
    "properties": {
      "field1": {
        "type": "long"
      },
      "field3": {
        "type": "double"
      }
    }
  }
}
```

1. Cela utilise le point de terminaison `_mappings` pour obtenir les mappages de l'index que nous avons créé.
2. Nous avons créé dynamiquement `my_type` dans la première étape de cet exemple.
3. `field2` est maintenant un `long` au lieu d'un `integer` parce que nous ne l'avons pas défini au départ. Cela *peut* s'avérer être un gaspillage de stockage sur disque.
4. `object1.field1` est maintenant un `double` pour la même raison que # 3 avec les mêmes ramifications que # 3.
    - Techniquement, un `long` peut être compressé dans beaucoup de cas. Cependant, un `double` ne peut pas être compressé car il s'agit d'un nombre à virgule flottante.
5. Nous avons également créé dynamiquement `my_other_type` dans la deuxième étape de cet exemple. Sa cartographie se révèle être la même parce que nous utilisions déjà le `long` et le `double` .
    - Rappelez-vous que `field1` *doit* correspondre à la définition de `my_type` (et il le fait).
    - `field3` est unique pour ce type, il n'a donc pas cette restriction.

Lire Différence entre les indices et les types en ligne:
https://riptutorial.com/fr/elasticsearch/topic/3412/difference-entre-les-indices-et-les-types

Chapitre 10: Grappe

Remarques

L'intégrité du cluster fournit de nombreuses informations sur le cluster, telles que le nombre de fragments alloués ("actifs") ainsi que le nombre de partitions non attribuées et déplacées. De plus, il fournit le nombre actuel de noeuds et de noeuds de données dans le cluster, ce qui peut vous permettre de rechercher des noeuds manquants (par exemple, si vous prévoyez qu'il y en a `15` , mais qu'il n'en affiche que `14` , il vous manque un noeud) .

Pour ceux qui connaissent Elasticsearch, les fragments "assignés" et "non attribués" peuvent les aider à détecter des problèmes.

Le champ le plus fréquemment vérifié à partir de Cluster Health est le `status` , qui peut être l'un des trois états suivants:

- rouge
- jaune
- vert

Les couleurs signifient chacune une et une seule chose très simple:

1. Le rouge indique qu'il vous manque *au moins* un fragment primaire.
    - Un fragment primaire manquant signifie qu'un index ne peut pas être utilisé pour écrire (indexer) de nouvelles données dans la plupart des cas.
        - Techniquement, vous pouvez toujours indexer tous les fragments primaires disponibles dans cet index, mais en pratique, cela signifie que vous ne pouvez pas le faire car vous ne contrôlez généralement pas quel fragment reçoit un document donné.
        - La recherche est toujours possible sur un cluster rouge, mais cela signifie que vous obtiendrez des résultats partiels si des index sont manquants dans les index.
    - Dans des circonstances normales, cela signifie simplement que le fragment primaire est alloué (`initializing_shards`).
    - Si un nœud vient de quitter le cluster (par exemple, parce que la machine qui l'exécute a perdu de la puissance), il est logique que vous manquiez *temporairement* certains fragments primaires.
        - Tout fragment de réplique pour ce fragment primaire sera promu comme fragment primaire dans ce scénario.
2. Jaune indique que tous les fragments primaires sont actifs, mais *qu'au moins* un fragment de réplique est manquant.
    - Une réplique manquante n'affecte que l'indexation si les paramètres de cohérence nécessitent une incidence sur l'indexation.
        - Par défaut, il n'y a qu'une seule réplique pour un serveur principal et l'indexation peut se produire avec un seul réplica manquant.
    - Dans des circonstances normales, cela signifie simplement que le fragment de

réplique est alloué (`initializing_shards`).
    - Un cluster à un seul nœud avec les répliques activées sera *toujours* jaune *au mieux* . Il peut être rouge si un fragment primaire n'a pas encore été attribué.
        - Si vous ne disposez que d'un seul nœud, désactivez les réplicas car vous n'en attendez aucun. Ensuite, il peut être vert.
3. Le vert indique que tous les fragments sont actifs.
    - La seule activité de fragment autorisée pour un cluster vert est la `relocating_shards` .
    - Les nouveaux index, et par conséquent les nouveaux fragments, feront passer le cluster du rouge au jaune et au vert, au fur et à mesure que chaque partition est allouée (primaire en premier, le rendant jaune, puis répliques si possible, le rendant vert).
        - Dans Elasticsearch 5.x et les versions ultérieures, les nouveaux index **ne** rendront **pas** votre cluster rouge à moins qu'il ne soit trop long à allouer.

Examples

Cluster Health sous forme de tableau, lisible par l'homme, avec en-têtes

L'exemple utilise la syntaxe HTTP de base. Tout `<#>` dans l'exemple doit être supprimé lors de sa copie.

Vous pouvez utiliser les API `_cat` pour obtenir une sortie tabulaire lisible par l'homme pour diverses raisons.

```
GET /_cat/health?v <1>
```

1. Le `?v` est facultatif, mais cela implique que vous voulez une sortie "verbeuse".

`_cat/health` existe depuis Elasticsearch 1.x, mais voici un exemple de sortie d'Elasticsearch 5.x:

Avec sortie verbeuse:

```
epoch      timestamp cluster       status node.total node.data shards pri relo init unassign
pending_tasks max_task_wait_time active_shards_percent
1469302011 15:26:51  elasticsearch yellow          1         1     45  45    0    0       44
0                  -                 50.6%
```

Cluster Health, sous forme de tableau, lisible par l'homme, sans en-têtes

L'exemple utilise la syntaxe HTTP de base. Tout `<#>` dans l'exemple doit être supprimé lors de sa copie.

Vous pouvez utiliser les API `_cat` pour obtenir une sortie tabulaire lisible par l'homme pour diverses raisons.

```
GET /_cat/health <1>
```

`_cat/health`

existe depuis Elasticsearch 1.x, mais voici un exemple de sortie d'Elasticsearch 5.x:

Sans sortie verbeuse:

```
1469302245 15:30:45 elasticsearch yellow 1 1 45 45 0 0 44 0 - 50.6%
```

Cluster Health sous forme de tableau, lisible par l'homme, avec en-têtes sélectionnés

L'exemple utilise la syntaxe HTTP de base. Tout `<#>` dans l'exemple doit être supprimé lors de sa copie.

Comme la plupart des API `_cat` dans Elasticsearch, l'API répond de manière sélective par un ensemble de champs par défaut. Cependant, d'autres champs existent de l'API si vous le souhaitez:

```
GET /_cat/health?help <1>
```

1. `?help` permet à l'API de renvoyer les champs (et noms abrégés) ainsi qu'une brève description.

`_cat/health` existe depuis Elasticsearch 1.x, mais voici un exemple de sortie d'Elasticsearch 5.x:

Champs disponibles à la date de création de cet exemple:

```
epoch                | t,time                                | seconds since 1970-01-01
00:00:00
timestamp            | ts,hms,hhmmss                         | time in HH:MM:SS

cluster              | cl                                    | cluster name

status               | st                                    | health status

node.total           | nt,nodeTotal                          | total number of nodes

node.data            | nd,nodeData                           | number of nodes that can
store data
shards               | t,sh,shards.total,shardsTotal         | total number of shards

pri                  | p,shards.primary,shardsPrimary        | number of primary shards

relo                 | r,shards.relocating,shardsRelocating  | number of relocating nodes

init                 | i,shards.initializing,shardsInitializing | number of initializing
nodes
unassign             | u,shards.unassigned,shardsUnassigned  | number of unassigned shards

pending_tasks        | pt,pendingTasks                       | number of pending tasks

max_task_wait_time   | mtwt,maxTaskWaitTime                  | wait time of longest task
pending
active_shards_percent | asp,activeShardsPercent              | active number of shards in
percent
```

Vous pouvez ensuite l'utiliser pour imprimer uniquement ces champs:

```
GET /_cat/health?h=timestamp,cl,status&v <1>
```

1. `h=...` définit la liste des champs à renvoyer.
2. `v` (verbose) définit que vous souhaitez imprimer les en-têtes.

La sortie d'une instance d'Elasticsearch 5.x:

```
timestamp cl            status
15:38:00  elasticsearch yellow
```

Santé des clusters basée sur JSON

L'exemple utilise la syntaxe HTTP de base. Tout `<#>` dans l'exemple doit être supprimé lors de sa copie.

Les API `_cat` sont souvent pratiques pour les humains pour obtenir des informations détaillées sur le cluster. Cependant, vous souhaitez souvent utiliser des résultats systématiquement analysables avec les logiciels. En général, les API JSON sont conçues à cet effet.

```
GET /_cluster/health
```

`_cluster/health` existe depuis Elasticsearch 1.x, mais voici un exemple de sortie d'Elasticsearch 5.x:

```
{
  "cluster_name": "elasticsearch",
  "status": "yellow",
  "timed_out": false,
  "number_of_nodes": 1,
  "number_of_data_nodes": 1,
  "active_primary_shards": 45,
  "active_shards": 45,
  "relocating_shards": 0,
  "initializing_shards": 0,
  "unassigned_shards": 44,
  "delayed_unassigned_shards": 0,
  "number_of_pending_tasks": 0,
  "number_of_in_flight_fetch": 0,
  "task_max_waiting_in_queue_millis": 0,
  "active_shards_percent_as_number": 50.56179775280899
}
```

Lire Grappe en ligne: https://riptutorial.com/fr/elasticsearch/topic/2069/grappe

Chapitre 11: Interface Python

Paramètres

Paramètre	Détails
les hôtes	Tableau d'hôtes sous la forme d'objet contenant des clés `host` et `port` . L' `host` par défaut est 'localhost' et le `port` est 9200. Un exemple d'entrée ressemble à `[{"host": "ip of es server", "port": 9200}]`
sniff_on_start	Booléen si vous souhaitez que le client sniffe les nœuds au démarrage, renifler signifie obtenir la liste des nœuds dans le cluster elasticsearch
sniff_on_connection_fail	Booléen pour déclencher le reniflage si la connexion échoue lorsque le client est actif
sniffer_timeout	différence de temps en secondes entre chaque reniflement
sniff_timeout	temps pour une seule demande de reniflage en quelques secondes
retry_on_timeout	Booelan pour si le client devrait temporiser le déclencheur en contactant un autre nœud elasticsearch ou simplement pour lancer une erreur
http_auth	Une authentification http de base peut être fournie ici sous la forme d'un `username:password` d' `username:password`

Examples

Indexer un document (c.-à-d. Ajouter un échantillon)

Installez la bibliothèque Python nécessaire via:

```
$ pip install elasticsearch
```

Connectez-vous à Elasticsearch, Créer un document (par exemple, saisie de données) et "Indexer" le document à l'aide d'Elasticsearch.

```
from datetime import datetime
from elasticsearch import Elasticsearch

# Connect to Elasticsearch using default options (localhost:9200)
es = Elasticsearch()
```

```
# Define a simple Dictionary object that we'll index to make a document in ES
doc = {
    'author': 'kimchy',
    'text': 'Elasticsearch: cool. bonsai cool.',
    'timestamp': datetime.now(),
}

# Write a document
res = es.index(index="test-index", doc_type='tweet', id=1, body=doc)
print(res['created'])

# Fetch the document
res = es.get(index="test-index", doc_type='tweet', id=1)
print(res['_source'])

# Refresh the specified index (or indices) to guarantee that the document
#  is searchable (avoid race conditions with near realtime search)
es.indices.refresh(index="test-index")

# Search for the document
res = es.search(index="test-index", body={"query": {"match_all": {}}})
print("Got %d Hits:" % res['hits']['total'])

# Show each "hit" or search response (max of 10 by default)
for hit in res['hits']['hits']:
    print("%(timestamp)s %(author)s: %(text)s" % hit["_source"])
```

Connexion à un cluster

```
es = Elasticsearch(hosts=hosts, sniff_on_start=True, sniff_on_connection_fail=True,
sniffer_timeout=60, sniff_timeout=10, retry_on_timeout=True)
```

Créer un index vide et définir le mappage

Dans cet exemple, nous créons un index vide (nous n'y indexons aucun document) en définissant son mappage.

Tout d'abord, nous créons une instance `ElasticSearch` et nous définissons ensuite le mappage de notre choix. Ensuite, nous vérifions si l'index existe et si ce n'est pas le cas, nous le créons en spécifiant les paramètres d' `index` et de `body` qui contiennent respectivement le nom d'index et le corps du mappage.

```
from elasticsearch import Elasticsearch

# create an ElasticSearch instance
es = Elasticsearch()
# name the index
index_name = "my_index"
# define the mapping
mapping = {
    "mappings": {
        "my_type": {
                "properties": {
                    "foo": {'type': 'text'},
```

```
                    "bar": {'type': 'keyword'}
                }
            }
        }
    }

# create an empty index with the defined mapping - no documents added
if not es.indices.exists(index_name):
    res = es.indices.create(
        index=index_name,
        body=mapping
    )
    # check the response of the request
    print(res)
    # check the result of the mapping on the index
    print(es.indices.get_mapping(index_name))
```

Mise à jour partielle et mise à jour par requête

Mise à jour partielle: Utilisée lorsqu'une mise à jour partielle du document est nécessaire, c'est-à-dire que dans l'exemple suivant, le `name` du champ avec l'ID `doc_id` va être mis à jour en "John". Notez que si le champ est manquant, il sera simplement ajouté au document.

```
doc = {
    "doc": {
        "name": "John"
    }
}
es.update(index='index_name',
          doc_type='doc_name',
          id='doc_id',
          body=doc)
```

Mise à jour par requête: utilisée lorsque nécessaire pour mettre à jour des documents satisfaisant à une condition, c'est-à-dire que dans l'exemple suivant, nous mettons à jour l'âge des documents dont le champ de `name` correspond à «John».

```
q = {
  "script": {
    "inline": "ctx._source.age=23",
    "lang": "painless"
  },
  "query": {
    "match": {
        "name": "John"
    }
  }
}

es.update_by_query(body=q,
                   doc_type='doc_name',
                   index='index_name')
```

Lire Interface Python en ligne: https://riptutorial.com/fr/elasticsearch/topic/2068/interface-python

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec Elasticsearch	Ahsanul Haque, Berto, Community, DJanssens, Dulguun, igo, KartikKannapur, manishrw, mightyteja, noscreenname, Onur, rafa.ferreira, RustyBuckets, sarvajeetsuman, SeinopSys, Shivkumar Mallesappa, Stephan-v, Suhas K, Sumit Kumar, Trilarion
2	Agrégations	Sid1199
3	Analyseurs	Bhushan Gadekar, Sid1199, Thomas
4	API de recherche	aerokite, Sid1199
5	Apprendre Elasticsearch avec kibana	sarvajeetsuman
6	Commandes Curl	Fawix, Mrunal Pagnis, Mrunal Pagnis
7	Configuration Elasticsearch	pickypg
8	Différence entre les bases de données relationnelles et Elasticsearch	Richa
9	Différence entre les indices et les types	pickypg
10	Grappe	Gerardo Rochín, pickypg
11	Interface Python	aidan.plenert.macdonald, christinabo, KartikKannapur, pickypg, Sumit Kumar