On va découvrir la base de données distribuée Elasticsearch. Son interface Kibana n'est malheureusement pas utilisable à cause de la nécessité d'une licence payante (greffon *xpack.security*). Cela enlève une part de confort mais ne change rien au fond du TP.

1. Présentation du travail avec Elasticsearch

Téléchargez le fichier `TP6.zip` et expansez-le dans votre compte. Il crée un dossier `TP6` pour faire ce TP : fichiers de données et scripts pour Elasticsearch. Ouvrez un terminal dans ce dossier.

NB: Il n'est pas nécessaire d'être connecté sur le serveur hadoop, car tout passe par http sur le port 9200. Il faut juste avoir le nom du serveur dans les scripts, variable `HOST`.

Ouvrez le script `test.sh` dans l'éditeur. Ce script lance des requêtes en *bash*. Ces requêtes sont groupées par fonction. Ces fonctions sont appelées l'une après l'autre à la fin du script. Dans chaque fonction, vous remarquerez comment les titres en gras sont affichés, et comment des contenus JSON sont redirigés vers *curl* (redirection `<<MOT...MOT`) : le mot clé `JSON` final **doit** être en début de ligne.

Lancez le script `test.sh`. Il va créer un index dans Elasticsearch, appelé `LLLL-tests` avec votre nom de compte à la place de `LLLL`. Commentez les appels à `schema` et `donnees`, décommentez la première requête, `get1`. Relancez le script ; comparez le résultat avec la requête. Puis commentez cette requête et décommentez la seconde, `recherche_url` ; comparez avec la requête. Idem pour `recherche_dsl`.

Avant de supprimer les données, ouvrez les URL suivants en mettant votre nom de compte : 

```
http://hadoop:9200/LLLL-tests/_search?q=valeur:7
http://hadoop:9200/LLLL-tests/_search?q=valeur:\-7
http://hadoop:9200/LLLL-tests/_search?q=valeur:"-7"
http://hadoop:9200/LLLL-tests/_search?q=valeur:>0
http://hadoop:9200/LLLL-tests/_search?q=nom:("a", "c")
http://hadoop:9200/LLLL-tests/_search?q=nom:(-"a")
http://hadoop:9200/LLLL-tests/_search?q=-nom:"a" AND -valeur:7
http://hadoop:9200/LLLL-tests/_search?q=nom:["a" TO "b"]
```

Ce sont des recherches par URL, voir le cours. Notez la présence du `\` devant le `-`, c'est parce que le signe `-` est aussi un opérateur pour exclure des termes de la recherche. Il faut mettre un `\\` dans le script, ou alors encadrer -7 par des `"-7"`, mais c'est moins commode en bash. Voyez comment agit le signe `-` pour bloquer des réponses, et comment employer les parenthèses pour grouper des conditions.

Pour finir, re-commentez les requêtes et décommentez `deleteall` et relancez le script pour supprimer les données. C'est ainsi que vous allez travailler.

2. Requêtes sur le fichier des arbres

On va travailler sur une variante du fichiers des arbres. Descendez dans le sous-dossier `arbres` du TP6 et ouvrez le script `arbres.sh` dans un éditeur.

2.1. Définition du schéma

La première opération consiste à définir le schéma. Il faudra terminer la fonction `schema()` du script `arbres.sh` avec la définition de tous les champs du fichier `arbres_tp6.csv`. Examinez-le afin de déterminer les noms et types des colonnes manquantes.

Vous aurez le choix entre `text` et `keyword` pour certains champs. Consulter le cours et faites comme vous pensez, ou mieux, lisez les exercices qui suivent avant de vous lancer.

IMPORTANT avant de passer à la suite, assurez-vous que le schéma est correctement créé. Il suffit d'ouvrir l'URL `http://hadoop:9200/LLLL-arbres`, il est affiché en gras à la fin de la fonction `schema`. Vous devez voir la totalité des champs avec les bons types.

Les erreurs les plus fréquentes sont un `:` oublié avant `{` ou une virgule oubliée à la fin d'un champ.

2.2. Injection des données

Maintenant que le schéma est prêt, il faut envoyer les données dans Elasticsearch. Commentez l'appel à `schema` et décommentez l'appel à `donnees`. La fonction `donnees` est déjà en place. Elle crée un fichier NDJSON à partir du CSV et l'envoie à Elasticsearch.

Une fois le schéma défini et les données injectées sans erreurs, vous pouvez commenter les appels à ces fonctions à la fin du script.

La première recherche, `recherche_url` est exprimée avec le langage « ligne » esquissé en cours. Relire les transparents pour avoir des éléments de syntaxe.

La deuxième requête `recherche_dsl` est écrite en DSL. Consultez le cours pour la syntaxe et les concepts.

IMPORTANT avant de passer à la suite, assurez-vous qu'il y a des données dans votre index. Il suffit d'ouvrir l'URL `http://hadoop:9200/LLLL-arbres/_search?pretty` qui est affiché en gras à la fin.

Voici maintenant quelques exercices.

3. Requêtes dans l'URL

Pour commencer, il faut écrire des requêtes avec le langage « ligne », comme `recherche_url`. Toute la requête est écrite dans l'URL.

3.1. Arbres du 5e arrondissement

Travaillez sur la fonction `exo_3_1`.

Pour désigner l'arrondissement qui est dans le champ adresse, on emploie la syntaxe `adresse.arrondissement`.

3.2. Nom commun contient Erable

Travaillez sur la fonction `exo_3_2`.

Il s'agit de trouver des arbres dont le nom commun contient « Erable » (sans accent).

Attention, il faut mettre un \ devant chaque espace : `Q='nom\ commun:Erable\ de\ Cappadoce'`

Dilemme:

- Si vous avez défini le champ `nom commun` en tant que `text`, ça va marcher. Elasticsearch peut chercher des fragments dans un texte.
- S'il est défini en `keyword`, il faut fournir le nom commun exact pour qu'il soit trouvé, et donc vous n'aurez aucun résultat.

Voyez que c'est contradictoire avec l'exercice suivant.

3.3. Nom commun est Erable de Montpellier

Travaillez sur la fonction `exo_3_3`.

Il s'agit de trouver des arbres dont le nom commun est « Erable de Montpellier » (sans accent).

- Si vous avez défini le champ `nom commun` en tant que `keyword`, ça va marcher. Elasticsearch peut chercher des documents identifiés par une telle chaîne.
- S'il est défini en `text`, vous allez recevoir un tas de résultats, parce que votre recherche contient plusieurs mots qui sont comparés aux données. Pour vous, les deux mots « Erable » et « Montpellier » sont les plus importants, mais pour ES, « de » compte autant. Du coup, il y a de nombreux résultats, avec des scores variés. Il est possible de ne garder que ceux ayant un score dépassant un seuil, mais ce n'est pas au programme.

Il est ennuyeux de voir qu'aucun choix de typage n'est satisfaisant. Il faudrait un type `keyword` dont les mots puissent être indexés, ou inversement un type `text` pouvant être interrogé de manière exacte.

Pour résoudre partiellement le problème, utilisez la technique des types multiples, présentée en cours. C'est à dire que le type principal du nom commun est `text` et il y a un sous-type nommé `"keyword"` qui est du type `keyword`. Ainsi, quand on a besoin du nom commun exact pour une agrégation ou un classement, on emploie `nom commun.keyword` et quand on cherche des mots dedans, on emploie `nom commun`.

```
{...
    "nom commun": {
        "type": "text",
        "fields": {
            "keyword": { "type": "keyword" }
        }
    },
```

Malheureusement, ça ne change rien au nombre de résultats, mais il devient possible de faire des tris et des agrégations alors que c'est du texte.

3.4. Nom commun contient Erable et Montpellier

Travaillez sur la fonction `exo_3_4`.

Il s'agit de trouver des arbres dont le nom commun contient les mots « Erable » et « Montpellier ». Vous verrez qu'il produit aussi un autre arbre, mais avec un score deux fois plus faible.

3.5. Hauteur supérieure à 40 mètres

Travaillez sur la fonction `exo_3_5`.

Il s'agit de trouver des arbres dont la hauteur atteint ou dépasse 40 mètres. Comme la hauteur est un type numérique, il ne trouve que les arbres voulus, pas d'autres qui feraient 39 mètres.

3.6. Arbres du Bois de Vincennes plantés avant 1850

Travaillez sur la fonction `exo_3_6`.

Il s'agit de trouver des arbres dont le lieu (dans l'adresse) est « Bois de Vincennes » et l'année de plantation antérieure à 1850.

3.7. Arbres du 18e siècle

Travaillez sur la fonction `exo_3_7`.

Il s'agit de trouver des arbres dont l'année de plantation est comprise entre 1701 et 1800 inclus. Utilisez soit les intervalles `[v1 TO v2]`, soit deux filtres avec comparaisons.

4. Requêtes avec DSL

On passe maintenant à DSL qui permet beaucoup plus de choses, mais pas très facilement, puisqu'il faut construire une requête en JSON sans avoir d'aide. Kibana proposerait, si on pouvait l'utiliser, quelques suggestions de saisie, mais pas suffisamment nombreuses.

Dans un premier temps, on va seulement retranscrire en DSL des requêtes précédentes, puis on verra les spécificités de DSL. Regarder la fonction `recherche_dsl` et comparez-la à `recherche_url`.

Avec ces requêtes, il n'est plus possible d'ouvrir un URL sur le navigateur, car il faut fournir une charge utile.

4.1. Hauteur supérieure à 40 mètres

Travaillez sur la fonction `exo_4_1`.

Il s'agit de trouver des arbres dont la hauteur atteint ou dépasse 40 mètres. Il faut utiliser un opérateur `"range"` et une borne `"gt"`.

4.2. Nom commun contient Erable

Travaillez sur la fonction `exo_4_2`.

Il s'agit de trouver des arbres dont le nom commun contient « Erable » (sans accent).

4.3. Nom commun contient Erable et Montpellier

Travaillez sur la fonction `exo_4_3`.

Il s'agit de trouver des arbres dont le nom commun contient les mots « Erable » et « Montpellier ». Il faut coder un opérateur booléen entre deux conditions `"must"`. Il n'est pas demandé d'éliminer les autres arbres.

Vous allez avoir du mal à coder la conjonction. En effet, la clause `"must"` doit avoir une liste en tant que valeur :

```
{
    "query": {
        "bool": {
            "must": [ LISTE ]
        }
    }
}
```

Cette liste est une collection d'objets, chacun entouré par des `{}`, comme `{ "match": { "genre": "Acer"}}`.

4.4. Platane mais pas commun, dans le 12e arrondissement

Travaillez sur la fonction `exo_4_4`.

Il s'agit de trouver des arbres dont le nom commun contient le mot « Platane » mais pas le mot « commun » et qui sont dans le 12e arrondissement.

4.5. Nom commun contient strictement Erable et Montpellier

Travaillez sur la fonction `exo_4_5`.

Il s'agit de trouver des arbres dont le nom commun contient les mots « Erable » et « Montpellier », mais pas un autre mot comme *Cappadoce*. Il faut alors utiliser la clause `"filter"` pour éliminer les réponses qui ne satisfont pas la totalité des critères. Il suffit d'emballer la question d'avant dans une deuxième requête booléenne avec un filtre.

4.6. Séquoia géants du Bois de Boulogne et du Parc Montsouris

Travaillez sur la fonction `exo_4_6`.

Il faut afficher les arbres dont le nom commun est `"Séquoia géant"` et qui sont situés (`adresse.lieu`) soit dans le Bois de Boulogne, soit dans le Parc Montsouris. Vous obtiendrez de nombreux résultats, mais avec un score rapidement décroissant.

5. Agrégations

Il s'agit de regrouper certaines informations de plusieurs documents, par exemple compter, additionner, moyenner, etc. La structure de la requête est donnée dans le cours. C'est un opérateur `"aggs"` qui permet de le faire.

Étudier la requête `agregation_dsl` et relire le cours.

5.1. Année de plantation du plus vieil arbre

Travaillez sur la fonction `exo_5_1`.

Afficher l'année de plantation du plus vieil arbre

Penser à changer le nom de la variable agrégée.

5.2. Lieux possédant des arbres

Travaillez sur la fonction `exo_5_2`.

Faire afficher la liste des lieux, en un seul exemplaire chacun, ayant des arbres (remarquables) à Paris. Ce sont les valeurs distinctes du champ `"adresse.lieu"`. Consulter le cours.

Vous verrez qu'en plus, ES vous affiche le nombre de documents dans chaque lieu.

6. Requêtes sur l'index des livres

Passez maintenant dans le dossier `livres` du TP6 pour des exercices beaucoup plus libres. Vous partez d'un fichier de données NDJSON prêt à être injecté dans Elasticsearch.

Commencez par définir le schéma des données. Vous n'avez pas le temps ? Ok, alors Elasticsearch est capable de déterminer les mappings de vos données, mais par contre, il ne se posera pas la question `text` ou `keyword` bien longtemps... Sautez directement à l'injection des données par la fonction `donnees`.

Si vous avez le temps de fignoler alors adoptez l'analyseur `french` pour les textes. Voir le transparent du cours intitulé « Analyse des textes ».

Voici maintenant quelques exercices. Vous avez entièrement le choix des techniques (URL ou DSL).

6.1. Livres parus entre 1850 et 1900

Afficher les six livres parus entre ces deux années incluses.

6.2. Livres incroyables écrits par Jules Verne

Afficher le ou les livres de Jules Verne dans lequel il est question dans le texte de quelque chose d'« incroyable ».

6.3. Livres d'humour comiques

Afficher les livres dont le texte contient les mots « humour » ou « comique ». La recherche sera plus fructueuse si vous avez spécifié l'analyseur `french` pour le texte. Sinon, vous n'aurez que les textes contenant exactement ces mots, et pas leurs pluriels.

7. Travail à rendre

Pensez à supprimer tous vos index à la fin.

Remontez au dessus du dossier `TP6` et compressez-le en `.tar.gz`. Ne le confondez pas avec le `TP6.zip` qui vous a été fourni. Votre archive doit contenir tous les scripts bash mis à jour que vous avez faits et modifiés. Déposez-la sur Moodle dans la zone de dépôt prévue.