DÉCOUVERTE ET UTILISATION D'ELASTICSEARCH

Introduction

Elasticsearch (parfois surnommé "ES") est le cœur central de la plate-forme d'analyse de logs la plus populaire à l'heure actuelle à savoir la pile ELK (Elasticsearch, Logstash et Kibana). Dans ce chapitre dédié à Elasticsearch, je vais vous fournir à vous nouveaux utilisateurs **les connaissances et les outils nécessaires pour commencer à utiliser Elasticsearch**.

Qu'est-ce qu'Elasticsearch ?

Mais en fait, **c'est quoi exactement Elasticsearch ?** Un indexeur ? Un moteur de recherche ? Une base de données ? Une solution de Big Data ? La vérité c'est que toutes ces réponses sont correctes et c'est ce qui fait l'intérêt d'Elasticsearch.

En effet, au fil des ans, Elasticsearch a été utilisé pour un nombre croissant de cas d'utilisation, allant de la simple recherche sur un site Web jusqu'à la collecte et l'analyse de données, d'analyses décisionnelles etc ...

Il a été initialement publié en 2010, c'est un outil d'analyse de données moderne basé sur le projet Apache Lucene. Entièrement open source et construit avec Java, il fait partie de la famille des **bases de données NoSQL**. Cela signifie qu'il stocke les données de manière non structurées et que vous ne pouvez pas utiliser des requêtes purement SQL pour les interroger.

L'intérêt de ce type de base de données et notamment celui d'Elasticsearch et qu'il vous permet de stocker, rechercher et analyser d'énormes volumes de données

rapidement et en temps quasi réel et de vous retourner des réponses en quelques millisecondes.

Ainsi, Elasticsearch est capable d'obtenir des réponses de recherche rapides car au lieu de rechercher le texte directement, il recherche un index. Il utilise une structure basée sur des Documents (nous verrons un plus loin ci-dessous la définition d'un Document) au lieu de tables et de schémas et est livré avec des API REST étendues pour stocker et rechercher les données. À la base, vous pouvez considérer Elasticsearch comme un serveur capable de traiter des requêtes JSON et de vous restituer les données sous le format JSON.

Dans le cadre de l'analyse des données, Elasticsearch est utilisé avec les autres composants de la pile ELK (Logstash et Kibana) et joue le rôle d'indexation et de stockage des données.

> **Information**
>
> Ce chapitre sur Elasticsearch pourrait également être considéré comme un **tutoriel NoSQL**. Cependant, contrairement à la plupart des bases de données NoSQL, Elasticsearch met fortement l'accent sur les capacités et les fonctionnalités de recherche, à tel point que le moyen le plus simple d'obtenir des données d'ES est de les rechercher à l'aide de l'API étendue d'Elasticsearch que nous verrons plus loin dans ce chapitre.

Les concepts et terminologies d'Elasticsearch

Pour mieux comprendre le fonctionnement d'Elasticsearch, abordons au préalable quelques **concepts et lexiques de base d'Elasticsearch** et sur la façon dont il organise

les données et ses composants backends.

Les Clusters

Nous retrouverons tout d'abord le cluster, qui est un ensemble d'un ou plusieurs nœuds Elasticsearch (serveurs). Il peut comprendre autant de nœuds que vous le souhaitez, ce qui le rend extrêmement évolutif.

La collection de nœuds contient toutes les données du cluster, et le cluster fournit une capacité d'indexation et de recherche sur tous les nœuds. En pratique, cela signifie que lorsque vous effectuez des recherches, vous n'avez pas à vous soucier d'un nœud en particulier sur lequel la donnée sera stockée.

Les Nodes (nœuds)

Un node représente un serveur unique qui stocke des données consultables et fait partie d'un cluster. Les nœuds participent aux capacités d'indexation et de recherche d'un cluster, ce qui signifie que lorsque des opérations sont effectuées sur le cluster, les nœuds collaborent pour répondre aux demandes.

Dans un cluster, différentes responsabilités sont attribuées aux différents types de nœuds:

- `Master nodes` : en charge de la gestion à l'échelle du cluster et des actions de configuration telle que l'ajout et la suppression de nœuds.
- `Data nodes` : stocke les données et exécute les opérations liées aux données telles que la recherche et l'agrégation.

- `Client nodes` : transmet les demandes de cluster au nœud maître et les demandes liées aux données aux Data nodes.
- `Tribe nodes` : agissent comme un Client node, effectuant des opérations de lecture et d'écriture sur tous les nœuds du cluster.
- `Ingestion nodes` : utilisés pour le prétraitement des Documents avant l'indexation (nouveau dans Elasticsearch 5.0).
- `Machine Learning Nodes` : Ce sont des nœuds disponibles sous la licence de base d'Elastic qui permettent des tâches d'apprentissage automatique.

Tous les nœuds sont capables par défaut d'être à la fois des Master nodes, Data nodes, Client nodes, Tribe nodes, Ingestion nodes ou Machine Learning Nodes (c'est le cas pour un cluster à un seul nœud). Cependant Il est recommandé de distinguer chaque nœud par un seul type, d'autant plus que les clusters grossissent.

Dans un environnement de développement ou de test, vous pouvez configurer plusieurs nœuds sur un seul serveur. En production, cependant en raison du nombre de ressources qu'un nœud Elasticsearch consomme, il est recommandé d'exécuter chaque node Elasticsearch sur un serveur distinct.

Les Fields (champs)

Les fields sont la plus petite unité de données individuelle dans Elasticsearch. Chaque field a un type de données défini et contient une seule donnée. Ces types de données incluent les types de données les plus communs (strings, numbers, dates, booleans), des types de données plus complexes (object and nested), des types de données géographiques (get_point et geo_shape) et les types de données spécialisés (token_count, join, rank_features, dense_vector, etc ..).

Il existe vraiment différentes variétés types de fields et de façons de les gérer, que vous pouvez retrouver sur cette page.

Les Documents

Les Documents sont des objets JSON qui sont stockés dans un index Elasticsearch et sont considérés comme l'unité de base de stockage. Dans le monde des bases de données relationnelles, les Documents peuvent être comparés à une ligne dans une table.

Les données dans les Documents sont définies avec des fields composés de clés et de valeurs. Une clé est le nom du field et une valeur peut être un élément de plusieurs types différents, comme une chaîne de caractères, un nombre, un booléen, un autre objet ou un tableau de valeurs.

Les Documents contiennent également des champs réservés qui constituent les métadonnées du Document tels que :

- `_index` : l'index où réside le Document.
- `_type` : le type que le Document représente.
- `_id` : l'identifiant unique du Document.

Les Documents sont donc l'unité d'information de base qui peut être indexée dans Elasticsearch et ils sont exprimés sous le format JSON. Chaque Document possède donc un identifiant unique et des données composées de fields, qui décrit le type d'entité dont il s'agit. Par exemple, un Document peut représenter un article d'encyclopédie, exemple ci-dessous :

```
{
  "_index" : "votre type d'index",
```

```
    "_type" : "votre type d'index",
    "_id" : "1",
    "_source" : {
      "created_at" : "2021-06-07T16:24:32.000Z",
      "title" : "mon article",
      "content" : "ceci est le contenu mon article",
      "author" : "Hatim"
    }
  }
```

Les Index

Les index, sont la plus grande unité de données dans Elasticsearch qui contiennent un ou plusieurs Documents et peuvent être comparés à une base de données dans le monde des bases de données relationnelles.

Poursuivant notre exemple sur les articles d'encyclopédie, vous pourriez avoir un index nommé "articles" contenant toutes les données liées à vos articles représentés sous forme de Documents qui contiennent des fields de type string sur le titre, le contenu et l'auteur, et un autre index nommé "utilisateurs" avec toutes les données liées à vos utilisateurs avec des Documents qui contiennent des fields de type string sur leur email, nom d'utilisateur et mot de passe.

Vous pouvez avoir autant d'index définis dans Elasticsearch que vous le souhaitez. Ceux-ci contiendront à leur tour des Documents propres à chaque index.

Les types (dépréciés)

Dans un index se trouvent des types de Documents. Un type représente une catégorie de Documents similaires. Un type se compose d'un nom et d'un Mapping, et où le Mapping n'a pas besoin d'être explicitement défini. Vous pouvez penser à un type équivalant à une table dans une base de données relationnelle telle que MySQL. Un index peut avoir un ou plusieurs types, et chacun peut avoir son propre

Mapping.

> **Attention**
>
> Différents types de documents étaient autorisés dans un seul index dans la version 6 d'Elasticsearch, mais cette fonctionnalité a été supprimée car cela a conduit à des problèmes divers.
>
> Un seul type de Document est autorisé par index dans la version 7. **La fonctionnalité devrait être complètement supprimée dans la version 8 d'Elasticsearch**. Voici l'explication des développeurs dans leur blog.

Les Mappings

Un type de Document a un Mapping similaire au schéma d'une table dans une base de données relationnelle. Il décrit les champs qu'un Document d'un type donné peut avoir avec leurs types de données, tels qu'une chaîne de caractères, entier, date, etc... Ils ont également des informations sur la façon dont les fields doivent être indexés et comment ils doivent être stockés par Lucene.

Grâce au mappage dynamique, il est facultatif de définir un mappage avant d'ajouter des Documents à un index. Si aucun mappage n'est défini, il sera déduit automatiquement lors de l'ajout d'un Document, en fonction de ses données.

Les Shards (fragments)

Nous allons maintenant discuter d'un nouveau terme appelé shards , c'est également un terme qui existe dans les bases de données relationnelles. Vous avez peut-être entendu parler du concept de "partitionnement" d'une base de données ?

Ne vous inquiétez pas si ce n'est pas le cas.

En effet, la taille de l'index est une cause fréquente de plantage d'Elasticsearch. Comme il n'y a pas de limite au nombre de Documents que vous pouvez stocker sur chaque index, un index peut occuper alors une quantité d'espace disque qui dépasse les limites de stockage d'un nœud. Dès qu'un index approche de cette limite, l'indexation commencera à échouer. Une façon de contrer ce problème consiste à diviser les index horizontalement en morceaux appelés shards (fragments) .

Ceci est utile si un index contient plus de données que le matériel d'un nœud ne peut en stocker. Un exemple pourrait être qu'un index contienne 1 téraoctet de données, mais que le nœud ait un disque dur de seulement 500 gigaoctets. Une shard peut ensuite être créée et stockée sur un autre nœud disposant de suffisamment d'espace pour cela.

Un shard est un index entièrement fonctionnel et indépendant et peut être stocké sur n'importe quel nœud au sein d'un cluster. Le nombre de shards peut être spécifié lors de la création d'un index, mais il est par défaut de 5.

Les shards permettent aussi une mise à l'échelle horizontale par volume de contenu, c'est-à-dire par espace d'index. De plus, le sharding permet de distribuer et de paralléliser les opérations entre les shards, ce qui augmente les performances d'un cluster Elasticsearch.

Les Replicas (répliques)

Alors que les shards améliorent l'évolutivité du volume de contenu pour les index, les Replicas garantissent une haute disponibilité. Une réplique est une copie d'un shard, qui peut prendre le relais en cas de défaillance d'un shard ou d'un nœud.

Un Replica ne réside jamais sur le même nœud que la shard d'origine, ce qui signifie que si le nœud donné échoue, le Replica sera disponible sur un autre nœud. Les répliques permettent également de mettre à l'échelle le volume de recherche, car les répliques peuvent gérer les requêtes de recherche.

Par défaut, Elasticsearch ajoute cinq shards principaux et une réplique pour chaque index, ce qui signifie que, sauf configuration contraire, qu'il y aura une réplique pour chaque shard principale, soit cinq au total.

D'autres concepts

Ce sont les principaux concepts que vous devez comprendre lorsque vous commencez à utiliser Elasticsearch, mais il existe également d'autres composants et termes sur Elasticsearch. Je ne peux pas tous les couvrir, je vous recommande donc de vous référer à la page des terminologies Elasticsearch pour plus d'informations.

Utilisation d'Elasticsearch

Installation et configuration

Vous trouverez mon tutoriel sur l'installation d'Elasticsearch dans mon article.

Les configurations Elasticsearch se font à l'aide d'un fichier de configuration dont l'emplacement dépend de votre système d'exploitation. Dans ce fichier, vous pouvez configurer les paramètres généraux (par exemple le nom du nœud), les paramètres réseau (par exemple l'hôte et le port), l'emplacement des données à stocker, la mémoire, les fichiers de logs, etc.

À des fins de développement et de test, les paramètres par défaut suffiront, mais il est recommandé de faire des recherches sur les paramètres que vous devez définir manuellement avant de passer en production. Par défaut le fichier de configuration d'Elasticsearch est le suivant: **/etc/elasticsearch/elasticsearch.yml**.

Autre chose, Elasticsearch ne s'exécutera pas automatiquement après l'installation et vous devrez le démarrer manuellement. La façon dont vous exécutez Elasticsearch dépend de votre système spécifique. Sur la plupart des systèmes Linux et Unix, vous avez juste à démarrer le service Elasticsearch (l'initialisation peut prendre un peu de temps) :

```
sudo systemctl start elasticsearch
```

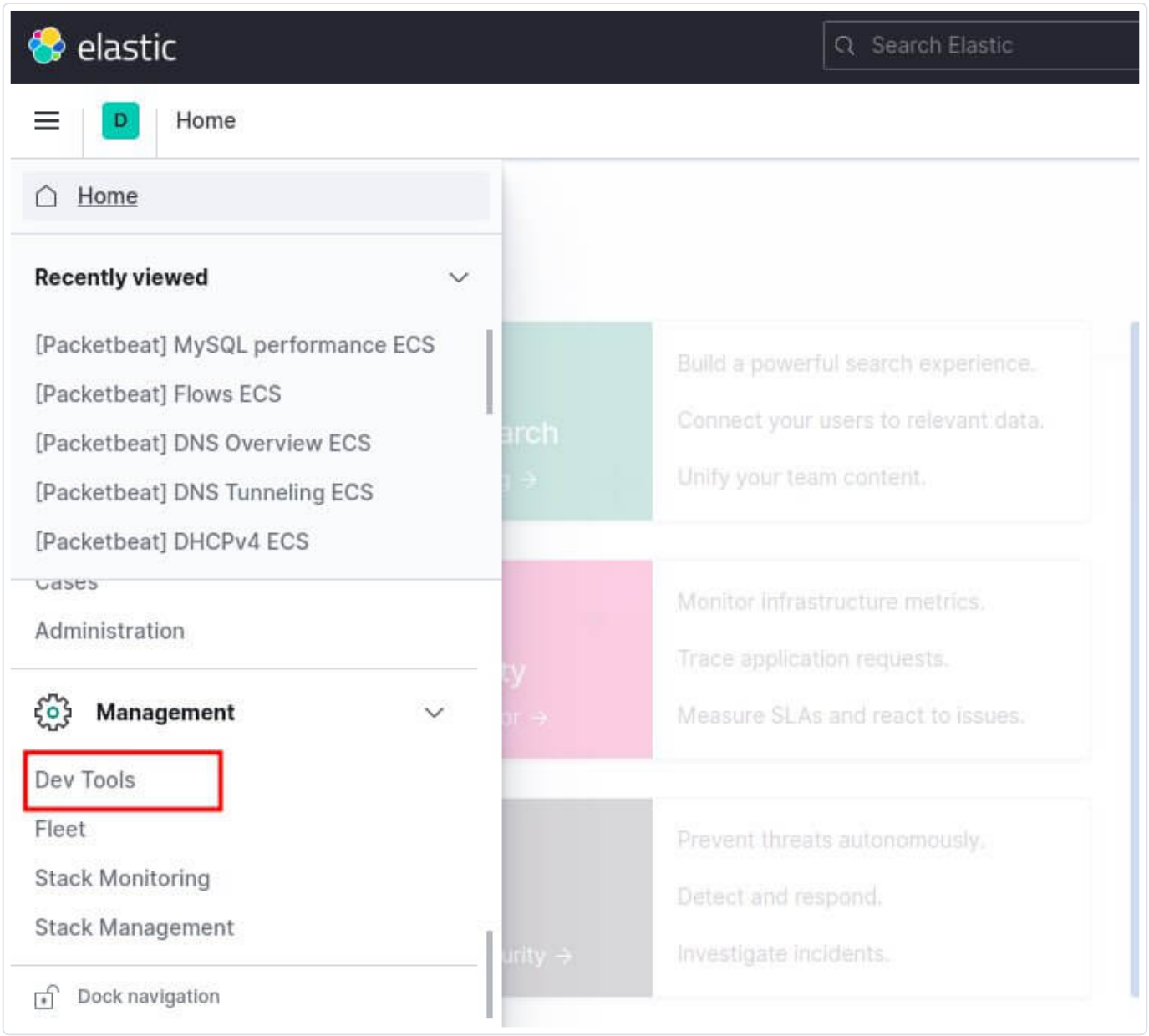
Pour confirmer que tout fonctionne correctement, pointez simplement avec la commande `curl` ou votre navigateur sur l'url et le port suivant http://localhost:9200, et vous devriez voir quelque chose comme la sortie suivante :

```
curl http://localhost:9200
```

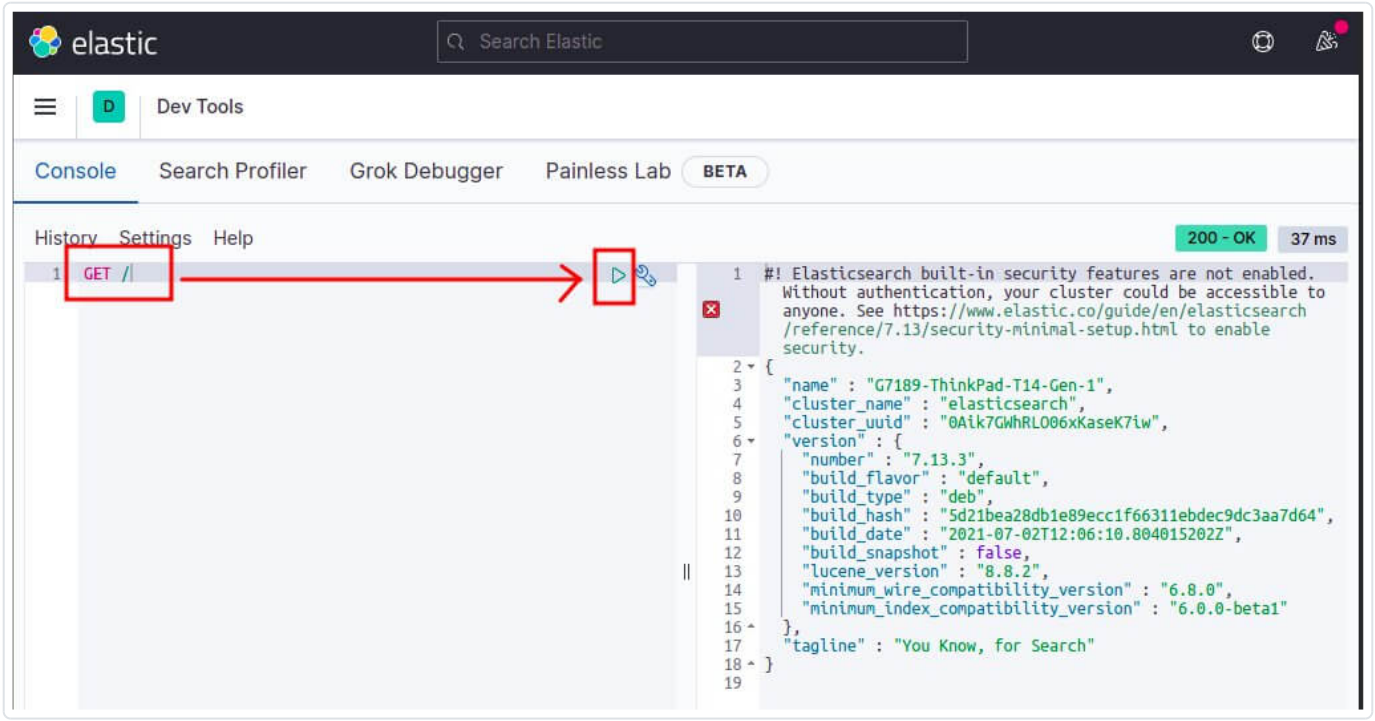
Résultat :

```
{
  "name" : "G7189-ThinkPad-T14-Gen-1",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "0Aik7GWhRLO06xKaseK7iw",
  "version" : {
    "number" : "7.13.3",
    "build_flavor" : "default",
    "build_type" : "deb",
    "build_hash" : "5d21bea28db1e89ecc1f66311ebdec9dc3aa7d64",
    "build_date" : "2021-07-02T12:06:10.804015202Z",
    "build_snapshot" : false,
    "lucene_version" : "8.8.2",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Vous pouvez également **communiquer avec Elasticsearch depuis la console Kibana**. Pour cela accéder à la page Kibana et depuis le menu gauche cliquez sur "Dev Tools":



Et exécutez une requête de type GET sur l'API Elasticsearch afin de récupérer des informations sur le node :



Si jamais vous rencontrez des problèmes d'initialisation, vérifier vos logs pour **déboguer le processus d'exécution d'Elasticsearch** :

```
sudo journalctl -f -u elasticsearch
```

Pour initialiser le service Elasticsearch à chaque démarrage de la machine, lancez la commande suivante :

```
sudo systemctl enable elasticsearch
```

Création de données dans Elasticsearch

L'indexation est le processus d'ajout de données dans Elasticsearch. En effet, lorsque vous fournissez des données à Elasticsearch, les données sont placées dans les index Apache Lucene . Cela est logique, car Elasticsearch utilise les index Lucene pour stocker et récupérer ses données. Bien que vous n'ayez pas besoin d'en savoir beaucoup sur Lucene, il est utile de savoir comment cela fonctionne lorsque vous commencez à devenir plus expert avec Elasticsearch.

Elasticsearch se comporte comme une API REST, vous pouvez donc utiliser méthode de type POST ou PUT pour y ajouter des données. Vous utiliserez la méthode PUT, lorsque vous connaissez déjà l'id de l'élément de données que vous souhaitez spécifier, ou la méthode POST si vous souhaitez qu'Elasticsearch génère pour vous un id automatiquement pour l'élément de données.

Dans notre exemple, nous allons établir un système d'articles d'encyclopédie. Nous allons donc créer un index qu'on nommera "articles" qui contiendra un Document avec des fields de type string sur le nom de l'auteur, le titre, la catégorie et le contenu de l'article ainsi que la date de publication qui sera de type date. Nous obtiendrons ainsi la requête suivante:

```
curl -X POST 'http://localhost:9200/articles/_doc?pretty' -H 'Content-Type: applicati
  {
    "created_at": "2021-06-07T16:24:32.000Z",
    "title": "mon article sur Elasticsearch",
    "content": "ceci est le contenu de mon article sur Elasticsearch",
    "category": "elasticsearch",
    "author": "Hatim"
  }
'
```

Résultat :

```
{
  "_index" : "articles",
  "_type" : "_doc",
  "_id" : "0YZ7uXoBzgM1NBNTNXt1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 3,
  "_primary_term" : 1
}
```

Comme vu sur le résultat, l'id est généré automatiquement pour nous. Pour le faire depuis la méthode PUT il suffit juste d'ajouter l'id à votre entrée:

```
curl -X PUT 'http://localhost:9200/articles/_doc/1?pretty' -H 'Content-Type: applicat
  {
    "created_at": "2021-05-07T16:24:32.000Z",
    "title": "mon second article sur Elasticsearch",
    "content": "ceci est le contenu de mon second article sur Elasticsearch",
    "category": "elasticsearch",
    "author": "Hatim"
  }
'
```

Résultat :

```
{
  "_index" : "articles",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 3,
  "_primary_term" : 1
}
```

Information

Le paramètre `pretty` de votre requête renvoie un JSON assez formaté (à utiliser uniquement pour le débogage !). Une autre option consiste à définir `?format=yaml` ce qui entraînera le retour du résultat au format YAML (parfois) plus lisible.

Pour ajouter plusieurs Documents en une seule requête, utilisez alors le paramètre _bulk en point de terminaison de votre requête. les fields de vos Documents au

format JSON doivent toujours être sur la même ligne et délimités par une nouvelle ligne. Chaque ligne doit se terminer par un caractère de nouvelle ligne, y compris la dernière ligne.

```
curl -X POST "localhost:9200/articles/_bulk?pretty" -H 'Content-Type: application/jso
{ "create": { } }
{ "created_at": "2021-08-07T16:24:32.000Z", "title": "mon article sur la stack ELK",
{ "create": { } }
{ "created_at": "2021-03-07T16:24:32.000Z", "title": "mon second article sur la stacl
'
```

Résultat :

```
{
  "took" : 10,
  "errors" : false,
  "items" : [
    {
      "create" : {
        "_index" : "articles",
        "_type" : "_doc",
        "_id" : "zoZ5uXoBzgM1NBNTantg",
        "_version" : 1,
        "result" : "created",
        "_shards" : {
          "total" : 2,
          "successful" : 1,
          "failed" : 0
        },
        "_seq_no" : 1,
        "_primary_term" : 1,
        "status" : 201
      }
    },
    {
      "create" : {
        "_index" : "articles",
        "_type" : "_doc",
        "_id" : "z4Z5uXoBzgM1NBNTantg",
        "_version" : 1,
        "result" : "created",
        "_shards" : {
          "total" : 2,
          "successful" : 1,
          "failed" : 0
        },
        "_seq_no" : 2,
        "_primary_term" : 1,
```

```
        "status" : 201
      }
    }
  ]
}
```

Récupération de données dans Elasticsearch

Pour voir la liste complète de vos index Elasticsearch, utilisez la requête suivante :

```
curl -X GET 'localhost:9200/_cat/indices?pretty'
```

Résultat :

```
green  open .kibana_7.13.3_001                _QiHUUwqQEW3EB5YgEa1zw 1 0    4571
yellow open apache-2021.07.12                 rk1U3br_TRyUenUu6U3plw 1 1      85
green  open .apm-agent-configuration          JMLcxwD8QqKj_7ZgQcKSdg 1 0       0
yellow open packetbeat-7.13.3-2021.07.14-000001 AlCvCInWRM6-B3mXUx4mVw 1 1 2770319
green  open .kibana_task_manager_7.13.3_001   ynWJoHQ4TIWrm5W_iy8RIQ 1 0      10 15
green  open .tasks                            nmDN_yhTSti1Q-s2kdQ2aQ 1 0      12
yellow open metricbeat-7.13.3-2021.07.13-000001 jHgARIOyRd-SfFfrWzMGPw 1 1   63684
green  open .kibana-event-log-7.13.3-000001   oSOQf0ChTTiSiGzPE1aq5Q 1 0       8
green  open .apm-custom-link                  wypmzo85R8uQ2UTcbnyxWQ 1 0       0
green  open .async-search                     6wnbxfSSREm4S1BkTIg09Q 1 0     174
yellow open filebeat-7.13.3-2021.07.12-000001 SSm52h8kTeeXcy4pOUM3Jg 1 1     936
yellow open articles                          VX_kMJDTSTaggmTVqJ0rmw 1 1       1
```

Dans cette sortie, nous retrouvons la liste de l'index que nous avons créé précédemment, un index Kibana et les index créés dans nos précédents chapitres.

Une fois que vous avez indexé vos données dans Elasticsearch, vous pouvez commencer à les récupérer et à les analyser. Encore une fois, via l'API REST Elasticsearch, nous utilisons la méthode GET.

Pour afficher tous les Documents d'un index dans Elasticsearch nous utiliserons la requête GET avec comme point de terminaison le mot _search :

```
curl -X GET 'localhost:9200/articles/_doc/_search?pretty'
```

Résultat :

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "articles",
        "_type" : "_doc",
        "_id" : "1",
        "_score" : 1.0,
        "_source" : {
          "created_at" : "2021-05-07T16:24:32.000Z",
          "title" : "mon second article sur Elasticsearch",
          "content" : "ceci est le contenu de mon second article sur Elasticsearch",
          "category" : "elasticsearch",
          "author" : "Hatim"
        }
      },
      {
        "_index" : "articles",
        "_type" : "_doc",
        "_id" : "zoZ5uXoBzgM1NBNTantg",
        "_score" : 1.0,
        "_source" : {
          "created_at" : "2021-08-07T16:24:32.000Z",
          "title" : "mon article sur la stack ELK",
          "content" : "ceci est le contenu de mon article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim"
        }
      },
      {
        "_index" : "articles",
        "_type" : "_doc",
        "_id" : "z4Z5uXoBzgM1NBNTantg",
        "_score" : 1.0,
        "_source" : {
          "created_at" : "2021-03-07T16:24:32.000Z",
          "title" : "mon second article sur la stack ELK",
          "content" : "ceci est le contenu de mon second article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim "
```

```
        }
      },
      {
        "_index" : "articles",
        "_type" : "_doc",
        "_id" : "0YZ7uXoBzgM1NBNTNXt1",
        "_score" : 1.0,
        "_source" : {
          "created_at" : "2021-06-07T16:24:32.000Z",
          "title" : "mon article sur Elasticsearch",
          "content" : "ceci est le contenu de mon article sur Elasticsearch",
          "category" : "elasticsearch",
          "author" : "Hatim"
        }
      }
    ]
  }
}
```

Le résultat contient un certain nombre de fields supplémentaires qui décrivent à la fois la recherche et le résultat. Voici un aperçu rapide de ces fields :

- `_took` : le temps en millisecondes que la recherche a pris.
- `_timed_out` : s'il y a eu un Time Out dans votre recherche.
- `_shards` : le nombre de shards Lucene recherchés, et leurs taux de réussite et d'échec.
- `_hits` : les résultats réels, ainsi que les métadonnées pour les résultats.
- `_score` : le score de pertinence de chaque Document trouvé dans votre recherche (plus le score est élevé plus le résultat sera pertinent).

Voici la commande pour récupérer un Document bien particulier via son id :

```
curl -X GET 'localhost:9200/articles/_doc/1?pretty'
```

Résultat :

```
{
  "_index" : "articles",
```

```
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "created_at" : "2021-05-07T16:24:32.000Z",
    "title" : "mon second article sur Elasticsearch",
    "content" : "ceci est le contenu de mon second article sur Elasticsearch",
    "category" : "elasticsearch",
    "author" : "Hatim"
  }
}
```

Vous avez également la possibilité d'afficher uniquement certains fields de votre Document. Supposons que nous souhaitons récupérer uniquement le nom de l'auteur et le titre de l'article, il faut alors exécuter une requête GET avec le paramètre _source suivie des noms des fields à filtrer :

```
curl -X GET 'localhost:9200/articles/_doc/1?pretty&_source=author,title'
```

Résultat :

```
{
  "_index" : "articles",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "author" : "Hatim",
    "title" : "mon second article sur Elasticsearch"
  }
}
```

Si les métadonnées d'une entrée ne vous intéressent pas, utilisez la commande suivante :

```
curl -X GET 'localhost:9200/articles/_doc/1/_source?pretty&_source=author,title'
```

Résultat :

```
{
  "author" : "Hatim",
  "title" : "mon second article sur Elasticsearch"
}
```

Rechercher des données dans Elasticsearch

Nous utiliserons également GET pour effectuer des recherches en appelant cette fois-ci le paramètre _search comme point de terminaison. Pour le moment récupérant tous nos Documents avec le filtre match_all, comme suit :

```
curl -X GET "localhost:9200/articles/_doc/_search?pretty" -H 'Content-Type: applicati
{
  "query": {
    "match_all": { }
  }
}
'
```

Résultat :

```
{
  "took" : 0,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 4,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "articles",
        "_type" : "_doc",
        "_id" : "y4ZSuXoBzgM1NBNTFHtr",
        "_score" : 1.0,
        "_source" : {
          "created_at" : "2021-07-23 16:21:46",
          "title" : "mon article sur la stack ELK",
```

```
          "content" : "ceci est le contenu de mon article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim"
        }
      },
      {
        "_index" : "articles",
        "_type" : "_doc",
        "_id" : "zIZSuXoBzgM1NBNTFHtr",
        "_score" : 1.0,
        "_source" : {
          "created_at" : "2021-07-23 16:21:46",
          "title" : "mon second article sur la stack ELK",
          "content" : "ceci est le contenu de mon second article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim "
        }
      },
      etc ...
    ]
  }
}
```

Si les métadonnées ne vous intéressent pas, alors vous pouvez utiliser paramètre filter_path pour réduire la réponse renvoyée par Elasticsearch. Ce paramètre prend une liste de filtres séparés par des virgules. Exemple :

```
curl -X GET "localhost:9200/articles/_doc/_search?pretty&filter_path=hits.hits._sourc
{
  "query": {
    "match_all": { }
  }
}
'
```

Résultat :

```
{
  "hits" : {
    "hits" : [
      {
        "_source" : {
          "created_at" : "2021-05-07T16:24:32.000Z",
          "title" : "mon second article sur Elasticsearch",
          "content" : "ceci est le contenu de mon second article sur Elasticsearch",
          "category" : "elasticsearch",
          "author" : "Hatim"
        }
```

```
      },
      {
        "_source" : {
          "created_at" : "2021-08-07T16:24:32.000Z",
          "title" : "mon article sur la stack ELK",
          "content" : "ceci est le contenu de mon article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim"
        }
      },
      etc ...
    ]
  }
}
```

Par défaut, la section `hits` de la réponse inclut jusqu'aux 10 premiers Documents qui correspondent à la recherche. Pour augmenter le nombre de Documents de votre recherche utilisez le paramètre `size` suivi du nombre de résultats que vous souhaitez afficher :

```
curl -X GET "localhost:9200/articles/_doc/_search?pretty&filter_path=hits.hits._sourc
{
  "query": {
    "match_all": { }
  }
}
'
```

Vous avez également la possibilité de trier vos résultats grâce au filtre sort. Dans notre cas nous souhaitons afficher les articles les plus récents en triant nos résultats par ordre décroissant depuis notre field `created_at` :

```
curl -X GET "localhost:9200/articles/_doc/_search?pretty&filter_path=hits.hits._sourc
{
  "query": {
    "match_all": { }
  },
  "sort": [ {"created_at": "desc"} ]
}
'
```

Résultat :

```
{
  "hits" : {
    "hits" : [
      {
        "_source" : {
          "created_at" : "2021-08-07T16:24:32.000Z",
          "title" : "mon article sur la stack ELK",
          "content" : "ceci est le contenu de mon article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim"
        }
      },
      {
        "_source" : {
          "created_at" : "2021-06-07T16:24:32.000Z",
          "title" : "mon article sur Elasticsearch",
          "content" : "ceci est le contenu de mon article sur Elasticsearch",
          "category" : "elasticsearch",
          "author" : "Hatim"
        }
      },
      {
        "_source" : {
          "created_at" : "2021-05-07T16:24:32.000Z",
          "title" : "mon second article sur Elasticsearch",
          "content" : "ceci est le contenu de mon second article sur Elasticsearch",
          "category" : "elasticsearch",
          "author" : "Hatim"
        }
      },
      {
        "_source" : {
          "created_at" : "2021-03-07T16:24:32.000Z",
          "title" : "mon second article sur la stack ELK",
          "content" : "ceci est le contenu de mon second article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim "
        }
      }
    ]
  }
}
```

Pour récupérer les Documents contenant une valeur spécifique dans un field vous utilisez le filtre match suivi du field et de la valeur à rechercher:

```
url -X GET "localhost:9200/articles/_doc/_search?pretty&filter_path=hits.hits._source
{
  "query": {
    "match": { "category": "ELK" }
```

```
  }
}
'
```

Résultat :

```
{
  "hits" : {
    "hits" : [
      {
        "_source" : {
          "created_at" : "2021-08-07T16:24:32.000Z",
          "title" : "mon article sur la stack ELK",
          "content" : "ceci est le contenu de mon article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim"
        }
      },
      {
        "_source" : {
          "created_at" : "2021-03-07T16:24:32.000Z",
          "title" : "mon second article sur la stack ELK",
          "content" : "ceci est le contenu de mon second article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim "
        }
      }
    ]
  }
}
```

Pour faire une recherche sur des fields spécifiques, vous utiliserez le filtre `fields`, comme suit:

```
curl -X GET "localhost:9200/articles/_doc/_search?pretty&filter_path=hits.hits._sourc
{
  "query": {
    "multi_match" : {
      "query":  "ELK",
      "fields": [ "title", "content" ]
    }
  }
}
'
```

Résultat :

```
{
  "hits" : {
    "hits" : [
      {
        "_source" : {
          "created_at" : "2021-08-07T16:24:32.000Z",
          "title" : "mon article sur la stack ELK",
          "content" : "ceci est le contenu de mon article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim"
        }
      },
      {
        "_source" : {
          "created_at" : "2021-03-07T16:24:32.000Z",
          "title" : "mon second article sur la stack ELK",
          "content" : "ceci est le contenu de mon second article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim "
        }
      }
    ]
  }
}
```

Nous avons également un autre type de requêtes qu'on surnomme "Range query". Ils permettent d'effectuer une recherche sur une plage de données. Dans notre cas nous souhaitons récupérer les articles créés à partir du 07/07/2021. Pour ce faire, nous utiliserons le filtre range avec l'option `gte`:

```
curl -X GET "localhost:9200/articles/_doc/_search?pretty&filter_path=hits.hits._sourc
{
  "query": {
    "range": {
      "created_at": {
        "gte": "2021-07-07"
      }
    }
  },
  "fields": [
    "created_at"
  ],
  "sort": [
    {
      "created_at": "desc"
    }
  ]
```

```
}
'
```

Résultat :

```
{
  "hits" : {
    "hits" : [
      {
        "_source" : {
          "created_at" : "2021-08-07T16:24:32.000Z",
          "title" : "mon article sur la stack ELK",
          "content" : "ceci est le contenu de mon article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim"
        }
      }
    ]
  }
}
```

les autres options disponibles pour ce type de filtres sont les suivantes:

- `gt` : supérieur à.
- `gte` : supérieur ou égal à.
- `lt` : moins de.
- `lte` : inférieur ou égal à.

Nous avons également un autre type de requêtes qu'on surnomme "Bool query". Il permettent de combiner plusieurs recherches sous forme de clause/conditions (correspond au `if` dans un langage de programmation standard) dans une seule et même requête.

Discutons d'abord de la structure générale de la requête Bool :

```
POST _search
{
  "query": {
    "bool": {
```

```
      "must": [...],
      "filter": [...],
      "must_not": [...],
      "should": [.. .]
    }
  }
}
```

Voici une explication des différentes options du filtre bool :

- `must` : la condition doit apparaître dans les Documents correspondants et contribuera à modifier le score de chaque résultat (correspond au `OU` logique dans un langage de programmation).
- `filtre` : la condition doit apparaître dans les Documents correspondants. Cependant, contrairement à `must`, le score de la requête sera ignoré.
- `should` : la condition doit apparaître dans le document correspondant (correspond au `AND` logique dans un langage de programmation).
- `must_not` : la condition ne doit pas apparaître dans les documents correspondants.

Dans cet exemple nous souhaitons récupérer tous les articles créés par l'auteur "Hatim" et qui ne sont pas de catégorie "elasticsearch" et créés à partir du 03/07/2021. Nous aurons ainsi la requête suivante :

```
curl -X GET "localhost:9200/articles/_doc/_search?pretty&filter_path=hits.hits._sourc
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "author": "Hatim"
          }
        },
        {
          "range": {
            "created_at": {
```

```
              "gte" : "2021-03-07"
            }
          }
        }
      ],
      "must_not": [
        {
          "match": {
            "category": "elasticsearch"
          }
        }
      ]
    }
  }
}
'
```

Résultat :

```
{
  "hits" : {
    "hits" : [
      {
        "_source" : {
          "created_at" : "2021-08-07T16:24:32.000Z",
          "title" : "mon article sur la stack ELK",
          "content" : "ceci est le contenu de mon article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim"
        }
      },
      {
        "_source" : {
          "created_at" : "2021-03-07T16:24:32.000Z",
          "title" : "mon second article sur la stack ELK",
          "content" : "ceci est le contenu de mon second article sur stack ELK",
          "category" : "ELK",
          "author" : "Hatim "
        }
      }
    ]
  }
}
```

Ce type de requêtes qui permettent des recherches plus avancées est nommé **requête DSL**. Nous avons pu voir quelques-unes d'entres elles, mais il existe un large éventail d'options disponibles dans ce type de recherche que vous pouvez combiner et assortir avec différentes options pour obtenir les résultats dont vous

avez besoin. Plus d'informations sur la page officielle ELK.

Modifier des données dans Elasticsearch

Pour ajouter un field à un Document déjà existant, utilisez la méthode POST avec le paramètre _update. Dans notre exemple, nous allons ajouter un field nommé "is_private" afin de mettre un de nos articles créé précédemment en privé:

```
curl -X POST "localhost:9200/articles/_update/1?pretty" -H 'Content-Type: applicatio
{
  "doc": {
    "is_private": true
  }
}
'
```

Résultat :

```
{
  "_index" : "articles",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 6,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 8,
  "_primary_term" : 1
}
```

Vérifions si notre field s'est bien enregistré dans notre Document:

```
curl -X GET 'localhost:9200/articles/_doc/1/_source?pretty'
```

Résultat :

```
{
  "created_at" : "2021-05-07T16:24:32.000Z",
  "title" : "mon second article sur Elasticsearch",
```

```
  "content" : "ceci est le contenu de mon second article sur Elasticsearch",
  "category" : "elasticsearch",
  "author" : "Hatim",
  "is_private" : true
}
```

C'est bien le cas ! D'ailleurs vous pouvez utiliser la même requête pour modifier la valeur d'un field. Dans cet exemple, nous allons passer notre article en mode public:

```
curl -X POST "localhost:9200/articles/_update/1?pretty" -H 'Content-Type: applicatio
{
  "doc": {
    "is_private": false
  }
}
'
```

Résultat :

```
{
  "_index" : "articles",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 7,
  "result" : "noop",
  "_shards" : {
    "total" : 0,
    "successful" : 0,
    "failed" : 0
  },
  "_seq_no" : 9,
  "_primary_term" : 1
}
```

Vérifions si notre field s'est bien enregistré:

```
curl -X GET 'localhost:9200/articles/_doc/1/_source?pretty'
```

Résultat :

```
{
  "created_at" : "2021-05-07T16:24:32.000Z",
  "title" : "mon second article sur Elasticsearch",
  "content" : "ceci est le contenu de mon second article sur Elasticsearch",
  "category" : "elasticsearch",
  "author" : "Hatim",
```

```
  "is_private" : false
}
```

Suppression des données dans Elasticsearch

Pour supprimer un Document d'Elasticsearch, c'est aussi simple que de rentrer des données dans Elasticsearch. La méthode HTTP à utiliser cette fois est sans surprise la méthode DELETE avec l'id du Document à supprimer:

```
curl -X DELETE 'localhost:9200/articles/_doc/1?pretty'
```

Résultat :

```
{
  "_index" : "articles",
  "_type" : "_doc",
  "_id" : "1",
  "_version" : 8,
  "result" : "deleted",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 10,
  "_primary_term" : 1
}
```

Si on tente de refaire une recherche sur notre Document nous obtiendrons une erreur "404 HTTP Not Found":

```
curl -X GET 'localhost:9200/articles/_doc/1/_source?pretty'
```

```

```

Erreur :

```
{
  "error" : {
    "root_cause" : [
      {
```

```
        "type" : "resource_not_found_exception",
        "reason" : "Document not found [articles]/[_doc]/[1]"
      }
    ],
    "type" : "resource_not_found_exception",
    "reason" : "Document not found [articles]/[_doc]/[1]"
  },
  "status" : 404
}
```

Pour supprimer un index, il suffit tout simplement d'utiliser une nouvelle fois la requête DELETE avec comme point de terminaison le nom de l'index à supprimer:

```
curl -X DELETE 'localhost:9200/articles?pretty'
```

Résultat :

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

Vérifions si notre index a bien été supprimé en vérifiant la liste des index:

```
curl -X GET 'localhost:9200/_cat/indices?pretty'
```

Résultat :

```
green  open .kibana_7.13.3_001                _QiHUUwqQEW3EB5YgEa1zw 1 0    4561
yellow open metricbeat-7.13.3-2021.07.13-000001 jHgARIOyRd-SfFfrWzMGPw 1 1   63684
green  open .apm-custom-link                  wypmzo85R8uQ2UTcbnyxWQ 1 0       0
green  open .kibana-event-log-7.13.3-000001   oSOQf0ChTTiSiGzPE1aq5Q 1 0       8
yellow open apache-2021.07.12                 rk1U3br_TRyUenUu6U3plw 1 1      85
green  open .apm-agent-configuration          JMLcxwD8QqKj_7ZgQcKSdg 1 0       0
green  open .async-search                     6wnbxfSSREm4S1BkTIg09Q 1 0     174
yellow open filebeat-7.13.3-2021.07.12-000001 SSm52h8kTeeXcy4pOUM3Jg 1 1     936
yellow open packetbeat-7.13.3-2021.07.14-000001 AlCvCInWRM6-B3mXUx4mVw 1 1 2770319
green  open .tasks                            nmDN_yhTSti1Q-s2kdQ2aQ 1 0      12
green  open .kibana_task_manager_7.13.3_001   ynWJoHQ4TIWrm5W_iy8RIQ 1 0      10 2
```

Bim, on ne le voit plus !

Conclusion

Ce chapitre avait pour but d'aider les débutants avec Elasticsearch et ne fournit que les étapes de base des opérations qu'on peut effectuer dans Elasticsearch. On peut cependant, commencer à répondre à notre question de départ "qu'est-ce qu'Elasticsearch ?"

Dans cet article, nous avons tenté de répondre à cette question à travers la compréhension de son fonctionnement et de son utilisation et nous n'avons qu'effleuré la surface pour apprendre tout ce qu'il y a à ce sujet. Mais sur la base de ce que nous avons couvert, nous pouvons résumer brièvement qu'Elasticsearch est à la base un moteur de recherche, dont l'architecture et les composants sous-jacents le rendent rapide et évolutif, au cœur d'un écosystème d'outils complémentaires qui ensemble peuvent être utilisés pour de nombreux cas d'utilisation, notamment la recherche, l'analyse, le traitement et le stockage des données.