Elasticsearch

RÉALISÉ PAR : MARWA KHARBICHI

Encadré par : Prof. KARIM LAMIA

Université Hassan 1er

mars 2025

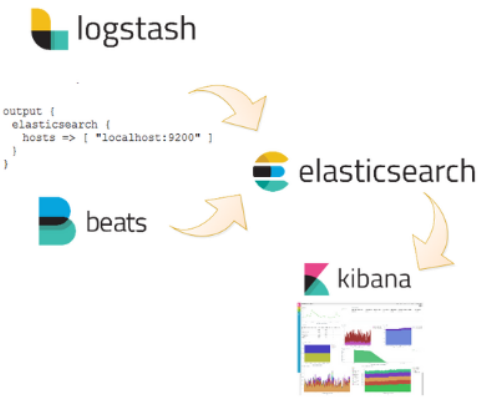
PLAN

Introduction

Elasticsearch est un moteur de recherche et d'analyse distribué et RESTful, basé sur Apache Lucene. Conçu pour la scalabilité horizontale, la fiabilité et la recherche en temps réel, il sert également de base de données NoSQL pour stocker, indexer et interroger efficacement des données structurées et non structurées.

Introduction

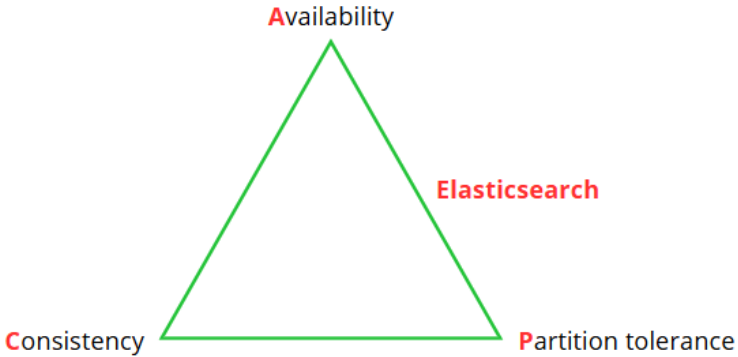
- Elasticsearch est indispensable dans l'écosystème ELK.



Introduction

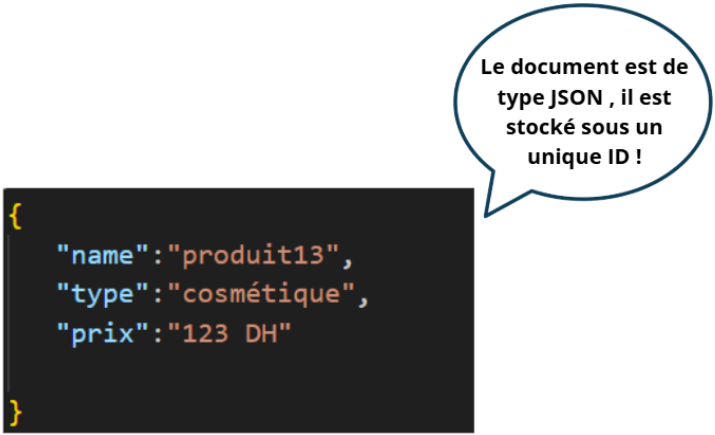
- Les entreprises qui utilisent elasticsearch :

Introduction

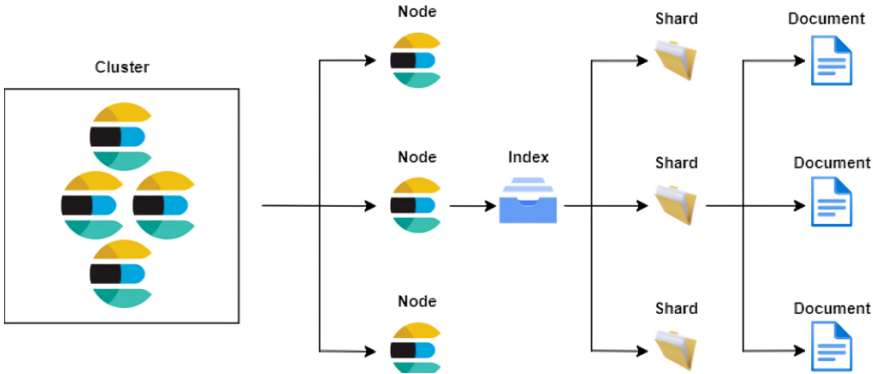


Introduction

- Les données sont stockées sous forme de documents.



Architecture d'Elasticsearch

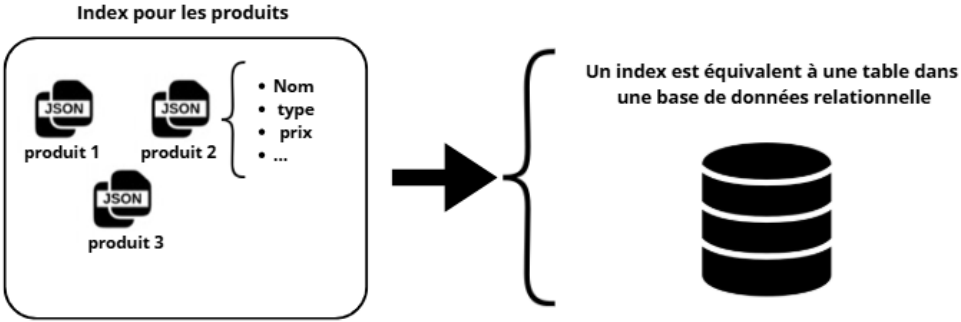


Architecture d'Elasticsearch

Un index est un peu comme une table dans une base de données sur un SGBD relationnel. Chaque index dispose d'un mapping, qui permet de définir la structure des types.

Le mapping est similaire à la définition de schéma dans une base de données relationnelle. Le mapping peut être définit manuellement, mais aussi généré automatiquement quand les documents sont indexés.

Architecture d'Elasticsearch

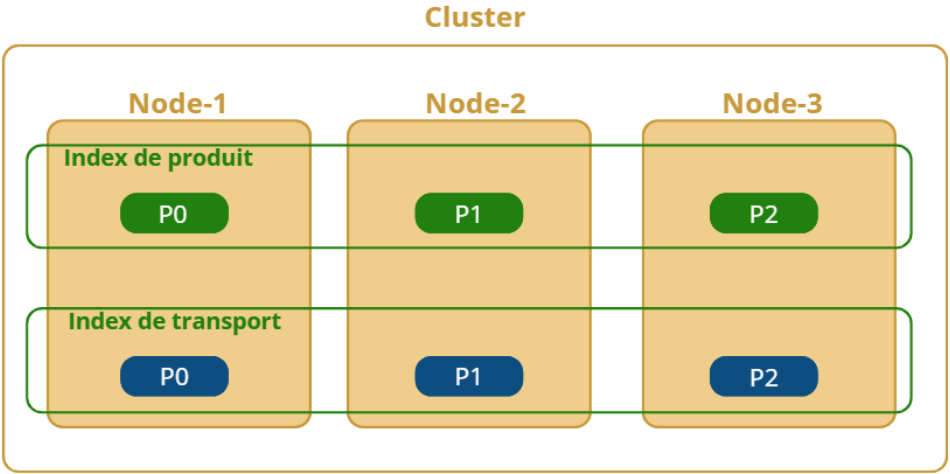


Architecture d'Elasticsearch

- C'est quoi un fragement ?

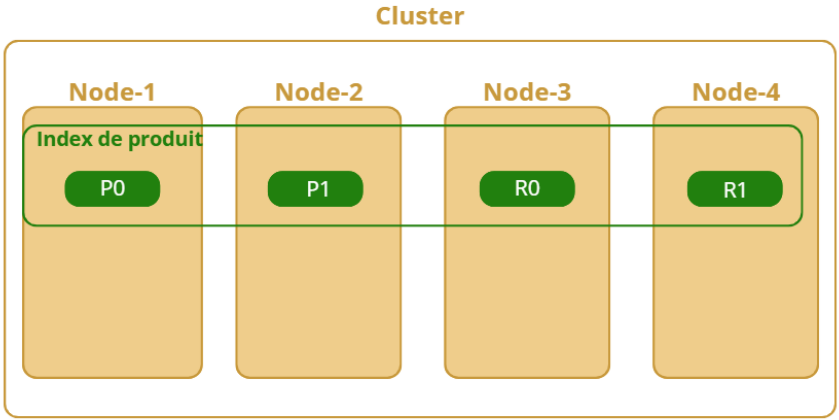
Un fragement ou bien "shard"est une partition de l'index qui correspond à une instance de Lucene. Les shards sont gérés de manière automatique par Elasticsearch, un shard peut être primaire ou être un réplicat. Les replica shards permettent d'améliorer les performances et d'éviter les erreurs en remplaçant une shard primaire en cas d'erreur.

Architecture d'Elasticsearch



Architecture d'Elasticsearch

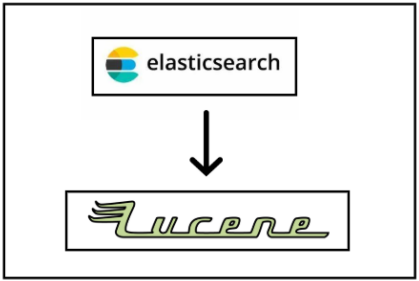
- Fragements et réplication :



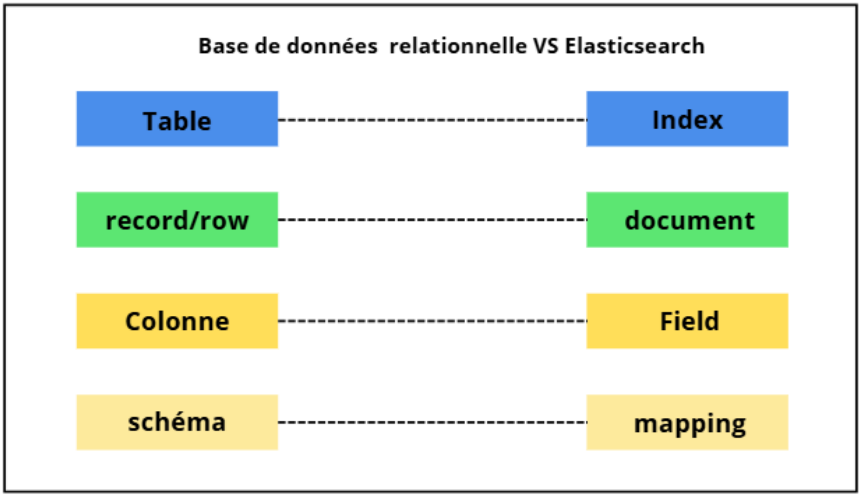
Architecture d'Elasticsearch

Rôle de lucene dans l'architecture d'Elasticsearch :

- Lucene est la bibliothèque java qui effectue les recherches textuelles , les indexations , les analyses de texte, etc.
- Chaque shard primaire ou répliqué dans Elasticsearch est en réalité un index Lucene.



Architecture d'Elasticsearch



Architecture d'Elasticsearch

Table d'employés

Name	Location	Age
John	Paris	22
Alice	New York	23
Dupont	CasaBlanca	25

Index d'employés


```
{
"Name":"John",
"Location":"Paris",
"Age": 22
}

{
"Name":"Alice",
"Location":"New York",
"Age": 23
}

{
"Name":"Dupont",
"Location":"CasaBlanca",
"Age": 25
}
```

Avantages

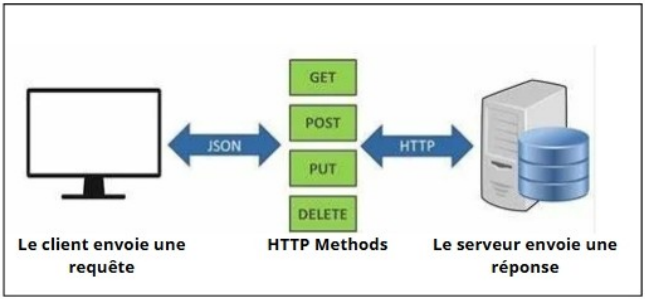
- **Scalabilité** : Distribué par nature, permet de gérer de grandes quantités de données.
- **Performance** : Recherche et analyse en temps réel.
- **Flexibilité** : Supporte des données structurées et non structurées.
- **Intégration** : Facilement intégrable avec d'autres outils (Kibana, Logstash, etc.).

Cas d'utilisation

- **Recherche full-text** : Moteur de recherche pour sites web ou applications.
- **Analyse de logs** : Centralisation et analyse des logs en temps réel.
- **Business Analytics** : Analyse de données métier pour des insights rapides.
- **Monitoring** : Surveillance des systèmes et applications.

Les méthodes HTTP pour l'indexation et la gestion des donnée

REST d'Elasticsearch est basée sur les principes du Transfert d'État Représentationnel (REST) qui permet d'effectuer des opérations CRUD (CREATE, READ, UPDATE, DELETE) .



Les méthodes HTTP pour l'indexation et la gestion des données

Opération	Méthode HTTP	Description	Exemple
Créer un index	PUT	Créer un nouvel index	PUT /mon_index
Indexer un document	PUT	Ajoute un document avec un ID spécifique. Si l'ID existe, le document est remplacé.	PUT /mon_index/_doc/1 { "name": "Produit A", "price": 100 }
Ajouter un document	POST	Ajoute un document avec un ID auto-généré	POST /mon_index/_doc { "name": "Produit B", "price": 200 }
Mettre à jour un document	POST	Met à jour partiellement un document existant	POST /mon_index/_update/1 { "doc": { "price": 150 } }
Récupérer un document	GET	Récupère un document par son ID.	GET /mon_index/_doc/1

Les méthodes HTTP pour l'indexation et la gestion des données

Opération	Méthode HTTP	Description	Exemple
Supprimer un document	DELETE	Supprime un document par son ID	DELETE /mon_index/_doc/1
Supprimer un index	DELETE	Supprime un index et tous ses documents	DELETE /mon_index
Rechercher des documents	GET ou POST	Exécute une requête de recherche	GET /mon_index/_search POST /mon_index/_search { "query": { "match_all": {} } }
Vérifier l'existence d'un document	HEAD	Vérifie si un document existe sans retourner son contenu	HEAD /mon_index/_doc/1

Query DSL

Query DSL (Domain Specific Language) est un langage de requête puissant et flexible utilisé dans Elasticsearch pour interroger et analyser les données. Il permet de construire des requêtes complexes en combinant différents types de requêtes et de filtres.

Query DSL

- Structure de base d'une requête Query DSL

```
GET /nom_de_l_index/_search
{
  "query": {
    "type_de_requête": {
      "champ": "valeur"
    }
  }
}
```

Query DSL

Requêtes de recherche en texte intégral (Full-Text Queries).

- match: Recherche en texte intégral avec analyse du texte.

```
GET /regions/_search
{
  "query": {
    "match": {
      "description": "France"
    }
  }
}
```

Query DSL

- multi_match: Recherche sur plusieurs champs.

```
GET /regions/_search
{
  "query": {
    "multi_match": {
      "query": "France",
      "fields": ["name", "description"]
    }
  }
}
```

Query DSL

Requêtes de termes (Term-Level Queries)

- term: Recherche un terme exact (sans analyse du texte).

```
"query": {
  "term": {
    "type.keyword": "région"
  }
}
```

Query DSL

- terms: Recherche plusieurs termes exacts.

```
GET /regions/_search
{
  "query": {
    "terms": {
      "name.keyword": ["Île-de-France", "Normandie", "Bretagne"]
    }
  }
}
```

Query DSL

- range : Recherche dans une plage de valeurs.

```
"range": {
  "date": {
    "gte": "2020-01-01",
    "lte": "2025-12-31"
  }
}
```

Query DSL

Requêtes booléennes (Boolean Queries)

- Combine plusieurs requêtes avec des opérateurs logiques (must, should, must_not, filter)

```
"query": {
  "bool": {
    "must": [
      { "match": { "description": "France" }},
      { "term": { "type.keyword": "région" }}
    ],
    "filter": [
      { "range": { "population": { "gte": 1000000 }}}
    ]
  }
}
```

Query DSL

Requêtes géospatiales (Geo Queries) :

Elasticsearch propose principalement deux types :

- geo_point : pour représenter un point géographique (latitude/longitude).
- geo_distance : pour représenter des formes géographiques complexes (polygones,lignes,cercles).

Query DSL

Indexation d'un point géographique (geo_point):

```
PUT /places
{
  "mappings": {
    "properties": {
      "name": { "type": "text" },
      "location": { "type": "geo_point" }
    }
  }
}
```

Query DSL

Ajout d'un document:

```
POST /places/_doc
{
  "name": "Café de Paris",
  "location": {
    "lat": 48.8566,
    "lon": 2.3522
  }
}
```

Query DSL

Requête geo_distance:Trouver les lieux à moins de 10km d'un point donné.

```
GET /places/_search
{
  "query": {
    "bool": {
      "filter": {
        "geo_distance": {
          "distance": "10km",
          "location": {
            "lat": 48.8566,
            "lon": 2.3522
          }
        }
      }
    }
  }
```

Query DSL

Exemple avec geo_shape :

```
PUT /regions
{
  "mappings": {
    "properties": {
      "name": { "type": "text" },
      "area": { "type": "geo_shape" }
    }
  }
}
```

Query DSL

Insertion d'un document représentant une région (polygone) :

```
POST /regions/_doc
{
  "name": "Zone Paris",
  "area": {
    "type": "polygon",
    "coordinates": [
      [
        [2.20, 48.90],
        [2.45, 48.90],
        [2.45, 48.80],
        [2.20, 48.80],
        [2.20, 48.90]  // On referme le polygone
      ]
    ]
```

Query DSL

Requete pour trouver les régions qui intersectent un point donné :

```
GET /regions/_search
{
  "query": {
    "geo_shape": {
      "area": {
        "shape": {
          "type": "point",
          "coordinates": [2.35, 48.85]
        },
        "relation": "intersects"
      }
    }
  }
}
```

Query DSL

Requêtes d'agrégation (Aggregation Queries)

- aggs : Permet de calculer des métriques ou de regrouper des données.

```
GET /regions/_search
{
  "size": 0,
  "aggs": {
    "types_de_region": {
      "terms": {
        "field": "type.keyword"
      }
    }
  }
}
```

Query DSL

- Tri des résultats (sort).

```
GET /regions/_search
{
  "size": 5,
  "sort": [
    {
      "population": {
        "order": "desc"
      }
    }
  ]
}
```

Query DSL

- Pagination : Limiter le nombre de résultats et paginer.

```
GET /regions/_search
{
  "from": 5,
  "size": 5,
  "sort": [
    {
      "population": {
        "order": "desc"
      }
    }
  ]
}
```

Query DSL

- Mise en surbrillance (highlight).

```
GET /regions/_search
{
  "query": {
    "match": {
      "description": "France"
    }
  },
  "highlight": {
    "fields": {
      "description": {}
    }
  }
}
```

Démonstration

Exemple 1: Création de l'index manip_1:

```
curl -u username:password -X PUT "localhost:9200/manip_1?pretty" -H
    "Content-Type: application/json" -d "{
  \"settings\": {
    \"number_of_shards\": 1,
    \"number_of_replicas\": 1
  },
  \"mappings\": {
    \"properties\": {
      \"name\": { \"type\": \"text\" },
      \"age\": { \"type\": \"integer\" },
      \"grade\": { \"type\": \"keyword\" },
      \"subjects\": { \"type\": \"text\" }
    }
  }
}"
```

Démonstration

Après la création d'un index (via l'invite de commande ou windows power shell) , on arrive à voir que l'index a été crée avec succès :

```
C:\Users\hp>curl -u elastic:elastic -X PUT "localhost:9200/manip_1?pretty" -H "Content-Type: application/json" -d "{ \"settings\": { \"number_of_shards\": 1, \"number_of_replicas\": 1 }, \"mappings\": { \"properties\": { \"name\": { \"type\": \"text\" }, \"age\": { \"type\": \"integer\" }, \"grade\": { \"type\": \"keyword\" }, \"subjects\": { \"type\": \"text\" } } } }"
{
  "acknowledged" : true,
  "shards_acknowledged" : true,
  "index" : "manip_1"
}
```

Démonstration

Pour insérer des données dans un index Elasticsearch, on utilise la méthode POST.Cette méthode permet d'ajouter un nouveau document à un index spécifique. Si vous spécifiez un ID de document (comme 1 dans l'exemple ci-dessous), Elasticsearch utilisera cet ID pour le document. Si vous ne spécifiez pas d'ID, Elasticsearch en générera un automatiquement.

Démonstration

Insertion de données dans l'index manip_1:

```
curl -X POST "localhost:9200/manip_1/_doc/1?pretty" -H "Content-
   Type: application/json" -d "{
  \"name\": \"Alice\",
  \"age\": 20,
  \"grade\": \"A\",
  \"subjects\": [\"Math\", \"Physics\"]
}"
```

Démonstration

Résultat après insertion :

```
C:\Users\hp>curl -u elastic:elastic -X POST "localhost:9200/manip_1/_doc/1?pretty" -H "Content-Type: application/json" -d "{ \"name\"
: \"Alice\",\"age\":20, \"grade\": \"A\", \"subjects\": [\"Math\",\"Physics\"]}"
{
  "_index" : "manip_1",
  "_id" : "1",
  "_version" : 1,
  "result" : "created",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 0,
  "_primary_term" : 1
}
```

Démonstration

Voici un autre exemple d'insertion de données:

```
curl -X POST "localhost:9200/manip_1/_doc/1?pretty" -H "Content-
   Type: application/json" -d "{
  \"name\": \"Bob\",
  \"age\": 22,
  \"grade\": \"B\",
  \"subjects\": [\"Chemistry\", \"Biology\"]
}"
```

Démonstration

Résultat après insertion de données:

```
C:\Users\hp>curl -u elastic:elastic -X POST "localhost:9200/manip_1/_doc/1?pretty" -H "Content-Type: application/json" -d "{ \"name\"
: \"Bob\",\"age\":22, \"grade\": \"B\", \"subjects\": [\"Chemistry\",\"Biology\"]}"
{
  "_index" : "manip_1",
  "_id" : "1",
  "_version" : 2,
  "result" : "updated",
  "_shards" : {
    "total" : 2,
    "successful" : 1,
    "failed" : 0
  },
  "_seq_no" : 1,
  "_primary_term" : 1
}
```

Démonstration

Pour effectuer une recherche on utilise la méthode GET pour récupérer des documents correspondant à des critères spécifiques.

```
curl -X GET "localhost:9200/manip_1/_search?pretty" -H "Content-
   Type: application/json" -d "{
  \"query\": {
    \"match\": {
      \"grade\": \"A\"
    }
  }
}"
```

Démonstration

Résultat de recherche:

```
C:\Users\hp>curl -u elastic:elastic -X GET "localhost:9200/manip_1/_search?pretty" -H "Content-Type: application/json" -d "{\"query\": {\"match\": {\"subjects\": \"Math
\" }}}"
{
  "took" : 12,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 1,
      "relation" : "eq"
    },
    "max_score" : 0.2876821,
    "hits" : [
      {
        "_index" : "manip_1",
        "_id" : "1",
        "_score" : 0.2876821,
        "_source" : {
          "name" : "Charlie",
          "age" : 22,
          "grade" : "A",
          "subjects" : [
            "Math",
            "Computer Science"
          ]
        }
      }
    ]
  }
}
```

Démonstration

Prenons un autre exemple :

```
PUT /utilisateurs
{
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 1
  },
  "mappings": {
    "properties": {
      "nom": {
        "type": "text"
      },
      "prenom": {
        "type": "text"
      },
      "age": {
        "type": "integer"
      },
      "region": {
        "type": "keyword"
      }
    }
```

Démonstration

- La méthode _bulk permet d'envoyer plusieurs opérations en une seule requête.

```
POST /utilisateurs/_bulk
{ "index": { "_id": 1 } }
{ "nom": "Dupont", "prenom": "Jean", "age": 30, "region": " le -de-
   France" }
{ "index": { "_id": 2 } }
{ "nom": "Martin", "prenom": "Marie", "age": 25, "region": "
   Provence-Alpes-Côte d'Azur" }
{ "index": { "_id": 3 } }
{ "nom": "Durand", "prenom": "Pierre", "age": 40, "region": "
   Auvergne-Rhône-Alpes" }
{ "index": { "_id": 4 } }
{ "nom": "Leroy", "prenom": "Sophie", "age": 35, "region": "Hauts-
   de-France" }
```

Démonstration

- La méthode _update permet de mettre à jour un document existant.

```
POST /utilisateurs/_update/1
{
  "doc": {
    "age": 32
  }
}
```

Démonstration

- La méthode DELETE permet de supprimer un document spécifique en utilisant son ID.

```
DELETE /utilisateurs/_doc/3
```

- On peut vérifier l'état de l'index en effectuant une recherche:

```
GET /utilisateurs/_search
{
  "query": {
    "match_all": {}
  }
}
```

Démonstration

Lé résultat attendu :


```
{
  "hits": {
    "total": {
      "value": 3,
      "relation": "eq"
    },
    "hits": [
      {
        "_id": "1",
        "_source": {
          "nom": "Dupont",
          "prenom": "Jean",
          "age": 32,
          "region": " le -de-France"
        }
      },
      {
        "_id": "2",
        "_source": {
          "nom": "Martin",
          "prenom": "Marie",
          "age": 25,
          "region": "Provence-Alpes-Côte d'Azur"
        }
      },
      {
        "_id": "4",
        "_source": {
          "nom": "Leroy",
          "prenom": "Sophie",
          "age": 35,
          "region": "Hauts-de-France"
        }
```

Index Inversé

L'index inversé est une structure de données qui permet à Elasticsearch d'effectuer des recherches textuelles rapides. Son fonctionnement repose sur deux étapes essentielles:

- Analyse du texte : Le texte est découpé en termes (tokens) via le processus d'analyse.
- Création de l'index : Pour chaque terme, on enregistre dans quelle document il apparaît.

Index Inversé

Elasticsearch utilise des analyseurs pour transformer le texte lors de l'indexation et de la recherche. Ce processus est crucial pour une recherche efficace.En effet , un analyseur est composé de trois éléments principaux:

- filtres de caractères.
- tokeniseur.
- filtres de tokens.

Index Inversé

1. Filtres de caractères : modifient le texte avant la tokenization.

Term	Documents
html_strip	Supprime les balises html
mapping	Remplace des caractères selon une configuration
pattern_replace	Remplace du texte via des expressions régulières

Index Inversé

- Voici un exemple sur l'utilisation du filtre mapping:

```
PUT /mon_index
{
  "settings": {
    "analysis": {
      "char_filter": {
        "my_char_filter": {
          "type": "mapping",
          "mappings": ["- => _", "é => e"]
        }
      }
    }
  }
}
```

Index Inversé

2. Les tokeniseurs : découpent le texte en termes (tokens).

Term	Documents
standard	Découpe selon les espaces et la ponctuation
whitespace	Découpe uniquement sur les espaces
keyword	Garde le texte entier comme un seul token
pattern	Découpe selon une expression régulière
edge_ngram	Génère des n-grammes pour la saisie semi-automatique

Index Inversé

- Voici un exemple de l'utilisation du tokeniseur pattern :

```
PUT /email_example
{
  "settings": {
    "analysis": {
      "tokenizer": {
        "email_tokenizer": {
          "type": "pattern",
          "pattern": "\\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}\\b",
          "group": 0
        }
      }
    }
  }
}
```

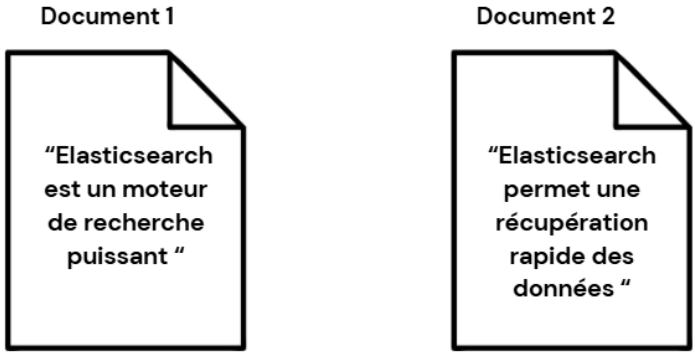
Index Inversé

3. Les filtres de tokens : Modifient les tokens générés par le tokeniseur.

Term	Documents
lowercase	Convertit en minuscules
stop	Supprime les mots vides (le, la, les, etc.)
synonym	Applique des synonymes
stemmer	Réduit à la racine des mots
asciifolding	Convertit les caractères accentués
ngram	Génère des sous-parties de mots

Index Inversé

- Voici un exemple simple illustrant le fonctionnement d'un index inversé.



Index Inversé

- L'index inversé pour ces documents ressemblerait à ceci :

Term	Documents
elasticsearch	1, 2
est	1
un	1
puissant	1
recherche	1
moteur	1
permet	2
rapide	2
données	2
récupération	2

Les Agrégations

Les agrégations de métriques calculent des valeurs numériques à partir des données, comme des moyennes, des sommes, des minimums, des maximums, etc.

Agrégation	Description
avg	Calcule la moyenne d'un champ numérique
sum	Calcule la somme des valeurs d'un champ numérique
min	Trouve la valeur minimale d'un champ numérique
max	Trouve la valeur maximale d'un champ numérique
stats	Retourne plusieurs métriques en une seule requête (count, sum, min, max, avg)

Les Agrégations

Prenons l'exemple de Calcul de la moyenne et la somme des prix des produits

```
GET /products/_search
{
  "size": 0,
  "aggs": {
    "average_price": {
      "avg": {
        "field": "price"
      }
    },
    "total_price": {
      "sum": {
        "field": "price"
      }
    }
  }
}
```

Les Agrégations

```
{
  "aggregations": {
    "average_price": {
      "value": 460.0  // Moyenne des prix
    },
    "total_price": {
      "value": 2300.0  // Somme des prix
    }
  }
}
```

Les Agrégations

Les agrégations de buckets regroupent les documents en "buckets" (groupes) en fonction de critères spécifiques, comme des intervalles, des termes, des plages de dates, etc.

Agrégation	Description
terms	Regroupe les documents par valeurs uniques d'un champ (par exemple, par catégorie)
range	Regroupe les documents en fonction de plages de valeurs (par exemple, des tranches de prix)
date_histogram	Regroupe les documents en fonction d'intervalles de temps (par exemple, par mois ou par année)
histogram	Regroupe les documents en fonction d'intervalles numériques

Les Agrégations

Exemple : Regrouper les produits par catégorie

```
GET /products/_search
{
  "size": 0,
  "aggs": {
    "by_category": {
      "terms": {
        "field": "category.keyword"
      }
    }
  }
}
```

Les Agrégations

Résultat : Regroupement par catégorie.


```
"aggregations": {
  "by_category": {
    "buckets": [
      {
        "key": "Electronics",
        "doc_count": 2
      },
      {
        "key": "Sports",
        "doc_count": 2
      },
      {
        "key": "Home Appliances",
        "doc_count": 1
      }
    ]
  }
```

Les Agrégations

Les agrégations de pipeline effectuent des calculs sur les résultats d'autres agrégations. Elles permettent de créer des analyses plus complexes en chaînant plusieurs agrégations.

Agrégation	Description
avg_bucket	Calcule la moyenne des résultats d'une agrégation de buckets
sum_bucket	Calcule la somme des résultats d'une agrégation de buckets
derivative	Calcule la dérivée (taux de changement) d'une agrégation
moving_avg	Calcule une moyenne mobile sur les résultats d'une agrégation

Les Agrégations

Exemple: Calculer la moyenne des prix moyens par catégorie

```
"size": 0,
"aggs": {
  "by_category": {
    "terms": {
      "field": "category.keyword"
    },
    "aggs": {
      "average_price": {
        "avg": {
          "field": "price"
        }
      }
    }
  },
  "avg_of_averages": {
    "avg_bucket": {
      "buckets_path": "by_category>average_price"  // Chemin vers l'agrégation à utiliser
    }
```

Les Agrégations

```
"aggregations": {
  "by_category": {
    "buckets": [
      {
        "key": "Electronics",
        "doc_count": 2,
        "average_price": {
          "value": 1000.0
        }
      },
      {
        "key": "Sports",
        "doc_count": 2,
        "average_price": {
          "value": 75.0
        }
      },
      {
        "key": "Home Appliances",
        "doc_count": 1,
        "average_price": {
          "value": 150.0
        }
      }
    ]
  },
  "avg_of_averages": {
    "value": 408.33  // Moyenne des prix moyens par catégorie
  }
```

Conclusion

- Elasticsearch est une solution puissante pour le stockage et la recherche de données.
- Son architecture distribuée et ses fonctionnalités de stockage en font un outil flexible et scalable.
- Idéal pour des cas d'utilisation variés, de la recherche full-text à l'analyse de logs.
- Une bonne compréhension des mécanismes de stockage est essentielle pour optimiser les performances.

Références

- https://www.elastic.co/elasticsearch
- https://www.elastic.co/guide/en/elasticsearch/reference/index.html