

Le mec au fond 🧑 , près de la fenêtre 🪟



# Introduction to Elasticsearch

For beginners, by a beginner

# Key Features

- 🔍 Real-time search and analytics
- 📈 Scalability
- ✓ High availability
- 🔓 Open-source
- 🔄 Flexible data models





## Un peu de terminologie

Relational Database	Elasticsearch
Database	Index
Table	Type
Row	Document
Column	Field
Schema	Mapping
SQL	Query DSL

💡 elastic will remove types

# Run Elasticsearch

```
docker pull elasticsearch:8.8.1
```

```
docker run -d --rm --name elasticsearch -p 9200:9200 -p 9300:9300 -e "discovery.type=sir
```



# API RESTful

Permet notamment de :

- 📁 Indexer un document
- 🔍 Rechercher un document
- ♻️ Mettre à jour un document
- 🗑️ Supprimer un document

[elastic.co/rest-apis.html](https://elastic.co/rest-apis.html)



# Création d'un index

L'URL de base se présente de la manière suivante

```
http://localhost:9200/_index/_doc/_id
```

👉 Ajoutons un index pour stocker des informations sur les voitures Tesla

```
curl -XPUT 'http://localhost:9200/tesla/'
```





# Mapping

👉 Ajoutons un mapping pour définir la structure de nos documents

```
curl -XPUT -H 'Content-Type: application/json' 'http://localhost:9200/tesla/_mapping' -c  
{  
  "properties": {  
    "name": {  
      "type": "text"  
    },  
    "code": {  
      "type": "keyword"  
    },  
    "description": {  
      "type": "text"  
    }  
  }  
}
```

Quand utiliser un type "text" ou "keyword" ? 🤔

- "text" : pour les champs qui doivent être analysés (recherche full-text)
- "keyword" : pour les champs qui ne doivent pas être analysés (recherche exacte)



# Indexation d'un document

👉 Ajoutons quelques models Tesla

```
curl -XPUT -H 'Content-Type: application/json' 'http://localhost:9200/tesla/_doc/1' -d '{
  "name": "Model S",
  "code": "S",
  "description": "LA berline électrique"
}'
```

```
curl -XPUT -H 'Content-Type: application/json' 'http://localhost:9200/tesla/_doc/2' -d '{
  "name": "Model 3",
  "code": "3",
  "description": "La voiture électrique des devs"
}'
```

```
curl -XPUT -H 'Content-Type: application/json' 'http://localhost:9200/tesla/_doc/3' -d '{
  "name": "Model X",
  "code": "X",
  "description": "Le SUV des richou"
}'
```

# Recherche dans Elasticsearch

> OÙ EST-CE QUE J'AI GARÉ MA MODEL S PLAID ? 🤔

👉 Ah, faisons une recherche rapide !

```
curl -XGET 'http://localhost:9200/tesla/_search?q=code:S'
```



# Recherche dans Elasticsearch

```
{
  "took":3,
  "timed_out":false,
  "_shards":{
    "total":1,
    "successful":1,
    "skipped":0,
    "failed":0
  },
  "hits":{
    "total":{
      "value":1,
      "relation":"eq"
    },
    "max_score":1.2039728,
    "hits":[
      {
        "_index":"tesla",
        "_id":"1",
        "_score":1.2039728,
        "_source":{
          "name":"Model S"
```

... NOMBRE TOTAL DE HITS

... L'INDEX D'OÙ IL PROVIENT

... L'ID DU DOCUMENT

... LA SOURCE DU DOCUMENT

# Requêtes plus complexes avec Query DSL

- Requête "match" : Recherche basique correspondant aux termes dans les champs des documents
- Requête "multi\_match" : Recherche un terme dans plusieurs champs différents
- Requête "bool" : Combine plusieurs requêtes avec des opérateurs logiques (AND/OR/NOT)
- Requête "aggregations" : Regroupe les données et fournit des statistiques sur ces groupes ("GROUP BY" etc)
- Et bien plus encore...
- [elastic.co/query-dsl.html](https://elastic.co/query-dsl.html)

# Exemple de requête "match"

👉 Allons à la recherche de notre Model S !

```
curl -XGET -H 'Content-Type: application/json' 'http://localhost:9200/tesla/_search' -d '{
  "query": {
    "match": {
      "code": "S"
    }
  }
}'
```



# Exemple de requête "bool"

👉 Et si on cherchait une Model S ou un Model 3 ?

```
curl -XGET -H 'Content-Type: application/json' 'http://localhost:9200/tesla/_search' -d '{
  "query": {
    "bool": {
      "should": [
        { "match": { "code": "S" } },
        { "match": { "code": "3" } }
      ]
    }
  }
}'
```



# Exemple de requête "multi\_match"

👉 Recherchons tous les modèles Tesla qui ont 'model' dans leur nom ou leur description

```
curl -XGET -H 'Content-Type: application/json' 'http://localhost:9200/tesla/_search' -d '{
  "query": {
    "multi_match": {
      "query": "model",
      "fields": ["name", "description"]
    }
  }
}'
```

# Exemple de requête avec "aggregations"

👉 Regardons combien de modèles Tesla nous avons dans notre index

```
curl -XGET -H 'Content-Type: application/json' 'http://localhost:9200/tesla/_search' -d '{
  "size": 0,
  "aggs": {
    "group_by_code": {
      "terms": {
        "field": "code"
      }
    }
  }
}'
```



# Les analyzers

👉 Quand on indexe un document, Elasticsearch analyse le texte et le stocke dans un format optimisé pour la recherche.

Prenons l'exemple suivant :

```
curl -XPUT -H 'Content-Type: application/json' 'http://localhost:9200/tesla/_doc/4' -d '{
  "name": "Model Y",
  "code": "Y",
  "description": "Le SUV compact"
}'
```

- Le champ "name" est analysé et stocké sous la forme "model" et "y"
- Le champ "description" est analysé et stocké sous la forme "suv" et "compact"
- Le champ "code" n'est pas analysé et stocké sous la forme "Y"
- Le champ "code" est donc plus adapté pour une recherche exacte
- Le champ "name" et "description" sont plus adaptés pour une recherche full-text
- [elastic.co/analysis.html](https://elastic.co/analysis.html)

# L'importance de l'analyse

👉 Prenons l'exemple suivant :

```
{  
  "description": "Outre la recharge à domicile et sur bornes ordinaires, la Model 3  
  dispose de bornes de recharges ultra-rapides appelées « Superchargeurs ». Elles permet  
  de récupérer 80 % de la charge en moins de 30 min ou 300 km en moins de 15 min sur la  
}
```

Avec l'analyse par défaut d'Elasticsearch, le champ "description" sera analysé et stocké sous la forme suivante :

```
{  
  "description": ["outre", "recharge", "domicile", "borne", "ordinaire", "model", "super  
}
```

- Lors d'une recherche, les termes recherchés sont analysés aussi, avec la même technique.
- Si on recherche "Bornes" au pluriel, on ne trouvera pas de résultat.
- Le token obtenu serait "bornes", qui n'existe pas dans notre index.
- L'utilisateur va donc devoir saisir les mots exacts : avec pluriels, accents, ligature...
- Cela n'est bien sûr pas acceptable !

# L'importance de l'analyse

En utilisant l'analyzeur français d'Elasticsearch, les tokens seront plutôt :

```
{  
  "description": ["recharg", "domicil", "born", "ordinaire", "model", "superchargeur", "c  
}
```

- Mieux, maintenant "bornes" est analysé et stocké sous la forme "born"
- On peut donc rechercher "bornes" ou "borne" et obtenir le même résultat
- [elastic.co/analysis-lang-analyzer.html](https://www.elastic.co/analysis-lang-analyzer.html)

# Les différentes étapes de l'analyse

- **Character filter** : Supprime les caractères spéciaux
- **Tokenizer** : Découpe le texte en tokens
- **Token filter** : Modifie les tokens obtenus

Le "tokenizer" par défaut est basique, tandis que l'"icu\_tokenizer", basé sur la librairie ICU, gère efficacement le standard Unicode. Les "token filters" permettent de modifier ces tokens :

- **"lowercase"** transforme tous les tokens en minuscules,
- **"elision"** supprime les élisions (ex : "l'homme" devient "homme"),
- **"stop"** élimine les tokens considérés comme du bruit (ex : en, au, du, par, est),
- **"stemmer"** réduit les mots à leur racine (ex : "marche", "marches", "marcher", "marchera" deviennent "march").

# Pipelines

👉 Les pipelines permettent de définir une suite d'opérations à effectuer sur un champ lors de l'indexation.

Utiles pour :

- Modifier/manipuler les données avant l'indexation
- Enrichir les données avec des informations supplémentaires
- Supprimer des données non désirées

Vous pouvez les voir comme un moyen de transformer et enrichir vos données "à la volée" lors de l'indexation.

[elastic.co/pipeline.html](https://elastic.co/pipeline.html)

# Crée un pipeline

👉 Créons un pipeline pour enrichir nos données Tesla

```
curl -XPUT -H 'Content-Type: application/json' 'http://localhost:9200/_ingest/pipeline/t
{
  "description": "Pipeline pour enrichir les données Tesla",
  "processors": [
    {
      "set": {
        "field": "brand",
        "value": "Tesla"
      }
    },
    {
      "set": {
        "field": "created_at",
        "value": "{{_ingest.timestamp}}"
      }
    }
  ]
}
```

# Ajouter un pipeline à l'indexation

👉 Ajoutons notre pipeline à l'indexation

```
curl -XPUT -H 'Content-Type: application/json' 'http://localhost:9200/tesla/_doc/4?pipeline={
  "name": "Model Y",
  "code": "Y",
  "description": "Le SUV compact"
}'
```

Ce qui nous donne :

```
{
  "_index": "tesla",
  "_id": "4",
  "_score": 0.6931471,
  "_source": {
    "name": "Model Y",
    "description": "Le SUV compact",
    "created_at": "2023-09-15T18:21:04.476592889Z",
    "code": "Y",
    "brand": "Tesla"
  }
}
```

# TODO

- [ ] Ajouter des slides sur les index templates ?
- [ ] Ajouter des slides sur les index lifecycle policies ?
- [ ] Ajouter des slides sur les snapshots ?
- [ ] Ajouter des slides sur les rollups ?
- [ ] Ajouter des slides sur les ingest pipelines ?
- [ ] Ajouter des slides sur les machine learning jobs ?