

# Formation Elasticsearch

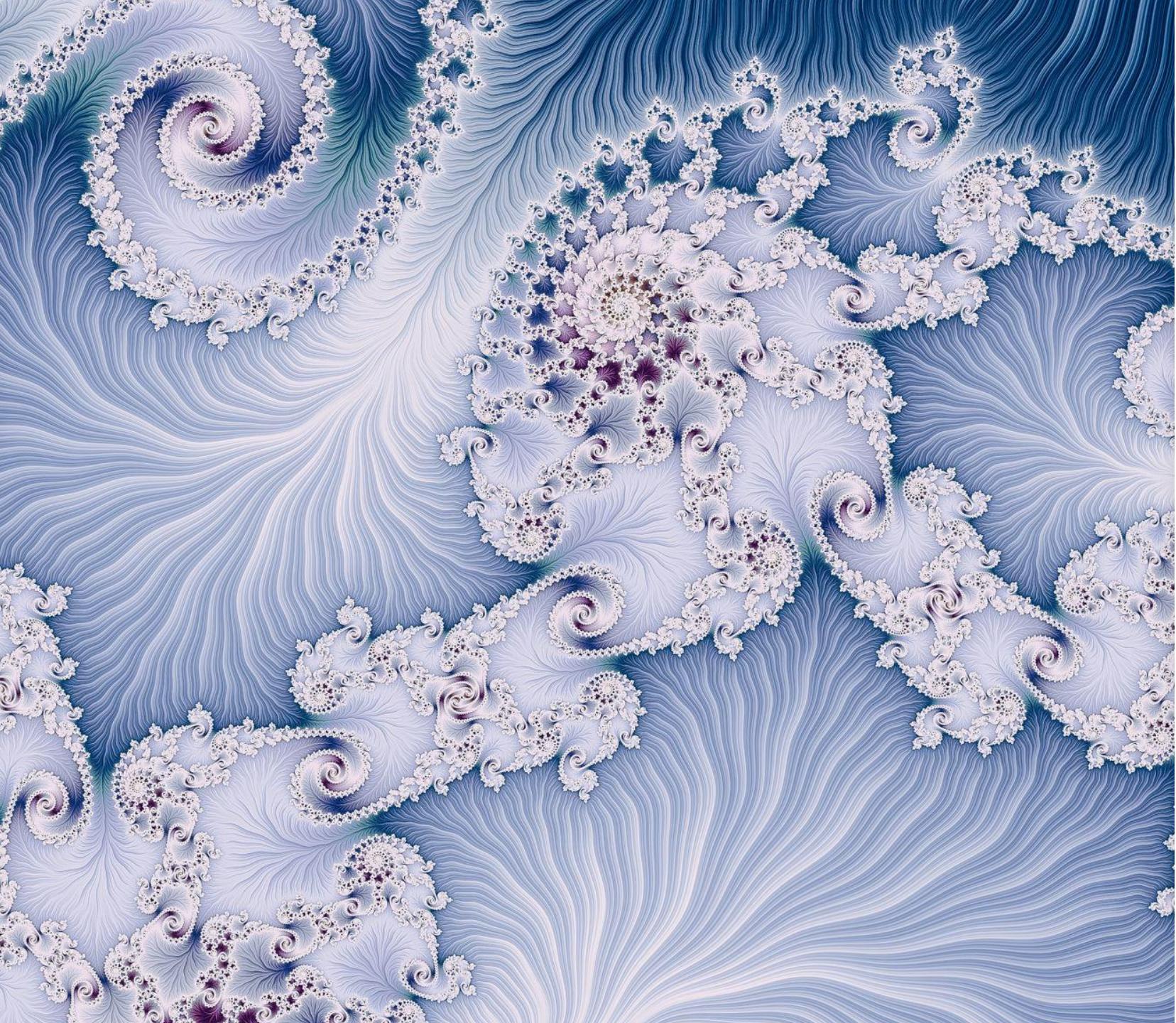
Ali Koudri – ali.koudri@gmail.com

# Plan

---

- Introduction
  - Installation et Configuration
  - Organisation des données
  - Indexation
  - Cycle de vie des documents
  - Scripting
  - Typage
  - Indexation avancée
  - Recherche
  - Agrégations
  - Monitoring et Administration
  - Maintenance et Sécurité
  - Beats
  - Logstash
  - Visualisation
  - Conclusion
- 

# Introduction



# Introduction

---

Elasticsearch est un puissant moteur de recherche et d'analyse open-source conçu pour l'évolutivité horizontale, la fiabilité et les capacités de recherche en temps réel.

Il est construit au-dessus d'Apache Lucene et fait partie de la pile Elastic, qui comprend également Logstash, Kibana et Beats.

Elasticsearch est largement utilisé pour une variété d'applications, y compris la recherche en texte intégral, l'analyse de données de journaux et d'événements, et l'analyse en temps réel.

# Introduction

---

- Dans une base de donnée, la donnée est interprétée pour ce qu'elle est
  - La réalité est plus nuancée: Fautes de frappe, imprécision, ambiguïté
- Dans la plupart des frameworks d'analyse de données, nous raisonnons sur des volumes de données statiques
  - Vs données reçues en temps réel
- La pile Elastic Stack sert justement à combler ces lacunes



# Fonctionnalités

- **Évolutivité et architecture distribuée:**
  - Elasticsearch est conçu pour évoluer horizontalement en distribuant les données sur plusieurs nœuds d'un cluster. Cette architecture lui permet de traiter efficacement de grands volumes de données.
- **Recherche texte intégral:**
  - Elasticsearch offre de puissantes capacités de recherche en texte intégral avec des fonctionnalités telles que l'évaluation de la pertinence, la mise en évidence et la prise en charge de requêtes complexes à l'aide de son DSL (langage spécifique au domaine).
- **Traitement des données en temps réel:**
  - Elasticsearch peut indexer et rechercher des données en temps quasi réel, ce qui en fait un outil idéal pour les applications nécessitant des informations actualisées.

# Fonctionnalités

- **API RESTful:**
  - Elle offre une API RESTful qui facilite l'interaction avec Elasticsearch à l'aide de requêtes HTTP. Cette API prend en charge des opérations telles que l'indexation, la recherche et la gestion des clusters.
- **Documents JSON sans schéma:**
  - Les données d'Elasticsearch sont stockées sous forme de documents JSON, ce qui permet des structures de données flexibles et dynamiques sans schémas prédéfinis.
- **Cadre d'agrégation:**
  - Elasticsearch fournit un cadre d'agrégation puissant pour effectuer des analyses de données complexes et générer des informations à partir de grands ensembles de données.
- **Haute disponibilité:**
  - Elasticsearch prend en charge des fonctionnalités telles que le partage et la réPLICATION automatiques pour garantir la haute disponibilité et la tolérance aux pannes.

# Cas d'usage principaux

- **Applications de recherche:**
  - Elasticsearch est couramment utilisé pour créer des fonctionnalités de recherche dans les sites web et les applications, permettant aux utilisateurs d'effectuer des recherches rapides et précises dans de vastes ensembles de données.
  -
- **Analyse des logs et des données d'événements:**
  - Elasticsearch est souvent utilisé en conjonction avec Logstash (pour l'ingestion de données) et Kibana (pour la visualisation) afin d'analyser les données de logs provenant de diverses sources, aidant ainsi les organisations à surveiller leur infrastructure et leurs applications.

# Cas d'usage principaux

- **Business Intelligence:**
  - Les organisations tirent parti des capacités d'agrégation d'Elasticsearch pour effectuer des tâches de veille stratégique, telles que l'analyse des tendances et la création de rapports.
- 
- **Commerce électronique:**
  - Les plateformes de commerce électronique utilisent Elasticsearch pour offrir aux utilisateurs des recherches de produits rapides, des recommandations personnalisées et une navigation à facettes.

# Comparaison avec le SQL

Caractéristique	Elasticsearch	Base de données SQL
Modèle de données	Orienté document (documents JSON)	Relationnel (tables avec lignes et colonnes)
Schéma	Sans schéma ou mappings dynamiques	Schéma fixe avec structure prédefinie
Langage de requête	DSL d'Elasticsearch (Domain Specific Language)	SQL (Structured Query Language)
Indexation	Optimisé pour la recherche en texte intégral avec index inversé	Indexation B-tree ou similaire pour requêtes structurées

# Comparaison avec le SQL

Caractéristique	Elasticsearch	Base de données SQL
Scalabilité	Scalabilité horizontale avec partitionnement et réPLICATION	Typiquement scalabilité verticale, certains supportent le partitionnement
Capacités de recherche	Recherche en texte intégral avancée, score de pertinence	Correspondances exactes, jointures complexes et transactions
Agrégation	Agrégations en temps réel puissantes	Agrégations via GROUP BY et fonctions d'agrégation
Cohérence des données	Cohérence éventuelle	Cohérence forte avec transactions ACID

# Comparaison avec le SQL

Caractéristique	Elasticsearch	Base de données SQL
Cas d'utilisation	Analyse de logs, analyses en temps réel, applications de recherche	Systèmes transactionnels, applications axées sur l'intégrité des données
Performance	Optimisé pour les charges de travail lourdes en lecture et les requêtes de recherche	Équilibré pour les opérations de lecture et d'écriture
Relations de données	Support limité pour les relations complexes	Support fort pour les relations complexes avec jointures
Format de stockage	Documents JSON	Lignes dans des tables

# Utilisation conjointe avec le SQL

## 1. Optimisation de la recherche

- **Cas d'utilisation:** Améliorer les capacités de recherche pour les applications qui reposent principalement sur des bases de données SQL mais qui nécessitent des fonctions de recherche avancées.
- **Mise en œuvre:** Indexer des tables ou des colonnes spécifiques de la base de données SQL dans Elasticsearch. Utiliser Elasticsearch pour les requêtes de recherche en texte intégral et SQL pour les opérations transactionnelles et les jointures complexes.

## 2. Synchronisation des données

- **Cas d'utilisation:** maintenir les données dans Elasticsearch à jour par rapport aux changements dans la base de données SQL.
- **Mise en œuvre:** Mettre en œuvre un mécanisme de synchronisation des données à l'aide d'outils comme Logstash, de scripts personnalisés ou de connecteurs tiers qui capturent les changements dans la base de données SQL et mettent à jour Elasticsearch en conséquence.

# Utilisation conjointe avec le SQL

## 3. Requête hybride

- **Cas d'utilisation:** Combiner la puissance analytique d'Elasticsearch avec les capacités transactionnelles des bases de données SQL.
- **Mise en œuvre:** Effectuer des recherches en texte intégral dans Elasticsearch et utiliser les résultats pour interroger des données structurées connexes dans la base de données SQL, ou vice versa.

## 4. Rapports et analyses

- **Cas d'utilisation:** Générer des rapports et effectuer des analyses qui nécessitent à la fois des données structurées provenant de bases de données SQL et des données non structurées provenant d'Elasticsearch.
- **Mise en œuvre:** Agréger les données des deux sources dans un outil de reporting ou un tableau de bord unifié qui peut gérer différents types de données.

## 5. Enrichissement des données

- **Cas d'utilisation:** Enrichir les résultats de recherche avec des données supplémentaires stockées dans une base de données SQL.
- **Mise en œuvre:** Utiliser Elasticsearch pour effectuer des recherches initiales, puis récupérer des détails supplémentaires dans la base de données SQL en fonction des résultats de la recherche.

# Histoire

---

- L'histoire de ES remonte à 1999 avec Lucene qui comprenait déjà de nombreuses briques pour la recherche, l'indexation et la visualisation d'information, dont:
  - Mahout, qui a contribué par la suite à Hadoop
  - Nutch, qui a donné lieu par la suite à HDFS
- ElasticSearch est une évolution de Lucene pour le passage à l'échelle
  - Stockage distribué
  - Apprentissage distribué
  - Interface REST
  - Recherche floue

# Licence

---

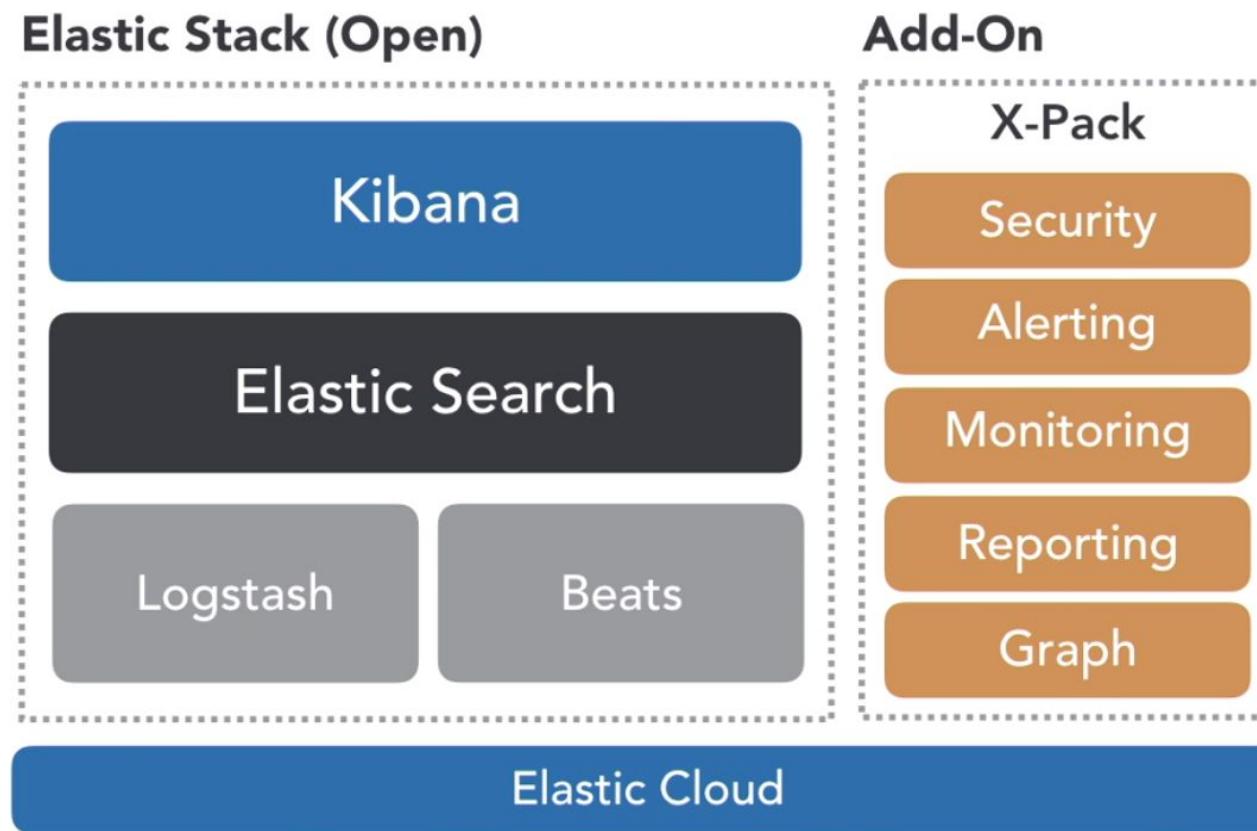
- Le cœur de la plateforme ES est en open-source
  - Large communauté
  - Très documentée
- Beaucoup de composants sont également en open-source et libre de droits
  - Cependant, beaucoup de composants sont payants (X-Pack)
- Le business model de ES repose sur:
  - Des composants custom
  - Du support
  - Des formations
  - Du consulting

# Utilisateurs notables

---

- Mozilla
- Adobe
- Gitlab
- Facebook
- Netflix
- Quora
- Wikimedia
- StackExchange
- ...

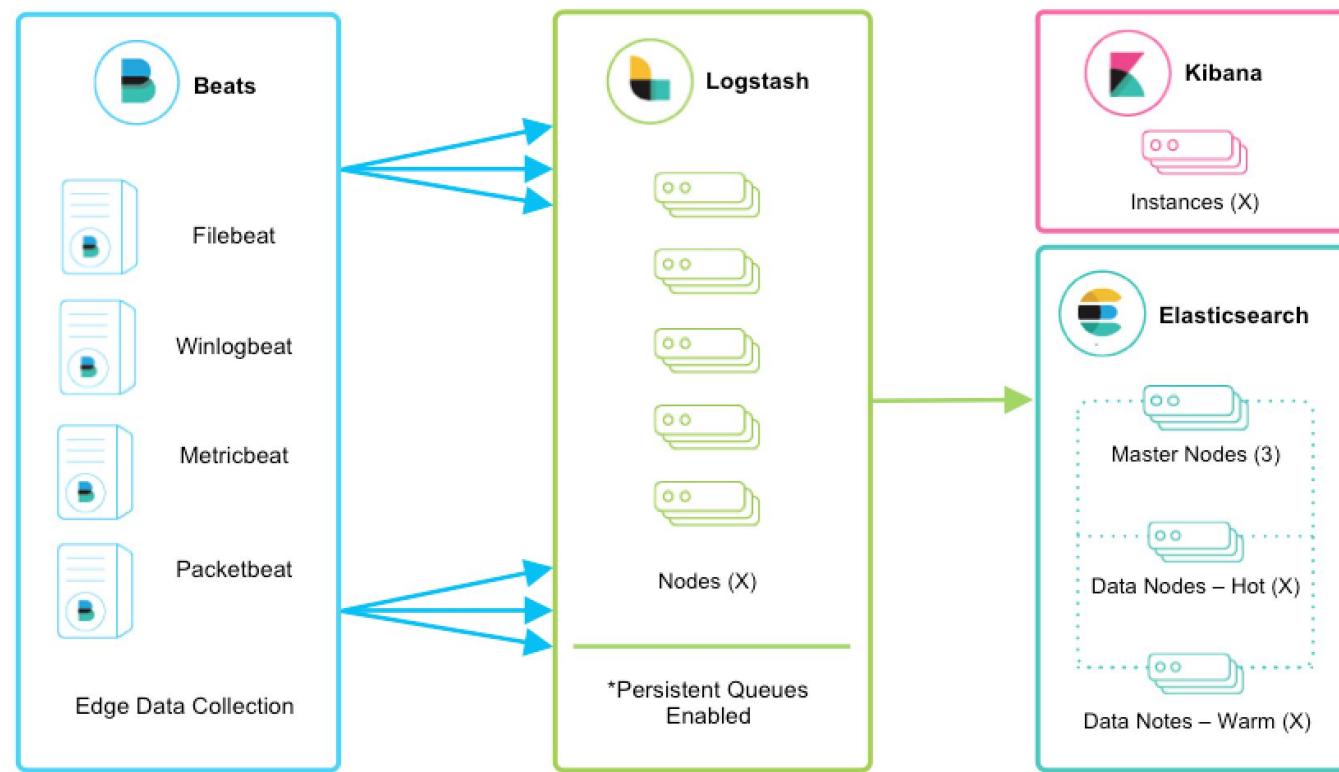
# Architecture Elastic Stack (ex-ELK)



# Architecture Elastic Stack

- **Elasticsearch** est le moteur de recherche et d'analyse distribué qui fournit un stockage de données unifié
- **Elastic Agent, Beats** et **Logstash** en sont les briques qui facilitent la collecte, l'agrégation et l'enrichissement de données
  - Elastic Agent, Beats => Agent
  - Logstash => ETL
- **Kibana** permet l'exploration, l'interaction et la visualisation des données
- Toutes ces briques fournissent de puissantes capacités de recherche, d'analyse et de monitoring pour tous les types de données
- Elastic Stack peut être déployé sur le cloud, avec la prise en charge par Amazon Web Services, Google Cloud et Microsoft Azure (fork OpenSearch)

# Elastic Stack – Fonctionnement Typique



# Exercice 1

Votre site e-commerce repose sur une architecture de microservices (catalogue, paiement, panier, recherche...) déployée sur Kubernetes.

Chaque fonctionnalité applicative est prise en charge par un ou plusieurs microservices indépendants, packagés sous forme de containers et orchestrés par le cluster Kubernetes.

- Vous devez concevoir une architecture d'observabilité avec la stack ELK afin de :
  - Surveiller l'état technique (santé, erreurs, ressources) de chaque microservice et du cluster Kubernetes.
  - Disposer d'une vision transverse métier (parcours d'achat, performance de recherche produit, transactions conclues, abandons panier, etc.).
  - Faciliter l'analyse rétrospective (« quand un incident métier se produit, comment le rattacher à un incident technique ou d'infrastructure ? »).

# Exercice 1 (Cont'd)

---

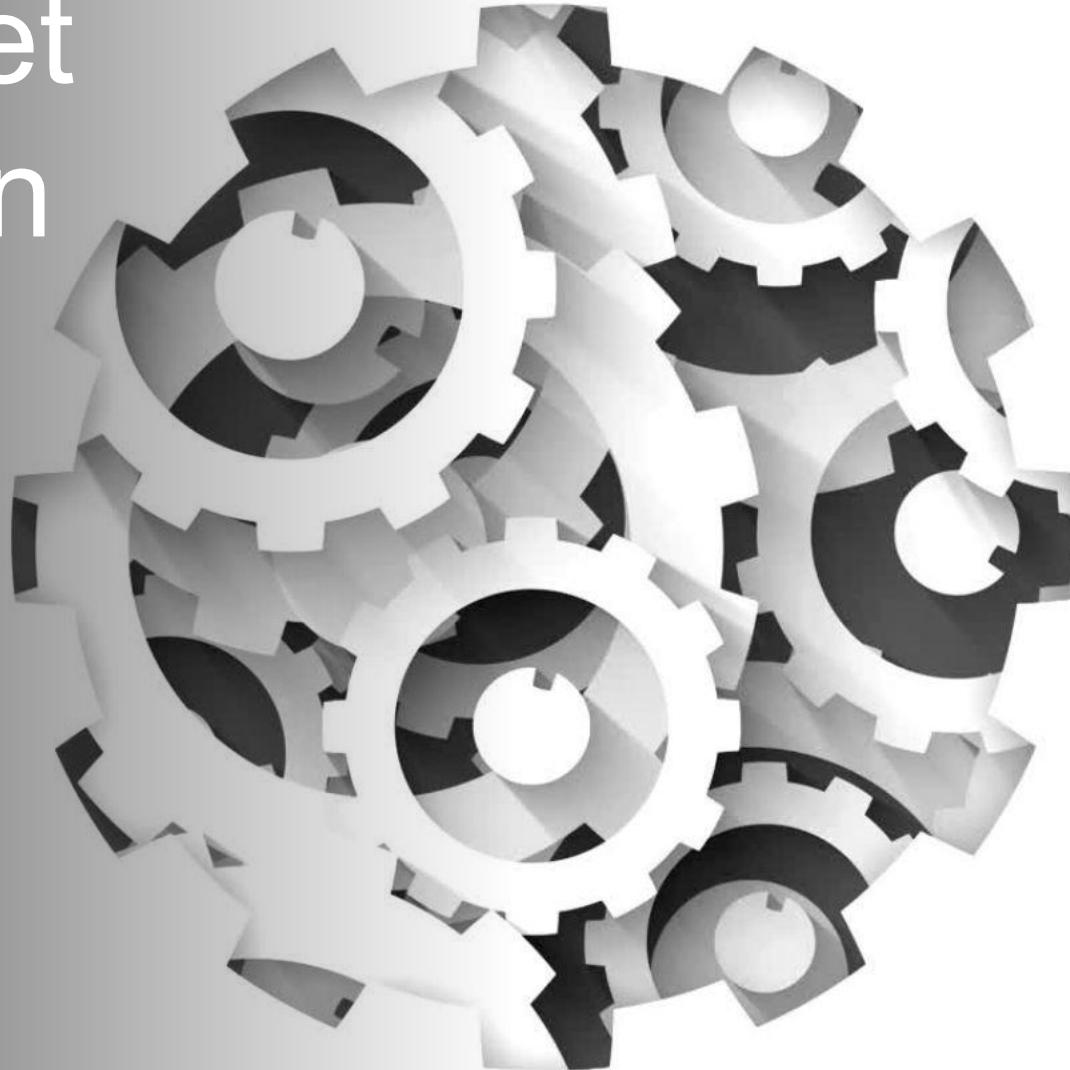
- Quels types de logs collecter ?
- Quelles métriques suivre pour une vue globale et granulaire ?
- Comment organiser la collecte sur Kubernetes ?
- Quelle articulation de la stack ELK ?
- Quels indicateurs métiers ET techniques faut-il corrélérer pour le pilotage applicatif ?

# Exercice 1 (Cont'd)

Rédigez (structure de fiche ou schéma de flux) :

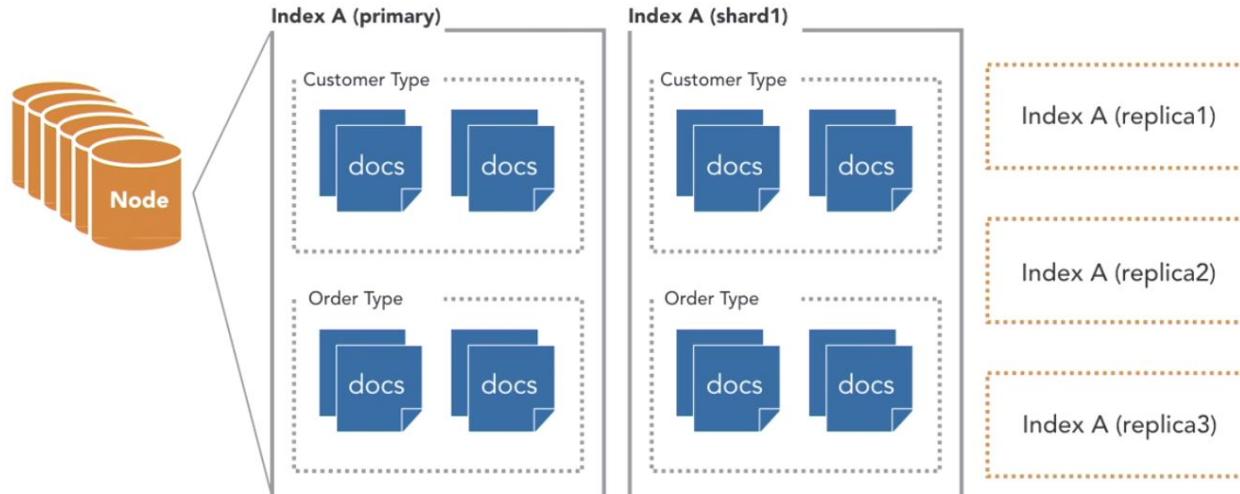
- L'architecture de monitoring cible (rôles de chaque brique ELK, points d'intégration dans Kubernetes, labels/méta-données à collecter pour la corrélation).
- La classification des logs/métriques/traces collectés (où, comment, pourquoi).
- Les dashboards et alertings critiques à réaliser pour à la fois la supervision technique quotidienne et le pilotage métier proactif.

# Installation et configuration



# Architecture ES

- ES a été pensé pour être distribué
- Un cluster ES est constitué de nœuds physiques
- Chaque nœud constitue un point de stockage fournissant des services d'indexation
- Un index représente une collection de documents partageant une même structure (type)
- Les documents d'un index sont stockés au format JSON – Les types sont inférés
- Un index peut être distribué sur plusieurs nœuds (shard) avec différentes répliques (replica)



# Prérequis

---

**Java (JVM)** : Elasticsearch nécessite une JVM, incluse par défaut dans la plupart des images ou packages récents.

**Ressources** : prévoir au minimum 4 Go de RAM, plusieurs vCPU, et un stockage rapide (SSD conseillé, surtout pour Elasticsearch).

**Docker** : pour la version conteneurisée, installez Docker et Docker Compose sur votre machine ou VM.

# Installation - Binaires

- Téléchargez et installez chaque composant depuis le site officiel Elastic.
  - <https://www.elastic.co/downloads/elasticsearch>
  - <https://www.elastic.co/downloads/beats>
  - <https://www.elastic.co/fr/downloads/logstash>
  - <https://www.elastic.co/fr/downloads/kibana>
- Démarrez chaque service manuellement et gérez la configuration via fichiers .yml spécifiques (elasticsearch.yml, kibana.yml).

# Configuration - Elasticsearch

- Documentation officielle sur la configuration :

<https://www.elastic.co/docs/deploy-manage/deploy/self-managed/configure-elasticsearch>

- Guide détaillé des paramètres :

<https://www.elastic.co/fr/elasticsearch/features>

# Configuration - Kibana

- Documentation de configuration générale :

<https://www.elastic.co/docs/deploy-manage/deploy/self-managed/configure-kibana>

- Paramètres de configuration :

<https://www.elastic.co/docs/reference/kibana/configuration-reference/general-settings>

- Pratiques pour la prise en main :

<https://www.jeveuxetredatascientist.fr/kibana-tout-savoir/>

# Configuration - Logstash

---

- Guide officiel de configuration de pipeline :  
<https://www.elastic.co/guide/en/logstash/current/configuration.html>
- Modèles et exemples sur le dépôt Github :  
<https://github.com/elastic/logstash>

# Configuration - Beats

- Documentation sur la configuration de Filebeat et Metricbeat :

<https://www.elastic.co/guide/en/beats/filebeat/current/configuration-filebeat-options.html>

<https://www.elastic.co/guide/en/beats/metricbeat/current/configuration-metricbeat-options.html>

- Introduction et modules :

[https://fr-wiki.ikoula.com/fr/ELK\\_-\\_Agents\\_Beats,\\_commandes\\_utiles](https://fr-wiki.ikoula.com/fr/ELK_-_Agents_Beats,_commandes_utiles)

# Conseils pratiques

- Pensez à ouvrir (en local) les ports essentiels : 9200 (Elasticsearch), 5601 (Kibana), 5044 ou autres pour Logstash selon vos pipelines.
- Utilisez Docker Compose pour centraliser toute la configuration (volumes pour la persistance Elasticsearch, liens réseaux, variables d'environnement).
- Pour des environnements de test, la configuration par défaut fonctionne bien. Pour la production, pensez à l'isolation réseau, la sécurité du cluster, le monitoring, les limites de ressources.
- Après installation, accédez à Kibana sur <http://localhost:5601> pour vérifier l'état de la stack.

# Installation - Docker

---

- curl -o get-docker.sh <https://get.docker.com>
- chmod +x get-docker.sh
- sudo ./get-docker.sh
- sudo usermod -aG docker \$USER
- docker compose up -d

# Exercice 2

---

Exécutez le fichier docker-compose.yml après l'avoir bien analysé

Depuis Kibana > Dev Tools, lancez les commandes suivantes et interprétez le résultat :

- GET /\_cat/health?v (Vérifier le statut green/yellow/red).
- GET /\_cat/nodes?v (Lister les 3 nœuds et leurs rôles).
- GET /\_cat/shards?v (Voir les shards des index internes, ex: .kibana).

# Travaux Préparatoires

Le dossier data contient un certain nombre de données sur lesquelles nous allons travailler.

Pour installer ces données, nous allons utiliser des librairies python :

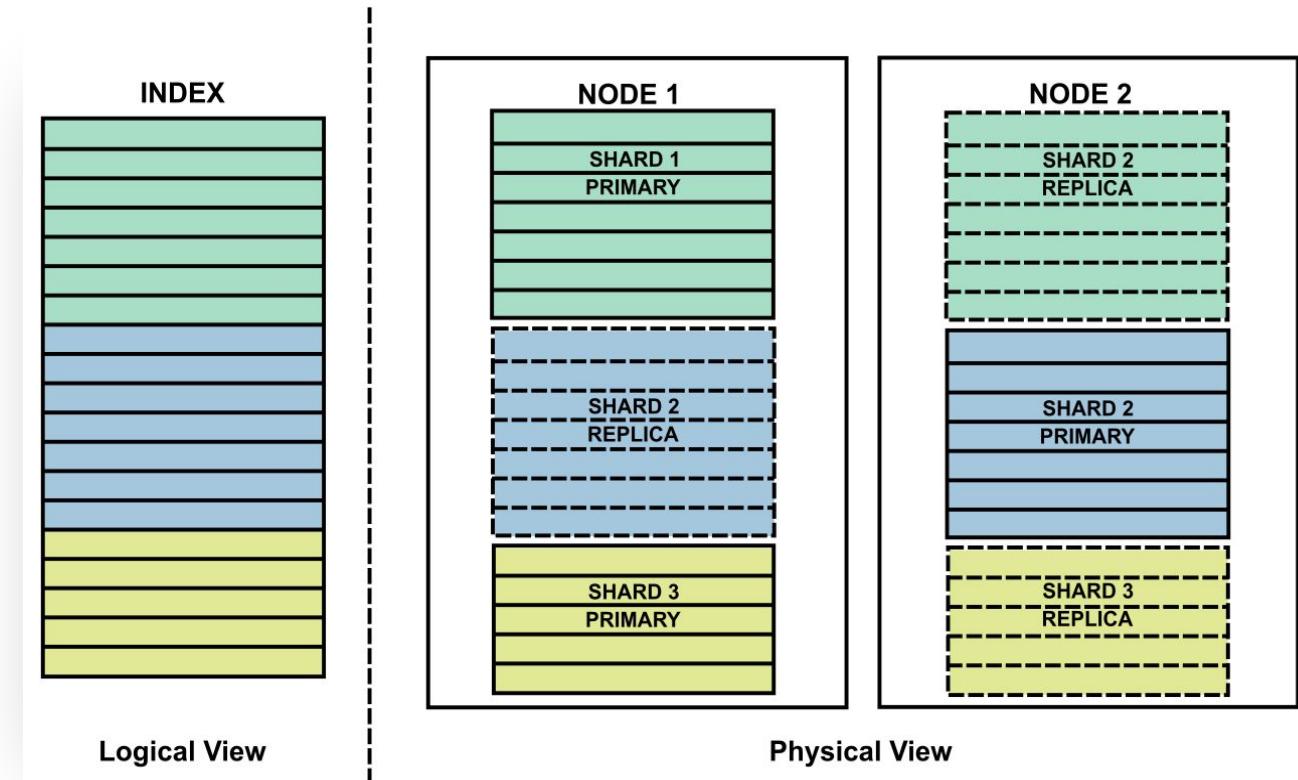
1. Créez un environnement virtuel : `virtualenv -p python3 .venv`
2. Activez cet environnement virtuel : `source .venv/bin/activate`
3. Installez les librairies : `pip install -r requirements.txt`
4. Installez les données
  - a. `python pushBulk.py data/accounts.ndjson accounts` <http://localhost:9200> # fichiers ndjson
  - b. `python pushES.py data/titanic.json titanic` <http://localhost:9200> # fichiers json

# Organisation des données



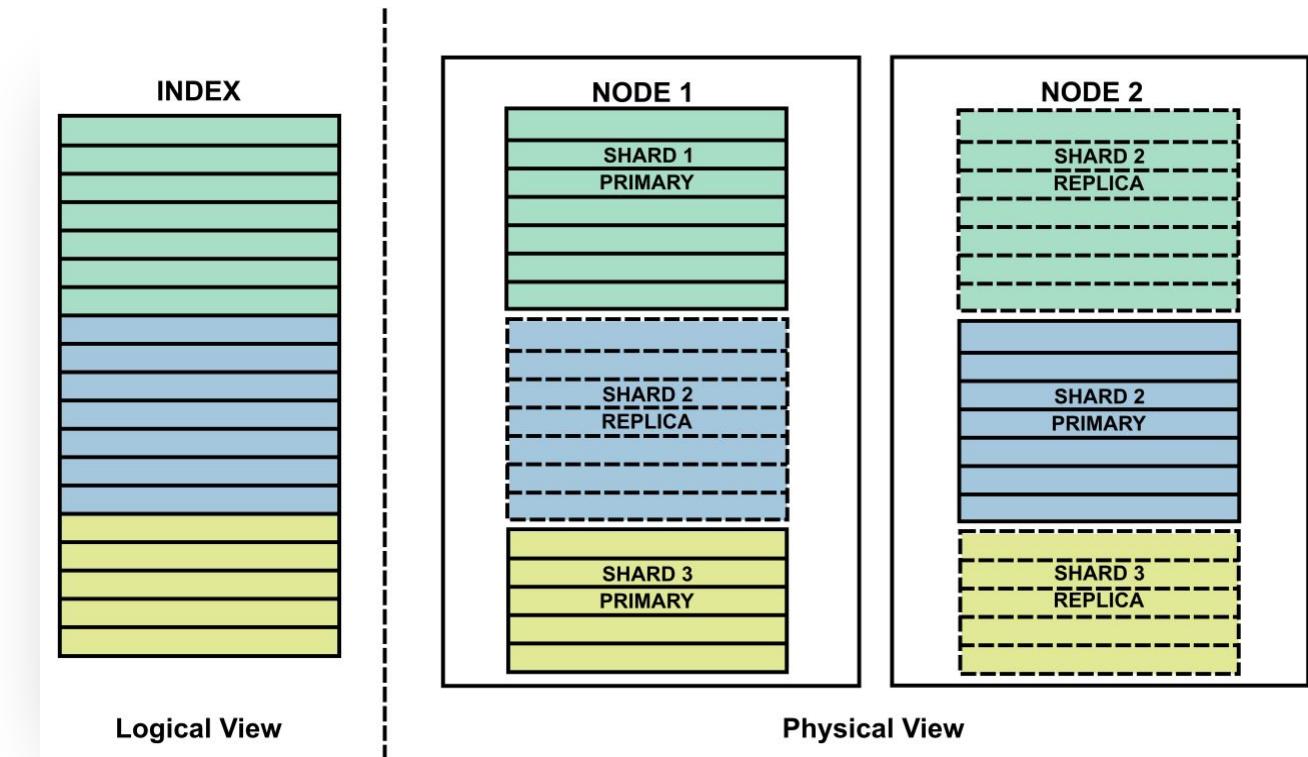
# Stockage de la donnée

- Un index est divisé en shards, et chaque shard est à son tour divisé en segments. Cette hiérarchie permet à Elasticsearch de gérer efficacement les grands ensembles de données.
- Lorsque les documents sont indexés, ils sont d'abord ajoutés à une mémoire tampon associée à un shard. Lorsque la mémoire tampon atteint une certaine taille ou après un intervalle spécifié, les données sont transférées sur le disque sous la forme d'un nouveau segment au sein de ce shard.



# Stockage de la donnée

- Au cours des recherches, Elasticsearch interroge indépendamment chaque shard d'un index. Au sein de chaque shard, l'opération de recherche implique l'interrogation de tous ses segments et l'agrégation des résultats.
- Pour maintenir les performances, Elasticsearch fusionne périodiquement des segments plus petits dans des segments plus grands au sein d'un shard. Ce processus réduit le nombre de segments, ce qui optimise à la fois la vitesse de recherche et l'efficacité du stockage.



# Document: Unité d'information de base

---

Dans Elasticsearch, un **document** est l'unité de base de l'information qui peut être indexée.

Chaque document est un **objet JSON**, un format structuré composé de champs et de valeurs.

Ces documents sont stockés dans un index et peuvent représenter n'importe quel type de données, comme un article de blog, un produit dans un catalogue de commerce électronique ou une entrée de journal.

# Caractéristiques d'un document

- **Sans schéma:**
  - Contrairement aux bases de données relationnelles traditionnelles qui nécessitent des schémas prédéfinis, Elasticsearch vous permet d'indexer des documents sans spécifier leur structure à l'avance. Cette flexibilité facilite le traitement des données dynamiques ou semi-structurées.
- **Format JSON:**
  - Les documents sont stockés au format JSON, qui est largement utilisé pour l'échange de données en raison de sa simplicité et de sa lisibilité. Ce format prend en charge différents types de données, notamment les chaînes de caractères, les nombres, les dates, les tableaux et les objets imbriqués.
- **Autonome:**
  - Chaque document contient toutes les informations nécessaires sur une entité ou un enregistrement, ce qui le rend autonome. Cela signifie que les données connexes sont généralement stockées dans le même document au lieu d'être réparties dans plusieurs tables ou collections.

# Caractéristiques d'un document

- **Indexation et recherche:**
  - Lorsqu'un document est indexé dans Elasticsearch, ses champs sont analysés et stockés dans un index inversé, ce qui permet d'effectuer des recherches en texte intégral rapides et efficaces. Vous pouvez effectuer des requêtes complexes sur ces documents en utilisant le DSL de requête d'Elasticsearch.
- **Versionnement et mises à jour:**
  - Elasticsearch prend en charge le versionnage des documents, ce qui vous permet de suivre les modifications au fil du temps et de gérer efficacement les mises à jour. Lorsqu'un document est mis à jour, Elasticsearch crée une nouvelle version du document plutôt que de modifier la version existante.

# Avantages du document

- **Flexibilité:** La capacité de stocker des structures de données diverses et complexes sans schémas prédéfinis permet un développement et une adaptation rapides à l'évolution des besoins.
- **Évolutivité:** Le stockage orienté document s'aligne bien avec l'architecture distribuée d'Elasticsearch, ce qui permet de stocker et d'extraire efficacement de grands volumes de données sur plusieurs nœuds.
- **Capacités de recherche riches:** L'approche orientée document facilite les fonctions de recherche avancées telles que la recherche en texte intégral, le filtrage et les agrégations.

# Documents : Actions

- **Indexation:** Cette action est utilisée pour ajouter un nouveau document à un index ou mettre à jour un document existant s'il existe déjà. Lorsque vous indexez un document, Elasticsearch analyse ses champs et le stocke dans un index inversé pour une recherche efficace.
- **Récupération:** Vous pouvez récupérer un document spécifique à l'aide de son identifiant unique (ID). Cette action renvoie le document avec ses métadonnées, telles que l'index et le type.
- **Mise à jour:** Cette action vous permet de modifier un document existant. Vous pouvez mettre à jour des champs spécifiques sans réindexer l'ensemble du document. Elasticsearch prend en charge les mises à jour partielles à l'aide de scripts ou en fournissant un document partiel.

# Documents: Actions

- **Suppression:** Cette action permet de supprimer un document d'un index à l'aide de son identifiant. Une fois supprimé, le document n'est plus consultable.
- **Opérations groupées (bulk):** Elasticsearch propose une API qui vous permet d'effectuer plusieurs opérations d'indexation, de mise à jour ou de suppression en une seule requête. Cela permet d'améliorer les performances lorsque l'on traite de gros volumes de données.
- **Recherche de documents**
  - La recherche dans Elasticsearch consiste à interroger un index pour récupérer les documents qui correspondent à des critères spécifiés.
  - Elasticsearch s'appuie sur un index inversé pour trouver rapidement les documents pertinents en fonction des termes de la requête, ce qui permet une recherche en texte intégral rapide et efficace.
  - Les résultats sont classés par ordre de pertinence, ce qui permet aux utilisateurs d'accéder d'abord aux informations les plus pertinentes.

# Document et métadonnées

- Dans ES, chaque donnée est stockée sur la forme d'un document
  - Équivalent à une ligne dans une base de données
- Un document est constitué de champs
  - Équivalent à une colonne dans une base de données
- Les données sont stockées avec leurs métadonnées

```
{  
  "pclass": 1,  
  "survived": true,  
  "sex": "female",  
  "age": 29  
}
```



```
{  
  "_index": "titanic",  
  "_type": "_doc",  
  "_id": "1",  
  "_version": 1,  
  "_seq_no": 1,  
  "_primary_term": 1,  
  "found": true,  
  "_source": {  
    "pclass": 1,  
    "survived": true,  
    "sex": "female",  
    "age": 29  
  }  
}
```

# Description des métadonnées

- **\_index**
  - **Description:** Le nom de l'index dans lequel le document est stocké.
  - **Objectif:** Identifie la collection de documents à laquelle ce document appartient. Un index est similaire à une base de données dans les bases de données relationnelles.
- **\_id**
  - **Description:** Identifiant unique du document dans son index.
  - **Objet:** Utilisé pour récupérer, mettre à jour ou supprimer un document spécifique. S'il n'est pas fourni lors de l'indexation, Elasticsearch génère automatiquement un identifiant.
- **\_type**
  - **Description:** Le type de document (obsolète dans Elasticsearch 7.x et supprimé dans 8.x).
  - **Objectif:** Historiquement utilisé pour classer les documents dans un index. Cependant, avec la suppression des types dans les versions plus récentes, chaque index ne contient plus qu'un seul type de document.

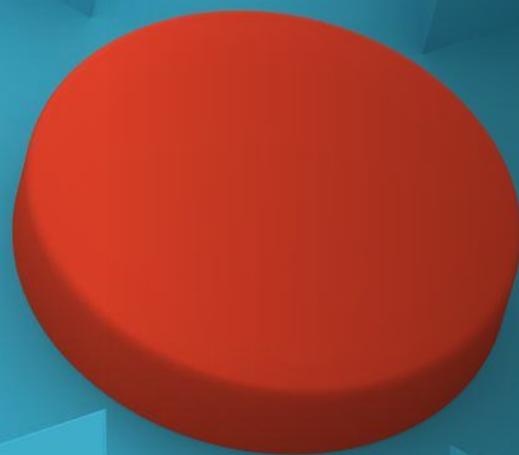
```
{  
  "_index": "titanic",  
  "_type": "_doc",  
  "_id": "1",  
  "_version": 1,  
  "_seq_no": 1,  
  "_primary_term": 1,  
  "found": true,  
  "_source": {  
    "pclass": 1,  
    "survived": true,  
    "sex": "female",  
    "age": 29  
  }  
}
```

# Description des métadonnées

- **\_version**
  - **Description:** Le numéro de version du document.
  - **Objectif:** Suivre les modifications apportées à un document au fil du temps. Chaque mise à jour d'un document incrémente son numéro de version, ce qui permet de gérer les mises à jour simultanées et de garantir la cohérence des données.
- **\_score**
  - **Description:** Score de pertinence calculé lors des requêtes de recherche.
  - **Objectif:** Indique dans quelle mesure un document correspond à une requête de recherche donnée. Les scores élevés indiquent de meilleures correspondances.
- **\_source**
  - **Description:** Le corps JSON original du document tel qu'il a été indexé.
  - **Objectif:** Stocke le contenu réel des données du document. Il peut être récupéré ou exclu des résultats de recherche en fonction des exigences de la requête.

```
{  
  "_index": "titanic",  
  "_type": "_doc",  
  "_id": "1",  
  "_version": 1,  
  "_seq_no": 1,  
  "_primary_term": 1,  
  "found": true,  
  "_source": {  
    "pclass": 1,  
    "survived": true,  
    "sex": "female",  
    "age": 29  
  }  
}
```

# Indexation



# Index : présentation

---

Dans Elasticsearch, un index est une structure de données fondamentale qui organise et stocke des documents de manière à permettre une recherche et une récupération efficaces.

Il est similaire à une table dans les systèmes de bases de données relationnelles, mais il est spécifiquement optimisé pour les opérations de recherche.

# Index : caractéristiques

## **Collection de documents:**

- Un index contient une collection de documents, chacun étant représenté par un objet JSON. Ces documents peuvent représenter différents types de données, tels que des journaux, des articles ou des produits.

## **Sans schéma:**

- Bien qu'Elasticsearch soit sans schéma, ce qui signifie que vous n'avez pas besoin de définir la structure de vos documents à l'avance, vous pouvez définir des mappings pour spécifier comment les champs doivent être indexés et recherchés.

## **Sharding et réPLICATION:**

- Un index est divisé en unités plus petites appelées « shards », qui sont distribuées sur les nœuds d'un cluster. Cela permet à Elasticsearch d'évoluer horizontalement et de traiter efficacement de grands volumes de données.
- Chaque nœud peut avoir des répliques pour assurer une haute disponibilité et une tolérance aux pannes.

# Index : caractéristiques

## **Index inversé:**

- Le cœur d'un index Elasticsearch est l'index inversé, qui associe les termes aux documents qui les contiennent. Cette structure permet des recherches rapides en texte intégral en identifiant rapidement les documents pertinents en fonction des termes de la requête.

## **Gestion de l'index:**

- Vous pouvez créer, supprimer et gérer des index à l'aide de l'API RESTful d'Elasticsearch. Les index peuvent également être configurés avec des paramètres qui contrôlent des aspects tels que l'allocation et l'analyse des shards.

## **Analyse :**

- Les indices permettent des agrégations complexes et des tâches d'analyse de données.

# Index prédefinis

## **\_mappings :**

- Les correspondances définissent la structure des documents dans un index, en spécifiant comment les champs doivent être indexés et quels types de données ils doivent avoir (par exemple, texte, mot-clé, date, nombre entier).
- En définissant les mappings à l'avance, vous pouvez contrôler la façon dont Elasticsearch analyse et stocke vos données, ce qui peut améliorer la pertinence et la performance des recherches.

## **\_settings:**

- Les paramètres de l'index comprennent des configurations telles que le nombre de shards et de répliques, l'intervalle de rafraîchissement et les paramètres d'analyse (par exemple, les analyseurs personnalisés).
- La prédéfinition de ces paramètres vous permet d'adapter l'index à vos besoins spécifiques, en optimisant des facteurs tels que la vitesse de recherche ou la tolérance aux pannes.

## **\_templates:**

- Les modèles d'index sont une fonctionnalité puissante d'Elasticsearch qui vous permet de définir des paramètres et des mappings qui sont automatiquement appliqués aux nouveaux index qui correspondent à un modèle spécifié.
- Les modèles sont particulièrement utiles pour gérer les index dans des environnements où de nouveaux index sont créés fréquemment, tels que les systèmes de gestion de journaux où les index peuvent être créés quotidiennement.

# Index de gestion

- **\_cat/indices**: Liste tous les index du cluster avec leur statut, nombre de documents, leur taille et d'autres détails.
- **\_cat/shards**: Fournit des informations sur les nuages de la grappe, y compris leur répartition entre les nœuds.
- **\_cat/nodes**: Affiche des détails sur chaque nœud du cluster, tels que le nom, l'adresse IP, les rôles et l'utilisation des ressources.
- **\_cat/health**: Donne un aperçu de l'état de santé de la grappe, y compris le nombre de nœuds et d'index, les cartes actives et les tâches en attente.
- **\_cluster/health**: Permet de vérifier rapidement l'état de santé général d'un cluster Elasticsearch. Il renvoie un état qui indique si le cluster fonctionne normalement ou s'il y a des problèmes qui nécessitent une attention particulière.

# Cycle de vie des documents



# Principes de fonctionnement

Les interactions avec ElasticSearch passent par des interfaces REST

Les commandes respectent la syntaxe du protocol HTTP

- <Verbe REST> <Index> <ID Service> <ID Document>
- Le verbe représente une des opérations CRUD du protocole HTTP
  - GET, POST, PUT, DELETE

L'identifiant du service désigne le service relatif à la gestion de l'index (stockage, recherche, ...)

- Exemple: [GET \\_cat/indices?v](#)

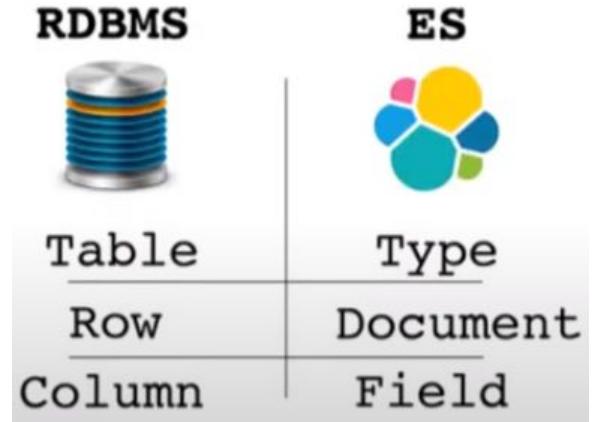
# Principes de fonctionnement

---

Dans la philosophie de ES, **tout est document**:

- Les données
- La configuration
- Les requêtes et les résultats
- ...

# TP



## Ajout d'un index

- Exécutez deux fois la commande ci-dessous avec des caractéristiques différentes – Qu'observez-vous ?

```
POST /persons/_doc/1
```

```
{
```

```
    "firstname": "John",
```

```
    "lastname": "Doe",
```

```
    "age": 32
```

```
}
```

## Obtention d'un document

- GET /persons/\_doc/1

# Quelques précisions

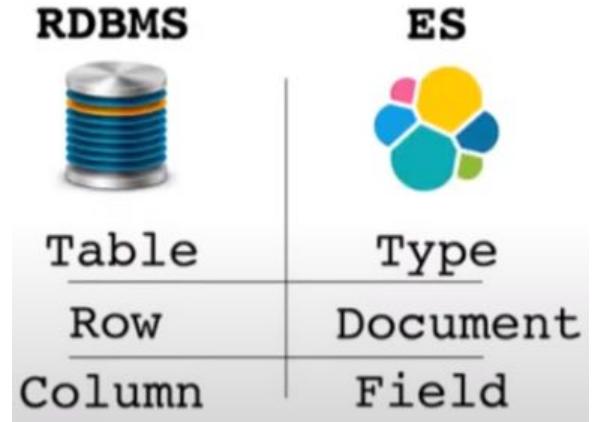
La configuration par défaut (1P + 1R) est le minimum requis pour la haute disponibilité.

- Si le nœud 1 tombe en panne, le nœud 2 promeut sa réplique R0 en tant que nouveau primaire P0 et votre cluster continue de fonctionner.
- Le nœud 3 sert alors à accueillir une nouvelle réplique qui sera créée.

Si votre objectif est d'avoir 3 copies de la donnée (1 primaire + 2 répliques) pour maximiser la tolérance aux pannes (vous pouvez perdre 2 nœuds), vous devez le spécifier.

```
PUT /persons/_settings
{
    "number_of_replicas": 2
}
```

# TP



## Suppression d'un index

- DELETE /persons

Remarque: À partir de la version 6

- un index => un type
- Pour des raisons de performance

# TP

---

- Exécutez la commande suivante: *HEAD /persons/\_doc/1*
- Exécutez la même commande avec un index inexistant
- Exécutez la commande suivante: *GET /persons*
- Ajoutez une personne avec une information supplémentaire (le sexe par exemple) et relancez la commande précédente. Qu'observez-vous ?
- Exécutez la commande *GET persons/\_search*

# Mise à jour

Les documents sont immutables

- La mise à jour crée en fait une copie modifiée
- La mise à jour des données peut se faire par l'appel à un script

**Question:** Pouvez-vous expliquer la raison de la syntaxe du second script ?

```
POST /persons/_update/1
{
  "doc": {
    "firstName": "Emmanuel"
  }
}
```

```
POST /persons/_update/1
{
  "script" : {
    "source": "ctx._source.age += params.num",
    "params": {
      "num": 2
    }
  }
}
```

# Mise à jour - Précisions

Le moteur d'Elasticsearch ne veut pas interpréter le code source de votre script à chaque fois.

Pour être rapide, il compile votre script painless ("source") en un code machine beaucoup plus rapide et le met en cache pour le réutiliser:

- Elasticsearch voit le script `"ctx._source.age += params.num"`. Il le compile une seule fois et le met en cache.
- Ce script est maintenant un "template" ou une "fonction" qui attend un paramètre.
- Quand vousappelez à nouveau avec "params": { "num": 5 }, Elasticsearch reconnaît le script, récupère la version compilée dans son cache, et lui passe simplement la nouvelle valeur 5.
- Vous pouvez faire 1 million de mises à jour avec 1 million de valeurs différentes, Elasticsearch n'utilisera qu'un seul script compilé.

# Mise à jour - Upsert

Il existe un mix entre l'insertion d'un document et sa mise à jour: **upsert**

La structure est la même que pour une mise à jour avec un champs supplémentaire contenant la structure à ajouter si elle n'existe pas

Si la donnée existe déjà, seul le bloc *script* est exécuté, sinon seul le bloc *upsert* est exécuté

```
POST /persons/_update/1
{
  "script" : {
    "source": "ctx._source.age += params.num",
    "params": {
      "num": 2
    }
  },
  "upsert": {
    "name": "Jane Doe",
    "age": 25,
    "created": "now"
  }
}
```

# Mise à jour par lot

ElasticSearch fournit les constructions pour modifier un ensemble de documents

Equivalent à un UPDATE WHERE en SQL

```
POST /persons/_update_by_query
{
  "script": {
    "source": "ctx._source.age += 1"
  },
  "query": {
    "match_all": {}
  }
}
```

# Suppressions unique et par lot

---

Suppression d'un document. Comme pour l'insertion ou la mise à jour, la suppression d'un document utilise le verbe HTTP de même nom

- DELETE /temperatures/\_doc/1

De la même manière que l'on peut mettre à jour un ensemble de documents, il est possible de supprimer un ensemble de documents répondant à certains critères

- POST /temperatures/\_delete\_by\_query {...}

# Batching

ElasticSearch peut exécuter des commandes en mode batch à travers le service `_bulk`

En laissant vide le champs `index`, vous demandez à Elasticsearch : "Indexe le document suivant et génère-moi un ID unique pour lui."

L'information la plus cruciale que vous pourriez y ajouter est votre propre `_id`.

```
POST /products/_bulk
{ "index": {} }
{ "name": "Espresso Machine", "price": 199, "in_stock": 5 }
{ "index": {} }
{ "name": "Milk Frother", "price": 149, "in_stock": 14 }
```

# Scripting



# Présentation

---

Les scripts dans Elasticsearch vous permettent d'écrire du code personnalisé qui peut être exécuté sur les nœuds du cluster pour effectuer diverses tâches, telles que :

- La transformation de données,
- Le calcul de valeurs personnalisées
- La mise en œuvre d'une logique spécifique lors de l'indexation,
- La mise à jour ou de la recherche de documents.



# Langages

---

Elasticsearch prend en charge plusieurs langages de script, notamment Painless (le langage de script par défaut), Groovy, JavaScript et Python (via des plugins).

Painless est spécialement conçu pour Elasticsearch et offre des performances optimisées et une sécurité accrue par rapport aux autres langages de script.

<https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-scripting-painless.html>

# Usages

**Mise à jour de documents** : Les scripts peuvent être utilisés lors de la mise à jour de documents pour modifier des champs spécifiques ou calculer de nouvelles valeurs basées sur la logique personnalisée.

**Agrégations** : Les scripts permettent de calculer des valeurs personnalisées ou de transformer des données pendant le processus d'agrégation.

**Scoring de pertinence** : Les scripts peuvent être utilisés pour personnaliser le calcul du score de pertinence des documents lors d'une recherche.

**Ingest Pipelines** : Les scripts peuvent être utilisés dans les pipelines d'ingestion pour transformer ou enrichir les données avant leur indexation.

# Scripts : Accès aux données et paramètres

Accès aux données dans les scripts :

- Dans un script, vous pouvez accéder aux données du document en cours en utilisant l'objet **ctx** (contexte) dans Painless ou l'objet doc dans d'autres langages.
- Vous pouvez accéder aux champs du document, effectuer des calculs ou des transformations sur ces champs et assigner de nouvelles valeurs.

Paramètres des scripts :

- Les scripts peuvent accepter des paramètres qui sont passés depuis la requête ou la configuration Elasticsearch.
- Les paramètres permettent de rendre les scripts réutilisables et de les adapter à différents scénarios sans avoir à modifier le code du script.

# Performance

- L'exécution de scripts peut avoir un impact sur les performances d'Elasticsearch, il est donc important d'optimiser les scripts et d'éviter les opérations coûteuses.
- Elasticsearch applique des mesures de sécurité pour limiter l'accès aux ressources système et prévenir l'exécution de code malveillant dans les scripts.

```
POST /mon_index/_update_by_query
{
  "script": {
    "source": "ctx._source.total = ctx._source.champ1 + ctx._source.champ2",
    "lang": "painless"
  },
  "query": {
    "match_all": {}
  }
}
```

# Utilisation avancée

Il est possible, par exemple, utiliser des scripts pour faire des recherches sur des propriétés dérivées

```
GET /_search
{
  "query": {
    "bool": {
      "filter": {
        "script": {
          "script": """
            double amount = doc['amount'].value;
            if (doc['type'].value == 'expense') {
              amount *= -1;
            }
            return amount < 10;
          """
        }
      }
    }
  }
}
```

Typage



# Types de donnée

Comme pour n'importe quel langage de programmation, ES permet de décrire différents types de donnée

- Les types primitifs (booléen, entier, réel, ...)
- Les types complexes (tableau, structure de donnée, ...)
- Les types géographiques (latitude, longitude, aire, ...)
- Types custom (numéro de téléphone, adresse ip, ...)

Différents types peuvent caractériser une chaîne de caractères:

- Keyword => un seul mot
- Text => une phrase

```
PUT /persons/_mapping
{
  "properties" : {
    "cat" : {
      "type": "keyword"
    }
  }
}
```

<https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-types.html>

# Types de donnée

ES infère le type des documents poussés

- [GET /persons/\\_mapping](#)

Il est toutefois possible de forcer le typage des documents

Mais il est impossible de changer le type des champs en route

```
PUT /persons/_mapping
{
  "properties" : {
    "cat" : {
      "type": "keyword"
    }
  }
}
```

# Mapping

Le mapping est aux documents ce que les schémas sont aux bases de données

Le mapping définit la structure d'un document

Le mapping est inféré, mais il peut également être défini au moment de la création de l'index (ou les deux)

- Il est par ailleurs recommandé de définir un mapping de manière explicite en production

GET /titanic/\_mapping

```
{  
  "titanic" : {  
    "mappings" : {  
      "properties" : {  
        "age" : {  
          "type" : "float"  
        },  
        "pclass" : {  
          "type" : "long"  
        },  
        "sex" : {  
          "type" : "text",  
          "fields" : {  
            "keyword" : {  
              "type" : "keyword",  
              "ignore_above" : 256  
            }  
          }  
        },  
        "survived" : {  
          "type" : "long"  
        }  
      }  
    }  
  }  
}
```

# Mapping Explicite

Définit la structure de vos documents en spécifiant les types de données pour chaque champ (par exemple, texte, mot-clé, date, nombre entier).

- Des mapping appropriés permettent d'éviter des problèmes tels que des affectations incorrectes de types de données, qui peuvent entraîner des erreurs ou des requêtes inefficaces.

Contribue à optimiser le stockage en spécifiant les champs qui doivent être indexés ou stockés.

- Les champs qui ne sont pas nécessaires à la recherche peuvent être exclus de l'indexation, ce qui permet de réduire la taille de l'index et d'améliorer les performances.

<https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-params.html>

# Mapping dynamique

Par défaut, Elasticsearch est configuré en mode `dynamic: true`. C'est le "mapping dynamique". Si vous indexez un document avec un nouveau champ, Elasticsearch va :

- Deviner le type de ce nouveau champ (ex: "hello" -> `text`, 5 -> `long`, "2025-01-01" -> `date`).
- Ajouter automatiquement ce nouveau champ au mapping de l'index.

**Inconvénients :**

- **Pollution du mapping** : Une simple faute de frappe ("`usarname`": "toto" au lieu de "`username`") crée un nouveau champ dans votre mapping.
- **Mauvais types** : Si votre premier document a un champ "`code_postal`": "75001", ES va le mapper en `text`. Le prochain document avec "`code_postal- "Mapping explosion" : Si une application envoie des champs dynamiques (ex: "event_id_123": true, "event_id_124": true...), vous pouvez vous retrouver avec des milliers de champs, ce qui dégrade les performances du cluster.`

# Mapping statique

Le mapping strict résout les problèmes en changeant les règles du jeu.

Si un index est configuré en dynamic: 'strict', lorsque vous essayez d'indexer un document contenant un champ inconnu :

- Elasticsearch rejette le document en entier.
- Il vous retourne une erreur de type `strict_dynamic_mapping_exception`.

Pourquoi est-ce une bonne chose ?

- **Contrôle de la Qualité** : Vous force à maintenir un schéma de données propre et défini. C'est le "mode production".
- **Détection d'Erreurs** : Vous alerte immédiatement si une application envoie des données mal formées ou inattendues.
- **Performance** : Empêche la "mapping explosion" et garantit que votre index reste optimisé.

# Mapping statique - Exemple

## Exercices:

- Définir le mapping des données contenues dans le fichier data/temperature.json
- Définir également le nombre de shards et de répliques à 2 au passage

```
PUT /persons
{
  "mappings": {
    "dynamic": "strict",
    "properties": {
      "firstName": { "type": "text" },
      "lastName": { "type": "text" },
      "age": { "type": "integer" }
    }
  }
}
```

# Mapping - Migration

Une fois le mapping défini, il est quasiment impossible de le changer

- À quelques exceptions près, comme l'ajout de contraintes d'intégrité

Si toutefois un changement dans le mapping est nécessaire, il faut alors réindexer l'ensemble des documents

- Avec la possibilité de filtrer les données (query) d'appliquer un traitement dans la réindexation (script)

```
POST /_reindex
{
  "source": {
    "index": "titanic"
  },
  "dest": {
    "index": "titanic_new"
  }
}
```

# Templates

- ES permet de définir un même mapping pour un ensemble d'index => template
- Un template permet de partager un même schéma pour un ensemble d'index sur la base d'un pattern de nommage (ex: access-logs-\*)
- Ils sont appliqués automatiquement à la création d'index correspondant au pattern
- Les templates sont éditables par le service /\_templates

**Exercice:** Définissez un template afin d'intégrer de nouveaux index de type "température"

```
PUT /_template/access-logs
{
  "index_patterns": ["access-logs-*"],
  "settings": {
    "number_of_shards": 2,
    "index.mapping.coerce": false
  },
  "mappings": {
    "properties": {
      "@timestamp": {
        "type": "date"
      },
      "url.original": {
        "type": "keyword"
      },
      "http.request.referrer": {
        "type": "keyword"
      },
      "http.response.status_code": {
        "type": "long"
      }
    }
  }
}
```

# Indexation avancée



# Indexation des documents

ES utilise une indexation inversée

- ES crée et met à jour un index des mots clefs à chaque fois qu'un document est chargé / modifié

Il référence tous les documents associés à ces mots clefs avec un score de pertinence, basé essentiellement sur le nombre d'occurrence du mot clef dans le document vs dans l'ensemble des documents (TF-IDF)

Il permet la recherche partielle, tenant compte:

- Des fautes d'orthographe
- De la proximité sémantique
- Des imprécisions

# Index inversé

---

Un index inversé est une correspondance entre des termes et les documents qui contiennent ces termes. Il fonctionne de la même manière qu'un index à la fin d'un livre, où l'on peut chercher un mot et trouver une liste de pages où ce mot apparaît.

Lorsqu'un document est indexé, ses champs de texte sont décomposés en termes individuels (jetons) par un processus appelé « tokenisation ». Ce processus consiste à supprimer la ponctuation, à convertir le texte en minuscules et à le diviser en mots ou en phrases.

# Index inversé

---

Chaque terme unique est ajouté à l'index inversé avec les références aux documents dans lesquels le terme apparaît. Ces références comprennent généralement l'identifiant du document et peuvent également inclure des informations supplémentaires telles que la fréquence du terme ou sa position dans le document.

Lorsqu'une requête est exécutée, Elasticsearch utilise l'index inversé pour identifier rapidement les documents qui contiennent les termes de la requête. Cela permet d'extraire rapidement les documents pertinents sans avoir à analyser chaque document de l'index.

# Index inversé: Avantages

**Rapidité:** l'index inversé permet à Elasticsearch d'effectuer des recherches très rapidement, même sur des ensembles de données volumineux.

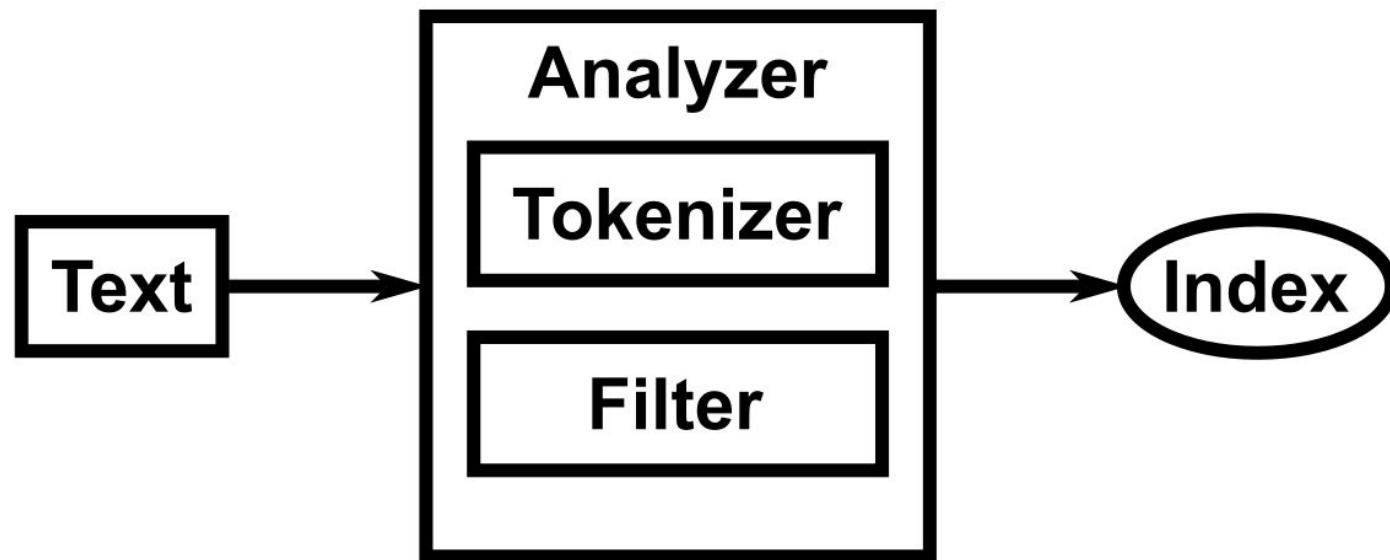
**Pertinence:** En stockant des informations supplémentaires telles que la fréquence et la position des termes, Elasticsearch peut calculer des scores de pertinence pour classer efficacement les résultats de recherche.

**Évolutivité:** La structure prend en charge le stockage et la récupération distribués, ce qui lui permet de traiter de grandes quantités de données sur plusieurs nœuds.

# Processus d'indexation

Lors du stockage, les documents sont analysés pour l'indexation

Chaque étape peut être personnalisée



# Analyseurs : Activités

## 1 – Tokenisation

- La tokenisation est le processus de décomposition d'une chaîne de texte en termes individuels, appelés « tokens ». Cette opération s'effectue généralement en divisant le texte au niveau des espaces blancs ou des signes de ponctuation.
- Elle permet à Elasticsearch d'indexer chaque terme séparément, ce qui offre des capacités de recherche en texte intégral efficaces.

## 2 - Mise en minuscules (Filtrage)

- Convertit tous les caractères du texte en minuscules.
- Permet de s'assurer que les recherches ne tiennent pas compte des majuscules et des minuscules, et que les résultats correspondent au texte saisi, qu'il soit en majuscules ou en minuscules.

# Analyseurs : Activités

## 3 - Suppression des mots vides (Filtrage)

- Élimine les mots courants (par exemple, « et », « le », « est ») qui ne sont souvent pas utiles pour la recherche.
- Réduit la taille de l'index et se concentrer sur des termes plus significatifs, améliorant ainsi l'efficacité et la pertinence de la recherche.

## 4 - Stemming (Filtrage)

- Réduire les mots à leur forme racine (par exemple, « running » devient « run »).
- Permet à différentes formes d'un mot de correspondre à un seul terme indexé, ce qui améliore la souplesse de la recherche et la mémorisation.

# Analyseurs : Activités

---

## 5 - Traitement des synonymes (Filtrage)

- Associe des synonymes à un terme commun (par exemple, « voiture » et « automobile »).
- Élargir les résultats de la recherche en incluant les documents qui contiennent des synonymes des termes recherchés.

## 6 - Suppression de la ponctuation (Filtrage)

- Supprime les signes de ponctuation du texte.
- Empêche la ponctuation d'affecter le processus d'indexation et de recherche, garantissant ainsi la cohérence.

# Exemple

```
POST /_analyze
{
  "text": "Hello 22, Do you enjoy LEARNING elasticsearch ?",
  "analyzer": "standard"
}
```



**Question:** que remarquez-vous ?

Ajoutez des virgules et des espaces, et observez le résultat de l'analyse

```
{
  "tokens": [
    {
      "token": "hello",
      "start_offset": 0,
      "end_offset": 5,
      "type": "<ALPHANUM>",
      "position": 0
    },
    {
      "token": "22",
      "start_offset": 6,
      "end_offset": 8,
      "type": "<NUM>",
      "position": 1
    },
    {
      "token": "do",
      "start_offset": 10,
      "end_offset": 12,
      "type": "<ALPHANUM>",
      "position": 2
    },
    ...
    {
      "token": "learning",
      "start_offset": 23,
      "end_offset": 31,
      "type": "<ALPHANUM>",
      "position": 5
    }
  ]
}
```

# Recherche et interprétation

ES permet d'associer du sens aux documents afin d'affiner les résultats des requêtes

Il peut adopter différentes stratégies de parsing des documents

```
GET /titanic/_analyze
{
  "tokenizer": "standard",
  "text": "akoudri@cenotelie.fr on https://cenotelie.fr"
}
```

```
GET /titanic/_analyze
{
  "tokenizer": "uax_url_email",
  "text": "akoudri@cenotelie.fr on https://cenotelie.fr"
}
```

# Analyseurs

Nous avons vu sur les planches précédentes le comportement de l'analyseur standard

- Génération d'un tableau de tokens
- La casse de chaque token est passée en minuscule
- Les mots "vides" sont supprimés
- ...

Il existe d'autres analyseurs prenant en compte les caractéristiques des informations stockées

- <https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html>
- **Exercice:** tester l'analyseur pour la langue française

# Analyseurs prédéfinis

## Analyseur standard

- L'analyseur par défaut d'Elasticsearch.
- Il utilise le tokenizer standard, qui divise le texte sur la base de la segmentation de texte Unicode.
- Applique des filtres standard tels que la minuscule et la suppression des mots vides.
- Convient au traitement de texte à usage général dans de nombreuses langues.

## Analyseur simple

- Un analyseur simple qui divise le texte en termes chaque fois qu'il rencontre des caractères autres que des lettres.
- Utilise le tokenizer de lettres.
- Convertit tous les tokens en minuscules.
- Utile pour les champs de texte simples où seules les lettres sont pertinentes.

# Analyseurs prédéfinis

## Analyseur d'espaces blancs

- Divise le texte en fonction des caractères d'espacement.
- Utilise le tokenizer d'espaces blancs.
- Convertit tous les tokens en minuscules.
- Idéal dans les cas où il est important de préserver la ponctuation, mais où l'insensibilité à la casse est nécessaire.

## Analyseur d'arrêt

- Similaire à l'analyseur simple mais supprime également les mots vides.
- Utilise le tokenizer stop.
- Applique un filtre de mots vides et met les tokens en minuscules.
- Convient aux textes en anglais pour lesquels les mots vides doivent être ignorés.

## Analyseur de mots-clés

- Traite l'ensemble de l'entrée comme un seul jeton.
- Utilise l'analyseur de mots-clés.
- Utile pour les champs où des correspondances exactes sont requises, tels que les identifiants ou les étiquettes.

# Analyseurs prédéfinis

## Analyseur de motifs

- Fractionne le texte en fonction d'un modèle d'expression régulière.
- Utilise le tokenizer de motifs, qui permet d'utiliser des motifs d'expressions rationnelles personnalisés pour diviser le texte.
- Option flexible pour les besoins de tokenisation personnalisés basés sur des motifs spécifiques.

## Analyseurs de langue

- Ensemble d'analyseurs adaptés à des langues spécifiques, telles que l'anglais, le français, l'espagnol, etc.
- Ils comprennent généralement des tokenizers et des filtres spécifiques à la langue, tels que le stemming et la suppression des mots vides.
- Idéal pour traiter des textes dans des langues spécifiques en appliquant les règles linguistiques appropriées.

# Analyseurs - Configuration

La configuration de l'analyseur peut être spécifiée sur le mapping

- Cela permet de déterminer comment les champs de texte sont traités lors de l'indexation et de l'interrogation.
- En choisissant le bon analyseur, vous pouvez améliorer la pertinence des recherches en vous assurant que le texte est tokenisé, normalisé et filtré de manière appropriée.
- Par exemple, l'utilisation d'analyseurs spécifiques à une langue peut améliorer les résultats de recherche en appliquant un stemming et une suppression des mots vides adaptés à la langue du texte.

Au-delà des différents analyseurs fournis de base, ES donne la possibilité de définir son propre analyseur

- À partir d'un analyseur existant et en précisant les filtres

```
PUT /persons
{
  "mappings": {
    "dynamic": "strict",
    "properties": {
      "age": {
        "type": "long"
      },
      "firstname": {
        "type": "text"
      },
      "lastname": {
        "type": "text"
      },
      "sex": {
        "type": "text"
      },
      "description": {
        "type": "text",
        "analyzer": "french"
      }
    }
  }
}
```

# Exercice

---

Définissez le mapping pour les données contenues dans le fichier  
data/shakespeare.json

Préciser les analyseurs adéquats pour ce mapping

# Bonus : Analyseur personnalisé

```
# Custom Tokenizer
PUT _ingest/pipeline/my_custom_tokenizer
{
  "description": "My custom tokenizer",
  "processors": [
    {
      "set": {
        "field": "_source.my_tokenizer",
        "value": {
          "type": "pattern",
          "pattern": ","
        }
      }
    }
  ]
}
```

```
# Custom Analyzer
PUT _ingest/pipeline/my_custom_analyzer
{
  "description": "My custom analyzer",
  "processors": [
    {
      "set": {
        "field": "_source.my_analyzer",
        "value": {
          "type": "custom",
          "tokenizer": "my_tokenizer",
          "filter": [
            "lowercase",
            "my_filter"
          ]
        }
      }
    }
  ]
}
```

# Recherche



# Présentation

Comme pour les opérations du CRUD, le requêtage passe par la même syntaxe

Les paramètres de la requête représentent un document JSON également

<https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html>

Testez les différentes requêtes et observez les résultats

```
GET /titanic/_search
```

```
GET /accounts/_count
{
  "query": {
    "match": {
      "state": "IL"
    }
  }
}
```

```
GET /titanic/_search
{
  "query": {
    "match_all": {}
  },
  "from": 10,
  "size": 10,
  "sort": [
    {
      "age": {
        "order": "desc"
      }
    }
  ]
}
```

# Paramètres

Le document représentant une requête est composé de cinq champs

**query:** représente les critères de la recherche

**size:** représente le nombre d'entrée max à renvoyer

**from:** combiné avec size, représente l'index de la première entrée (shift)

**\_source:** permet de sélectionner les champs à renvoyer (tous par défaut)

- Représenté par un tableau
- Utilise des wild-cards et les mots-clefs include / exclude pour faire une sélection plus fine

**sort:** permet de trier les résultats selon un ou plusieurs champs particulier(s)

# Composants du DSL

Le DSL d'Elasticsearch comprend plusieurs composants permettant de construire des requêtes :

- **Requêtes de correspondance (match)**: Utilisées pour la recherche en texte intégral, elles permettent de faire correspondre les documents qui contiennent des termes spécifiques.
- **Requêtes de termes (term)**: Utilisées pour les correspondances exactes sur des données structurées.
- **Requêtes d'intervalle (range)**: Utilisées pour trouver des documents dont les valeurs se situent dans une plage spécifiée.
- **Requêtes booléennes (bool)**: Combinent plusieurs requêtes à l'aide d'opérateurs logiques tels que `must`, `should`, et `must_not`.
- **Requêtes d'agrégation (aggs)**: Utilisées pour l'analyse des données, par exemple pour calculer des moyennes ou additionner des valeurs.

# Exemple

Testez les deux requêtes suivantes. Que remarquez-vous ?

Quels avantages y a-t-il à utiliser des filtres plutôt que des queries ?

```
GET /titanic/_search
{
  "query": {
    "match": {
      "sex": "Female"
    }
  }
}
```

```
GET /titanic/_search
{
  "query": {
    "bool": {
      "filter": {
        "match": {
          "sex": "Female"
        }
      }
    }
  }
}
```

# Recherche "floue" et pertinence

Utilisez le contexte query lorsque la pertinence (le \_score) est importante.

Il est idéal pour la recherche "full-text", comme ce que ferait un utilisateur dans une barre de recherche :

- Calcule un score (\_score) : Plus un document est pertinent, plus son score est élevé.
- Analyse le texte : Il utilise l'analyseur pour découper les termes (ex: "Machine à café" devient "machine", "café").
- N'est pas mis en cache (principalement) : Le score doit être calculé à chaque fois.
- Clauses typiques : match, multi\_match, query\_string.

```
GET /titanic/_search
{
  "query": {
    "match": {
      "name": "John Smith"
    }
  }
}
```

# Filtrage "strict" et performance

Utilisez le contexte filter pour tout ce qui est une question binaire (oui/non), des données structurées, des plages ou des valeurs exactes.

- Ne calcule pas de score : La réponse est juste "oui" ou "non".
- N'analyse pas le texte (en général) : Il cherche la valeur exacte (ex: le tag "Cuisine" ou le prix 199).
- Extrêmement rapide et mis en cache : Elasticsearch adore les filtres. Il peut mettre en cache le résultat (une simple liste d'ID de documents) pour le réutiliser lors de requêtes futures.
- Clauses typiques : term, terms, range, exists.

```
GET /titanic/_search
{
  "query": {
    "bool": {
      "filter": [
        { "term": { "sex": "female" } },
        { "term": { "pclass": 1 } }
      ]
    }
  }
}
```

# Combinaison recherche floue et stricte

La meilleure pratique est de toujours combiner les deux en utilisant une bool query.

- Mettez vos clauses de pertinence (ex: match) dans la section must (ou should).
- Mettez vos clauses de filtrage (ex: term, range) dans la section filter.

Ce qui se passe :

- Phase 1 (Filter) : Elasticsearch exécute d'abord le bloc filter (rapide, mis en cache). Il identifie l'ensemble des passagers de 3ème classe qui n'ont pas survécu.
- Phase 2 (Query) : Seulement sur cet ensemble de résultats, il exécute la requête match du bloc must pour calculer la pertinence du nom "John Smith".
- Le résultat est la liste des "John Smith" les plus pertinents, mais qui garantit qu'ils sont tous de 3ème classe et ont survécu.

```
GET /titanic/_search
{
  "query": {
    "bool": {
      "must": [
        { "match": { "name": "John Smith" } }
      ],
      "filter": [
        { "term": { "pclass": 3 } },
        { "term": { "survived": 1 } }
      ]
    }
  }
}
```

# Recherches basiques

- Exécutez ces différentes requêtes et expliquez le résultat
- Pour la deuxième requête, comment expliquer la différence de score ?

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

```
GET /courses/_search
{
  "query": {
    "match": {
      "name" : "Computer"
    }
  }
}
```

```
GET /courses/_search
{
  "query": {
    "exists": {
      "field": "professor.email"
    }
  }
}
```

```
GET /courses/_search
{
  "query": {
    "range": {
      "students_enrolled": {
        "gte": 10,
        "lte": 20
      }
    }
  }
}
```

# Recherche et interprétation

- Dans une requête, on peut demander au moteur de recherche d'interpréter les documents afin de déterminer le résultat
  - Le mot clef **match** active l'interprétation des documents dans la requête
  - Le mot clef **term** désactive l'interprétation des documents dans la requête
- **Exercice:** Exécuter les deux requêtes ci-dessous. Qu'observez-vous ?

```
GET /titanic/_search
{
  "query": {
    "match": {
      "sex": "Female"
    }
  }
}
```

```
GET /titanic/_search
{
  "query": {
    "term": {
      "sex": "Female"
    }
  }
}
```

# Clauses must, must\_not et should

```
GET /titanic/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "match": {
            "pclass": 1
          }
        },
        {
          "match": {
            "parch": 2
          }
        }
      ]
    }
  }
}
```

```
GET /titanic/_search
{
  "query": {
    "bool": {
      "must_not": [
        {
          "match": {
            "pclass": 1
          }
        },
        {
          "match": {
            "parch": 2
          }
        }
      ]
    }
  }
}
```

```
GET /titanic/_search
{
  "query": {
    "bool": {
      "should": [
        {
          "match": {
            "pclass": 1
          }
        },
        {
          "match": {
            "parch": {"query": 2, "boost": 3}
          }
        }
      ]
    }
  }
}
```

- La clause **must** est utilisée pour spécifier les conditions que les documents doivent remplir pour être inclus dans l'ensemble des résultats.
- La clause **must\_not** est utilisée pour spécifier les conditions que les documents ne doivent pas remplir pour être inclus dans l'ensemble des résultats.
- La clause **should** est utilisée pour spécifier des conditions facultatives. Les documents qui correspondent à ces conditions sont considérés comme plus pertinents mais ne sont pas obligés de correspondre.

Question: d'après vous, à quoi sert la clause **boost** ?

# Recherche sur plusieurs champs

La clause **multi\_match** d'Elasticsearch est un type de requête polyvalent utilisé pour rechercher un texte donné dans plusieurs champs d'un document. Elle étend les capacités de la requête **match** en vous permettant de spécifier plusieurs champs à rechercher simultanément, ce qui la rend idéale pour les scénarios où le terme recherché peut apparaître dans différentes parties d'un document.

**multi\_match** prend en charge plusieurs types de requêtes, qui déterminent la manière dont les termes de recherche sont comparés aux champs. Les types les plus courants sont les suivants

- **best\_fields**: recherche chaque champ individuellement et sélectionne le score du champ qui correspond le mieux.
- **most\_fields**: Prend en compte tous les champs correspondants et combine leurs résultats.
- **cross\_fields**: Traite les champs comme un seul champ combiné, utile pour les champs qui représentent des données similaires (par exemple, le prénom et le nom de famille).
- **phrase**: Permet de faire correspondre des phrases dans plusieurs champs.
- **phrase\_prefix**: Similaire à phrase mais prend en charge la correspondance des préfixes.

```
GET /courses/_search
{
  "query": {
    "multi_match": {
      "fields": ["name^2", "professor.email"],
      "query": "computer",
      "type": "best_fields"
    }
  }
}
```

# Recherche de séquences

La requête `match_phrase` est utilisée pour trouver des documents dans lesquels l'expression spécifiée apparaît exactement telle qu'elle a été fournie, avec les termes dans le même ordre et adjacents les uns aux autres.

- **Correspondance exacte des phrases:** la requête recherche les documents contenant la séquence exacte des termes spécifiés dans la requête. Par exemple, la recherche de « Elasticsearch tutorial » ne trouvera que les documents où ces deux mots apparaissent ensemble dans cet ordre.
- **Proximité:** Vous pouvez spécifier un paramètre « `slop` » pour permettre une certaine flexibilité dans le positionnement des termes. Ce paramètre permet de faire apparaître un certain nombre d'autres mots entre les termes.

La requête `match_phrase_prefix` est similaire à `match_phrase` mais permet de faire correspondre le préfixe au dernier terme de la phrase. Cette fonction est utile pour les fonctions d'autocomplétion ou de recherche au fil de l'eau.

- **Correspondance des préfixes:** la requête correspond aux documents dans lesquels l'expression spécifiée apparaît, le dernier terme étant un préfixe. Par exemple, la recherche de « Elasticsearch tut » correspondrait à des documents contenant des expressions telles que « Elasticsearch tutorial » ou « Elasticsearch tutoring ».
- **Proximité:** Comme `match_phrase`, il peut également utiliser un paramètre `slop` pour permettre une certaine flexibilité dans le positionnement des termes.

# Recherche de séquences : Query\_String

La requête query\_string est une requête de recherche en texte intégral puissante qui permet aux utilisateurs de spécifier des recherches complexes en utilisant une syntaxe de requête compacte.

**Syntaxe avancée** : Cette requête prend en charge une syntaxe riche, permettant l'utilisation d'opérateurs booléens (AND, OR, NOT), de caractères génériques, de recherches floues, et plus encore. Par exemple, "elasticsearch AND (tutorial OR guide)" recherchera des documents contenant "elasticsearch" et soit "tutorial", soit "guide".

**Recherche multi-champs** : Par défaut, la requête s'applique à tous les champs indexés, mais vous pouvez spécifier des champs particuliers. Par exemple, "title:elasticsearch  
description:tutorial" recherchera "elasticsearch" dans le champ titre et "tutorial" dans le champ description.

**Analyse** : La requête est analysée et tokenisée selon l'analyseur par défaut de l'index, ce qui permet une recherche plus flexible mais peut parfois conduire à des résultats inattendus si l'utilisateur n'est pas familier avec le processus d'analyse.

# Recherche de séquences : Fuzzy

La requête fuzzy est utilisée pour trouver des documents contenant des termes similaires au terme de recherche, permettant ainsi de gérer les fautes d'orthographe et les variations légères.

**Distance de recherche** : La similarité est basée sur la distance de Levenshtein, qui mesure le nombre de caractères à changer pour transformer un mot en un autre. Par exemple, une recherche fuzzy pour "elosticsearch" pourrait trouver des documents contenant "elasticsearch".

**Paramètre de fuzziness** : Vous pouvez spécifier le degré de "fuzziness" autorisé, généralement exprimé en nombre de caractères qui peuvent être modifiés. Par exemple, avec un fuzziness de 2, "book" pourrait correspondre à "cook", "books", ou "booki".

**Performance** : Les requêtes fuzzy peuvent être coûteuses en termes de performance, surtout sur de grands ensembles de données, car elles nécessitent de comparer de nombreuses variations possibles.

# Recherche de séquences : Wildcard

La requête wildcard permet d'effectuer des recherches en utilisant des caractères génériques pour remplacer un ou plusieurs caractères dans le terme de recherche.

**Caractères génériques** : Utilise "\*" pour remplacer zéro ou plusieurs caractères, et "?" pour remplacer exactement un caractère. Par exemple, "elast\*" correspondrait à "elastic", "elasticsearch", etc., tandis que "te?t" correspondrait à "test" ou "text".

**Sensibilité à la casse** : Les requêtes wildcard sont généralement sensibles à la casse, sauf si elles sont appliquées à un champ analysé qui effectue une normalisation de la casse.

**Performance** : Les requêtes wildcard, en particulier celles commençant par un caractère générique (par exemple, "\*search"), peuvent être très coûteuses en termes de performance car elles nécessitent de scanner tous les termes de l'index.

**Limitation** : En raison de leur impact potentiel sur les performances, il est généralement recommandé d'utiliser les requêtes wildcard avec parcimonie et d'éviter les motifs trop larges ou commençant par un caractère générique si possible.

# Recherche de séquences : Exemples

```
GET /shakespeare/_search
{
  "query": {
    "query_string": {
      "default_field": "text_entry",
      "query": "(to be) AND (not to be)"
    }
  }
}
```

```
GET /shakespeare/_search
{
  "query": {
    "match_phrase": {
      "text_entry": "to be or not to be"
    }
  }
}
```

```
GET /courses/_search
{
  "query": {
    "match_phrase_prefix": {
      "course_description": {
        "query": "artificial intelligence, machine learning",
        "slop": 1
      }
    }
  }
}
```

<https://www.elastic.co/guide/en/elasticsearch/reference/current/full-text-queries.html>

# Recherche dans une liste / une intervalle

```
GET /titanic/_search
{
  "query": {
    "terms": {
      "pclass": [1, 2]
    }
  }
}
```

```
GET /titanic/_search
{
  "query": {
    "range": {
      "age": {
        "gte": 60,
        "lt": 80
      }
    }
  }
}
```

```
GET /courses/_search
{
  "query": {
    "range": {
      "course_publish_date": {
        "lt": "2016"
      }
    }
  }
}
```

Que remarquez-vous sur l'exécution de la 3ème requête ?

# Exercices

---

- Affichez les 10 premiers documents de l'index des températures.
- Recherchez les documents où la température moyenne est supérieure à 20°C.
- Trouvez les documents où la température moyenne est comprise entre 10°C et 15°C, triés par date croissante.
- Affichez les documents où l'incertitude sur la température moyenne est inférieure à 1°C.
- Recherchez les documents où la ville est "Abidjan" ou "Paris", en utilisant une requête booléenne.

# Exercices

---

- Trouvez les documents où la température moyenne est supérieure à 25°C et la date est dans le mois de juin, quelle que soit l'année.
- Recherchez les documents où le nom de la ville commence par "New" (par exemple, New York, New Delhi), en utilisant une requête de préfixe.
- Affichez les documents où la latitude est supérieure à 40°N et la longitude est inférieure à 20°E, en utilisant une requête booléenne avec des requêtes géographiques.
- Trouvez les documents où le nom du pays contient "États" ou "Royaume", en utilisant une requête de correspondance avec l'opérateur "OR".

# Exercices

---

- Rechercher dans l'index shakespeare le document qui match avec l'entrée "love"
- Rechercher dans l'index shakespeare le document qui match avec la phrase exacte "to be or not to be"
- Rechercher dans l'index shakespeare le document qui match avec l'entrée "love" prononcé par "Romeo"
- Rechercher le terme "death" dans les champs "text\_entry" et "play\_name", mais donne un poids double aux correspondances dans "play\_name"

# Update / Delete by query

Nous avons vu dans les bases que l'on pouvait mettre à jour / supprimer des documents à partir de leur index

Nous pouvons appliquer ce que nous avons appris dans cette section afin de sélectionner un sous-ensemble de documents sur lesquels appliquer ces opérations

```
POST /temperatures/_delete_by_query
{
  "query": {
    "range": {
      "paris": {
        "lte": 0
      }
    }
  }
}
```

```
POST /temperatures/_update_by_query
{
  "query": {
    "match_all": {}
  },
  "script": {
    "source": "ctx._source.cat = 'regular'",  

    "lang": "painless"
  }
}
```

# Agrégations



# Présentation

Les agrégations dans Elasticsearch sont une fonctionnalité puissante qui vous permet d'effectuer des analyses de données complexes et de générer des informations à partir de vos données indexées.

Elles vous permettent de résumer, de regrouper et d'effectuer des calculs sur vos données, à l'instar des fonctions GROUP BY et aggregate de SQL.

Les agrégations peuvent être imbriquées les unes dans les autres, ce qui permet d'effectuer des requêtes hiérarchiques complexes. Par exemple, vous pouvez imbriquer une agrégation métrique à l'intérieur d'une agrégation de buckets pour calculer la valeur moyenne d'un champ dans chaque bucket.

# Types

---

- **Agrégations de groupes de documents (Bucket Aggregations)** : Regrouper les documents dans des groupes en fonction de critères spécifiques. Les exemples incluent `terms`, `range`, `histogram`, et `date_histogram`.
- **Agrégations métriques**: Calculer des mesures telles que la somme, la moyenne, le minimum, le maximum et le nombre sur des champs numériques. Exemples : `avg`, `sum`, `min`, `max`, et `stats`.
- **Agrégations de pipeline**: Effectuer des calculs sur les résultats d'autres agrégations. Exemples : `derivative`, `moving_avg`, et `bucket_script`.
- **Agrégations matricielles**: Opèrent sur des données multidimensionnelles, comme le calcul des corrélations ou de la covariance.

# Agrégations

```
GET /titanic/_search
{
  "aggs": {
    "sex": {
      "terms": {
        "field": "sex.keyword"
      }
    }
  }
}
```

Exécutez la requête ci-dessus et observez le résultat

Pourquoi utiliser `.keyword` dans les agrégations ?

- **Correspondance exacte** : les agrégations comme les **termes** nécessitent une correspondance exacte (indépendante des résultats de l'analyse) pour regrouper les documents en fonction de valeurs uniques. L'utilisation d'un champ mot-clé permet de s'assurer que l'agrégation opère sur les valeurs exactes de la chaîne de caractères plutôt que sur les jetons analysés (exple: `firstClass => "first class"`)
- **Efficacité**: Les champs de mots-clés sont optimisés pour des opérations telles que les agrégations et les tris, ce qui les rend plus efficaces que l'utilisation de champs de texte analysés.

# Agrégations - Exemples

```
GET /titanic/_search
{
  "size": 0,
  "aggs": {
    "sex": {
      "terms": {
        "field": "sex.keyword"
      }
    }
  }
}
```

```
GET /titanic/_search
{
  "size": 0,
  "aggs": {
    "sex": {
      "terms": {
        "field": "sex.keyword"
      },
      "aggs": {
        "age_moyen": {
          "avg": {
            "field": "age"
          }
        }
      }
    }
  }
}
```

```
GET /titanic/_search
{
  "size": 0,
  "aggs": {
    "sex": {
      "terms": {
        "field": "sex.keyword"
      },
      "aggs": {
        "age_moyen": {
          "avg": {
            "field": "age"
          }
        }
      }
    },
    "classe": {
      "terms": {
        "field": "pclass"
      }
    }
  }
}
```

```
GET /titanic/_search
{
  "size": 0,
  "aggs": {
    "sex": {
      "terms": {
        "field": "sex.keyword"
      },
      "aggs": {
        "age_moyen": {
          "stats": {
            "field": "age"
          }
        }
      }
    }
  }
}
```

Exécutez ces requêtes et observez le résultat

# Agrégations - Exemples

Il est possible de raisonner par catégorie comme illustré dans l'exemple ci-contre

```
GET /titanic/_search
{
  "size": 0,
  "aggs": {
    "sex": {
      "terms": {
        "field": "sex.keyword"
      },
      "aggs": {
        "age_ranges": {
          "range": {
            "field": "age",
            "ranges": [
              {
                "from": 0,
                "to": 20
              },
              {
                "from": 20,
                "to": 40
              },
              {
                "from": 40,
                "to": 60
              },
              {
                "from": 60,
                "to": 80
              }
            ]
          },
          "aggs": {
            "age_avg": {
              "avg": {
                "field": "age"
              }
            }
          }
        }
      }
    }
  }
}
```

# Exercices

```
GET /titanic/_search
{
  "size": 0,
  "aggs": {
    "age-stats": {
      "stats": {
        "field": "age"
      }
    },
    "fare-stats": {
      "stats": {
        "field": "fare"
      }
    }
  }
}
```

## Exercice

- Trouver l'âge moyen des hommes mineurs
- Trouver la classe moyenne des femmes âgées de plus de 60 ans
- Tip: combiner query et aggs

# Observations

- Dans l'exercice précédent, nous avons identifié des sous-ensembles avant de pouvoir faire des calculs dessus
- C'est une étape consommatrice de ressources qui peut être évitée en précisant les filtres au niveau des agrégations

```
GET /titanic/_search
{
  "size": 0,
  "aggs": {
    "minors": {
      "filter": {
        "range": {
          "age": {
            "lt": 18
          }
        }
      },
      "aggs": {
        "age_moyen": {
          "avg": {
            "field": "age"
          }
        }
      }
    }
  }
}
```

# Fonctions statistiques

ES implante de nombreuses fonctions statistiques

<https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics.html>

```
GET /titanic/_search
{
  "size": 0,
  "aggs": {
    "ages": {
      "histogram": {
        "field": "age",
        "interval": 5
      }
    }
  }
}
```

```
GET /titanic/_search
{
  "size": 0,
  "aggs": {
    "pct_age": {
      "percentiles": {
        "field": "age",
        "percents": [
          25,
          50,
          75
        ]
      }
    }
  }
}
```

# Exercices

---

- Affichez le nombre total de documents dans l'index "temperatures".
- Trouvez la température moyenne à Paris.
- Comptez le nombre de documents pour chaque ville.
- Trouvez la température maximale pour chaque ville.
- Pour chaque ville, calculez la température moyenne par année.

# Exercices

---

- Trouvez les 3 villes avec la température moyenne la plus élevée.
- Pour chaque ville, calculez la température moyenne, minimale et maximale.
- Divisez les températures en 5 intervalles égaux et comptez le nombre de documents dans chaque intervalle pour chaque ville.
- Pour chaque ville, trouvez l'année avec la plus grande variation de température (différence entre le maximum et le minimum).

# Géolocalisation

La géolocalisation est une fonctionnalité puissante d'Elasticsearch qui permet de stocker, rechercher et analyser des données géospatiales.

Elasticsearch prend en charge différents types de données géospatiales comme les points géographiques (latitude/longitude), les formes (lignes, polygones) et même les données plus complexes comme les relations de proximité.

Avec ces capacités, Elasticsearch permet de construire des applications géolocalisées performantes comme :

- Des moteurs de recherche de points d'intérêt
- Des analyses de données géomarketing
- Du suivi et de l'optimisation de flottes de véhicules
- Des systèmes de recommandation basés sur la localisation

# Géolocalisation - Caractéristiques

## Types de champs géographiques :

- Le type "geo\_point" pour stocker des coordonnées latitude/longitude
- Le type "geo\_shape" pour représenter des formes plus complexes comme des lignes, polygones, multi-polygones

## Indexation de données géospatiales :

- Lors de l'indexation, les coordonnées sont converties dans un format interne optimisé pour les requêtes géospatiales
- Des algorithmes spécialisés comme les arbres geohash et les grilles de quadtree sont utilisés pour un accès rapide aux données

# Géolocalisation - Caractéristiques

## Requêtes géospatiales :

- Recherche de documents dans un rayon autour d'un point (geo\_distance)
- Recherche de documents dans une zone rectangulaire (geo\_bounding\_box)
- Recherche de documents qui croisent ou sont contenus dans des formes arbitraires (geo\_shape)

## Agrégations géospatiales :

- Regroupement des documents par grille géographique (geohash\_grid)
- Calcul de la distance moyenne des documents par rapport à un point (geo\_centroid)

## Tri par distance :

- Les résultats peuvent être triés par distance croissante ou décroissante par rapport à un point de référence

# Géolocalisation - Exemples

```
GET /temperatures/_search
{
  "query": {
    "geo_bounding_box": {
      "location": {
        "top_left": {
          "lat": 6.0,
          "lon": -4.0
        },
        "bottom_right": {
          "lat": 5.0,
          "lon": -3.0
        }
      }
    }
  }
}
```

```
GET /temperatures/_search
{
  "query": {
    "geo_distance": {
      "distance": "100km",
      "location": {
        "lat": 5.63,
        "lon": -3.23
      }
    }
  }
}
```

# Géolocalisation - Exercices

- Trouvez toutes les stations météo situées dans la "péninsule ibérique" (approximativement l'Espagne et le Portugal).
- Trouvez toutes les stations météo dans un rayon de 500km autour de Paris (France).
- Trouvez toutes les stations météo situées à l'intérieur d'un polygone qui dessine approximativement l'Italie.
- Trouvez toutes les stations "tropicales" (température moyenne supérieure à 25°C) qui sont aussi situées dans un rayon de 1000km autour de Dakar (Sénégal).
- Combien de stations se trouvent dans les zones suivantes par rapport à Berlin (Allemagne) : "Proche" (0-500km), "Moyen" (500-2000km) et "Loin" (>2000km).

# X-Pack



# Présentation

X-Pack est une extension officielle de la Suite Elastic (ELK) qui regroupe plusieurs fonctionnalités avancées initialement proposées sous forme de modules séparés, désormais intégrées nativement à la stack Elastic.

## Atouts dans la Stack ELK

- X-Pack a été conçu pour renforcer l'offre Elastic par l'ajout de fonctions essentielles à l'exploitation en production, la supervision à large échelle et la sécurité des données.
- Depuis la version 6.3, de nombreuses fonctionnalités sont incluses par défaut dans les distributions Elastic, avec certaines options gratuites et d'autres payantes suivant la politique de licence.

# Add-Ons X-Pack Populaires

**Security** : Fournit des fonctionnalités de sécurité avancées, y compris l'authentification, l'autorisation basée sur les rôles, le chiffrement SSL/TLS, et l'audit.

**Monitoring** : Offre des outils de surveillance en temps réel pour suivre les performances et la santé de votre cluster Elasticsearch.

**Alerting** : Permet de configurer et de gérer des alertes basées sur des conditions spécifiques dans vos données Elasticsearch.

**Reporting** : Permet de générer et de planifier des rapports PDF à partir des visualisations et des tableaux de bord de Kibana.

**Machine Learning** : Fournit des capacités d'apprentissage automatique pour la détection d'anomalies et les prévisions sur vos données Elasticsearch.

# Add-Ons X-Pack Populaires

**Graph** : Permet d'explorer les relations dans vos données à travers des visualisations de graphes interactifs.

**SQL** : Offre une interface SQL pour interroger Elasticsearch, permettant une intégration plus facile avec les outils BI existants.

**Watcher** : Un système d'alerte et de notification qui permet de détecter des changements dans les données Elasticsearch et d'y réagir.

**APM (Application Performance Monitoring)** : Fournit des outils pour surveiller et optimiser les performances des applications.

# Monitoring et administration



# X-Pack Monitoring

---

X-Pack Monitoring est le module intégré à la Suite Elastic qui permet de surveiller en continu l'état et la performance de vos clusters Elasticsearch, Kibana, Logstash et Beats depuis des dashboards dédiés dans Kibana.

## Cas d'usage typiques

- Surveiller les pics d'utilisation du CPU ou de la heap Elasticsearch et anticiper les problèmes de scalabilité.
- Suivre la disponibilité et la charge des pipelines Logstash ou des agents Beats pour détecter d'éventuels « goulets d'étranglement ».
- Mettre en place du reporting automatisé sur les performances du cluster ou sur les indices volumineux.
- Diagnostiquer rapidement les pannes ou incohérences dans les flux de données entre applications et Elastic Stack.

# X-Pack Monitoring - Fonctionnalités

**Collecte automatique** des métriques système et applicatives pour chaque brique Elastic Stack : état des nœuds, indices, shards, pipelines Logstash, agents Beats.

**Visualisation en temps réel** dans Kibana via des vues synthétiques et détaillées : santé du cluster, niveaux d'utilisation des ressources (CPU, RAM, stockage), flux de données, taux d'erreur ou de latence par service.

**Historique des indicateurs** : conservation des stats, possibilité de suivre l'évolution de la stack sur plusieurs jours/semaines pour l'analyse post-mortem ou la capacité planning.

**Alerting couplé** : création d'alertes sur les métriques critiques (CPU, heap, nombre de shards, taux d'erreur), notifications par email/webhook lorsque certains seuils sont atteints.

# X-Pack Monitoring – Mise en œuvre

**Activation** : dans les fichiers de configuration (elasticsearch.yml, kibana.yml), activez le bloc monitoring

- xpack.monitoring.enabled: true
- Il est possible de configurer des exporteurs pour stocker les métriques monitoring dans un cluster dédié.

**Accès** : connectez-vous à Kibana, rendez-vous dans le module « Monitoring » pour une vue globale ou détaillée de la santé et des performances de vos instances Elastic Stack.

**Gestion multi-cluster** : X-Pack Monitoring gère la supervision centralisée de plusieurs clusters (utile en production ou pour séparer le monitoring de la production et du développement).

**Sécurité** : X-Pack Monitoring fonctionne en synergie avec les autres modules X-Pack pour garantir l'intégrité des données de supervision et leur accès à travers des droits d'utilisateur.

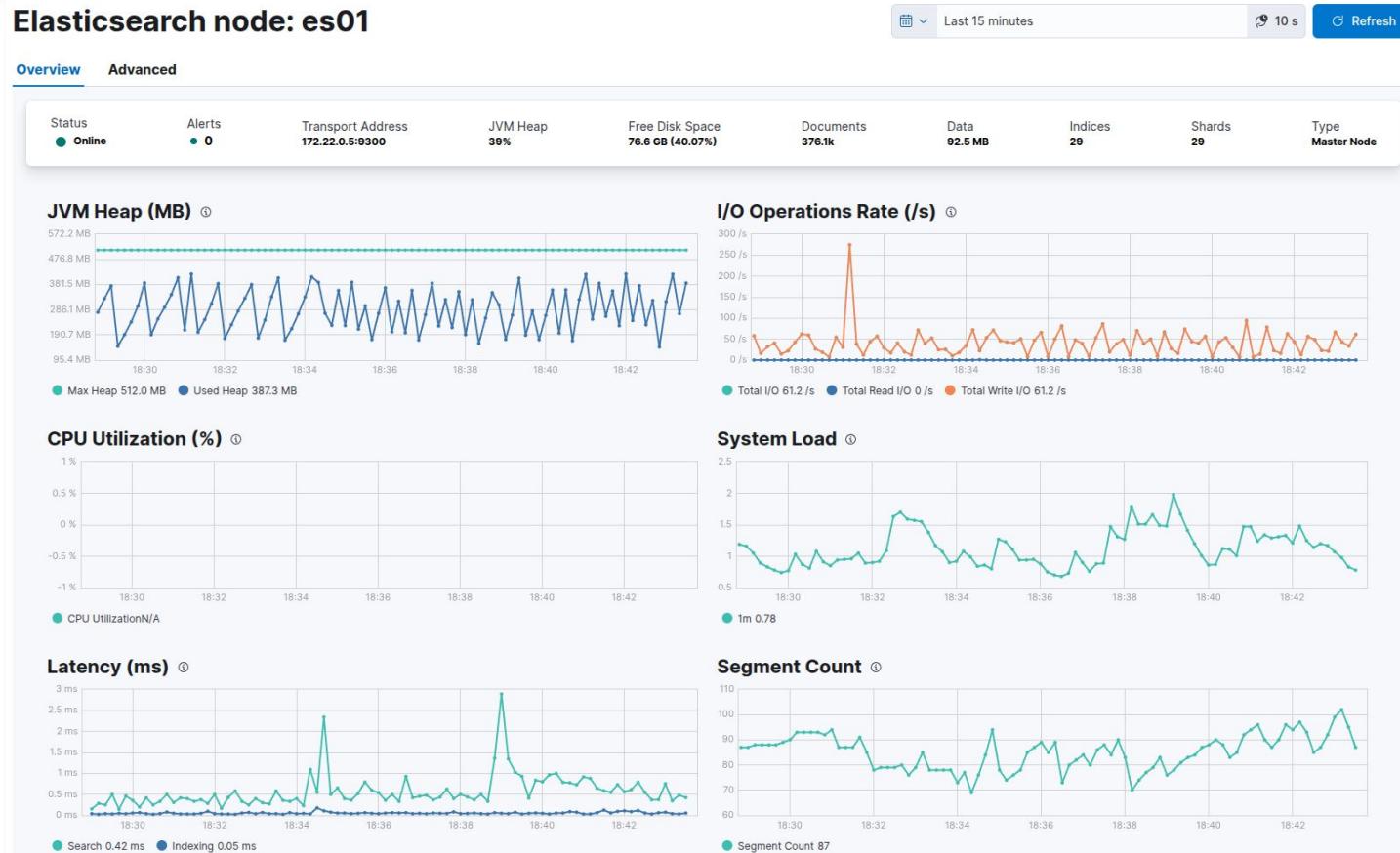
# Travaux Pratiques

---

Ouvrir le "Stack Monitoring" dans Kibana.

Identifier le nœud "master" actuel et simuler une charge (ex: en lançant une grosse requête d'agrégation) et observer l'impact sur le CPU et la JVM Heap.

# Monitoring - Exemple



# API Snapshot

L'API Snapshot d'Elasticsearch permet d'effectuer des sauvegardes (« snapshots ») incrémentales des indices, du cluster ou d'objets spécifiques, et de restaurer tout ou partie de la donnée à l'état précédemment sauvegardé.

```
PUT /_snapshot/mon_repo
{
  "type": "fs",
  "settings": { "location": "/chemin/du/repo", "compress": true }
}
```

1 - déclaration

2- création

```
PUT /_snapshot/mon_repo/mon_snapshot
{
  "indices": "index1,index2",
  "ignore_unavailable": true,
  "include_global_state": true
}
```

3- restauration

```
POST /_snapshot/mon_repo/mon_snapshot/_restore
{
  "indices": "index1",
  "rename_pattern": "index1",
  "rename_replacement": "restored_index1"
}
```

# API Snapshot - Fonctionnement

Les snapshots sont utilisés pour la sauvegarde, la restauration après incident, ou la migration de données entre clusters.

La sauvegarde est incrémentale : seuls les nouveaux segments des indices depuis le dernier snapshot sont enregistrés, ce qui optimise l'espace disque et accélère les opérations.

Un snapshot inclut les principaux shards des indices ciblés, l'état du cluster et éventuellement les « feature states » (ex : historisation Kibana, sécurité, etc.).

Les snapshots sont déposés dans un « repository » : il faut d'abord le déclarer (filesystem local/NFS, S3, etc.) puis utiliser l'API avec cURL ou directement via Kibana.

# Exercices

1. Sauvegarder l'index *temperatures* sur le serveur minio
2. Supprimer l'index *temperatures*
3. Restorer l'index *temperatures*

PS: Lancer docker-compose.secured.yml au préalable

```
PUT /_snapshot/minio_backup
{
  "type": "s3",
  "settings": {
    "bucket": "elk-backup",
    "client": "default",
    "endpoint": "http://minio:9000",
    "protocol": "http",
    "path_style_access": true
  }
}
```

# Considérations de performance

## Mapping et analyse

- **Mises en correspondance des champs:** Définir des correspondances explicites pour les champs afin d'éviter la surcharge de la correspondance dynamique. Utilisez les types de données appropriés (par exemple, **keywords** pour les correspondances exactes, **text** pour la recherche en texte intégral).
- **Analyseurs:** Choisissez les bons analyseurs pour vos champs de texte afin d'optimiser la pertinence et les performances de la recherche. Les analyseurs personnalisés peuvent être adaptés à des cas d'utilisation spécifiques.

## Cache

- **Cache de requêtes:** Exploiter le cache de requêtes d'Elasticsearch pour les requêtes fréquemment exécutées afin de réduire les temps de réponse.
- **Cache de données de champ:** Optimiser l'utilisation du cache de données de champ en utilisant **doc\_values** pour les agrégations sur les champs de mots clés.

# Considérations de performance

## Optimisation des requêtes

- **Contexte des filtres:** Utilisez des filtres plutôt que des requêtes lorsque l'évaluation de la pertinence n'est pas nécessaire, car les filtres sont mis en cache et plus rapides.
- **Éviter les caractères génériques:** Réduire au minimum l'utilisation de caractères génériques dans les requêtes, car ils peuvent être gourmands en ressources.
- **Pagination:** Utilisez les API `search_after` ou `scroll` pour une pagination profonde au lieu d'utiliser les valeurs large `from/size`.

## Gestion des ressources

- **Taille du tas:** Allouer suffisamment de mémoire de tas aux nœuds Elasticsearch, typiquement 50% de la RAM disponible, mais pas plus de 32GB pour éviter les problèmes de JVM.
- **E/S disque:** Utiliser des disques SSD rapides pour le stockage des données Elasticsearch afin d'améliorer les performances d'indexation et de recherche.
- **Utilisation du CPU:** Surveillez l'utilisation du processeur et optimisez la complexité des requêtes ou la distribution des nuages si les nœuds sont constamment surchargés.

# Considérations de performance

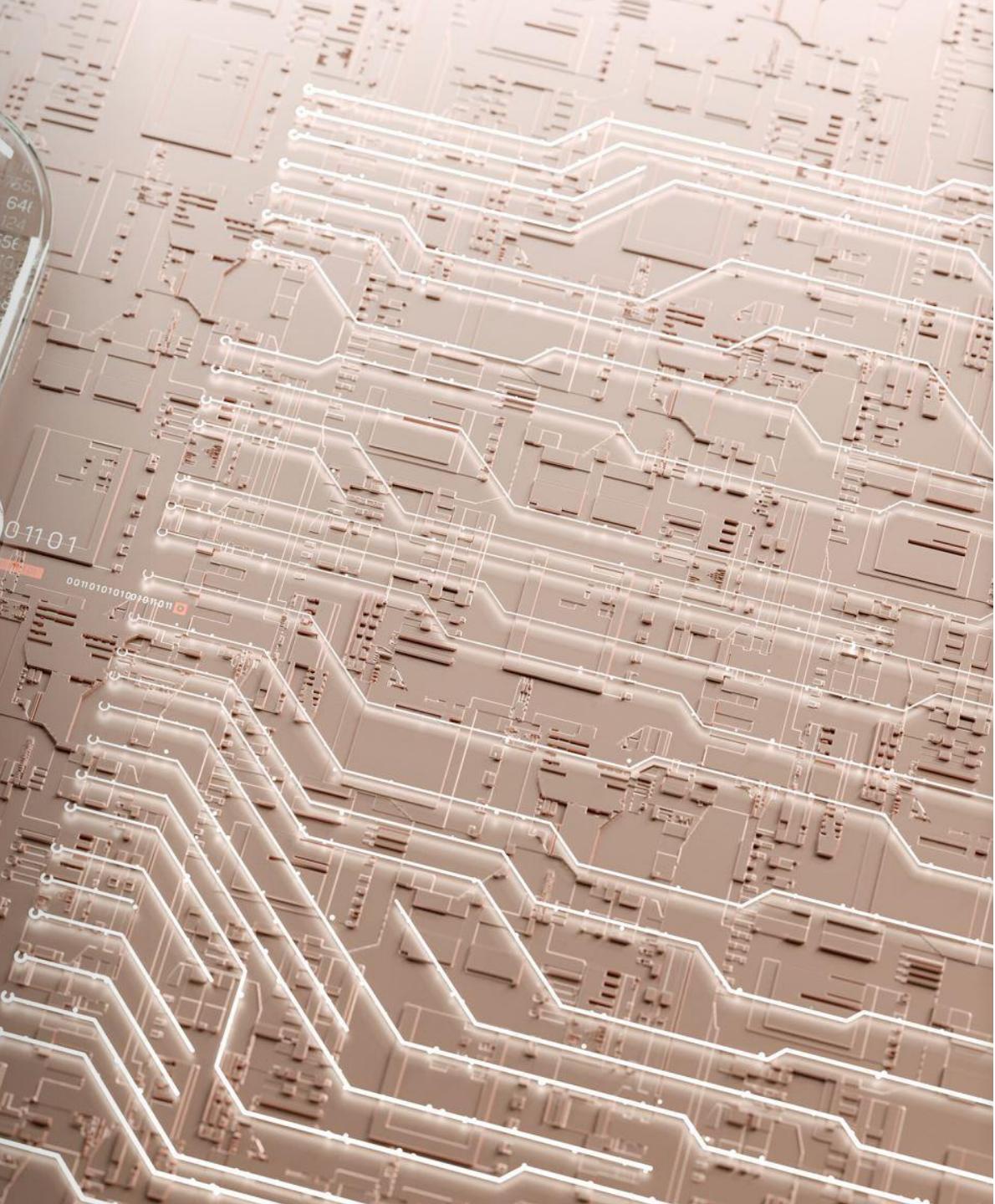
## Configuration du cluster

- **Rôles des nœuds:** Attribuer des rôles spécifiques aux nœuds (par exemple, maître, données, acquisition) afin d'optimiser l'utilisation des ressources et la stabilité.
- **Équilibrage de la charge:** Répartir la charge de manière homogène entre les nœuds en équilibrant les grappes et les répliques.

## Surveillance et maintenance

- **Outils de surveillance:** Utilisez des outils comme Kibana, Elastic Stack Monitoring ou des solutions tierces pour surveiller l'état de santé du cluster, les mesures de performance et l'utilisation des ressources.
- **Maintenance régulière:** Effectuez des tâches de maintenance régulières telles que la fusion de segments, la réduction d'index et l'effacement d'anciens index afin de maintenir des performances optimales.

# Maintenance et sécurité



# Rolling Upgrade

Le « Rolling Upgrade » est une méthode de mise à jour du cluster qui évite tout arrêt, en procédant node par node.

- Chaque nœud est arrêté, mis à jour, puis réintégré au cluster pendant que les autres continuent à servir les requêtes.
- Cela permet d'assurer la continuité de service, la reprise automatique des shards et la synchronisation des données.
- Cette procédure exige que la version cible soit compatible avec la version courante (vérifier la documentation officielle avant chaque upgrade), et que les réplications soient bien configurées pour garantir aucune perte de données pendant les basculements.

# Rolling Upgrade - Tuto

1. Vérifiez la compatibilité des versions
  - o Consultez la documentation officielle pour connaître les chemins de migration autorisés (les upgrades directs peuvent être limités, souvent d'une seule version majeure à la suivante).
2. Sauvegardez impérativement vos données
  - o Effectuez un snapshot complet de vos indices et de l'état du cluster avant toute opération.
3. Désactivez temporairement certaines opérations
  - o Suspendre les tâches non critiques et les indexations lourdes.
  - o Désactivez temporairement l'allocation automatique des shards

```
# Désactiver / Réactiver le sharding
PUT _cluster/settings
{
  "persistent": {
    "cluster.routing.allocation.enable": "none | all"
  }
}
```

# Rolling Upgrade - Tuto

1. Arrêtez le premier nœud à mettre à jour
  - De préférence commencez par les nœuds du « data tier » le moins critique (ex : cold ou frozen), puis progressez vers les hot/warm/master.
2. Mettez à jour le binaire Elasticsearch sur ce nœud
  - Installez la nouvelle version (RPM/DEB, tar.gz, Docker tag...), mettez à jour la configuration si besoin.
3. Redémarrez le nœud et attendez qu'il soit totalement réintégré
  - Contrôlez via l'API \_cat/nodes ou Kibana que le nœud est bien reconnu et en état « green » ou « yellow » acceptable.

# Rolling Upgrade - Tuto

1. Répétez l'opération sur chaque nœud du cluster
  - Traitez chaque nœud un par un, sans jamais arrêter plusieurs nœuds simultanément (maintien de la haute disponibilité des données et du cluster).
2. Réactivez l'allocation automatique
  - Une fois tous les nœuds migrés et stables, relancez l'allocation automatique des shards
3. Contrôlez la santé du cluster (vérifiez les logs, l'état des shards, l'intégrité des indices)
  - Vérifiez l'absence de warnings, de shards non alloués, et restaurez les applications/services clients si tout est opérationnel.

# ILM – Index Lifecycle Management

**ILM gère le cycle de vie des indices** : il automatise les phases de gestion des données (hot, warm, cold, delete), en appliquant des politiques selon l'âge, la taille ou l'usage des indices.

**Rollover des indices** : ILM permet le « rollover » automatique, c'est-à-dire la création de nouveaux indices de destination au fil du temps ou après avoir atteint certaines limites. Cela facilite la gestion, l'archivage et l'optimisation du stockage et des performances d'indexation.

**Optimisation sur les différents tiers** : les indices peuvent migrer entre les phases hot/warm/cold/frozen pour adapter leur stockage et leur disponibilité selon le besoin, sans intervention manuelle.

# ILM - Phases

Phase	Usage	Caractéristiques
<b>Hot</b>	Ecriture intensive	Indices récents, toutes les opérations autorisées
<b>Warm</b>	Lecture optimisée, peu d'écriture	Réduction du nombre de shards, optimisation ressource
<b>Cold</b>	Lecture seule, stockage moins coûteux	Suppression des répliques inutiles, stockage long terme
<b>Delete</b>	Suppression	Effacement automatique des indices arrivés en fin de vie

# ILM - Avantages

---

**Automatisation** : plus besoin de scripts externes, tout est géré par le cluster selon la politique définie en YAML ou depuis Kibana.

**Observabilité** : suivi de l'état de chaque index via l'API `_ilm/explain`.

**Adaptabilité** : support des data streams, intégration avec Beats et Logstash pour appliquer des politiques dès la création des indices.

# ILM – Mise en œuvre

Une « policy » décrit les différentes phases (hot, warm, cold, delete/frozen) et les actions à effectuer pour chaque phase.

Pour mettre en oeuvre la politique ci-contre, il est nécessaire de passer par le point d'accès suivant, avec la méthode PUT:

- PUT /\_ilm/policy/my\_policy

```
{  
  "policy": {  
    "phases": {  
      "hot": {  
        "actions": {  
          "rollover": {  
            "max_size": "50gb",  
            "max_age": "30d"  
          }  
        }  
      },  
      "warm": {  
        "actions": {  
          "allocate": {  
            "number_of_replicas": 1  
          },  
          "forcemerge": {  
            "max_num_segments": 1  
          }  
        }  
      },  
      "cold": {  
        "actions": {  
          "freeze": {}  
        }  
      },  
      "delete": {  
        "min_age": "90d",  
        "actions": {  
          "delete": {}  
        }  
      }  
    }  
  }  
}
```

# ILM – Mise en œuvre

Il est ensuite possible d'associer la politique ILM à un « index template », ce qui permet d'automatiser la gestion des nouveaux indices.

```
PUT _template/my_template
{
  "index_patterns": [ "logs-*" ],
  "settings": {
    "number_of_shards": 1,
    "index.lifecycle.name": "my_policy",
    "index.lifecycle.rollover_alias": "logs"
  }
}
```

# Curator : pour les besoins avancés

---

Curator reste utile pour les tâches complexes ou spécifiques non couvertes par ILM, comme la gestion fine des snapshots, le close/open sélectif, ou l'application d'actions sur des patterns d'indices précis.

Il fonctionne en externe et peut être lancé via cron, mais il ne gère plus les indices administrés par ILM par défaut (option `allow_ilm_indices` nécessaire pour le forcer).

Utile aussi pour les versions antérieures d'Elasticsearch ou pour des opérations de migration et de maintenance non quotidiennes.

# Exercice

---

Pour des logs applicatifs (index logs-appli-YYYY.MM.DD), définissez une politique ILM :

- Hot : Les données restent 7 jours.
- Warm : Après 7 jours, elles passent en "warm" (lecture seule).
- Delete : Après 30 jours, elles sont supprimées.

# Authentification et autorisation

---

**Authentification** : vérification de l'identité des utilisateurs via login/mot de passe, clé API, ou jeton JWT.

**Autorisation (RBAC)** : attribution de rôles aux utilisateurs, permettant de restreindre les actions sur le cluster, les index, ou les dashboards Kibana (lecture, écriture, gestion, etc.).

Les **mappings** des rôles peuvent se faire via l'API, le fichier de configuration ou les interfaces Kibana, et il est possible de synchroniser avec LDAP/Active Directory/SSO.

# Chiffrement des communications

---

TLS/SSL : chiffrement des échanges entre les nœuds du cluster, entre le client et le cluster, et entre Kibana et Elasticsearch.

Les certificats garantissent que seuls les nœuds ou clients autorisés peuvent se connecter, limitant les risques d'intrusion ou de data leakage sur le réseau.

# Chiffrement des données

---

Elasticsearch propose le chiffrement des données stockées (« at rest ») via des fonctionnalités natives ou intégrées chez certains fournisseurs cloud (ex : AWS, Google).

Les données sont protégées même en cas d'accès non autorisé au disque ou au système de fichiers sous-jacent.

# Audit

---

X-Pack inclut l'audit des accès et modifications pour tracer l'ensemble des opérations et répondre aux exigences de conformité (RGPD, PCI, etc.).

Il est possible de configurer des logs d'audit des requêtes, changements de rôles, tentatives d'accès, etc.

# Bonnes pratiques

---

Désactivez les accès non sécurisés (HTTP sans TLS, accès anonymous).

Activez la sécurité dès le déploiement (ne jamais exposer un cluster Elasticsearch sans protection sur Internet).

Segmentez les réseaux (firewall, VPC, démilitarisation) pour limiter la surface d'attaque.

Mettez à jour régulièrement les dépendances et plugins.



# Exercice

---

Créer un rôle titanic\_reader :

- Privilège read sur l'index titanic.
- Privilège monitor sur le cluster.

Créer un utilisateur analyste avec le mot de passe password123.

- Assigner le rôle titanic\_reader à l'utilisateur analyste.
- Tester : Se connecter à Kibana en tant qu'analyste.

Essayer de lire (GET /titanic/\_search) et d'écrire (POST /titanic/\_doc/999 {...}) dans l'index.

# Beats



# Présentation

Beats est une famille d'agents de collecte de données légers ("lightweight shippers").

- **Rôle** : S'installe sur vos serveurs, conteneurs, ou VM pour collecter des données opérationnelles.
- **Léger** : Écrit en langage Go. Conçu pour une performance maximale avec une empreinte CPU et mémoire minimale.
- **Spécialisé** : Chaque "Beat" est spécialisé dans la collecte d'un type de données spécifique.
- **Destination** : Envoie les données collectées directement vers Elasticsearch ou vers Logstash pour un traitement plus poussé.

# Motivations

## Performance & Légereté

- Impact minimal sur les ressources de vos serveurs de production, ce qui est crucial car l'agent de collecte ne doit pas ralentir l'application.

## Garantie de Livraison (At-Least-Once)

- Beats mémorise l'état de lecture de chaque fichier dans un fichier "registry".
- En cas d'arrêt (de l'agent, du serveur, du réseau), il reprend exactement là où il s'était arrêté. Aucune ligne de log n'est perdue.

# Motivations

---

## Gestion de la Contre-pression (Backpressure)

- Si Logstash ou Elasticsearch sont surchargés et répondent lentement, Filebeat ralentit automatiquement sa lecture.
- Cela évite de saturer le pipeline d'ingestion ("Logstash OOM").

## Simplicité & Intégration

- Configuration simple via un fichier YAML.
- S'intègre nativement avec Elasticsearch et Logstash.

# Architecture

Un Beat est configuré autour de **3 composants principaux** :

- **Inputs (Entrées)**
  - D'où viennent les données ?
  - Ex: type: log (fichiers), type: syslog, type: container...
  - Définit les chemins d'accès : paths: ["/var/log/nginx/\*.log"]
- **Processors (Processeurs)**
  - Transformations légères effectuées à la source (sur l'agent).
  - Ex: Ajouter des métadonnées (tags, infos cloud), supprimer des champs, décoder du JSON.
- **Outputs (Sorties)**
  - Où vont les données ?
  - output.elasticsearch: { ... } (Envoi direct à ES)
  - output.logstash: { ... } (Envoi à Logstash pour enrichissement)
  - output.console: (Pratique pour le débogage !)

# Agents spécialisés

## **Filebeat** (Le plus populaire)

- Collecte et "suit" (tail) les fichiers journaux (logs).
- Gère la rotation des logs, les coupures, les redémarrages.
- Ex: logs Nginx, logs applicatifs, logs système...

## **Metricbeat**

- Collecte les métriques système et services.
- Ex: CPU, RAM, disque, Docker, Nginx, Kafka, MySQL...

# Agents spécialisés

## Packetbeat

- Analyse le trafic réseau en temps réel (sniffer).
- Ex: Monitorer les requêtes HTTP, les transactions SQL, les appels DNS...

## Heartbeat

- Monitoring de disponibilité ("uptime").
- Vérifie activement que vos services sont joignables (ping ICMP, TCP, HTTP).

## Auditbeat

- Audit de sécurité et d'intégrité des fichiers.
- Ex: Déetecte les modifications de fichiers critiques (ex: /etc/passwd), audite les appels système.

# Les modules

Pour les services courants (Nginx, MySQL, Apache, Kafka...), Filebeat propose des Modules.

Un module s'occupe de TOUT pour vous :

- Configuration de l'Input : Sait où trouver les bons fichiers de log (/var/log/nginx/access.log, etc.).
- Parsing : Charge un Pipeline Ingest dans Elasticsearch pour parser les logs (une alternative à Logstash pour les cas standards).
- Dashboards : Installe des Dashboards Kibana pré-construits pour visualiser immédiatement vos données.

# Les modules : Commandes

**Lister les modules** : *filebeat modules list*

**Activer un module** : *filebeat modules enable nginx*

**Configurer un module** : Éditer le fichier de configuration correspondant

- Exemple: /etc/filebeat/modules.d/nginx.yml

# Travaux Pratiques

---

1. Ouvrez le fichier docker-compose.secured.yml.
2. Décommentez les services suivants :
  - a. logstash
  - b. filebeat
  - c. nginx-log-generator
3. Relancez la stack pour prendre en compte les nouveaux services
4. Que se passe-t-il ?

# Travaux Pratiques

Étudiez le fichier filebeat/filebeat.nginx.yml et répondez aux questions suivantes :

- Quel type d'input est utilisé ? (type: log)
- Quel(s) fichier(s) Filebeat surveille-t-il ? (paths: - /var/log/nginx/access.log)
- Où Filebeat envoie-t-il les données ? (output.logstash: hosts: ["logstash:5044"])
- Pourquoi n'envoie-t-on pas directement à output.elasticsearch ?

# Logstash



# Présentation

---

**Rôle** : C'est un ETL (Extract, Transform, Load) pour vos logs et événements. Il ingère, transforme, enrichit et envoie les données.

**Architecture** : Son fonctionnement est basé sur des pipelines configurables.

**Puissance** : Extrêmement flexible grâce à plus de 200 plugins (inputs, filters, outputs).

# Cas d'usage

---

Parser des logs non-structurés (ex: logs Apache, Nginx, applicatifs).

Enrichir les données (ex: ajouter des infos GeolP, parser un User-Agent).

Normaliser les données (ex: renommer des champs, corriger des types).

Router les données vers différentes destinations (ex: Elasticsearch, S3, ...).

# Motivations

---

## Parsing Complexe (Grok)

- Transformer du texte brut (log Nginx) en champs structurés (JSON).

## Enrichissement de Données

- geoip: Transformer une client\_ip en pays, ville et coordonnées GPS.
- useragent: Transformer un "User Agent" en OS, Mapsur, appareil.
- jdbc: Interroger une base de données SQL pour ajouter des métadonnées (ex: trouver un user\_name depuis un user\_id).

# Motivations

---

## Routage Conditionnel Avancé

- Envoyer les logs de PROD vers un index Elasticsearch et les logs de DEV vers un autre.
- Si un log contient [ERROR], l'envoyer aussi vers PagerDuty ou Slack.

## Gestion de Multiples Sources / Destinations

- Collecter des données depuis Beats, des flux Kafka, des files SQS, des BDD...
- ...et les envoyer vers Elasticsearch, S3, HDFS, etc.

# Architecture

La configuration de Logstash est simple et se résume à 3 composants :

- **input** (Entrée)
  - Comment les données entrent dans Logstash.
  - Ex: beats, tcp, kafka, file, jdbc...
- **filter** (Filtre)
  - Le cœur de Logstash. C'est ici que la magie opère.
  - Une série de transformations, d'enrichissements, de parsings.
  - Ex: grok, mutate, geoip, date, json...
- **output** (Sortie)
  - Où les données transformées sont envoyées.
  - Ex: elasticsearch, s3, file, stdout (pour le debug)...

# Grok (Filtre)

---

grok est un filtre très puissant. Il parse du texte brut pour en extraire des champs structurés.

Il utilise des expressions régulières (regex) pré-packagées et nommées.

**Exemple** : %{NUMBER:response\_code} va chercher un nombre et le stocker dans un champ nommé response\_code.

# Travaux Pratiques

---

Ouvrez le fichier logstash/pipeline/nginx.conf et répondez aux questions suivantes:

- Comment log Nginx est parsé ?
- Comment est enrichie la donnée ?
- Comment sont récupérées les informations relatives aux producteurs ?
- Comment sont traitées les dates ?
- Comment sont envoyées les données transformées ?



# Visualisation



# Présentation

---

Kibana est un interface web qui vous permet de :

- Explorer vos données brutes (Logs, Métriques).
- Visualiser les données sous forme de graphiques, cartes, tables...
- Monitorer la santé de votre stack et de vos applications.
- Administrer la sécurité, le cycle de vie des données, et plus encore.

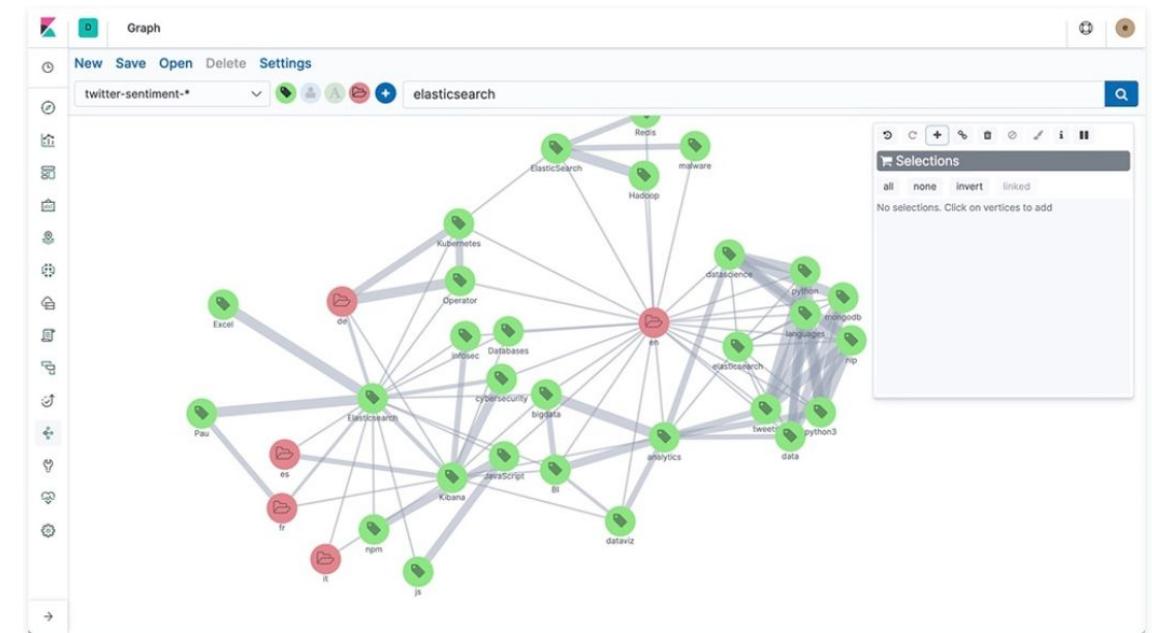
# Présentation

KIBANA intègre les diagrammes classiques

- Camembert
  - Histogramme
  - Donut
  - Nuage de points
  - Réseaux
  - ...

Il permet l'analyse de différents types de données

- Géographiques
  - Suite temporelles
  - ...



# Apps

---

L'application **discover** permet d'avoir une vue d'ensemble des données à afficher

L'application **visualize** permet de définir des vues sur les données

L'application **graph** permet de définir des visualisations mettant en avant des relations entre données

L'application **map** permet de définir des visualisations basées sur des cartes

L'application **dashboard** permet d'agréger des vues afin de construire un tableau de bord

L'application **canvas** permet de construire des dashboards beaucoup plus élaborés

L'application **machine learning** permet d'implanter des algos de ML (application payante)

# Apps

---

L'application **App Search** permet de définir des fonctionnalités de recherche, incluant des statistiques sur les recherches

L'application **Workplace Search** permet d'étendre les fonctionnalités de recherche à d'autres applications (git, twitter, ...)

L'application **metrics** permet d'agréger des métriques provenant de différentes sources (configurées avec Beats)

L'application **logs** permet de visualiser les logs provenants de différentes sources

L'application **APM** permet de visualiser les performances des différentes briques logicielles d'une architecture

# Apps

L'application **Uptime** permet de visualiser la disponibilité des différentes briques logicielles d'une architecture

L'onglet **Security** contient un certain nombre d'applications dédiés à l'analyse de la sécurité des applications

L'application **Ingest Manager** permet de configurer des agents pour l'ingestion de données

L'application **Stack Monitoring** permet de surveiller l'ensemble de la pile ES

L'application **Stack Management** permet de configurer la pile ES pour la visualisation

# Workflow type

Pour passer des logs bruts à un dashboard décisionnel, le processus est presque toujours le même :

1. **Data View** (Vue de Données) : Dire à Kibana quel(s) index Elasticsearch il doit lire (ex: logstash-nginx-\*).
2. **Discover** (Découvrir) : Explorer les logs bruts, filtrer, chercher des erreurs.
3. **Visualize** (Visualiser) : Créer un graphique (un camembert, une carte, une courbe...) à partir des données. L'outil principal pour cela est Lens.
4. **Dashboard** (Tableau de Bord) : Assembler plusieurs visualisations sur un seul écran pour raconter une histoire.

# Étape 1 - Data View

---

1. Aller dans Stack Management > Kibana > Data Views.
2. Cliquer sur "Create Data View".
3. Nommer la vue : Logs Nginx
4. Utiliser le pattern d'index : logstash-\*
5. Choisir le champ de temps : @timestamp

# Étape 2 - Discover

---

L'application Discover est votre loupe. C'est l'outil de "debug" et d'exploration par excellence.

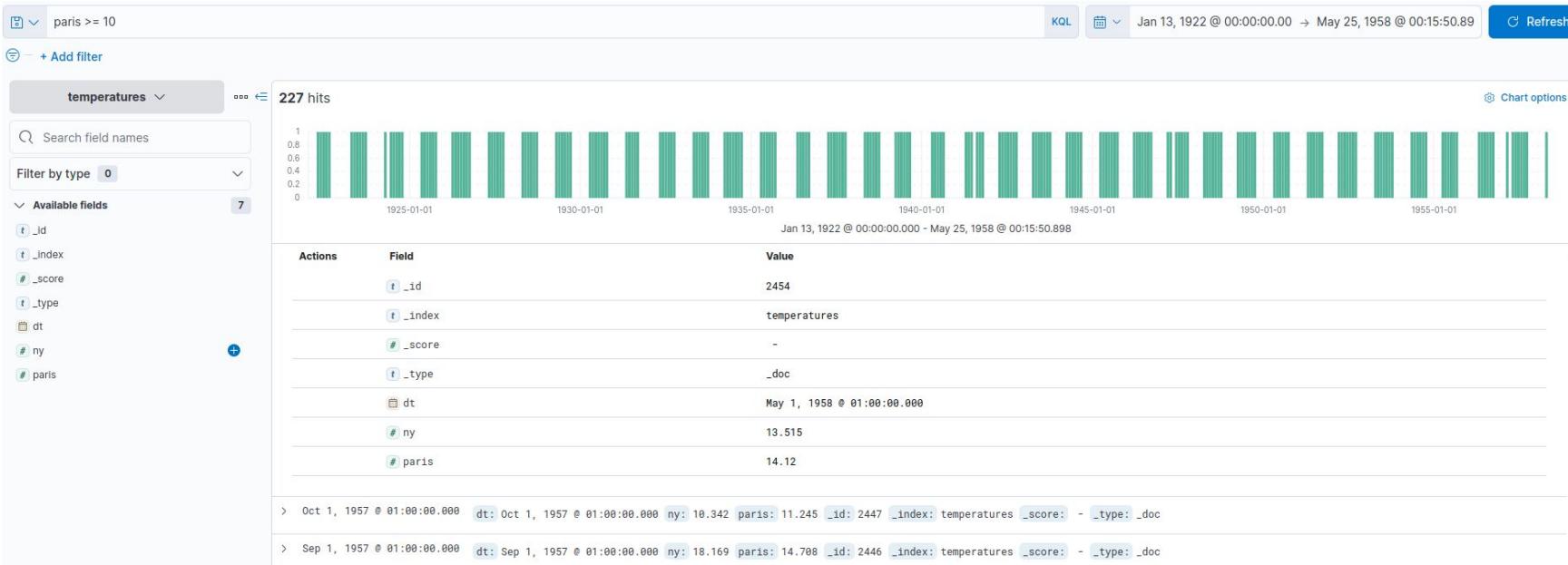
Elle vous permet de :

- Voir le flux de logs en temps réel.
- Filtrer par plage de temps (ex: "les 15 dernières minutes").
- Chercher en texte plein (ex: response: 404 ou "Login failed").
- Filtrer avec le langage KQL (Kibana Query Language) :
  - response: 404 AND geoip.country\_name: "France"
  - bytes > 5000
- Inspecter un document log en JSON vs. en Table.

# Discover

Kibana offre une interface user-friendly pour explorer vos données

Il fournit un langage de requête plus évolué (**KQL**), plus concis que les requêtes JSON



# Kibana Query Language (KQL)

---

**Présentation** : KQL (Kibana Query Language) est le langage de filtrage par défaut que vous utilisez dans la barre de recherche de Kibana.

**Objectif** : Conçu pour être simple, intuitif et facile à apprendre pour les utilisateurs qui ne sont pas experts en base de données.

**Auto-complétion** : Son principal atout ! Kibana vous suggère les champs (geoip.country\_name, response...) et les opérateurs (:, >, and...) pendant que vous tapez.

# KQL : Syntaxe

La syntaxe KQL se lit presque comme une phrase.

**Principe de base :** champ: valeur

**Texte exact :**

- response: 404 (Trouve les documents où le champ response est exactement 404)
- geoip.country\_name: "France" (Utilisez des guillemets " si la valeur contient des espaces, des majuscules ou des caractères spéciaux)

**Opérateurs Booléens :**

- and : response: 404 and verb: "POST"
- or : useragent.os.name: "Windows 10" or useragent.os.name: "Windows 11"
- not : not response: 200

# KQL : Comparaison et wildcards

## Comparaisons numériques (>, <, >=, <=) :

- bytes > 10000 (Trouve tous les logs où la taille de la réponse était supérieure à 10 000 bytes)
- response >= 400 (Trouve toutes les erreurs client et serveur)

## Wildcard (\*) :

- request: /api/v1/\* (Trouve toutes les requêtes API de la v1)
- useragent.name: Chro\* (Trouve "Chrome", "Chromium", etc.)

## Vérifier l'existence d'un champ :

- geoip.location: \* (Trouve tous les logs où l'enrichissement GeoIP a fonctionné et a ajouté un champ location)
- not geoip.location: \* (Trouve les logs où l'enrichissement a échoué ! Très utile pour le débogage.)

# KQL - Exercices

---

Sur l'index temperatures, afficher toutes entrées correspondantes aux critères suivants:

- Dates entre 1800 et 1815
- Temperatures à Paris comprises entre 10 et 15

Sur les données du titanic, afficher les hommes ayant survécu

# Étape 3 - Visualize

---

**Lens** est l'outil de visualisation moderne de Kibana. Il est conçu pour être rapide, intuitif et puissant (par "glisser-déposer").

**Le principe :**

- Choisissez votre type de graphique (camembert, barre, ligne, carte...).
- Glissez-déposez les champs que vous voulez analyser.
- Lens configure le graphique pour vous.

# Visualisation

La visualisation utilise les différents types d'agrégation supportés par ELK

- **Bucket** => ensemble de documents satisfaisant certains critères, peuvent être imbriqués
- **Metric** => Calcul des métriques sur un ensemble de documents, peuvent être combinés aux buckets
- **Pipeline** => Permet de chaîner des métriques

# Visualisation - Bucket

---

**Définition** : C'est un groupe de documents qui partagent un critère ou une caractéristique commune.

**Analogie** : Imaginez que vous avez une énorme boîte de briques LEGO de toutes les couleurs (vos documents).

- Faire une agrégation par "Termes" sur la couleur, c'est trier ces briques dans des seaux (buckets) séparés : un seau pour les rouges, un seau pour les bleues, un seau pour les jaunes.
- Chaque seau (bucket) contient un ensemble de documents.

# Visualisation - Types de bucket

**Terms (Termes)** : Un bucket par valeur unique (ex: response: 200, response: 404...).

**Range (Intervalle)** : Un bucket par intervalle défini (ex: bytes: 0-1000, bytes: 1001-5000...).

**Date Histogram** : Un bucket par intervalle de temps (ex: par minute, par heure, par jour...).

**GeoHash** : Un bucket par zone géographique (ce qui alimente les cartes dans Kibana).

# Visualisation - Metric

---

**Définition** : C'est un calcul mathématique effectué sur un ensemble de documents (Agrégation)

**Analogie** : Reprenons nos seaux (buckets) de briques LEGO triées par couleur.

- Poser la question "Combien y a-t-il de briques dans le seau rouge ?" est une métrique (count).
- Si chaque brique a un poids, "Quel est le poids moyen des briques dans le seau bleu ?" est une métrique (avg).
- "Quelle est la brique la plus lourde dans le seau jaune ?" est une métrique (max).

# Visualisation - Metrics courants

**Count** : Nombre de documents (souvent implicite).

**Sum** : Somme d'une valeur (ex: `sum(bytes)` pour la bande passante totale).

**Avg** : Moyenne d'une valeur (ex: `avg(bytes)` pour la taille moyenne des requêtes).

**Min / Max** : Valeur minimale ou maximale.

**Cardinality** : Nombre de valeurs uniques (ex: nombre d'adresses IP uniques).

# Pipeline

---

Le terme "Pipeline" (ou "chaîne de traitement") fait référence à une série d'étapes de traitement appliquées à une donnée.

Il y a deux contextes majeurs dans la stack ELK :

- Le Pipeline Logstash (le plus courant)
- Le Pipeline d'Ingestion (Elasticsearch)



# Pipeline Logstash

C'est le cœur de Logstash. Le fichier nginx.conf est un exemple parfait de pipeline. Il se compose de trois étapes :

1. **input** : D'où vient la donnée ? (Ex: de Filebeat, sur le port 5044).
2. **filter** : Que fait-on à la donnée ? C'est la transformation. (Ex: grok pour parser, geoip pour enrichir, mutate pour nettoyer).
3. **output** : Où va la donnée transformée ? (Ex: vers Elasticsearch).

# Pipeline ES

Concept similaire au pipeline Logstash, mais qui s'exécute directement sur un nœud Elasticsearch au moment de l'indexation (avant que le document ne soit stocké).

- Il est composé de processors (l'équivalent des filters de Logstash).
- Il est plus léger que Logstash, mais moins puissant.
- Les modules Filebeat (ex: filebeat modules enable nginx) utilisent souvent ces pipelines d'ingestion pour parser les données sans avoir besoin de Logstash.

# Étape 4 - Dashboard

Un Dashboard est une collection de visualisations (créées avec Lens) et de recherches (sauvegardées depuis Discover) sur une seule page.

- C'est le produit final que vous montrez à votre équipe, votre manager, ou votre client.
- Les dashboards sont interactifs : cliquer sur "France" dans un graphique filtrera tous les autres graphiques du dashboard pour ne montrer que les données de France.
- Ils sont partageables (PDF, PNG, lien web).

# Exercice de Synthèse : Analyse de Logs Web

- 
1. Générer des logs d'accès Nginx.
  2. Collecter ces logs avec Filebeat.
  3. Transformer et enrichir ces logs avec Logstash (Grok, GeolIP).
  4. Stocker les données structurées dans Elasticsearch.
  5. Visualiser les résultats dans un Dashboard Kibana.

# Conclusion

Name \_\_\_\_\_

Signature \_\_\_\_\_

Date \_\_\_\_\_



# Synthèse

Au cours de cette formation, nous n'avons pas seulement appris la théorie. Nous avons construit un pipeline de logs complet et réaliste.

- **Collecte** (Filebeat) : Nous avons collecté les logs Nginx bruts, de manière fiable et légère, depuis leur source.
- **Transformation** (Logstash) : Nous avons utilisé la puissance de Logstash pour :
  - Parser le texte brut avec grok.
  - Enrichir les données avec geoip (localisation) et useragent (appareil).
  - Nettoyer et structurer les données.
- **Stockage & Analyse** (Elasticsearch) : Nous avons indexé ces données structurées dans un moteur de recherche ultra-rapide, les rendant interrogables en temps réel.
- **Visualisation** (Kibana) : Nous avons exploré nos logs avec KQL et créé un Dashboard (avec Lens) qui transforme des lignes de log en insights décisionnels (cartes, graphiques...).

# À retenir

**Beats** : L'agent de collecte léger et fiable à la source.

**Logstash** : Le pipeline ETL (input, filter, output) pour le parsing et l'enrichissement complexes.

**Elasticsearch** : Le cœur de la stack. Un moteur de recherche JSON distribué. On y gère :

- index / \_mapping : Le "comment" les données sont stockées.
- Query DSL / Agrégations (bucket, metric) : Le "comment" on interroge les données.

**Kibana** : La fenêtre sur vos données.

- KQL : Votre outil d'exploration quotidien dans Discover.
- Lens : L'outil le plus simple pour créer des visualisations.