Faculté des Sciences Agadir

Master d'excellence-Analytique des données
et intelligence artificielle

Module :

SYSTEMS REPARTIS & DISTRIBUES

Projet N°7

Moteur de recherche distribué

Réalisé par :

- KINAD Kawtar
- MOUSTIKE Imane
- KHAIR Latifa
- KENNOUZ Ayoub

Encadré par :

• Pr. Idraiss Jaafar

Année académique : 2024-2025

Table des Matières

Introduction

Contexte et Problématique

À l'ère du Big Data, où plus de **328,77 millions de téraoctets** de données sont générés chaque jour (*Statista, 2024*), les solutions de recherche centralisées montrent des limites criantes en termes de performance et de scalabilité. Les architectures traditionnelles peinent à gérer des requêtes complexes sur des volumes massifs, avec des temps de réponse inacceptables (>1s pour 10+ millions de documents) et des risques élevés de panne.

Les **systèmes distribués** émergent comme la réponse à ces défis. En répartissant le traitement sur plusieurs nœuds, ils garantissent :

- Une **latence optimale** (<200 ms même sous forte charge)
- Une **scalabilité horizontale** (ajout transparent de nœuds)
- Une **haute disponibilité** (réplication automatique des données)

Objectif du Projet

Ce projet vise à concevoir un **moteur de recherche de nouvelle génération** combinant :

- **Elasticsearch 8.x** pour l'indexation distribuée et la recherche full-text
- **Apache Solr 9.x** comme alternative comparable
- **Kibana** pour la visualisation avancée des données
- **Logstash** pour l'ingestion et le prétraitement
- **Java/Python** pour l'intégration système

- ***Apache Solr*** est mentionné à titre de comparaison mais ne sera **pas utilisé dans l'implémentation finale** afin de simplifier l'architecture et de favoriser l'intégration fluide des outils ***Elastic***.

Valeur Ajoutée

Contrairement aux solutions existantes, notre implémentation apportera :

1. Un mécanisme d'**auto-scaling** basé sur la charge
2. Des **connecteurs optimisés** pour des sources hétérogènes
3. Une **API unifiée** Java/Python pour une intégration simplifiée

Structure du Document

Ce cahier des charges organise notre démarche en 5 axes :

1. Spécifications fonctionnelles
2. Architecture technique
3. Plan de développement
4. Métriques de succès
5. Planning et livrables

Ce document expose de manière détaillée les objectifs fonctionnels et techniques de notre projet, ainsi que le plan de développement prévu pour sa mise en œuvre

> ***"Dans l'économie de la connaissance, la capacité à retrouver instantanément l'information pertinente crée un avantage compétitif décisif. Notre solution positionne cette capacité à l'ère du pétaoctet."***

Spécifications Fonctionnelles

Le moteur de recherche distribué que nous allons concevoir doit répondre aux besoins d'**indexation rapide**, de **recherche efficace**, et de **visualisation claire** des données dans un environnement massivement distribué. Les fonctionnalités attendues se répartissent comme suit :

Fonctionnalités principales

1. **Ingestion des données**
    - Le système doit permettre **l'importation de données structurées et non structurées** (ex. : fichiers texte, JSON, logs).
    - Il doit supporter la **mise à jour automatique des données indexées** en cas de modification des sources.
    - Les sources peuvent inclure des **API publiques dynamiques** (ex. : News API, Reddit API ,Wikipedia API...) pour simuler un flux de contenu en temps réel.

2. **Indexation distribuée**
    - Les documents doivent être **analysés et indexés automatiquement** via un moteur distribué (ex : Elasticsearch ou Solr).
    - L'indexation doit être répartie sur plusieurs nœuds pour assurer la **scalabilité horizontale**.

3. **Recherche distribuée**
    - L'utilisateur doit pouvoir soumettre des requêtes (mots-clés, phrases, filtres).
    - Le système retourne les **résultats les plus pertinents en temps réel**, même sous charge élevée.
    - Les résultats doivent inclure : **titre, extrait, score de pertinence, date**.

4. **Interface de recherche**

   - Une interface simple (ou ligne de commande) doit permettre à l'utilisateur de saisir une requête et de visualiser les résultats.
   - Une interface de **dashboard (via Kibana)** présentera les statistiques liées aux recherches, aux performances du système, etc.
   - Dans un premier temps, **Kibana** fera office d'interface utilisateur pour interagir avec les données indexées. Une **interface personnalisée (web)** développée en **Python (Flask)** pourra être ajoutée dans une seconde étape si le temps le permet.

5. **Visualisation & tableaux de bord**

   - Les données indexées seront **visualisées dans Kibana** sous forme de graphiques, histogrammes, nuages de mots...
   - Des **dashboards personnalisables** doivent pouvoir être créés pour le monitoring de l'activité.

Cas d'utilisation

Pour illustrer les fonctionnalités du moteur de recherche distribué, voici quelques cas d'usage représentatifs :

- **Cas d'utilisation 1 – Recherche simple**
  Un utilisateur accède à l'interface de recherche, saisit un mot-clé, et obtient instantanément une liste de documents pertinents avec des extraits et la date de création.

- **Cas d'utilisation 2 – Indexation automatique**
  Un administrateur ajoute un nouveau fichier de logs dans le dossier source. **Logstash** détecte le fichier, le prétraite, et l'envoie automatiquement à **Elasticsearch** pour indexation.

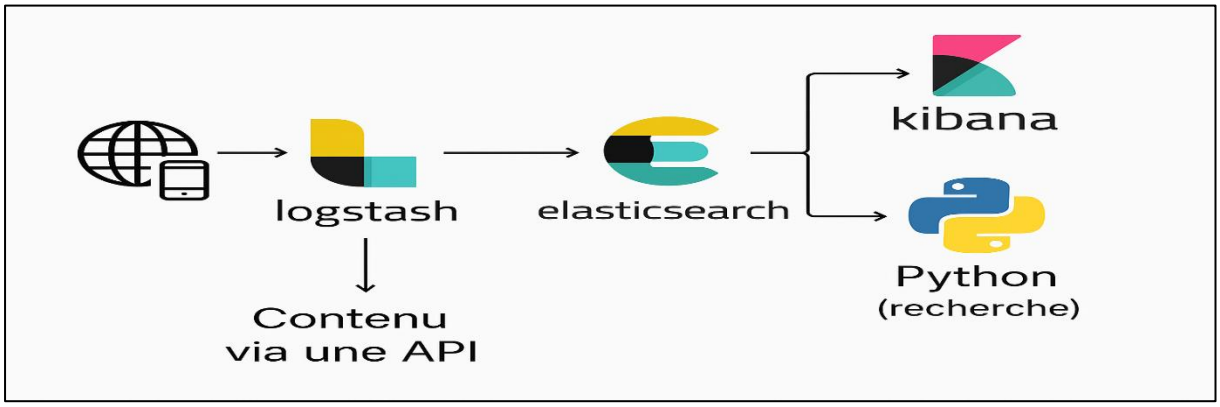
- **Cas d'utilisation 3 – Visualisation dans Kibana**
  Un analyste consulte les statistiques d'utilisation du moteur via **Kibana** : nombre de requêtes traitées, temps de réponse moyen, et documents les plus consultés.

Architecture Technique

L'architecture proposée repose sur un ensemble de composants distribués, interconnectés, qui assurent une ingestion efficace des données, une indexation scalable, une recherche rapide et une visualisation intuitive. L'ensemble du système est pensé pour supporter de grandes volumétries tout en restant résilient.

Schéma global de l'architecture

Ce schéma illustre l'architecture globale du moteur de recherche distribué, depuis l'ingestion des données jusqu'à la visualisation des résultats via Kibana :



Composants principaux

Composant	Rôle principal
Logstash	Ingestion des données (fichiers logs, JSON, CSV...) et prétraitement avant indexation.
Elasticsearch	Moteur de recherche distribué chargé d'indexer et de traiter les requêtes full-text.
Kibana	Interface de visualisation pour les données indexées et les statistiques.
API Java/Python	Permet l'intégration et l'interrogation du moteur depuis des systèmes tiers.
Cluster de nœuds	Répartition des charges entre plusieurs nœuds Elasticsearch pour assurer la scalabilité et la résilience.

Flux de données

1. Les données sources (ex. : fichiers logs) sont détectées automatiquement par Logstash.
2. Elles sont nettoyées, transformées et envoyées à Elasticsearch.
3. Elasticsearch les indexes et les rend disponibles à la recherche.
4. Kibana permet de visualiser les données et les statistiques en temps réel.
5. Une API REST (Java ou Python) permet aux utilisateurs ou aux applications de soumettre des requêtes personnalisées.

Enchaînement des traitements – cycle complet de recherche

Le processus global de fonctionnement du moteur de recherche s'effectue en deux grands temps:

1. Phase d'Indexation (préparation des données)

Ce processus est lancé automatiquement ou périodiquement.

- Collecte des données via *Logstash* depuis des sources dynamiques (API REST, fichiers logs, etc.).
- Nettoyage et transformation des données au format JSON.
- Envoi des données à *Elasticsearch* pour indexation.
- Indexation distribuée dans *Elasticsearch* (répartition automatique entre nœuds et les données sont divisées en **shards** (partitions), et chaque shard peut être répliqué pour assurer la disponibilité et la résilience.).
- Les données deviennent disponibles pour la recherche.

2. **Phase de Recherche (à chaque requête utilisateur)**

Ce processus est déclenché à chaque fois qu'un utilisateur effectue une recherche.

- L'utilisateur saisit une requête (mot-clé ou expression).
- Cette requête est transmise via l'interface Kibana ou via une API REST personnalisée (Flask/Python).
- Elasticsearch analyse la requête (tokenisation, analyse linguistique, etc.).
- Le moteur recherche les correspondances les plus pertinentes dans les index.
- Les résultats sont retournés à l'utilisateur avec score, extrait, et date.
- Les statistiques de recherche sont mises à jour et consultables sur Kibana.

Considérations techniques

- **Scalabilité horizontale :** ajout de nœuds Elasticsearch à chaud.
- **Résilience :** réplication automatique des shards.
- **Tolérance aux pannes :** si un nœud tombe, un autre prend le relais.
- **Sécurité :** Authentification de l'accès à l'API et à Kibana (si nécessaire)

Plan de Développement

Le développement du moteur de recherche distribué se déroulera selon une approche itérative et incrémentale. Chaque étape vise à construire une version fonctionnelle du système, en assurant la qualité, la scalabilité et l'adaptabilité aux besoins utilisateurs.

Étapes clés du développement :

1. Configuration de l'environnement

Mettre en place les outils et l'infrastructure nécessaires pour le projet.

1.1. Mise en place du cluster Elasticsearch

- Installer **Elasticsearch** sur plusieurs nœuds (minimum 3 pour la haute disponibilité).
- Configurer le **cluster Elasticsearch** (nom, découverte des nœuds, etc.).
- Vérifier la connectivité et la communication entre les nœuds du cluster.

1.2. Installation de Logstash et Kibana

- Installer Logstash pour la gestion de l'ingestion des données.
- Installer Kibana pour la visualisation des données et le monitoring du cluster Elasticsearch.

1.3. Choix de l'environnement de développement

- Choisir entre Java ou Python pour le développement de l'API REST.
- Configurer l'environnement de développement (IDE, gestion de versions, etc.).

2. Ingestion et indexation

2.1. Création des pipelines Logstash

- Configurer Logstash pour ingérer des données depuis des **sources diverses** (fichiers JSON, API REST, fichiers logs, etc.).
- Définir les **filtrages**, **nettoyages** et **transformation** des données (par exemple, formatage des dates, extraction de champs, etc.).

2.2. Mapping et Indexation dans Elasticsearch

- Définir le **mapping** d'index pour chaque type de données (par exemple, texte, date, nombres).
- Indexer les données dans Elasticsearch à l'aide de Logstash.

2.3. Tests de scalabilité

- Tester l'ajout et la suppression de nœuds du cluster Elasticsearch.
- Vérifier l'équilibrage des données entre les nœuds après ajout/suppression.

3. Développement de l'API

3.1. Développement de l'API REST

- Concevoir une API RESTful pour permettre les requêtes utilisateur
- Choisir et implémenter les frameworks en Java (Spring Boot) ou en Python (Flask).

3.2. Fonctionnalités de l'API

- Implémenter des requêtes simples et recherches full-text (recherche par mot-clé, expression).
- Implémenter des filtres pour affiner les résultats (par exemple, date, catégorie, etc.).
- Gestion des erreurs et des codes de statut HTTP appropriés pour l'API.

3.3. Intégration avec Elasticsearch

- Créer des requêtes Elasticsearch à partir des entrées utilisateur (via l'API).

4. Visualisation dans Kibana

- Conception de dashboards personnalisés pour visualiser les données indexées dans Elasticsearch (par exemple, graphiques, diagrammes, etc.).
- Configurer Kibana pour suivre des requêtes populaires, temps de réponse, documents les plus consultés.
- Monitoring de la charge système et du cluster.

5. Tests et optimisation

- Scénarios de charge : simulation de milliers de requêtes.
- Analyse des performances : latence, disponibilité, consommation mémoire.
- Optimisations : sharding, caching, balance des nœuds.

6. Documentation & livrables

- Documenter l'architecture du système, l'API et les fonctionnalités.
- Fournir des instructions pour la configuration, le déploiement et l'extension du moteur de recherche.
- Créer un guide utilisateur sur l'utilisation du moteur de recherche et de l'API.
- Compilation du projet et packaging.
- Présentation finale du moteur (démo).

Métriques de succès

Les **métriques de succès** mesureront l'efficacité du moteur de recherche en fonction de plusieurs critères :

- **Temps de réponse des requêtes**

  Le système doit répondre aux requêtes en moins de 200 ms, même avec un grand volume de données, garantissant ainsi une expérience utilisateur rapide et fluide.

- **Scalabilité**

  Le moteur doit pouvoir ajouter de nouveaux nœuds au cluster sans perte de performance, assurant ainsi une montée en charge facile et efficace.

- **Précision des résultats**

  Les résultats de recherche doivent être pertinents et bien classés, avec un taux de pertinence mesuré par l'exactitude des résultats retournés pour diverses requêtes.

- **Disponibilité et résilience**

  Le moteur de recherche doit être disponible à tout moment, avec une tolérance aux pannes assurée par la réplication des données et la gestion automatique des défaillances des nœuds.

- **Utilisation des ressources**

  Le système doit gérer efficacement les ressources, en particulier la mémoire et le processeur, pour maintenir des performances optimales sans surcharge.