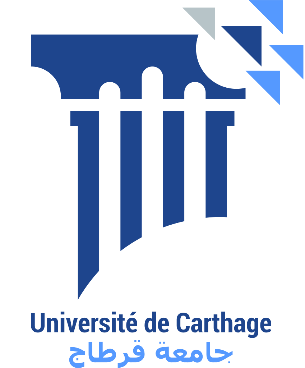
**République Tunisienne**

**Ministère de l’enseignement supérieur et de la Recherche scientifique**

**Université de Carthage**

**Institut des Hautes Etudes Commerciales de Carthage**





**Rapport de Projet**

Mastère : **Big Data Analytics & E-Commerce**

**Système de Surveillance et Analyse de Logs en temps réel pour Datacenter Hadoop**

**Réalisé par**

Hana Mzoughi & Senda Trifi

**Matière :**

* **Framework Avancés de développement Web**
* **Framework Avancés d’extraction et d’analyse des Big Data**

**Professeurs académiques :**

* **Mme Sondes Fayech**

**rR**

**Année universitaire :** 2024-2025

### ****Plan de Rapport pour l'Application****

#### ****1. Introduction****

1.1. Contexte du projet  
1.2. Problématique et objectifs  
1.3. Description générale de l'application

#### ****2. Analyse des Besoins****

2.1. Identification des utilisateurs cibles  
2.2. Spécifications fonctionnelles  
2.3. Contraintes techniques et non techniques

#### ****3. Architecture et Technologies****

3.1. Vue d'ensemble de l'architecture  
3.2. Technologies utilisées (Kafka,Pyspark, Spark Streaming, React, ELK, etc.)  
3.3. Justification des choix techniques

#### ****4. Développement de l'Application****

4.1. Backend

4.2. Frontend

#### ****5.Perspectives et Améliorations****

5.1. Fonctionnalités futures  
5.2. Optimisation des performances  
5.3. Extensions possibles de l'application

#### ****6. Conclusion****

6.1. Résumé des réalisations  
6.2. Impact attendu

### ****1. Introduction****

#### ****1.1. Contexte du projet****

Avec la croissance des systèmes complexes et le besoin accru d'automatisation, la surveillance et la gestion des données en temps réel sont devenues essentielles. Ce projet vise à concevoir une application de monitoring des logs Hadoop pour deux data center.  
L’objectif principal est d’assurer une détection rapide des erreurs critiques, d’envoyer des alertes en temps réel et de fournir des visualisations utiles pour la prise de décision.

#### ****1.2. Problématique et objectifs****

**Problématique** : Comment collecter, traiter et analyser efficacement les logs en temps réel afin d’identifier les erreurs critiques et de générer des alertes pertinentes ?

* **Objectifs** :
  + Capturer les logs des deux data centers.
  + Filtrer et traiter les erreurs critiques, notamment les blocs contenant des anomalies liées à Java.
  + Stocker les données pour une visualisation claire et exploitable.
  + Fournir un système d'alerte réactif en cas d’erreur. (envoie des emails et visualisation)

#### ****1.3. Description générale de l'application****

L'application repose sur une architecture Lambda qui combine à la fois le traitement par **batch** et en **temps réel**. intégrant des technologies modernes comme, Zookeeper, Kafka,Pyspark, Spark Streaming,l'ELK Stack et React. Elle permet de :

* Collecter les logs en temps réel via Kafka.
* Traiter et filtrer les logs avec Spark Streaming.
* Appliquer un pipeline Logstash utilisant des filtres Grok pour structurer les données de manière optimale avant leur indexation.
* Indexer les données dans Elasticsearch pour une visualisation à l'aide de Kibana.
* Fournir un tableau de bord interactif, développé avec React, intégré à Kibana pour la gestion des alertes et des statistiques en temps réel.

### ****2. Analyse des Besoins****

#### ****2.1. Identification des utilisateurs cibles****

Les principaux utilisateurs de cette application incluent :

* Les administrateurs système, responsables de la maintenance des data centers.
* Les ingénieurs en infrastructure, souhaitant des analyses détaillées des performances des serveurs.
* Les managers IT, recherchant des rapports synthétiques sur l’état des systèmes.

#### ****2.2. Spécifications fonctionnelles****

L’application doit :

* Collecter les logs de manière continue et fiable.
* Identifier les erreurs critiques et les structurer pour faciliter leur analyse.
* Stocker les logs normaux et erronés dans des indexes séparés.
* Générer des alertes en temps réel (affichage dans le frontend et notifications par e-mail).
* Permettre une visualisation claire des données à travers des tableaux de bord interactifs.

#### ****2.3. Contraintes techniques et non techniques****

* **Techniques** :
  + Gestion de grands volumes de données générées par les deux data centers.
  + Assurer une faible latence dans le traitement des logs en temps réel.
* **Non techniques** :
  + Simplicité d’utilisation pour les utilisateurs non techniques.
  + Disponibilité élevée du système pour garantir son efficacité en continu.

### ****3. Architecture et Technologies****

#### ****3.1. Vue d’ensemble de l’architecture****

#### L’architecture est composée de plusieurs couches :

1. **Collecte des logs via Kafka** : Les logs des deux data centers sont collectés et acheminés vers le système centralisé.
2. **Traitement en temps réel via Spark Streaming** : Les logs du data center 2, qui contient des erreurs critiques, sont traités en temps réel pour une détection rapide des anomalies.
3. **Traitement par batch pour le data center 1** : Les logs du data center 1, qui ne contiennent pas d'erreurs critiques, sont traités en mode batch afin d'optimiser les ressources.
4. **Indexation et stockage dans Elasticsearch** : Les données issues des traitements en temps réel et par batch sont indexées quotidiennement dans Elasticsearch, avec un index distinct par jour (format : datacenter2-YYYY-MM-DD) pour faciliter la gestion et l’analyse temporelle des logs.
5. **Visualisation et gestion via Kibana et une interface React** : L’architecture repose sur une approche Lambda, combinant le traitement batch et temps réel, pour offrir une visualisation complète et interactive des logs et des alertes grâce à Kibana et une interface React dédiée.

#### ****3.2. Technologies utilisées****

 **Kafka** : Collecte continue des logs avec gestion des topics.

**Pyspark :** pour le traitement batch et streaming.

 **Spark Streaming** : Filtrage et structuration des données en temps réel pour le data center 2.

 **Logstash** : Structuration et préparation des logs du data center 1 en mode batch grâce à des filtres adaptés, tels que Grok.

 **Elasticsearch** : Indexation rapide et recherche efficace pour les données issues des traitements en batch et en temps réel.

 **React** : Interface utilisateur moderne et intuitive pour la gestion des alertes et la visualisation des données.

#### ****3.3 Justification des choix techniques****

* **Kafka** pour sa scalabilité et sa fiabilité dans le traitement des flux de données.
* **Spark Streaming** pour sa capacité à traiter des volumes massifs en temps réel.
* **Elasticsearch** pour des performances optimales d'indexation.

**4. Développement de l’Application**

#### 4.1 Backend

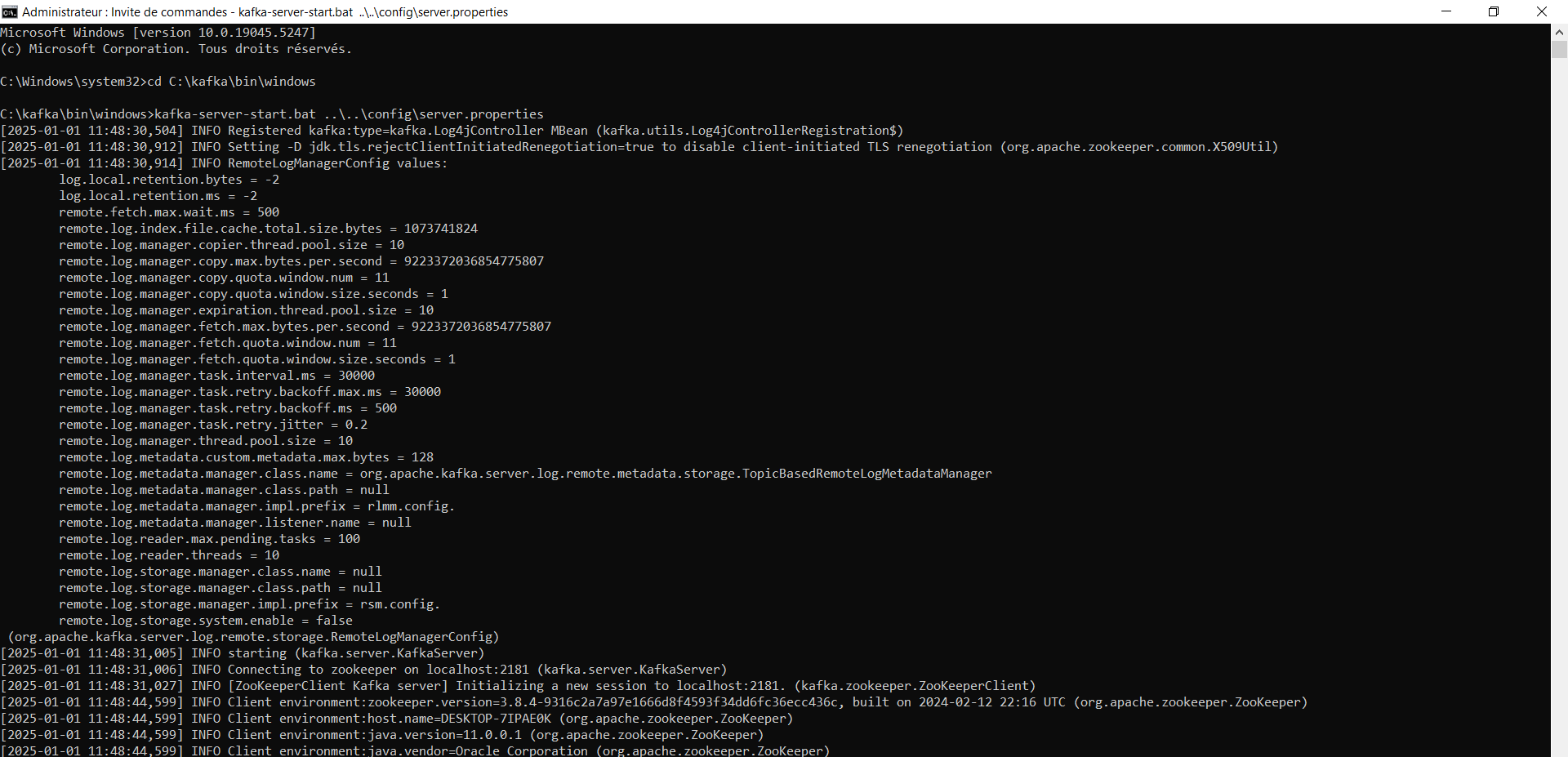
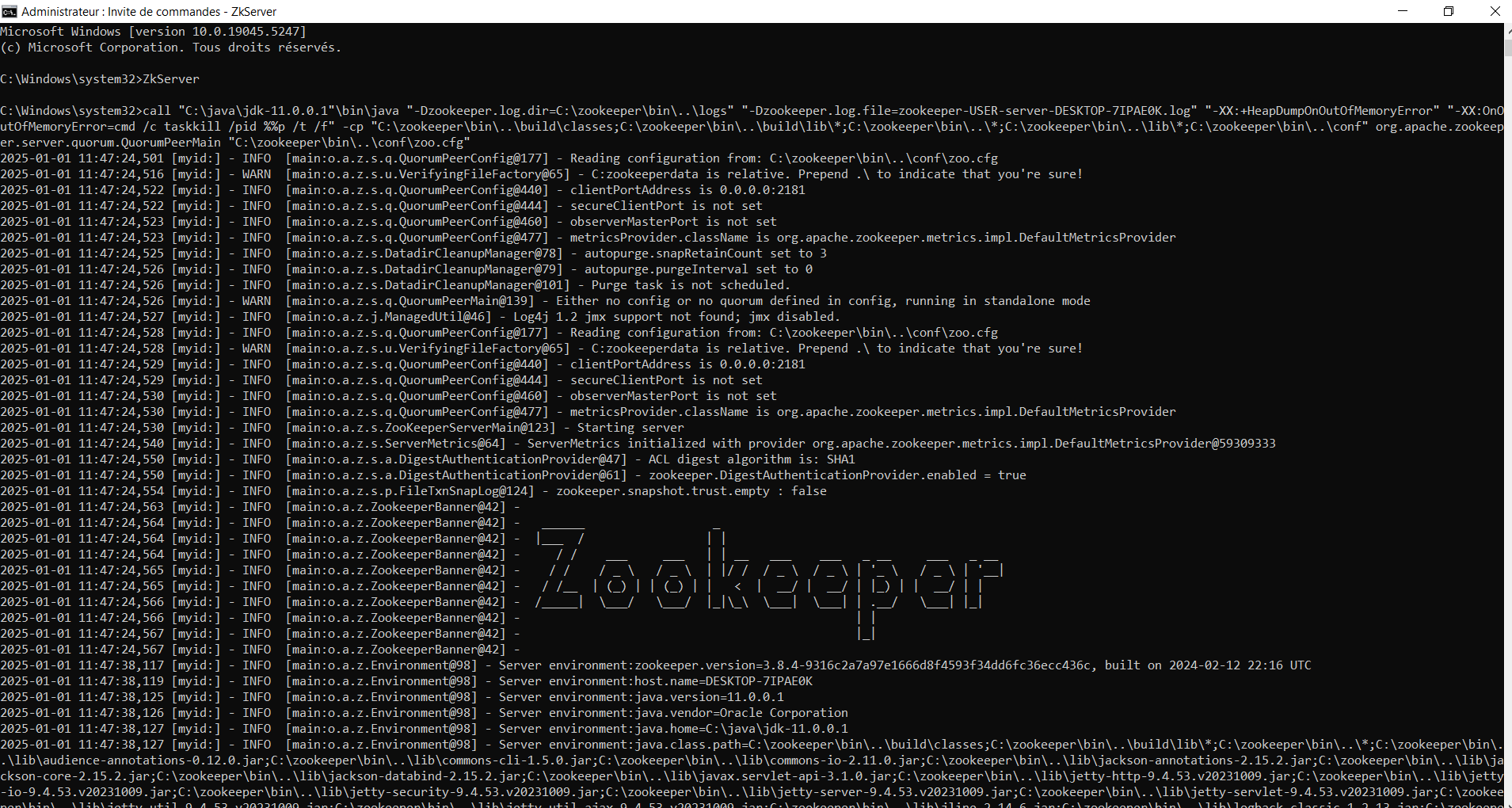
Le backend gère la collecte, le traitement et l’indexation des logs en temps réel. Il est conçu pour garantir un flux de données fluide et une intégration cohérente avec le frontend, en utilisant principalement Kafka, Spark Streaming ,logstash et Elasticsearch.

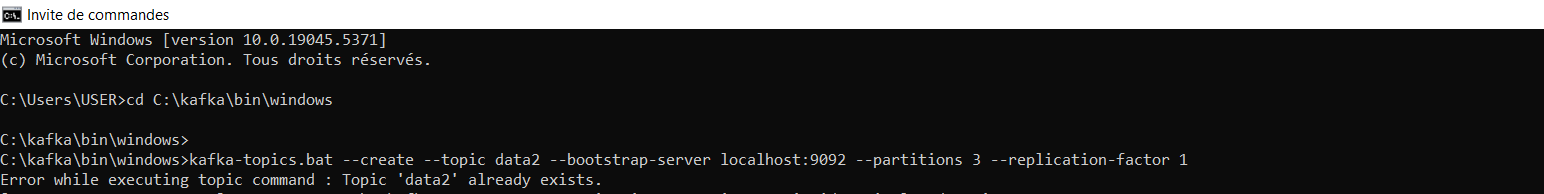
**Collecte des logs (Kafka) :**

* **Configuration de Kafka** :
  + Chaque data center a un topic Kafka dédié.
  + Les producteurs Kafka consomment les logs générés et les publient dans les topics correspondants.
* **Fonctionnalité clé :**
  + Kafka agit comme une file d’attente distribuée, assurant la fiabilité et la scalabilité des flux de données.

**Étapes générales d'initialisation de l'application :**

Le processus commence par le lancement du **Zookeeper server (ZkServer)**, essentiel pour la coordination des clusters Kafka. Ensuite, le service Kafka est démarré pour permettre la gestion des messages. Une fois Kafka opérationnel, les **topics nécessaires** au traitement des logs sont créés en fonction des besoins du projet. Enfin, une vérification des topics créés est effectuée en les listant pour s'assurer qu'ils ont été correctement configurés et sont prêts à recevoir des messages.







**Traitement des logs pour le data center 1**  
Pour le data center 1, qui ne contient pas d'erreurs critiques, le processus commence par la création d’un **producer Kafka** chargé de lire les logs depuis un fichier et de les stocker dans un topic Kafka dédié, nommé "data1". Une fois les logs stockés, un **consumer Kafka** consomme ces logs bruts et les envoie directement vers Elasticsearch sans aucune structuration initiale.

Lorsqu'ils atteignent Elasticsearch, un **pipeline Logstash** est déclenché pour structurer et indexer les logs par jour. Le pipeline inclut :

* **Input** : Extraction des logs bruts depuis l’index "logs" d’Elasticsearch.
* **Filter** : Structuration des données avec le filtre Grok pour extraire des champs clés comme la date, l’heure, le PID, le niveau de log (Level) et le message. Des transformations supplémentaires sont effectuées pour créer un champ @timestamp basé sur les champs Date et Time.
* **Output** : Les logs structurés sont réindexés dans un index quotidien, nommé "datacenter1-structured-logs-YYYY.MM.dd".

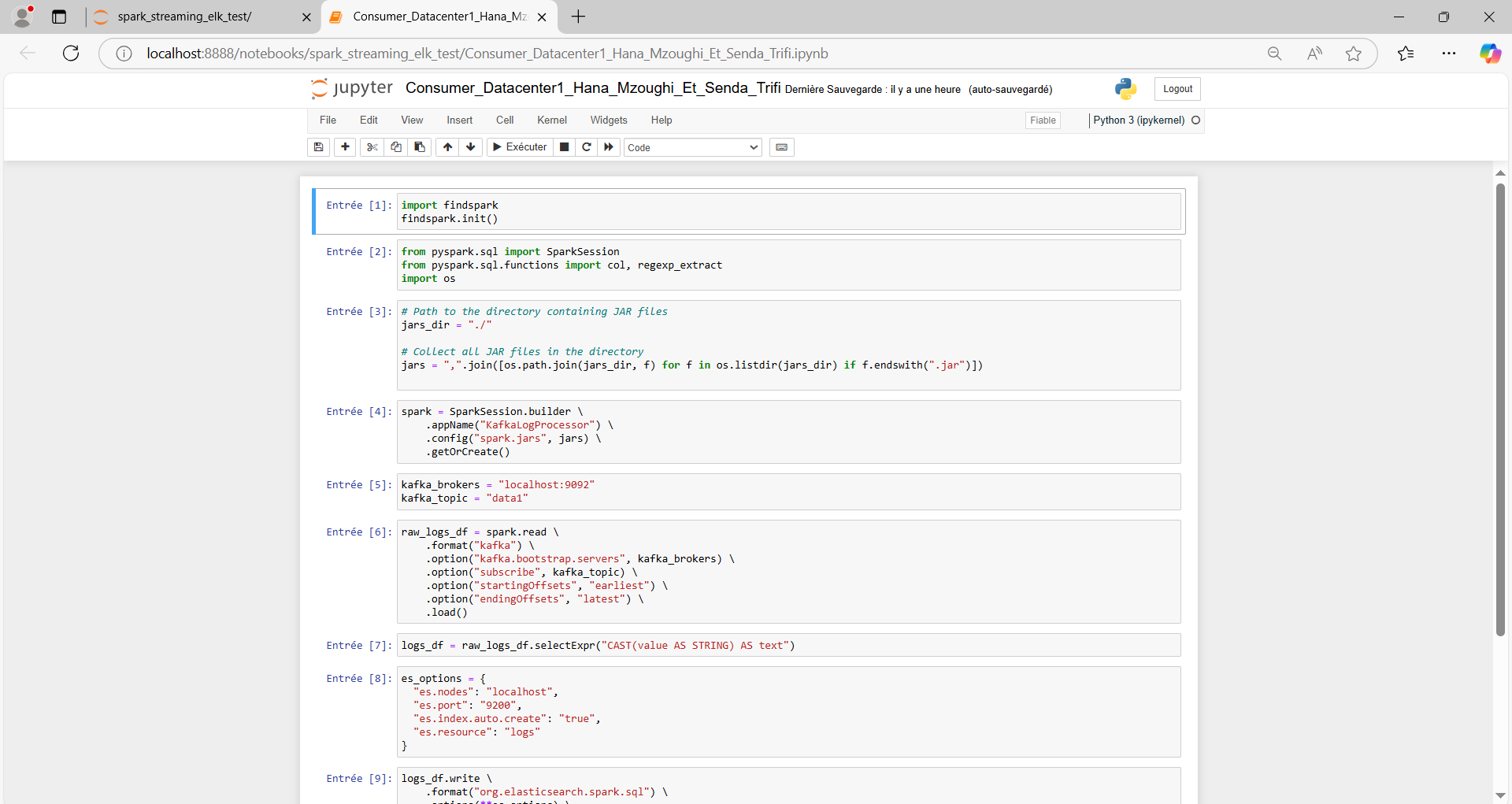
**Problèmes rencontrés avec Logstash et leur résolution**  
Lors de la mise en place du pipeline Logstash pour structurer les logs du data center 1, j’ai rencontré des problèmes liés à la consommation élevée des ressources, notamment la mémoire. Ces limitations ont entraîné des **timeouts fréquents** lors du traitement des données, ce qui impactait la performance globale du système.

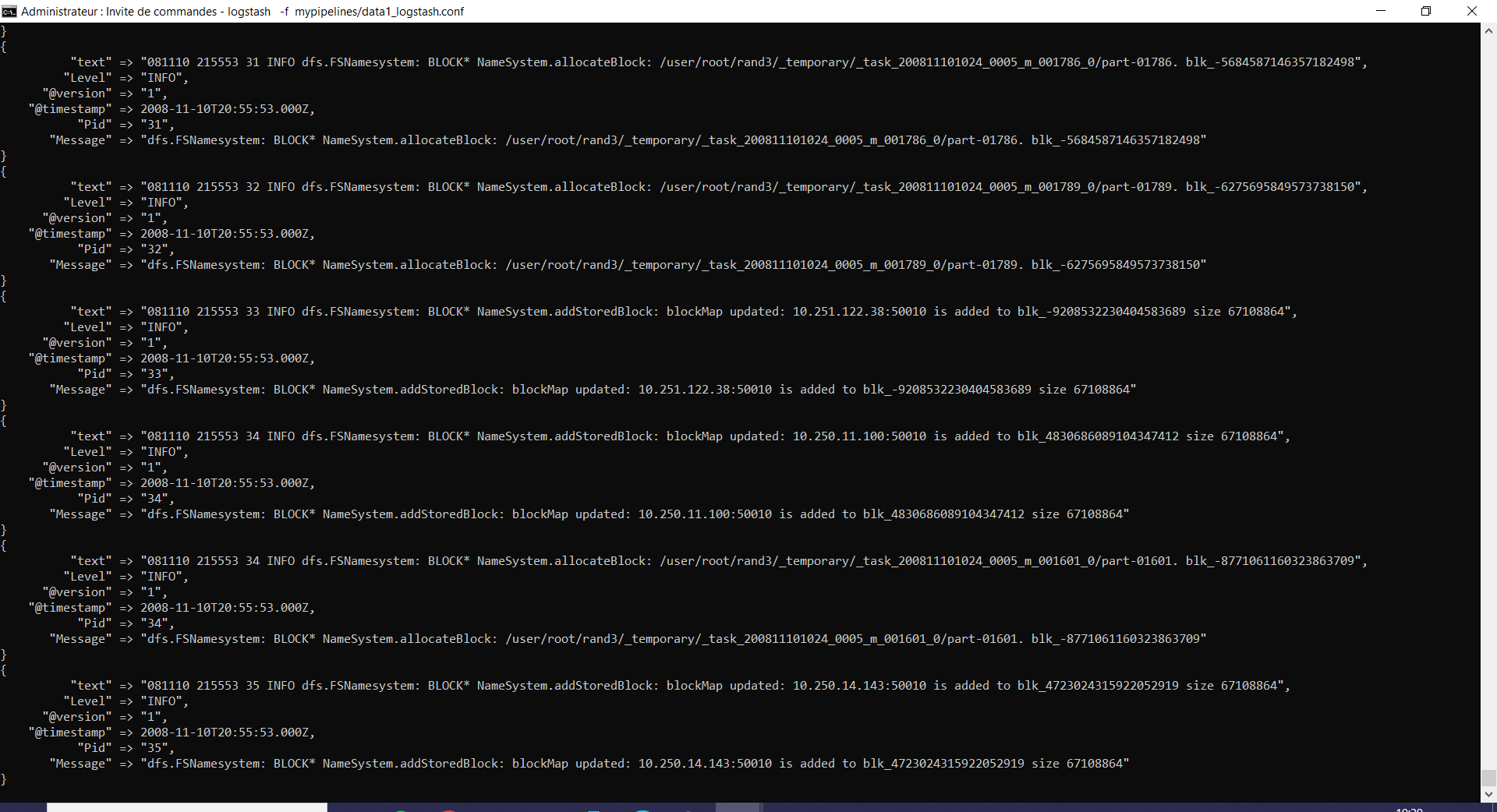
Après une analyse approfondie, j’ai identifié que la configuration par défaut de Logstash ne fournissait pas suffisamment de mémoire pour gérer efficacement le volume des logs à traiter. Pour résoudre ce problème, j’ai modifié le fichier de configuration **jvm.options** de Logstash afin d’augmenter la mémoire maximale allouée à la JVM. La valeur par défaut de 2 Go a été portée à **8 Go**, ce qui a considérablement amélioré les performances et réduit les erreurs de timeout.

Résultats de producer pour remplir le topic dans le consumer console :

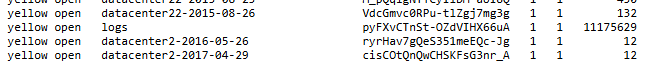


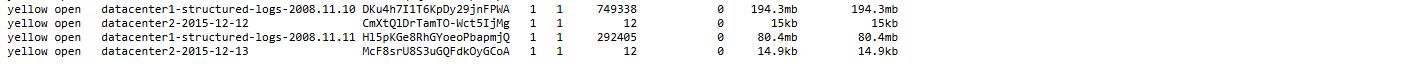
Consumer vers elastic :



Les réultats de lancement de logstash :

Le premier indexe créer par pyspark :

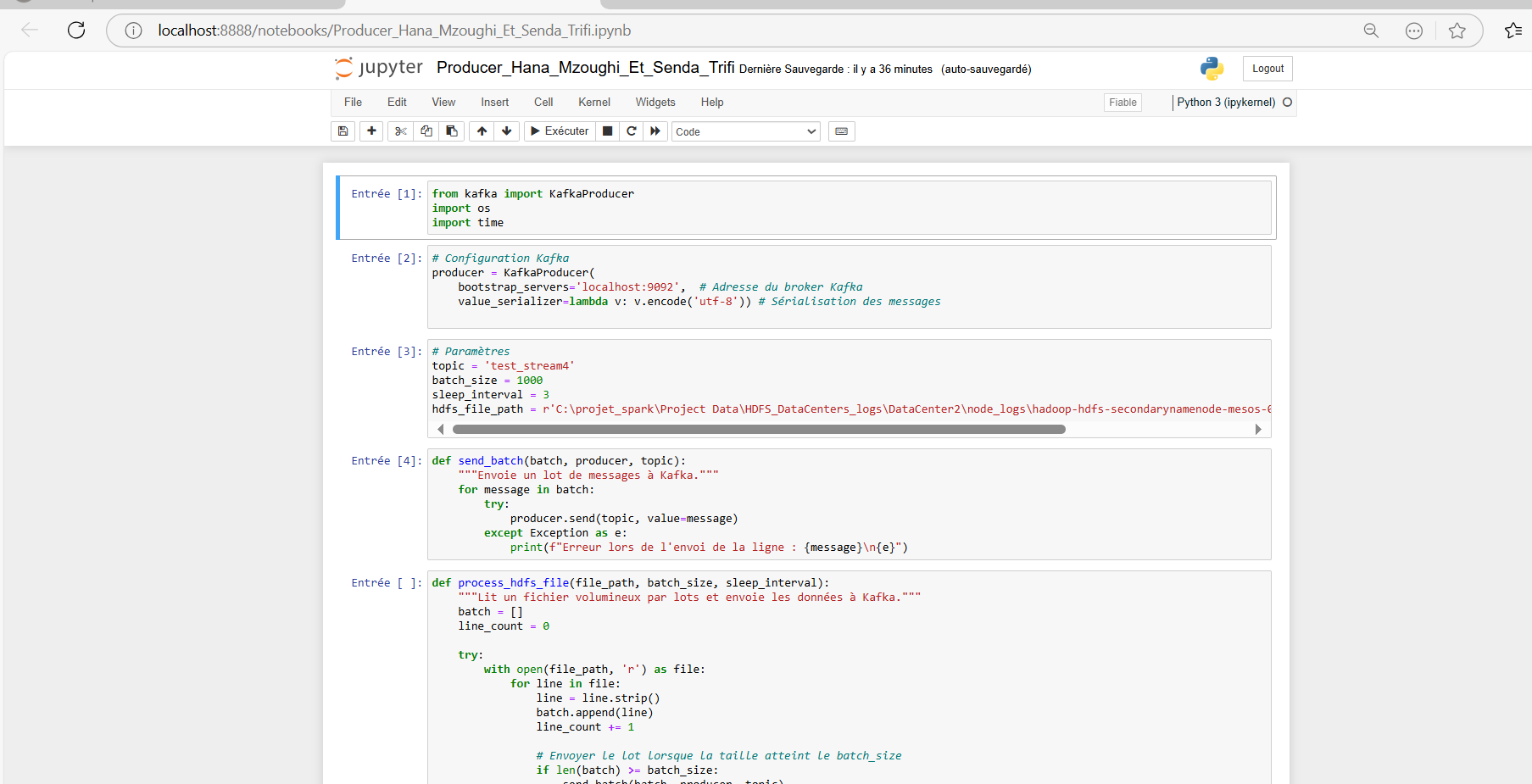


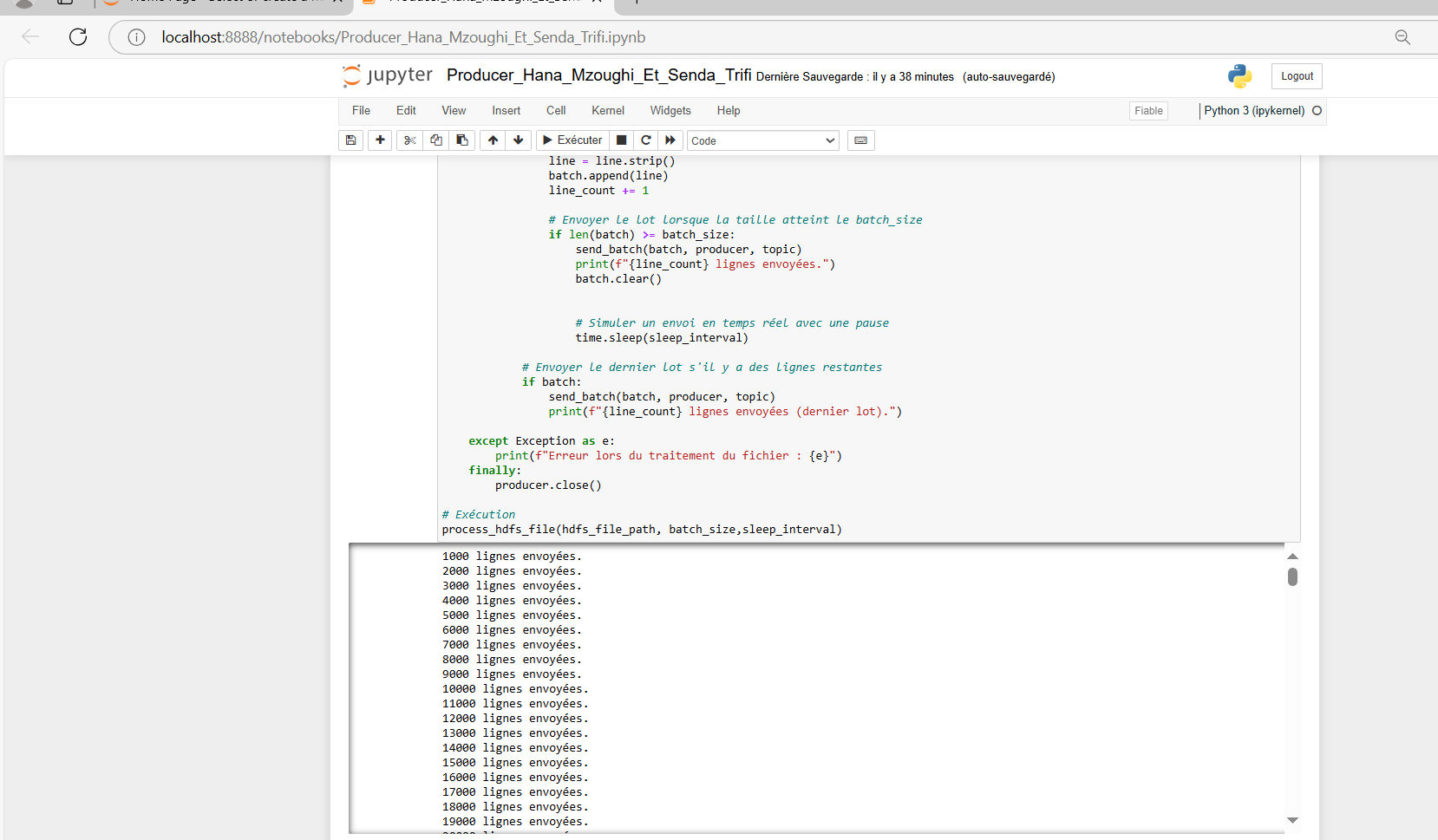
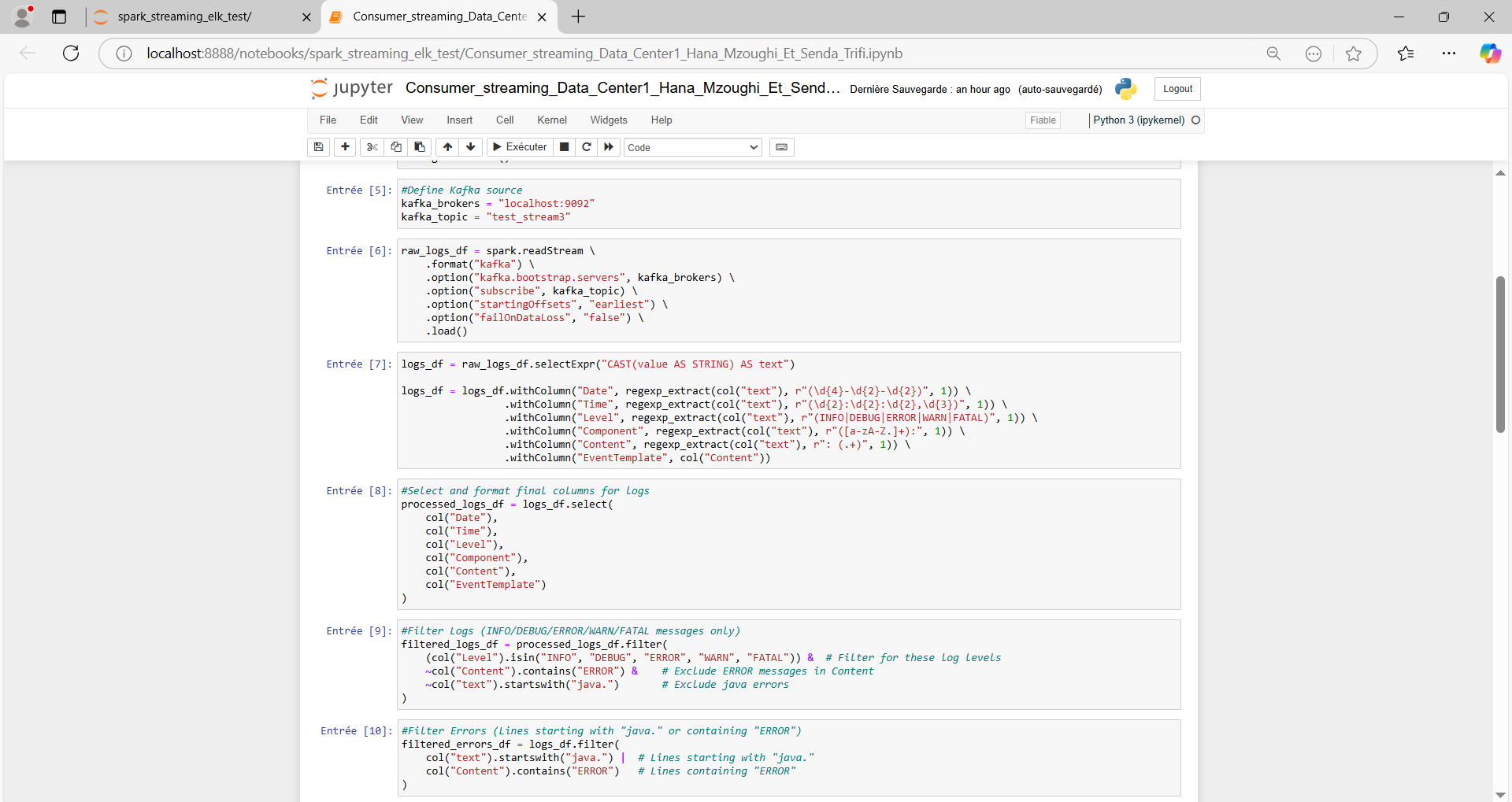
Les indexes créer par logstash :

**Traitement des logs pour le data center 2**  
Pour le data center 2, le processus commence par la création d’un **producer Kafka** et d’un **consumer**, comme pour le data center 1. Toutefois, la **structuration des logs** pour ce data center est effectuée à l’aide de **Spark Streaming**, car Logstash, étant plus gourmand en ressources, n'était pas adapté pour ce volume de données. Avec Spark Streaming, les logs sont traités en temps réel, permettant une détection rapide des erreurs critiques.

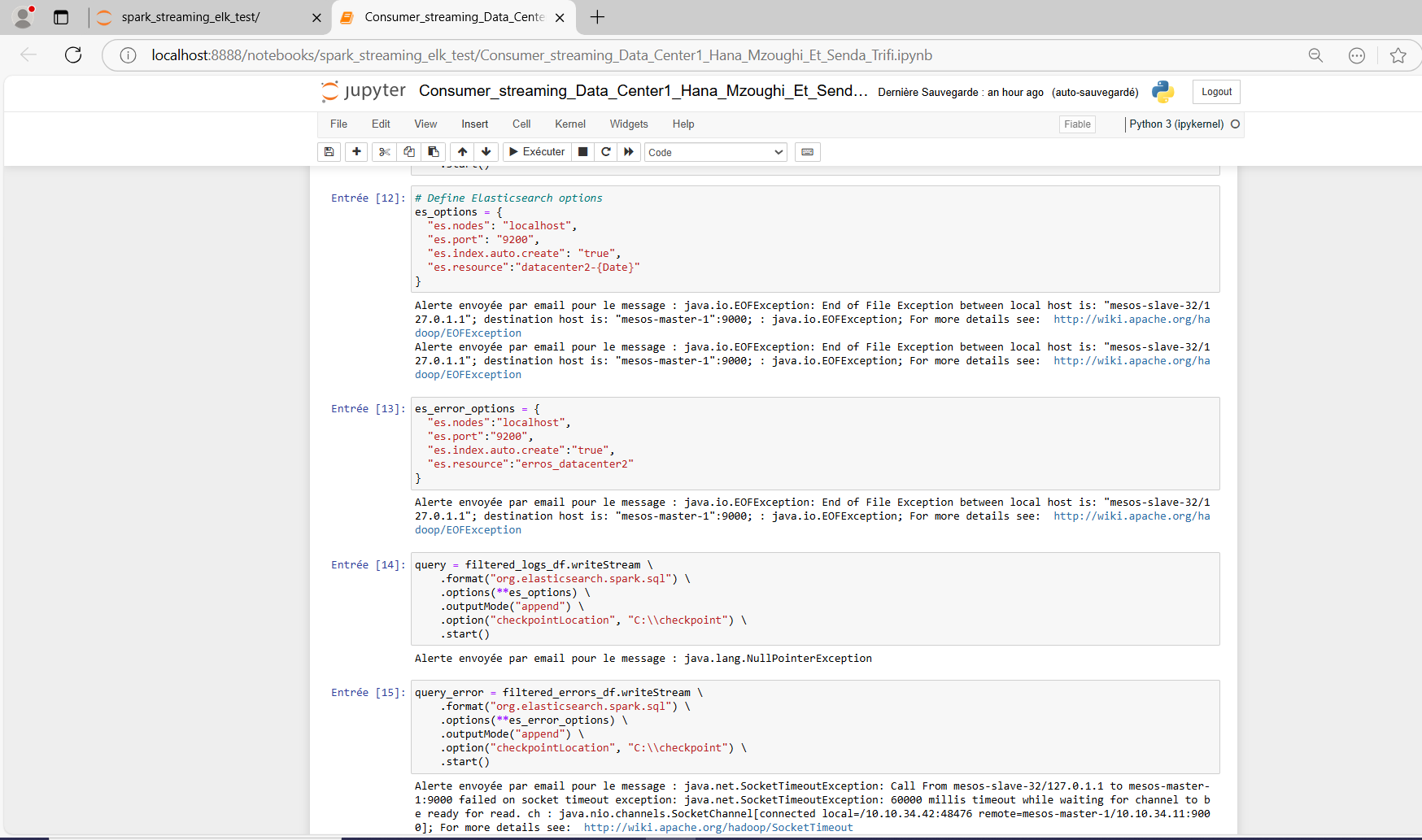
Lors de ce traitement, les erreurs sont **filtrées** et envoyées par **email** pour permettre une intervention rapide. En parallèle, les données sont **indexées quotidiennement** dans Elasticsearch, avec un index distinct pour chaque jour (par exemple, datacenter2-YYYY-MM-DD). Cette approche garantit que les logs sont structurés et stockés de manière optimale pour une analyse et une visualisation efficaces.

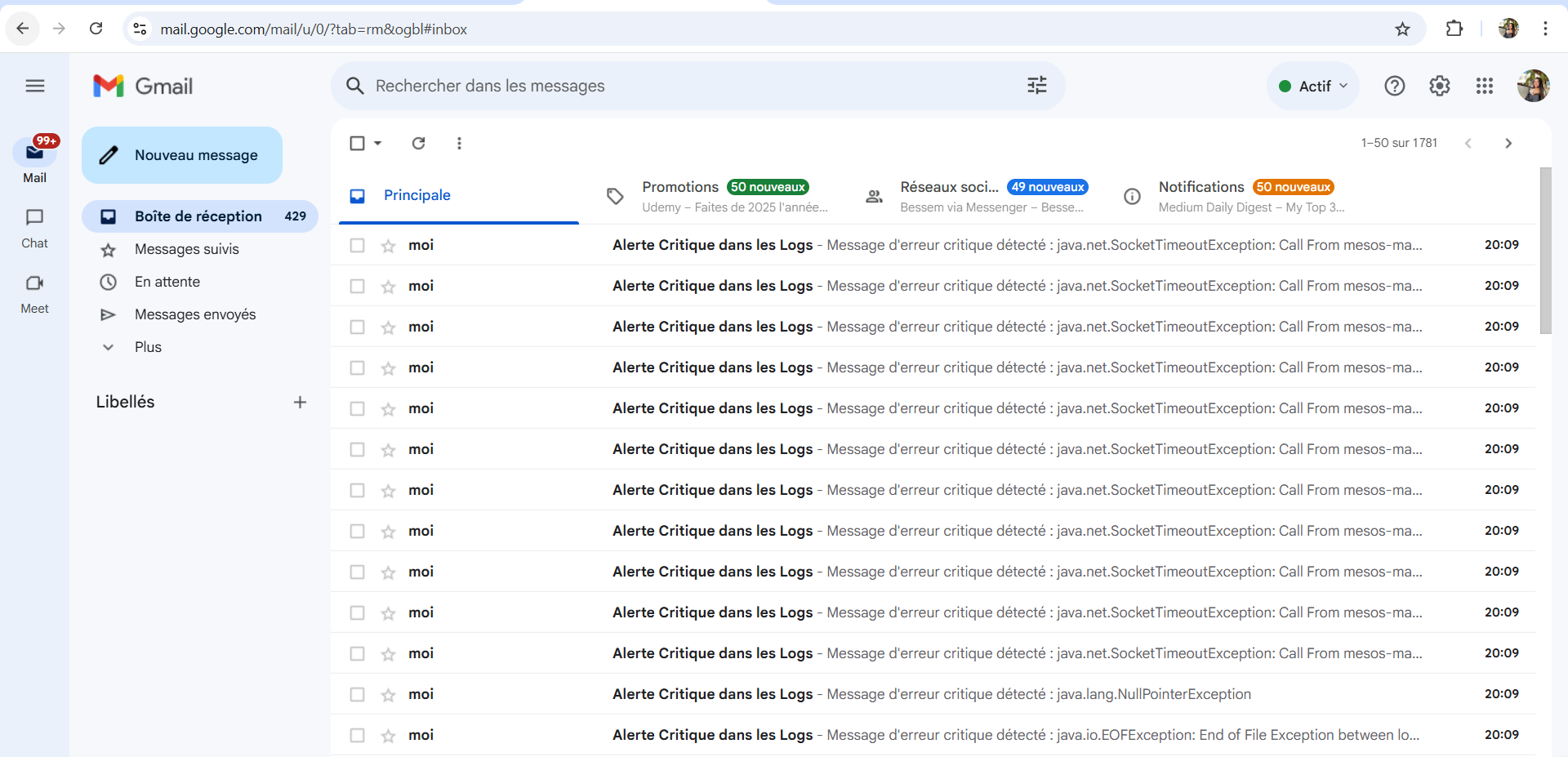
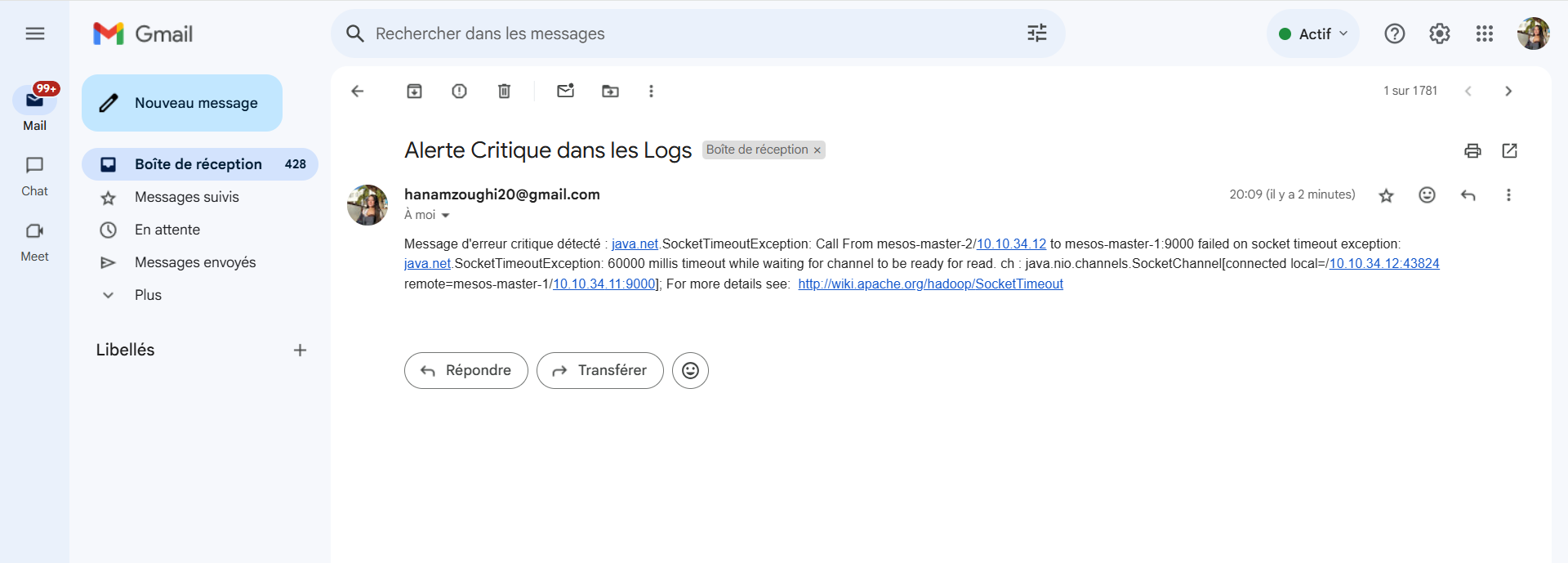
Ce processus en temps réel permet une gestion réactive des erreurs et assure que les logs sont constamment mis à jour et prêts pour l’analyse dans Kibana.

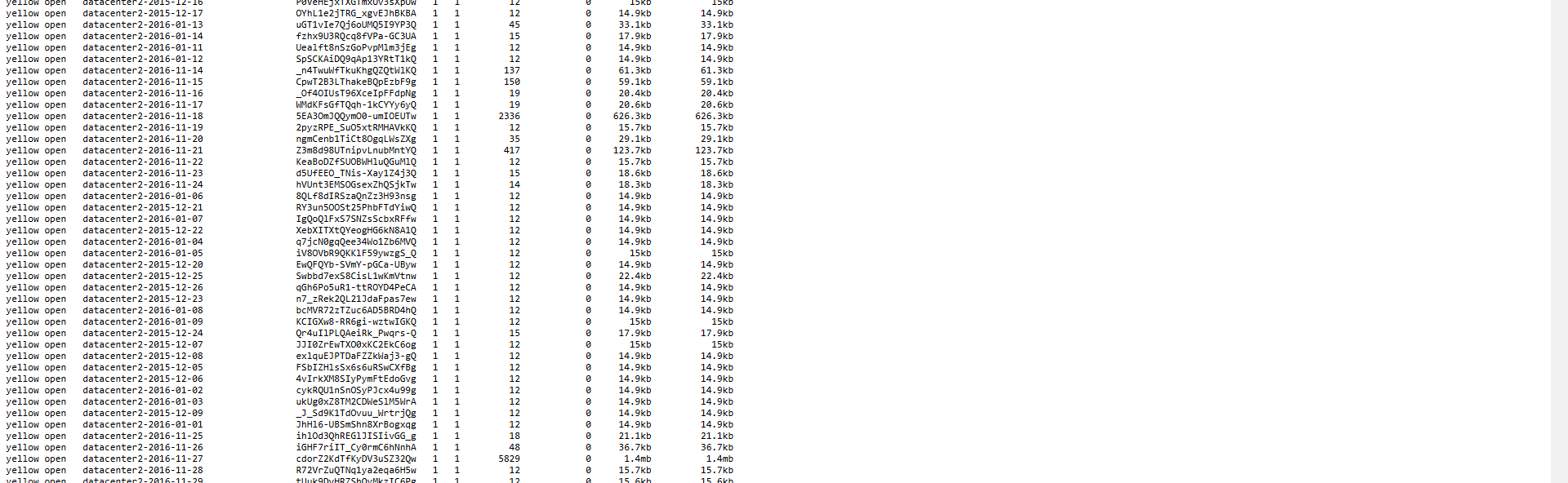










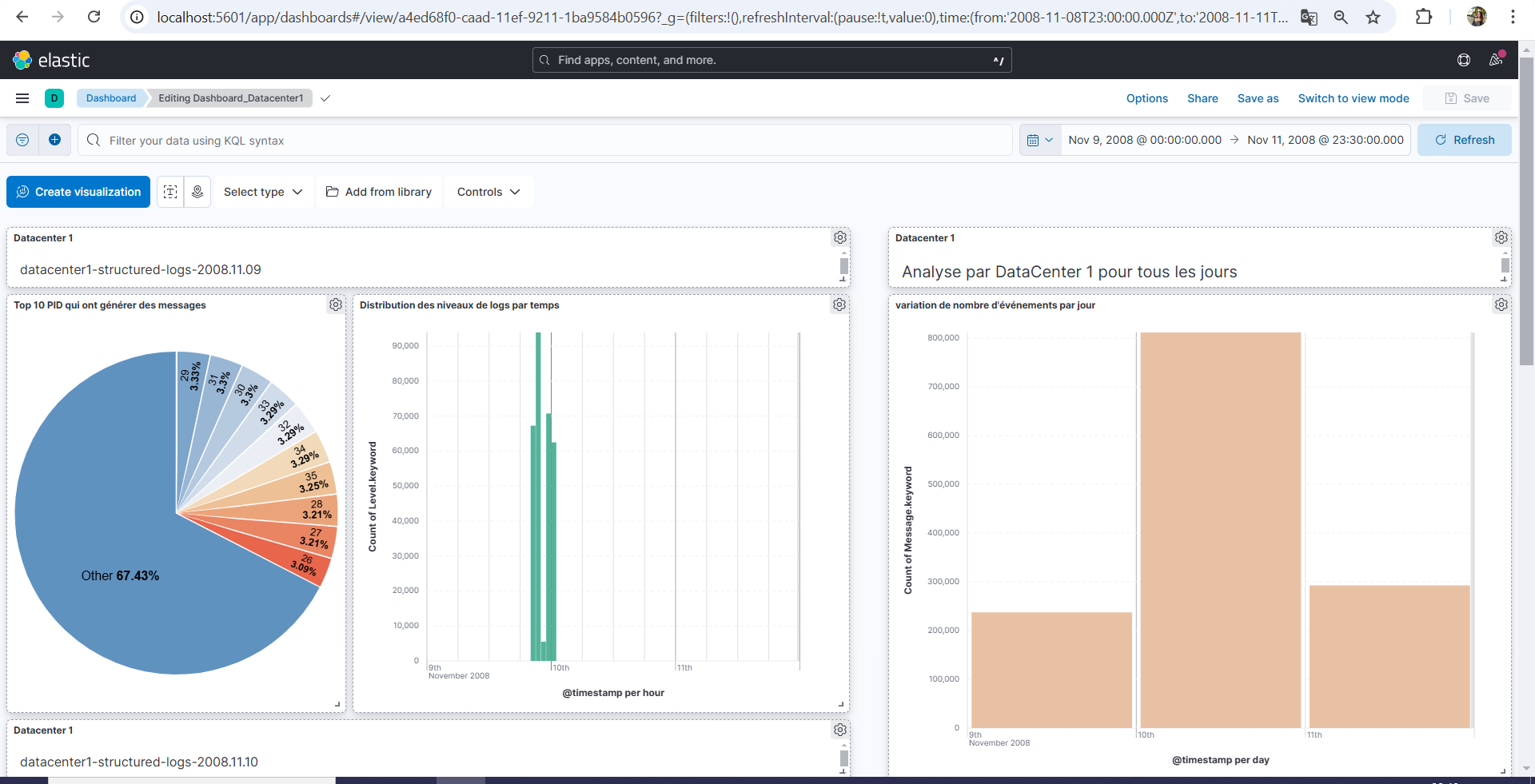


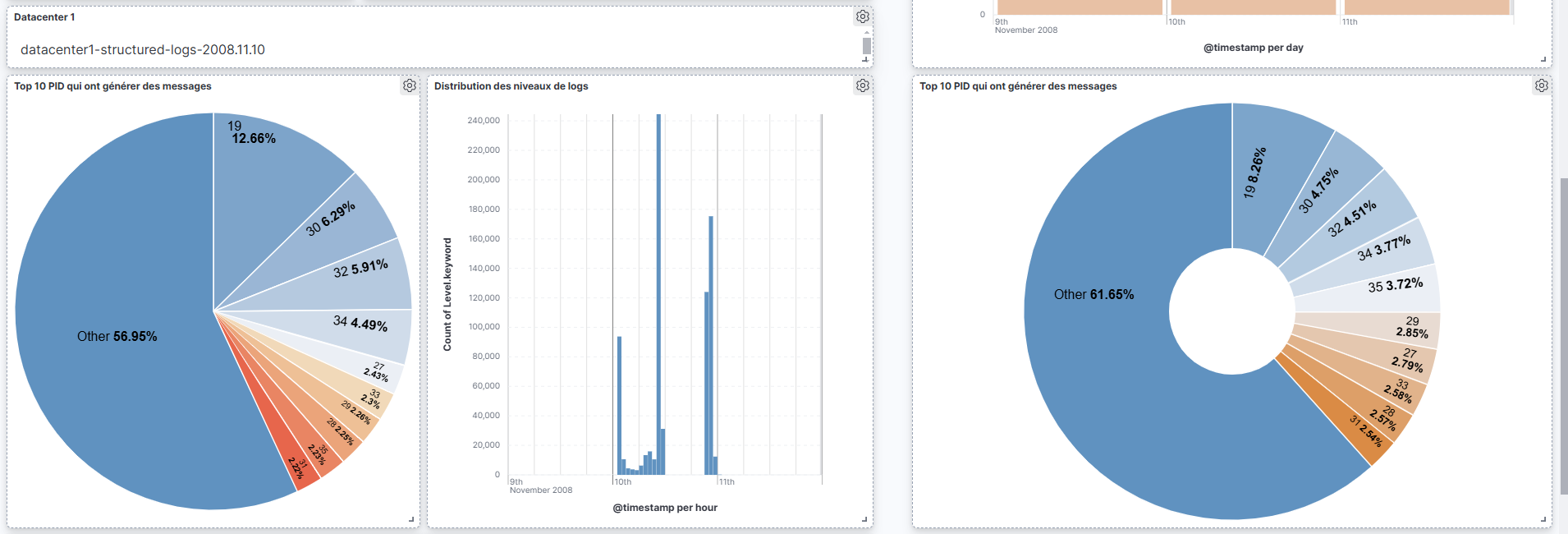
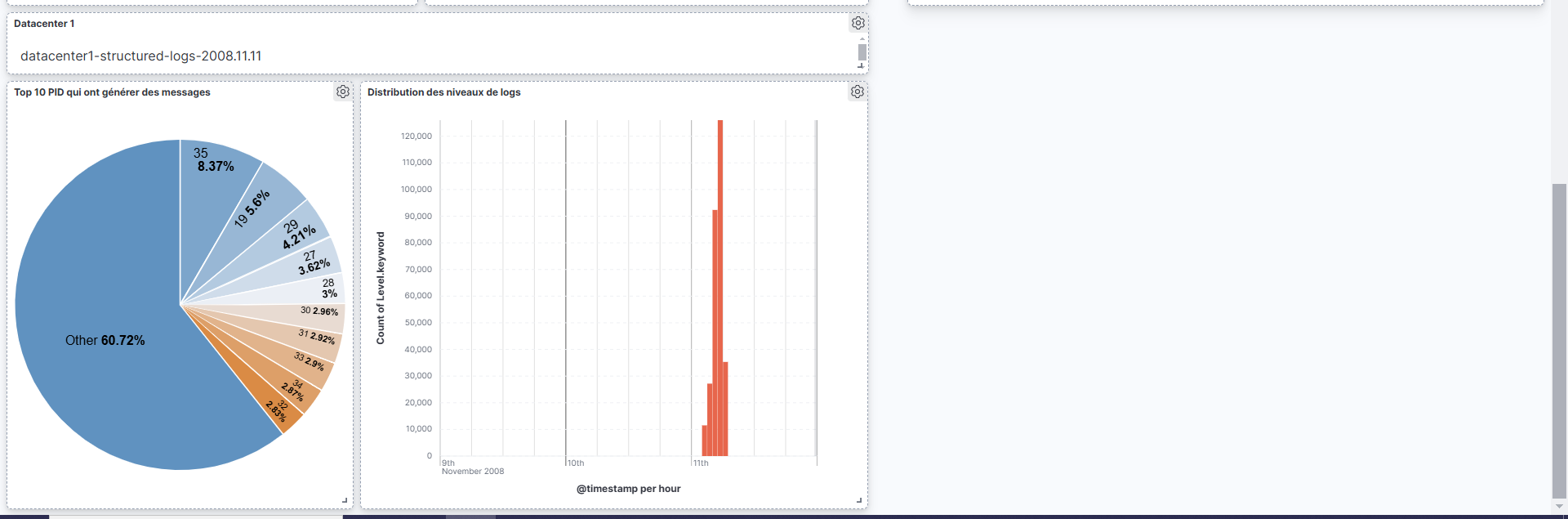
#### ****4.2. Frontend****

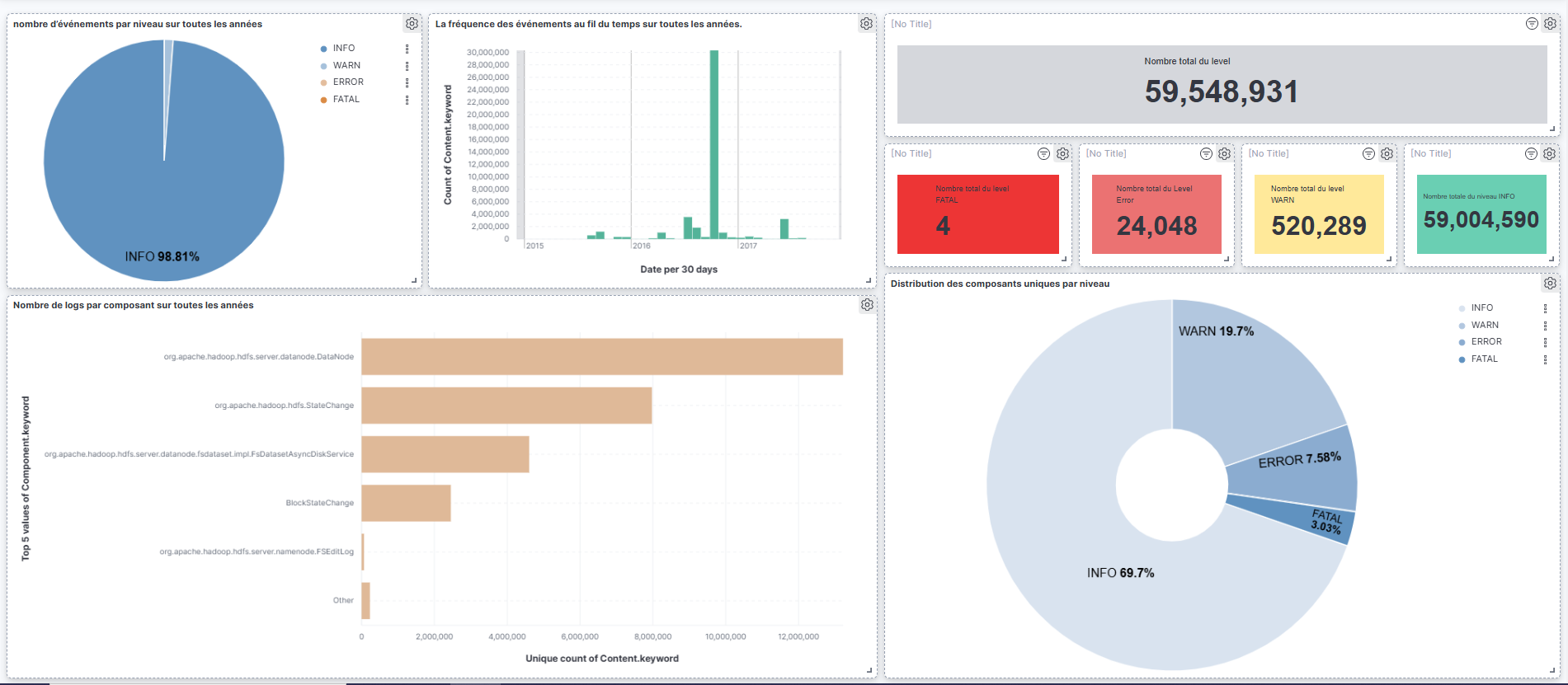
* Création d’un tableau de bord interactif avec Reac et Kibana.
* Système de notification en temps réel pour les alertes critiques.

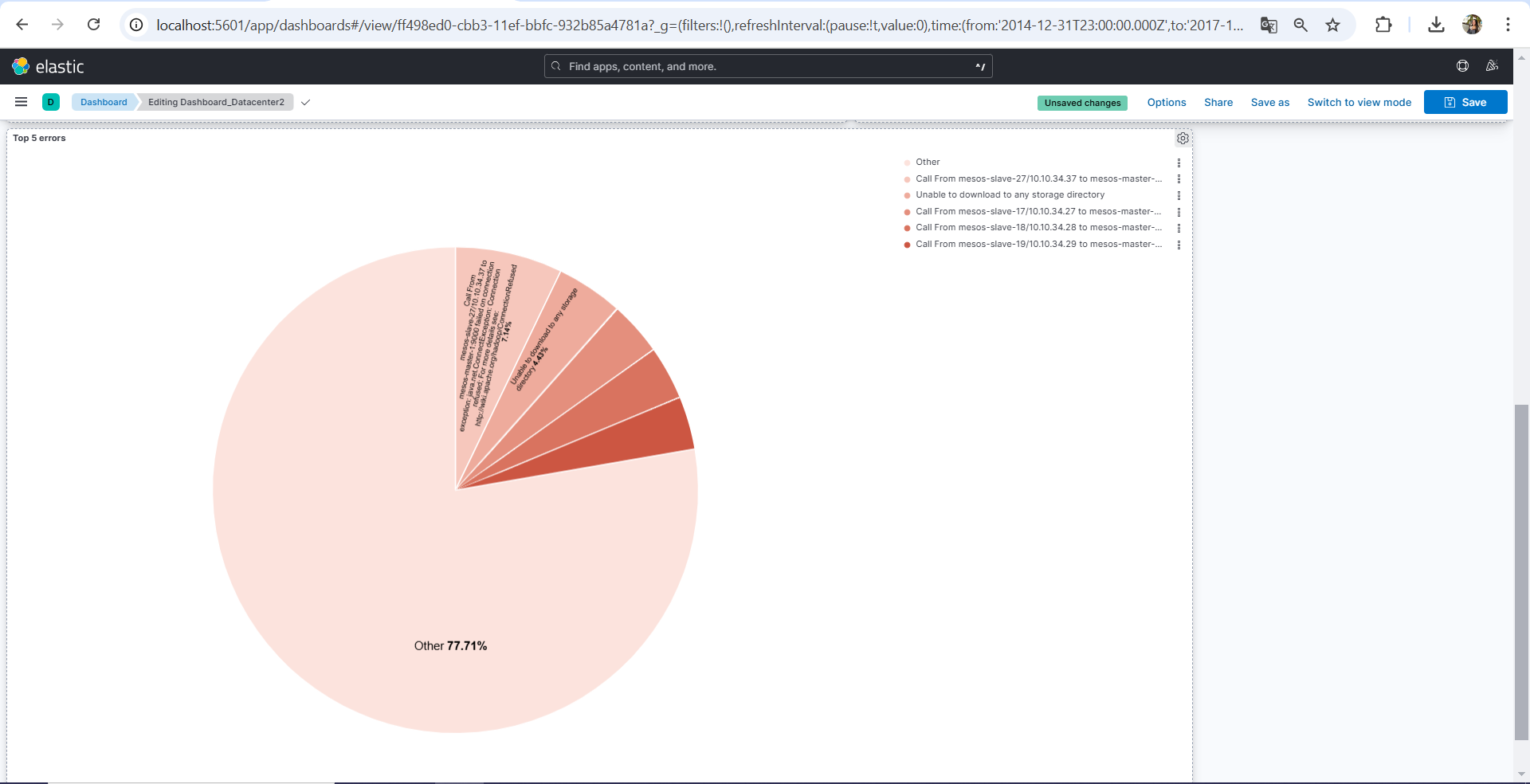
Une fois les logs correctement structurés et indexés, ils sont visualisés dans Kibana selon des axes d’analyse spécifiques. Ces visualisations mettent l’accent sur des éléments comme le **niveau des logs (Level)**, les **messages**, et les **PID par jour**, en utilisant différents types de graphiques, notamment des histogrammes et des diagrammes en anneau, pour une interprétation claire et efficace des données.

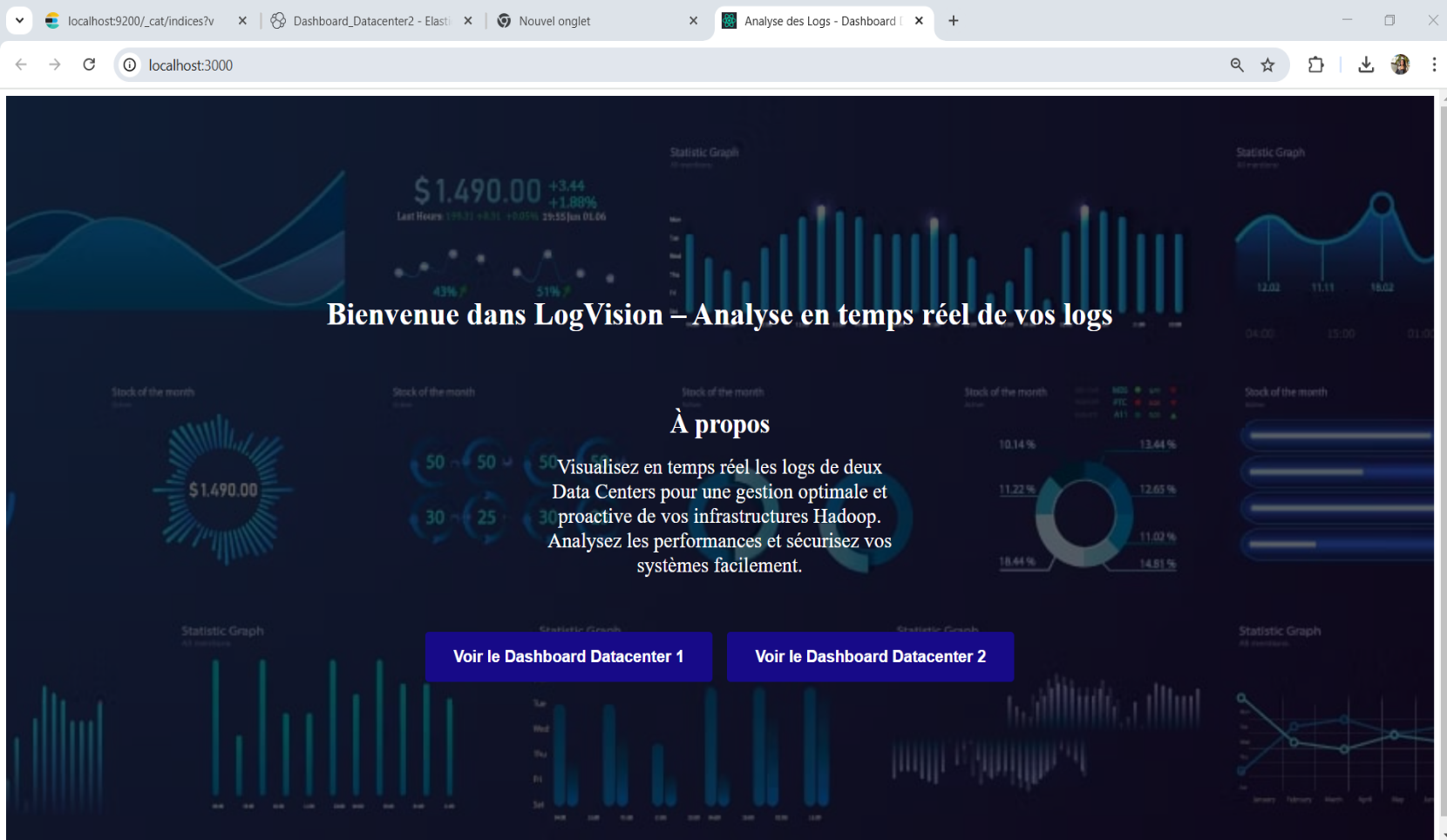
Une interface React a été développée pour intégrer et afficher les dashboards Kibana, offrant ainsi une page d'accueil interactive et personnalisée. Cette intégration permet aux utilisateurs de visualiser et d'interagir avec les données en temps réel, tout en bénéficiant d'une navigation fluide et d'une expérience utilisateur optimisée.











## 

## 

## 5. Perspectives et Améliorations

### 5.1 Fonctionnalités futures

* Intégration d’un système de prédiction des pannes basé sur le Machine Learning.

### 5.2 Optimisation des performances

* Réduction supplémentaire de la latence et amélioration des visualisations.

### 5.3 Extensions possibles

* Adaptation du système pour surveiller d’autres types d’environnements.

## 6. Conclusion

### 6.1 Résumé des réalisations

Le projet a permis de développer un système de monitoring en temps réel, robuste et fiable.

### 6.2 Impact attendu

* Amélioration de la gestion des infrastructures.
* Réduction des temps d’intervention en cas de panne