

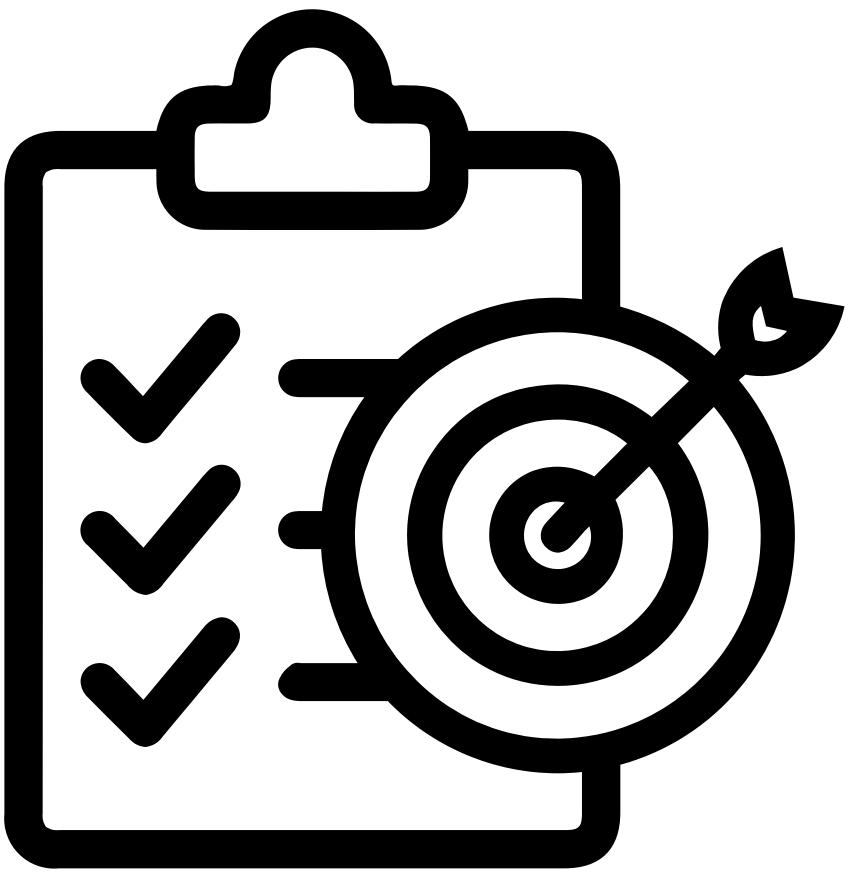
# Search Engine with Elasticsearch pour l'analyse des logs CI/CD

**Présentée par:**  
BOUTANFIT Salma  
ELIDRISSI Asma  
JENNANE Salma

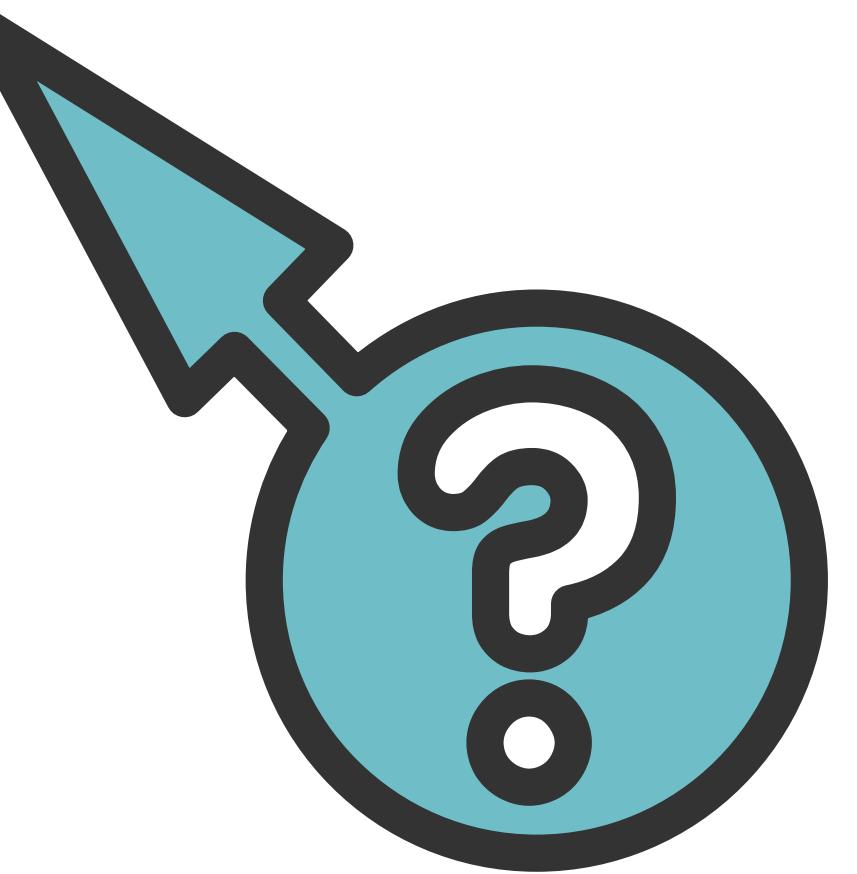
**Jury:**  
Mr. Noureddine  
KERZAZI

# Plan

- 1 Contexte & Problématique**
- 2 Objectifs du projet**
- 3 Analyse des données**
- 4 Architecture du système**
- 5 Recherche sémantique & RAG**
- 6 Résultats et démonstration**
- 7 Conclusion et perspectives**



# 1 Contexte & problématique



# 1.Contexte & problématique



**CI/CD génère des volumes massifs de logs**  
**Exemple : Mozilla CI**  
**~500 000 builds en 20 jours**  
**Analyse manuelle X impossible**

# 1.Contexte & problématique

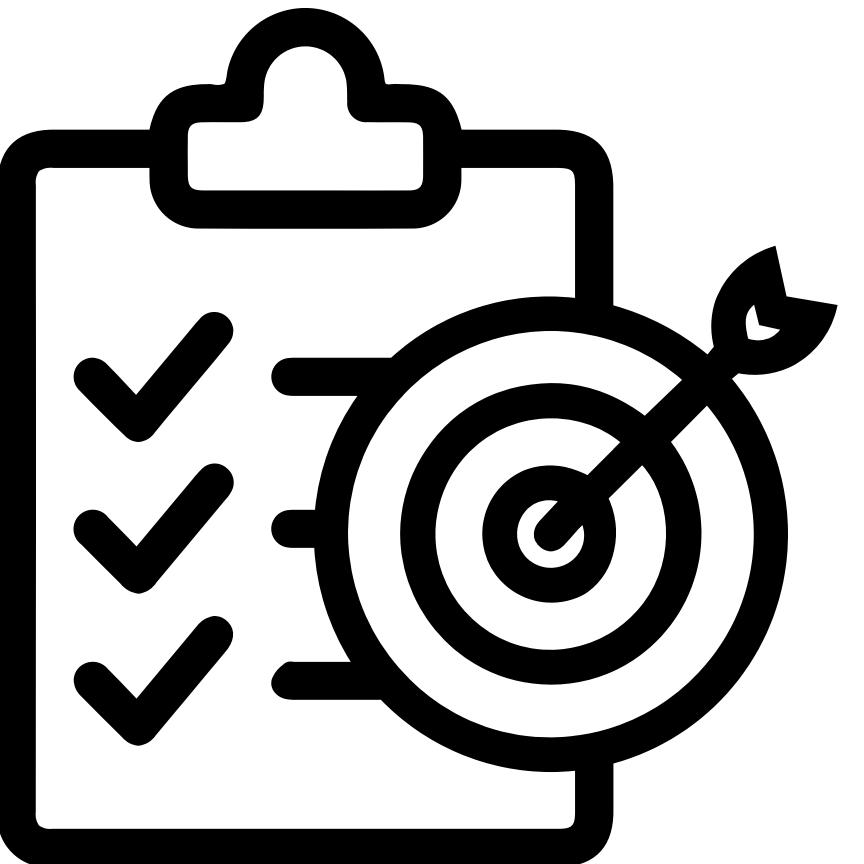


Quels sont les défis de la gestion des logs ?

Volume très élevé, logs non structurés, recherche limitée, pas d'analyse sémantique et besoin de réponses rapides pour DevOps.



# 2 objectifs du projet



- ✓ Pipeline d'ingestion temps réel
- ✓ Structuration intelligente des logs
- ✓ Visualisation avec Kibana
- ✓ Recherche sémantique via RAG
- ✓ Interface conversationnelle en langage naturel

# 3 Analyse des données

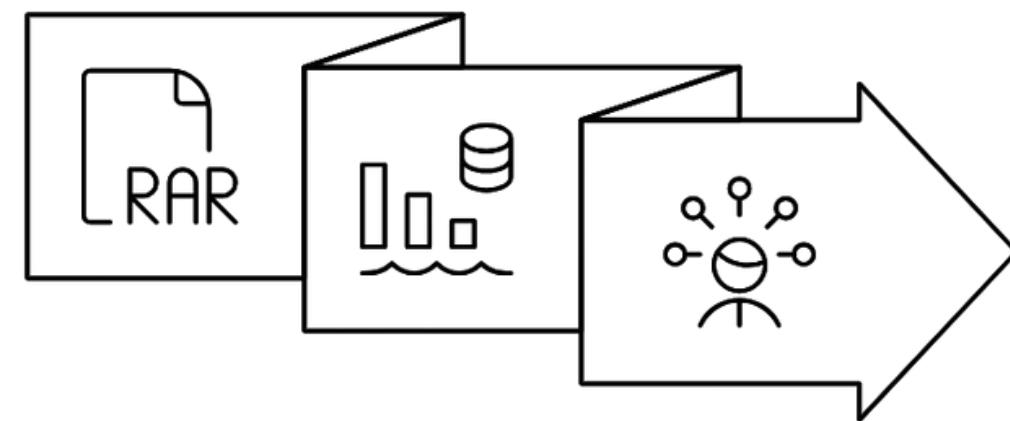


## Données utilisées (Mozilla CI)

Logs CI Mozilla  
20 jours de données  
Structure complexe  
Formats hétérogènes

## Extraction des Données

Collecte des données de logs à partir de fichiers RAR



## Analyse des Logs

Traitement des logs pour extraire des informations significatives

## Génération d'Insights

Structure de 4 niveaux

Log\_Analyzer\_Colab.ipynb

File Edit View Insert Runtime Tools Help

commands + Code + Text ▶ Run all ▾

🔍 Analyseur de Structure des Logs Mozilla CI

Ce notebook analyse la structure des logs pour identifier les patterns et préparer le parsing

📦 1: Installation des dépendances

```
[ ] # Installation des dépendances  
!pip install -q rarfile  
!apt-get install -qq unrar  
  
print("✅ Dépendances installées avec succès!")
```

✅ Dépendances installées avec succès!

UPLOAD 2: Upload des fichiers .rar

3 fichiers .rar:

- log-2018-06-01.rar
- log-2018-06-19.rar
- log-2018-06-20.rar

## **L'analyse a révélé une structure hiérarchique en 4 niveaux :**

Niveau 1 - Les métadonnées : dans le header, on trouve le builder, le statut du build, et les timestamps.

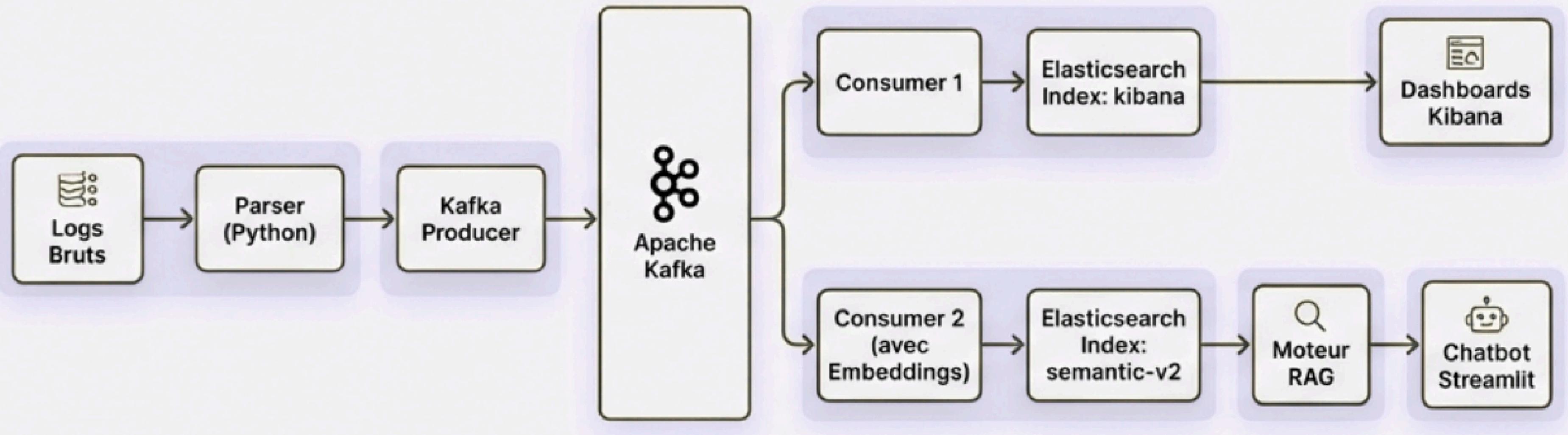
Niveau 2 - Les erreurs : détection via regex des mots-clés ERROR, WARNING, avec calcul d'un taux d'erreur.

Niveau 3 - Les métriques de performance : extraction des valeurs CPU, I/O, et durées via des patterns spécifiques.

Niveau 4 - Le contexte d'exécution : inférence de la plateforme (Linux, Windows, Mac) et du type de test à partir du nom du builder."

# 4 Architecture du système





```

    < MOZILLA-CI-LOG-ANALYSIS
      < data
      < scripts
        + extract_rar.py
        + parse_all_logs.py
      < src
        < ai
          > __pycache__
          + __init__.py
          + rag_engine_semantic.py
        > elasticsearch
        < kafka
          + __init__.py
          + consumer_semantic.py
          + consumer.py
          + producer.py
        < parser
          > __pycache__
          + __init__.py
          + log_parser.py
        < utils
          > __pycache__
          + __init__.py
          + timestamp_utils.py
        .env
        .gitignore
        + app_chatbot.py
        + benchmark_rag.py
        { benchmark_report.json
        docker-compose.yml
        requirements.txt
    
```

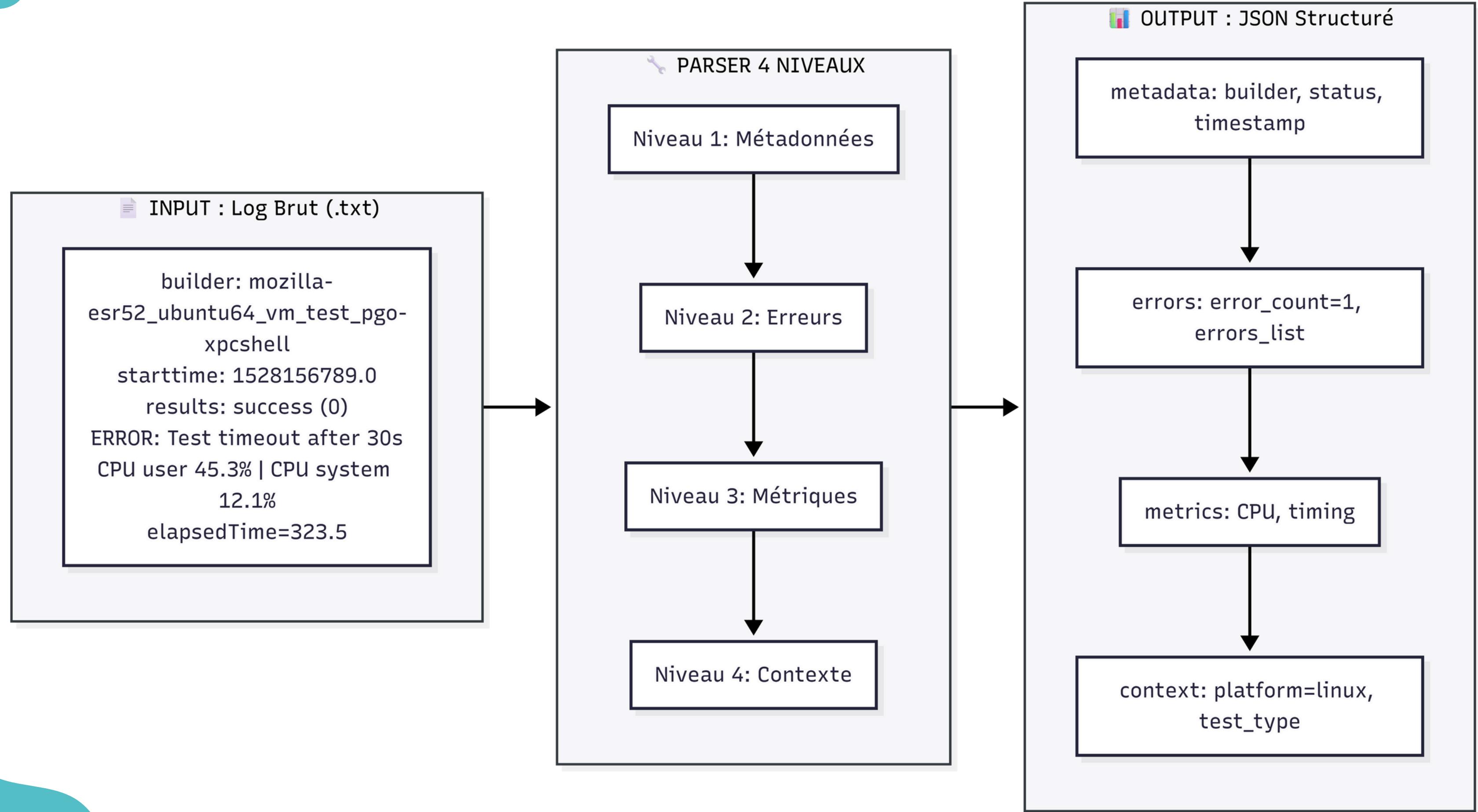
```

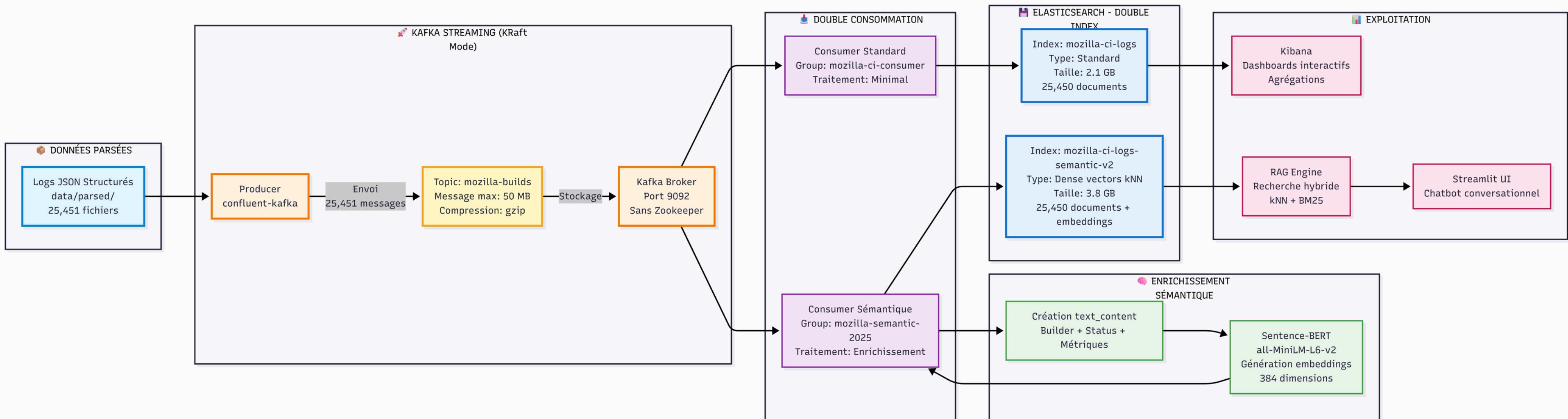
    < MOZILLA-CI-LOG-ANALYSIS
      < data
        < extracted
        < parsed
          > day_01
          > day_02
          > day_03
          > day_04
          > day_05
          > day_06
          > day_07
          > day_08
          > day_09
          > day_10
          > day_11
          > day_12
          > day_13
          > day_14
          > day_15
          > day_16
          > day_17
          > day_18
          > day_19
          > day_20
          > raw
    
```

```

    < data
      < extracted
      < parsed
        < day_01
          { 129546168_2018-05-31-23-37-13.json
          { 129546169_2018-05-31-23-46-03.json
          { 129546170_2018-05-31-23-11-59.json
          { 129546171_2018-05-31-23-17-31.json
          { 129546172_2018-05-31-23-17-49.json
          { 129546173_2018-05-31-23-11-54.json
          { 129546174_2018-05-31-23-17-33.json
          { 129546175_2018-05-31-23-46-24.json
          { 129546176_2018-05-31-23-56-06.json
          { 129546177_2018-05-31-23-12-44.json
          { 129546178_2018-05-31-23-17-24.json
          { 129546179_2018-05-31-23-46-05.json
          { 129546180_2018-05-31-23-46-14.json
          { 129546181_2018-05-31-23-41-15.json
          { 129546182_2018-05-31-23-50-42.json
          { 129546183_2018-05-31-23-41-08.json
          { 129546184_2018-05-31-23-17-26.json
          { 129546185_2018-05-31-23-17-26.json
          { 129546186_2018-05-31-23-17-33.json
          { 129546187_2018-05-31-23-17-30.json
          { 129546188_2018-05-31-23-46-34.json
          { 129546189_2018-05-31-23-46-02.json
          { 129546190_2018-05-31-23-17-49.json
          { 129546191_2018-05-31-23-46-05.json
          { 129546192_2018-05-31-23-46-06.json
          { 129546193_2018-05-31-23-46-06.json
    
```



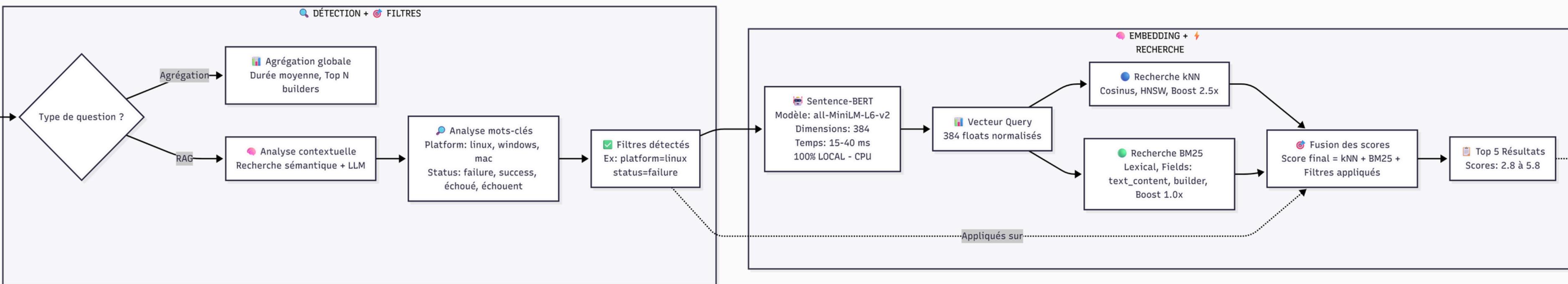


# 5 Recherche sémantique & RAG



## QUESTION UTILISATEUR

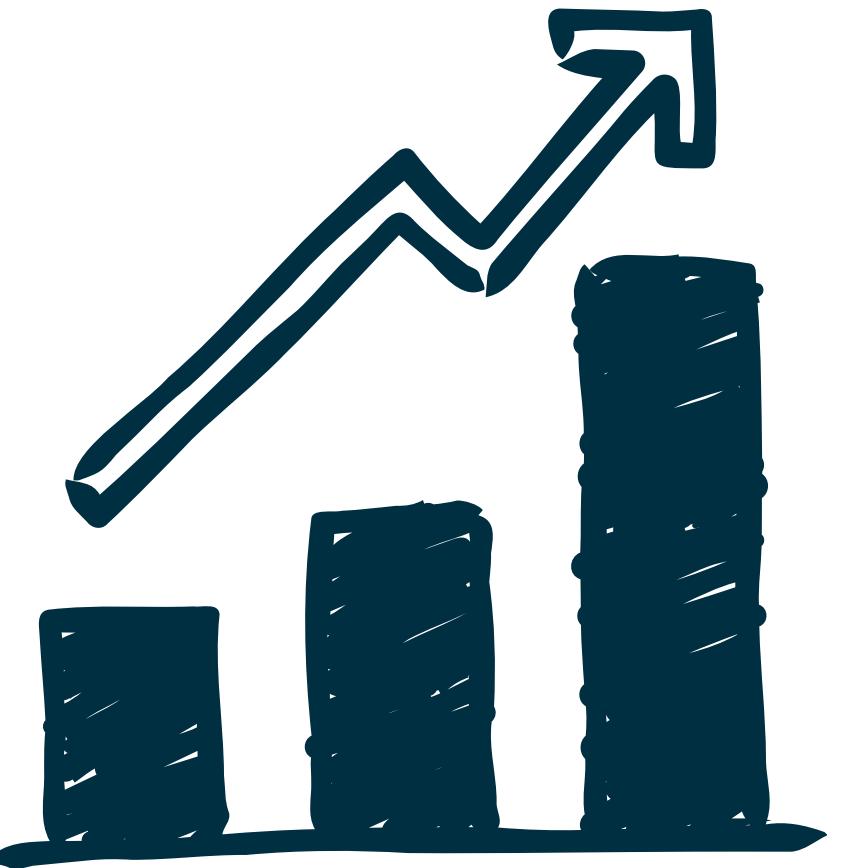
Question en langage naturel  
Ex: Quels builds sont lents sur Linux ?



## PERFORMANCE MESURÉE

Recherche Hybride: 216.9 ms  
kNN seul: 585.6 ms  
Gain: 2.7x  
BM25 pré-filtre efficacement → moins de candidats kNN

# 6 Résultats et démonstration



# Index Management

[Index Management docs](#)

[Indices](#)   [Data Streams](#)   [Index Templates](#)   [Component Templates](#)   [Enrich Policies](#)

Update your Elasticsearch indices individually or in bulk. [Learn more.](#)

[Include rollup indices](#)

[Include hidden indices](#)



Search

[Reload indices](#)

[Create index](#)

<input type="checkbox"/> Name	Health	Status	Primaries	Replicas	Docs count	Storage size	Data stream
<a href="#">mozilla-ci-logs</a>	<span style="color: yellow;">● yellow</span>	open	1	1	25445	4.66gb	
<a href="#">mozilla-ci-logs-semantic-v2</a>	<span style="color: green;">● green</span>	open	1	0	25450	4.96gb	

```
PS C:\Users\bouta\Downloads\mozilla-ci-log-analysis> python src/kafka/producer.py
```

KAFKA PRODUCER

Kafka: localhost:9092

Topic: mozilla-builds

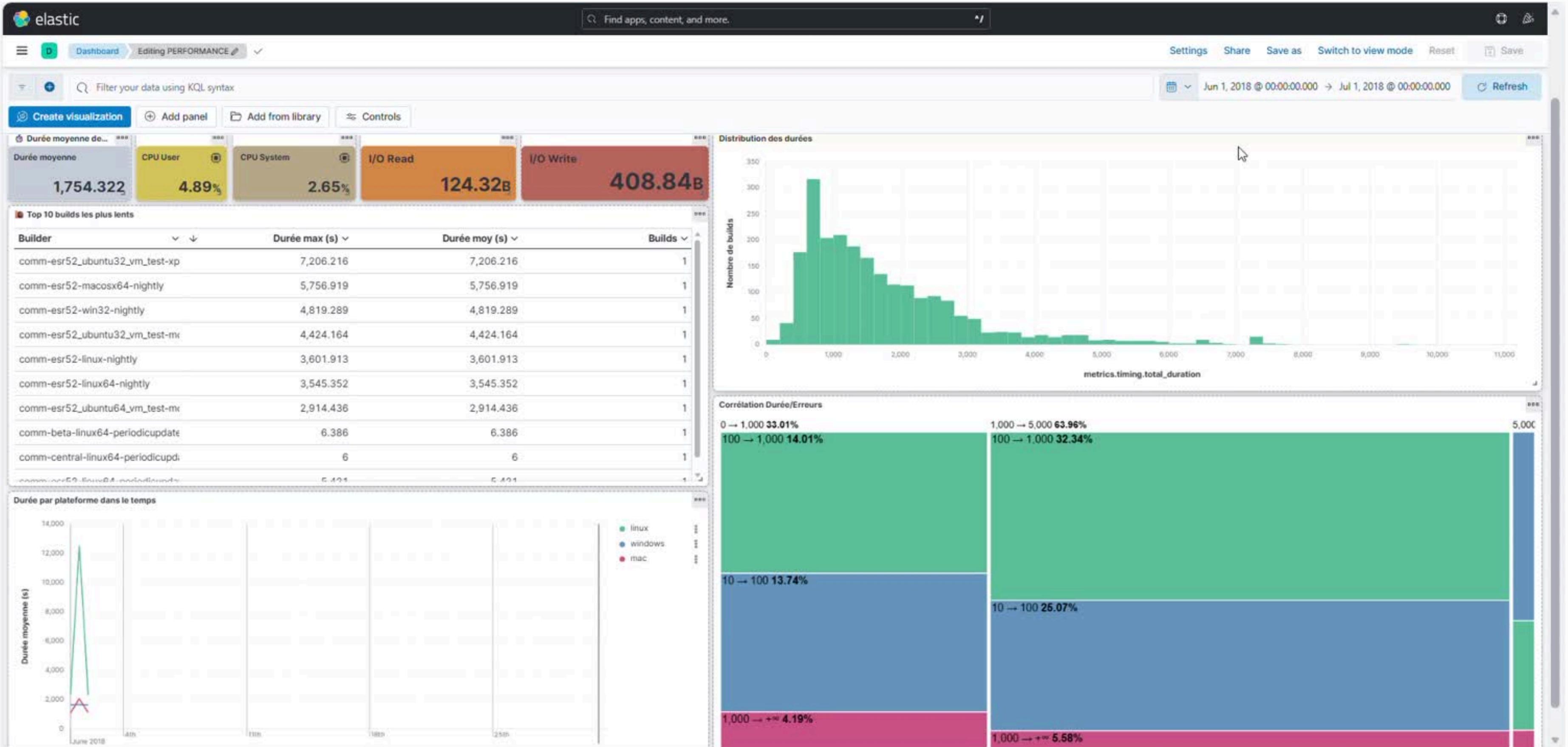
Source: c:\users\bouta\Downloads\mozilla-ci-log-analysis\data\parsed

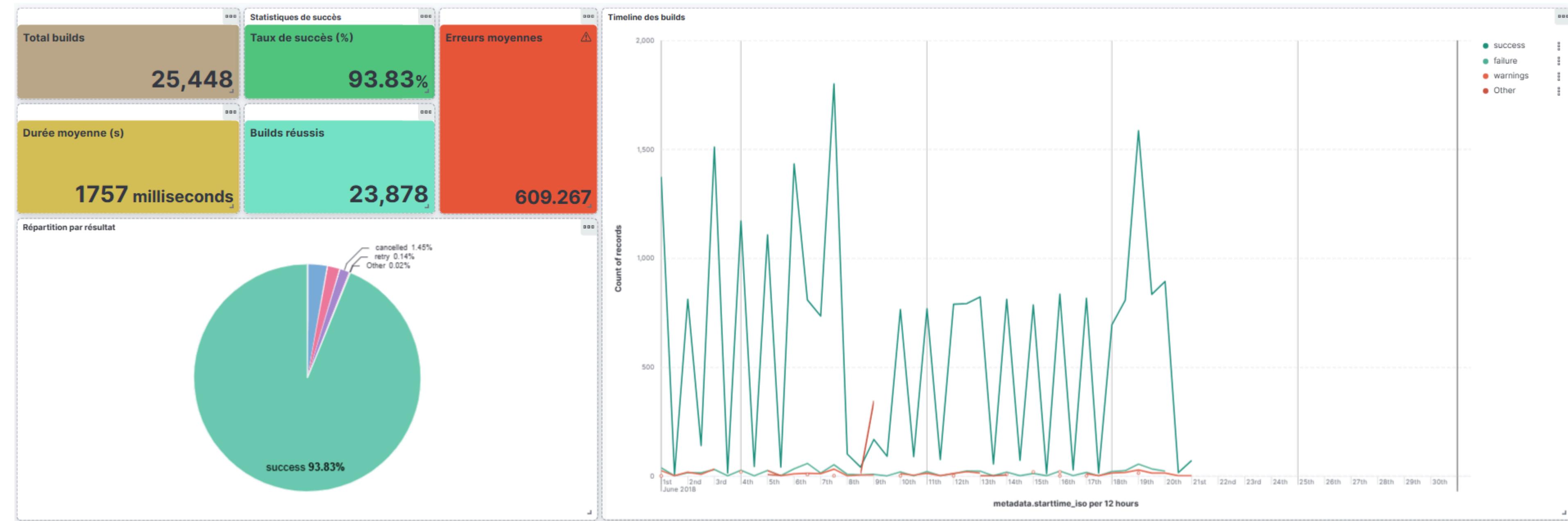
⌛ Attente de Kafka...

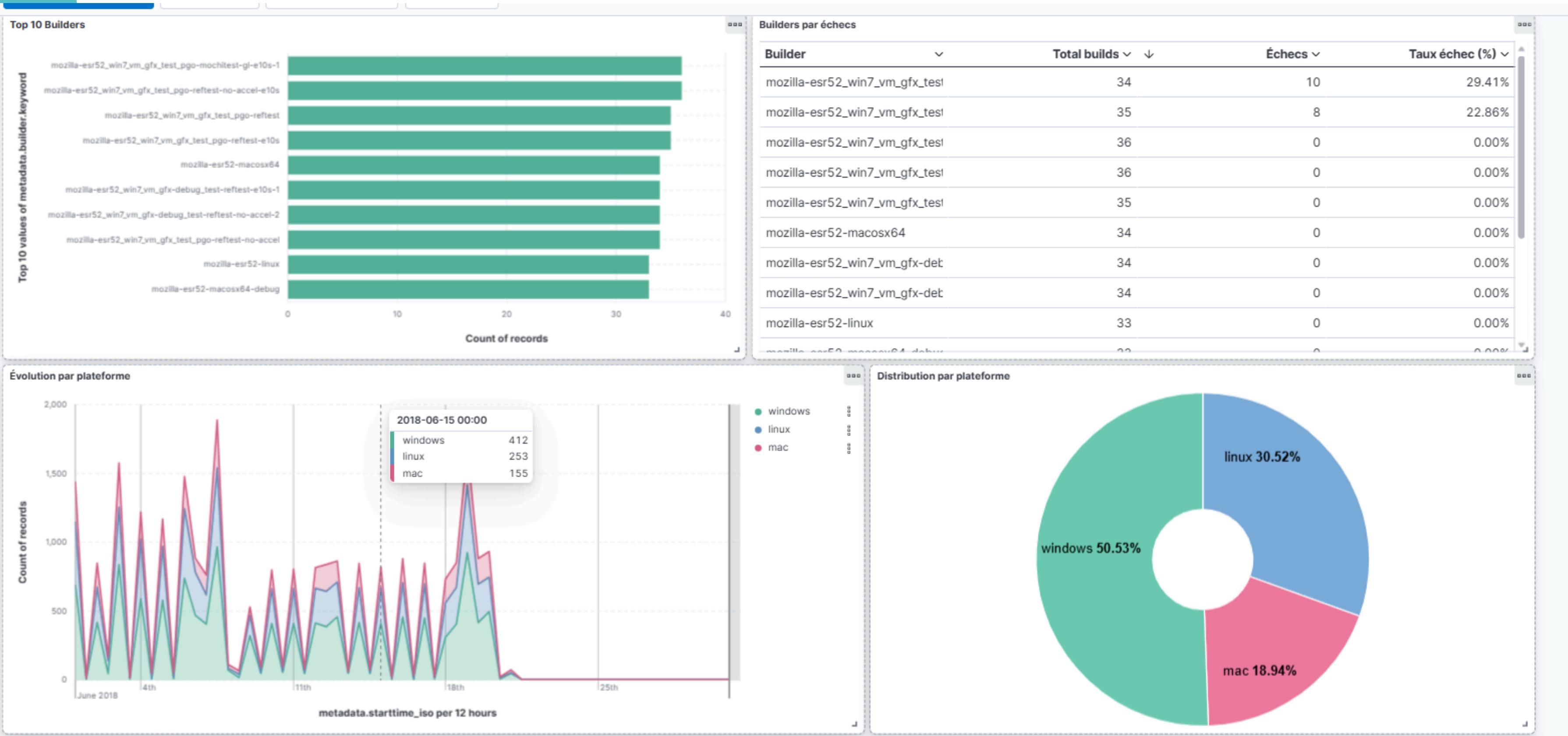
✓ Connexion à Kafka...

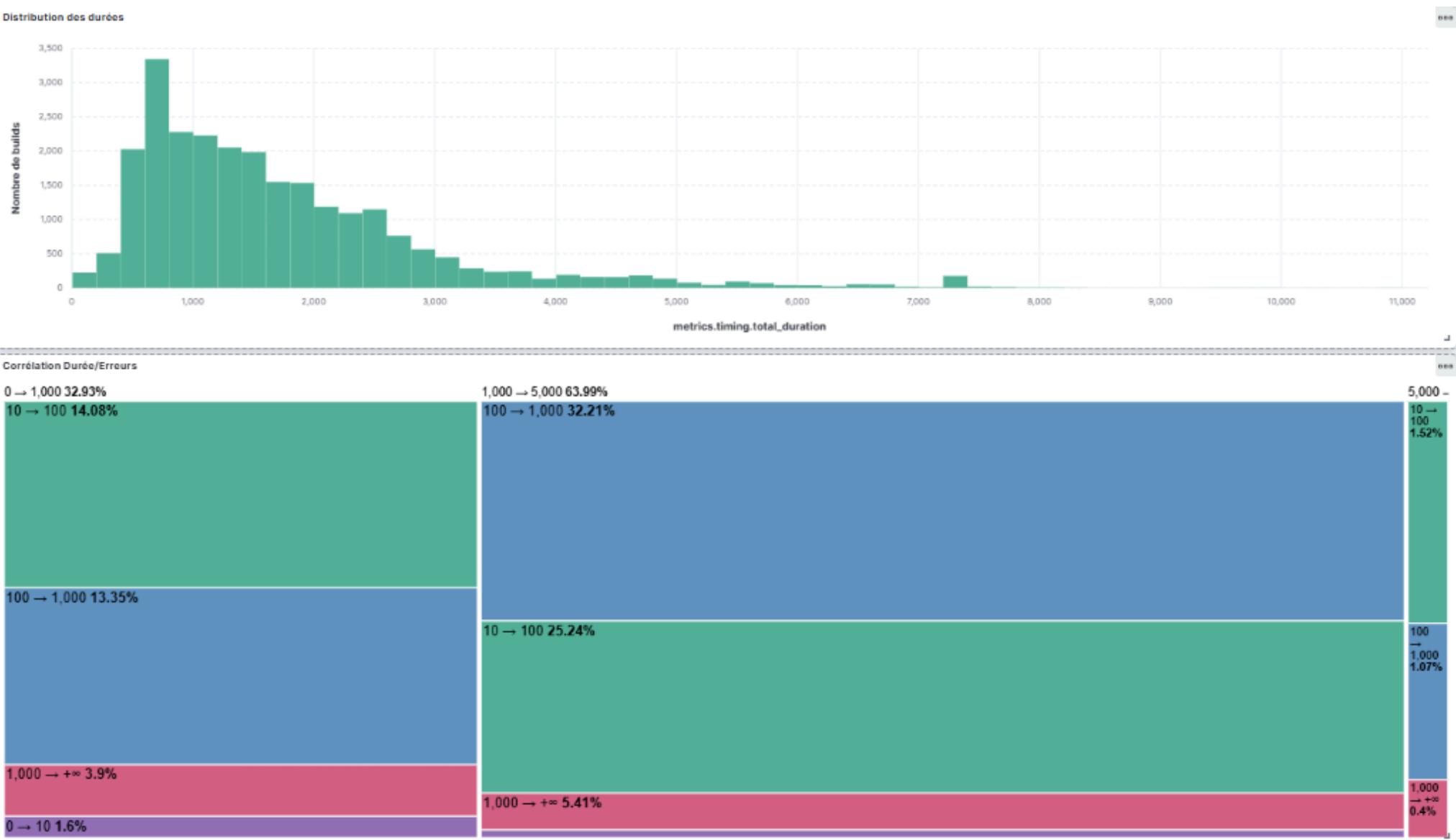
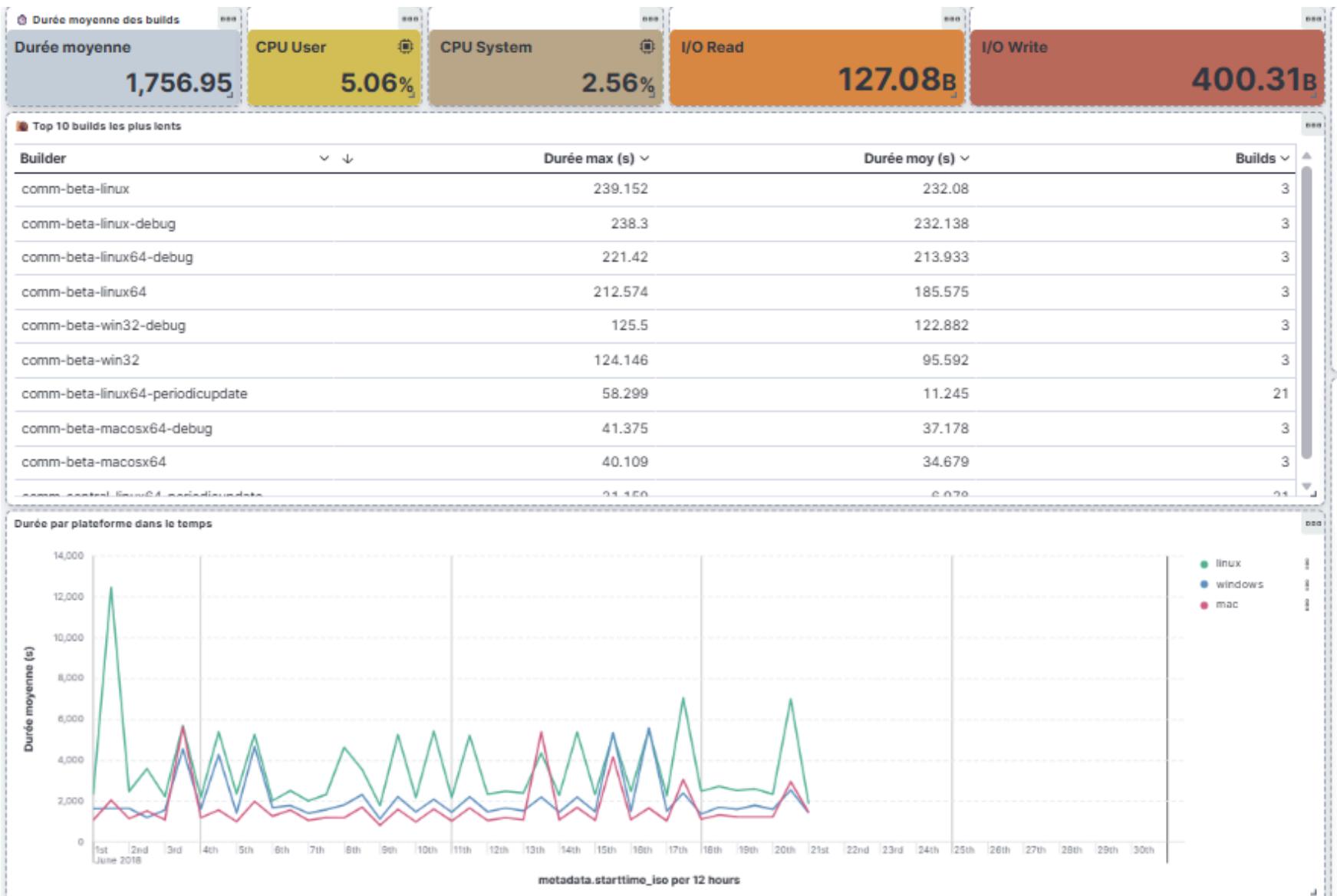
📄 25,451 fichiers JSON trouvés

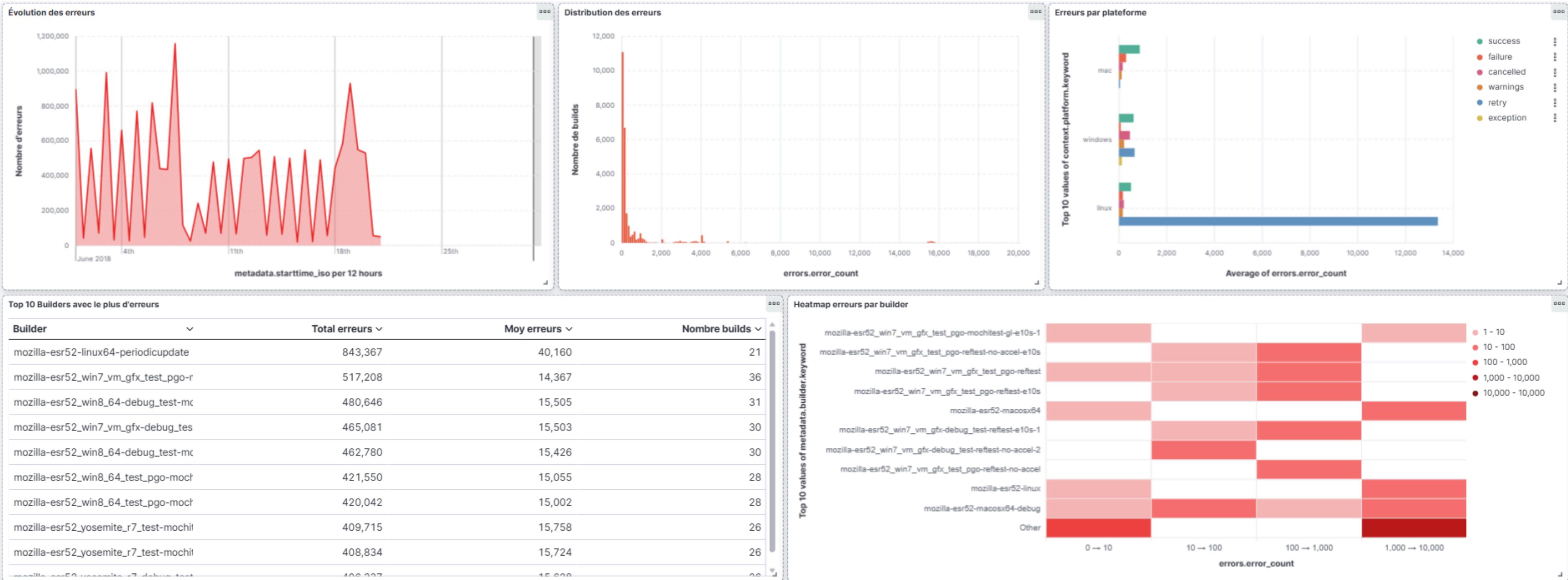
```
[100/25451] 100 envoyés
[200/25451] 200 envoyés
[300/25451] 300 envoyés
[400/25451] 400 envoyés
[500/25451] 500 envoyés
[600/25451] 600 envoyés
[700/25451] 700 envoyés
[800/25451] 800 envoyés
[900/25451] 900 envoyés
[1000/25451] 1000 envoyés
[1100/25451] 1100 envoyés
[1200/25451] 1200 envoyés
[1300/25451] 1300 envoyés
[1400/25451] 1400 envoyés
[1500/25451] 1500 envoyés
[1600/25451] 1600 envoyés
[1700/25451] 1700 envoyés
[1800/25451] 1800 envoyés
[1900/25451] 1900 envoyés
[2000/25451] 2000 envoyés
[2100/25451] 2100 envoyés
[2200/25451] 2200 envoyés
[2300/25451] 2300 envoyés
[2400/25451] 2400 envoyés
```











localhost:8501

Deploy :

# moz://a

## Mozilla CI Assistant

Analysez vos logs CI/CD avec l'intelligence artificielle

**Configuration**

Mode de recherche ?

Hybride (Recommandé)

Sémantique pur

**Statistiques**

Questions 0 Historique 0

Effacer l'historique

Statistiques détaillées

Questions suggérées

Builds les plus lents Erreurs sur Linux Compare Windows vs Linux

Résumé des problèmes Builds avec CPU élevé Tests qui échouent

Posez votre question sur les logs CI/CD... Envoyer

Capacités

Recherche intelligente  
Comprend les synonymes et filtre par plateforme

Analyse avancée  
Performance, erreurs et comparaisons

Contexte conversationnel  
Se souvient de vos questions

Propulsé par Elasticsearch, Sentence Transformers et Mistral  
Recherche sémantique 100% locale

Quelle est la durée moyenne des builds ?

Vous

Quelle est la durée moyenne des builds ?

Assistant

La durée moyenne globale des builds est de 1757 secondes (29 minutes et 17 secondes), calculée sur 25,450 builds.

▼ Voir les logs sources

#### 1. mozilla-esr52\_yosemite\_r7-debug\_test-mochitest-e10s-devtools-chrome-7

- Status: `warnings`
- Durée: `1379.968898s`
- Score: `3.23`

#### 2. mozilla-esr52\_yosemite\_r7-debug\_test-mochitest-e10s-devtools-chrome-7

- Status: `warnings`
- Durée: `1391.8425630000002s`
- Score: `3.23`

#### 3. mozilla-esr52\_yosemite\_r7-debug\_test-mochitest-e10s-devtools-chrome-7

- Status: `warnings`
- Durée: `1389.079563s`
- Score: `3.22`

#### 4. mozilla-esr52\_yosemite\_r7-debug\_test-mochitest-e10s-devtools-chrome-8

- Status: `warnings`
- Durée: `1285.242652s`
- Score: `3.22`

#### 5. mozilla-esr52\_yosemite\_r7-debug\_test-mochitest-e10s-devtools-chrome-2

- Status: `failure`
- Durée: `1817.044012s`
- Score: `3.22`

Assistant

Les 5 builders avec le plus d'erreurs sont :

▼ Voir les logs sources

#### 1. try\_win10\_64-debug\_test-gtest

- Status: `failure`
- Durée: `407.467s`
- Score: `2.87`

#### 2. mozilla-esr52\_win8\_64-debug\_test-crash-test-e10s

- Status: `failure`
- Durée: `957.222s`
- Score: `2.81`

#### 3. mozilla-esr52\_win8\_64-test\_pgo-gtest

- Status: `failure`
- Durée: `228.488s`
- Score: `2.81`

#### 4. mozilla-esr52\_win8\_64-test\_pgo-gtest

- Status: `failure`
- Durée: `235.097s`
- Score: `2.81`

#### 5. mozilla-esr52\_win8\_64-test\_pgo-gtest

5 builders avec le plus d'erreurs ?

Assistant

La durée moyenne globale des builds est de 1757 secondes (29 minutes et 17 secondes), calculée sur 25,450 builds.

▼ Voir les logs sources

#### 1. mozilla-esr52\_yosemite\_r7-debug\_test-mochitest-e10s-devtools-chrome-7

- Status: `warnings`
- Durée: `1379.968898s`
- Score: `3.23`

#### 2. mozilla-esr52\_yosemite\_r7-debug\_test-mochitest-e10s-devtools-chrome-7

- Status: `warnings`
- Durée: `1391.8425630000002s`
- Score: `3.23`

#### 3. mozilla-esr52\_yosemite\_r7-debug\_test-mochitest-e10s-devtools-chrome-7

- Status: `warnings`
- Durée: `1389.079563s`
- Score: `3.22`

#### 4. mozilla-esr52\_yosemite\_r7-debug\_test-mochitest-e10s-devtools-chrome-8

- Status: `warnings`
- Durée: `1285.242652s`
- Score: `3.22`

Vous

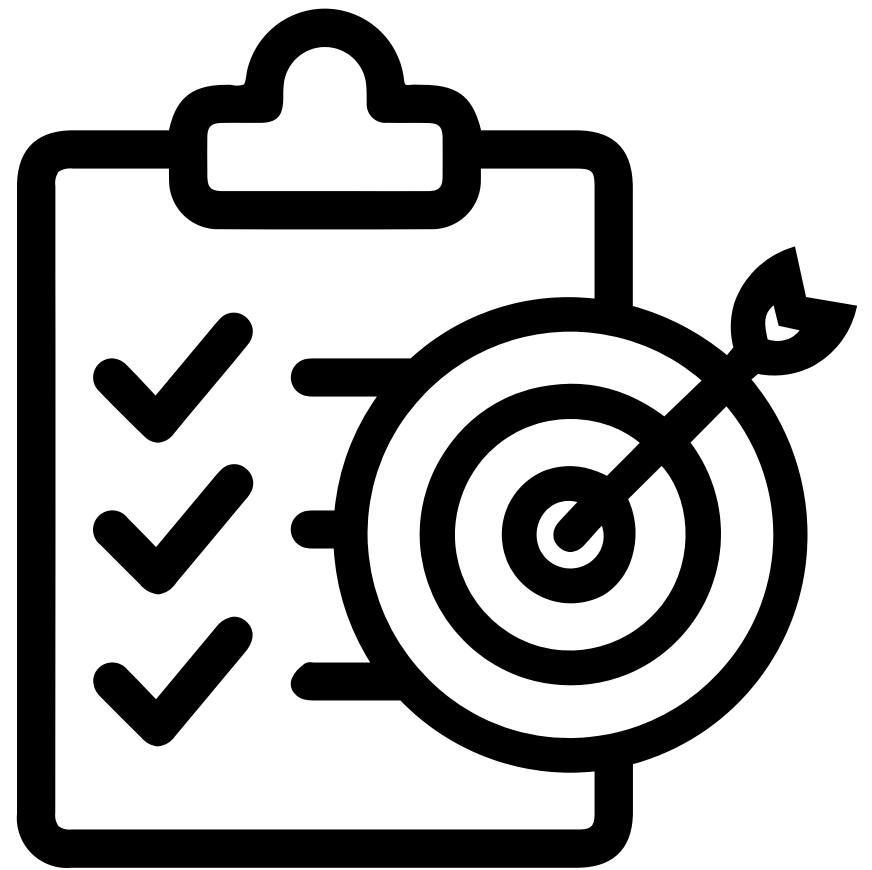
Combien de développeurs travaillent sur Mozilla ?

Assistant

Information insuffisante dans ces 5 logs. Les logs ne contiennent pas d'informations sur le nombre de développeurs travaillant sur Mozilla.

7

# Conclusion et perspective



## Objectifs atteints

- ✓ Pipeline temps réel opérationnel (Kafka + ES)
- ✓ Double indexation (viz + RAG)
- ✓ RAG hybride performant
- ✓ 0 erreur sur 25,450 documents
- ✓ Système 100% local
- ✓ Interface conversationnelle accessible

## AXES D'AMÉLIORATION



- Latence LLM : 70s sur CPU  
→ Solution : GPU réduirait à 2-5s (gain 10-30x)
- Pas de métriques ROUGE/BLEU  
→ Solution : Dataset de test annoté
- Débit ingestion : 6.1 docs/s  
→ Solution : Batch indexing ES (5-10x)
- **Fine-tuning Sentence-BERT sur logs CI/CD**

## Extensions fonctionnelles

**Alerting temps réel** : ajouter un consumer dédié au monitoring proactif

- Kafka Streams : Déetecter les anomalies en streaming
- Règles configurables : Alerter si taux d'échec > seuil
- Notifications : Email, Slack, PagerDuty

Exemple : "Alerte : 5 builds Linux consécutifs ont échoué avec timeout"

## **Analyse prédictive** : Exploiter l'historique pour prédire les échecs

1. Features engineering : Extraire métriques temporelles (CPU trend, durée relative)
2. Modèle ML : Random Forest ou XGBoost pour classification (success/failure)
3. Intégration RAG : "Ce build a 78% de risque d'échouer (CPU élevé, historique récent)"

# Merci pour votre attention

