École Nationale Supérieure
d'Informatique et d'Analyse des Systèmes
- Rabat

Projet d'Information Retrieval

Surveillance et Analyse en Temps Réel des Logs de Build Firefox à l'aide de la Pile ELK et du Machine Learning

Élèves :
Salma BOUTANFIT
Asma EL IDRISSI
Salma JENNANE

Jury :
Monsieur Noureddine
Kerzazi

Année Universitaire 2024-2025

Remerciements

Nous tenons à exprimer notre profonde gratitude à Monsieur **Nourredine Kerzazi** pour la qualité de son enseignement dans le module **Information Retrieval**. Son sens de l'explication, sa rigueur méthodologique ainsi que sa disponibilité nous ont permis d'acquérir une meilleure compréhension des concepts abordés et d'aborder ce projet avec une vision claire et structurée.
Nous lui sommes également reconnaissants pour ses conseils et orientations qui ont joué un rôle essentiel dans la réalisation de ce travail.
Nous tenons enfin à remercier nos familles et nos camarades pour leur soutien, leur patience et leurs encouragements tout au long de ce projet.

Résumé

Dans un contexte où les systèmes informatiques génèrent des volumes importants de données de journalisation, l'analyse et la surveillance de ces logs deviennent essentielles pour assurer la stabilité et la performance des applications. Ce projet s'inscrit dans cette optique et porte sur la mise en place d'une solution de monitoring en temps réel basée sur la pile **ELK** (Elasticsearch, Logstash, Kibana) pour l'analyse des logs de build du navigateur Firefox.
Nous avons commencé par définir un schéma de données et un mapping adapté, puis configuré une ingestion automatisisée des logs afin de simuler un flux continu. Par la suite, des visualisations ont été développées dans **Kibana** pour faciliter l'exploration et le suivi des indicateurs clés de performance. Enfin, un module de **détection d'anomalies** a été intégré à l'aide des fonctionnalités de machine learning d'Elasticsearch, permettant d'identifier des comportements inhabituels dans les temps d'exécution des builds.
Les résultats obtenus montrent la pertinence de l'approche ELK pour la supervision des systèmes en temps réel et ouvrent la voie à des extensions telles que l'alerte automatique ou l'analyse prédictive.

Mots-clés : ELK, Elasticsearch, Logstash, Kibana, Logs, Monitoring, Machine Learning, Détection d'anomalies, Information Retrieval, Tableau de bord.

Abstract

With the increasing volume of system-generated log data, real-time monitoring and analysis have become essential to ensure application performance, reliability, and operational stability. This project focuses on designing and implementing a real-time monitoring solution based on the **ELK** stack (Elasticsearch, Logstash, Kibana) to process and analyze Firefox build logs.
We defined an appropriate data mapping structure, configured an automated log ingestion pipeline to simulate continuous data flow, and developed interactive dashboards in **Kibana** for performance visualization. Additionally, an **anomaly detection** model was integrated using Elasticsearch's machine learning capabilities to identify abnormal build execution patterns.
The results demonstrate the effectiveness of the ELK stack for real-time log supervision and highlight the potential for further enhancements such as automatic alerting and predictive analysis.

Keywords : ELK, Elasticsearch, Logstash, Kibana, Log Analysis, Real-time Monitoring, Machine Learning, Anomaly Detection, Information Retrieval.

Liste des Abréviations

API : Application Programming Interface
CPU : Central Processing Unit
CSV : Comma-Separated Values
ELK : Elasticsearch, Logstash, Kibana
ETL : Extract, Transform, Load
HTTP : HyperText Transfer Protocol
JSON : JavaScript Object Notation
KPI : Key Performance Indicator
ML : Machine Learning
RAM : Random Access Memory
URI : Uniform Resource Identifier
URL : Uniform Resource Locator

Introduction générale

Avec l'évolution rapide des systèmes informatiques et l'augmentation constante du volume de données générées par les applications et les infrastructures, la gestion et l'analyse des journaux (*logs*) sont devenues des opérations essentielles. Les logs constituent une source précieuse d'informations permettant de diagnostiquer des dysfonctionnements, d'évaluer la performance d'un système et de garantir la continuité des services. Cependant, leur volume important, leur nature hétérogène et leur arrivée en flux continu rendent leur traitement manuel difficile, voire impossible.
Dans ce contexte, des solutions de **monitoring** et d'analyse en temps réel sont indispensables. La pile **ELK** (Elasticsearch, Logstash, Kibana) s'impose aujourd'hui comme une solution open-source puissante et largement adoptée pour la collecte, l'indexation, la recherche et la visualisation des données de logs. Elle offre une approche centralisée permettant de transformer des données brutes en informations exploitables.
Ce projet s'inscrit dans le cadre du module **Information Retrieval** et vise à mettre en place un système complet de supervision basé sur la pile ELK pour l'analyse des logs de build du navigateur Firefox. Il s'agit d'automatiser l'ingestion des logs, de concevoir des tableaux de bord interactifs pour le suivi des indicateurs de performance et d'intégrer un module de **détection d'anomalies** afin d'identifier automatiquement des comportements inhabituels dans les processus de build.
Le présent rapport est structuré comme suit : dans un premier temps, nous présenterons le contexte général ainsi que les concepts fondamentaux liés à la pile ELK et à l'analyse des logs. Ensuite, nous décrirons l'architecture mise en place et les différentes étapes d'implémentation. Nous présenterons par la suite les résultats obtenus, avant de conclure en mettant en évidence les apports du projet ainsi que les perspectives d'amélioration futures.

Table des matières

Table des figures

Chapitre 1

Contexte Général et Fondements Théoriques

1.1 Contexte Général

Avec l'évolution des infrastructures numériques, les systèmes informatiques modernes produisent une quantité croissante de données liées à leur fonctionnement. Parmi ces données, les journaux de bord, appelés *logs*, occupent une place centrale. Ils constituent une source d'information essentielle pour observer le comportement d'un système, analyser des incidents, suivre la performance, et assurer la continuité de service.

Dans les environnements distribués et fortement automatisés, les logs sont générés en volumes massifs et en temps réel, ce qui rend leur traitement manuel difficile voire impossible. Il devient alors nécessaire de disposer d'outils avancés permettant la **collecte**, l'**indexation**, l'**analyse** et la **visualisation** de ces données.

1.2 Les Journaux (Logs) : Rôle et Nature

1.2.1 Définition et Structure

Un log est une information textuelle enregistrée automatiquement par un programme informatique pour témoigner d'un événement. Chaque entrée de log contient généralement :

— un **horodatage** (date et heure de l'évènement),
— le **niveau de sévérité** (ex. `INFO`, `WARNING`, `ERROR`),
— la **source ou le module concerné**,
— un **message descriptif**.

1.2.2 Types de Logs

Parmi les logs les plus courants, on distingue :

— **Logs système** : événements du système d'exploitation,
— **Logs applicatifs** : messages produits par une application,
— **Logs réseau** : traces d'échanges entre équipements,
— **Logs de sécurité** : authentification, accès, audits.

1.2.3 Limites de l'Analyse Manuelle

Le volume, la diversité et la vitesse de génération rendent la lecture humaine inefficace. Il devient donc nécessaire d'automatiser :

— La centralisation des logs,
— Leur transformation pour un format uniforme,

— Leur stockage sous une forme indexée,
— Leur visualisation et leur interprétation.

1.3 La Pile ELK

La pile **ELK** est une architecture open-source de traitement de logs, composée de :
— **Elasticsearch** : moteur d'indexation et de recherche,
— **Logstash / Filebeat** : ingestion et transformation des logs,
— **Kibana** : visualisation et analyse.

1.3.1 Architecture Globale

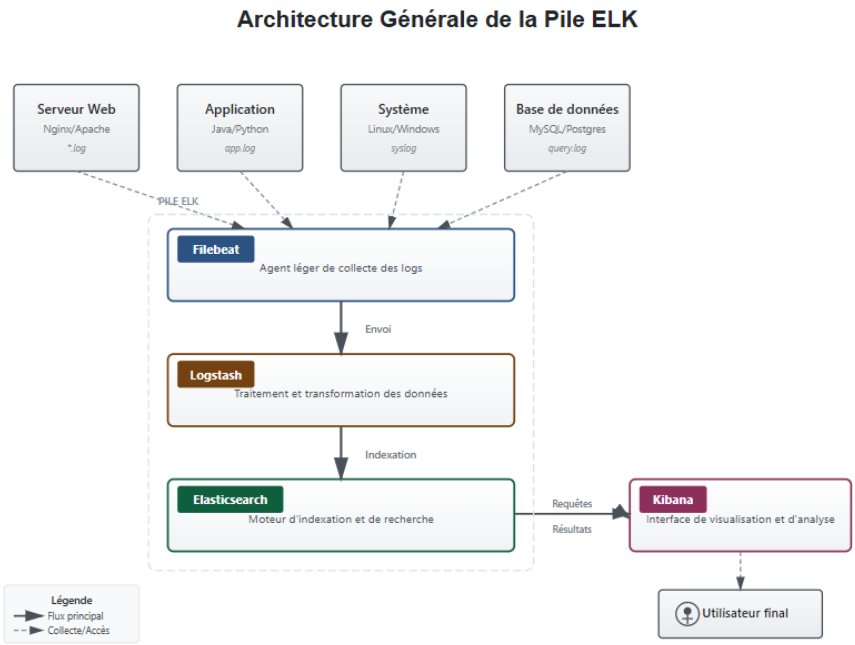

FIGURE 1.1 – Architecture générale de la pile ELK

1.3.2 Elasticsearch

Elasticsearch stocke les données sous forme de documents JSON indexés, permettant une recherche rapide. Il repose sur la structure **Index → Document → Champ**, facilitant la requêtage par expressions ou filtres.

1.3.3 Logstash et Filebeat

— **Filebeat** est un expéditeur léger placé sur les serveurs d'origine pour lire les fichiers de logs.
— **Logstash** peut filtrer, enrichir ou transformer les données avant stockage.

1.3.4 Kibana

Kibana permet de créer :
— des graphiques d'évolution,
— des tableaux de bord personnalisés,
— des requêtes interactives sur les données.

1.4 Monitoring en Temps Réel

Le monitoring vise à observer en continu le fonctionnement d'un système afin d'anticiper, détecter et résoudre les défaillances. Il implique :
— l'analyse de tendances,
— l'identification des goulots de performance,
— la supervision des anomalies.

1.5 Détection d'Anomalies par Machine Learning

1.5.1 Concept d'Anomalie

Une anomalie est une observation qui s'écarte significativement du comportement normal appris. Exemples :
— temps de build anormalement long,
— augmentation soudaine d'erreurs,
— séquence d'événements inattendue.

1.5.2 Détection Automatisée dans Elasticsearch

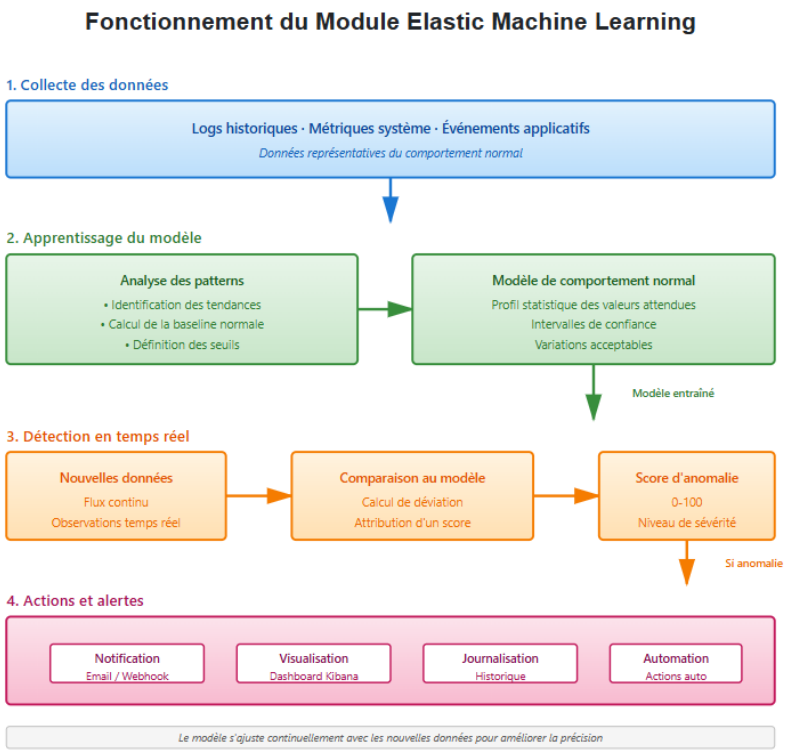

FIGURE 1.2 – Fonctionnement du module Elastic Machine Learning

Elasticsearch permet de :
— apprendre un modèle de comportement normal,
— détecter en continu des écarts,
— attribuer un **score d'anomalie**.

1.6 Conclusion

Ce chapitre a présenté les concepts essentiels liés à l'analyse des logs, à la pile ELK et à la détection d'anomalies. Ces éléments constituent la base théorique nécessaire pour aborder l'architecture et l'implémentation exposées dans les chapitres suivants.

Chapitre 2

Analyse des Besoins et Architecture du Système

2.1 Objectifs du Projet

L'objectif principal de ce projet est de mettre en place un système complet de **supervision et d'analyse en temps réel** des journaux de build du navigateur Firefox. Dans le cadre du module **Information Retrieval**, ce projet vise à mobiliser des connaissances théoriques et pratiques autour de la collecte, du traitement, de l'indexation et de la visualisation de données textuelles issues de logs.

Plus précisément, le projet doit permettre de :

— **automatiser l'ingestion** des fichiers logs au fur et à mesure de leur génération,
— **indexer** les données dans une base optimisée pour la recherche,
— **analyser** les métriques liées aux performances des builds,
— **visualiser** ces métriques dans des tableaux de bord dynamiques,
— **détecter automatiquement** des comportements anormaux grâce au Machine Learning.

2.2 Description des Données de Build Firefox

Les données traitées dans ce projet sont issues des processus de compilation et de test du navigateur Firefox. Chaque étape du build génère des logs contenant des informations précieuses sur :

— le **déroulement des opérations** (étapes de compilation, tests, liens...),
— la **durée d'exécution** de chaque étape,
— les **messages d'erreurs ou d'avertissements**,
— les **états de réussite ou d'échec**.

Ces logs sont produits en continu, ce qui nécessite une solution permettant de les exploiter au fur et à mesure de leur apparition, afin de détecter rapidement d'éventuels ralentissements ou anomalies.

2.3 Analyse des Besoins Fonctionnels

Le système doit offrir les fonctionnalités suivantes :

— **Collecte automatique** des logs depuis leur source,
— **Transformation et normalisation** du format pour une indexation cohérente,
— **Stockage et indexation** optimisés pour la recherche full-text,
— **Consultation interactive** via des dashboards,
— **Déclenchement d'alertes** en cas de comportement anormal.

2.4 Besoins Non Fonctionnels

Au-delà des fonctionnalités, le système doit répondre à des exigences de qualité :
— **Temps réel** : le traitement doit s'effectuer avec une latence minimale,
— **Scalabilité** : capacité à traiter des volumes croissants de logs,
— **Robustesse** : continuité en cas de flux irrégulier,
— **Simplicité d'exploitation** : maintenance et déploiement simplifiés.

2.5 Choix Technologiques

La pile **ELK** (Elasticsearch, Logstash, Kibana) a été retenue car elle répond naturellement aux besoins identifiés. Cependant, dans ce projet, **Filebeat** est privilégié à Logstash pour l'ingestion des logs, car :
— Filebeat est léger, rapide et facile à configurer,
— il est conçu spécifiquement pour l'expédition de logs,
— il réduit la charge système par rapport à Logstash.
Elasticsearch permet :
— une **indexation efficace** des données sous forme de documents JSON,
— des **recherches avancées** via Query DSL,
— une **analyse statistique** des performances.
Kibana constitue la couche de visualisation permettant de construire des dashboards interactifs pour l'interprétation des données.

Enfin, le module **Elastic Machine Learning** offre la possibilité de détecter automatiquement des anomalies dans les temps de build en analysant les tendances dans les données historiques.

2.6 Architecture de la Solution

2.6.1 Schéma d'Architecture Globale

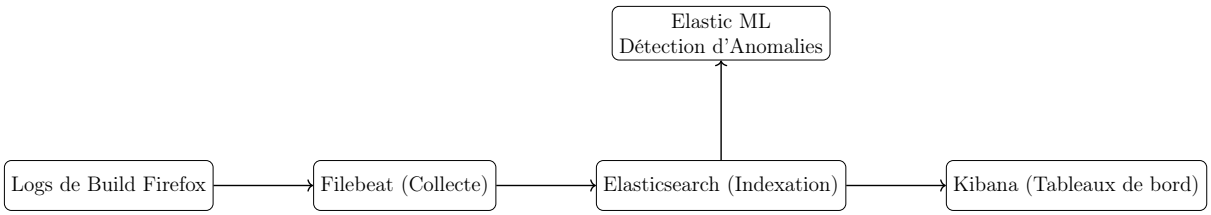

FIGURE 2.1 – Architecture globale de la solution implémentée.

2.7 Conclusion du Chapitre

Ce chapitre a permis de clarifier les besoins du projet, de présenter les caractéristiques des données traitées, et de justifier les choix technologiques adoptés. L'architecture proposée pose ainsi les fondations de la solution. Le chapitre suivant sera consacré à l'implémentation détaillée.

Chapitre 3

Implémentation de la Solution

Introduction

Cette phase a pour objectif de déployer une architecture fonctionnelle capable de collecter, transformer et indexer les journaux de build de Firefox en temps réel. L'implémentation repose sur quatre composants principaux : Filebeat pour la collecte des logs, Logstash pour le parsing et l'enrichissement, Elasticsearch pour l'indexation et Kibana pour l'exploration et l'analyse. L'ensemble de l'environnement est déployé dans un cluster ELK conteneurisé via `docker-compose`.

3.1 Déploiement de l'environnement ELK

L'environnement complet est constitué de quatre services Docker : Elasticsearch, Logstash, Kibana et Filebeat. Le déploiement par conteneurisation permet l'isolation de chaque composant, la reproductibilité de l'environnement et un redéploiement rapide en cas de modification.

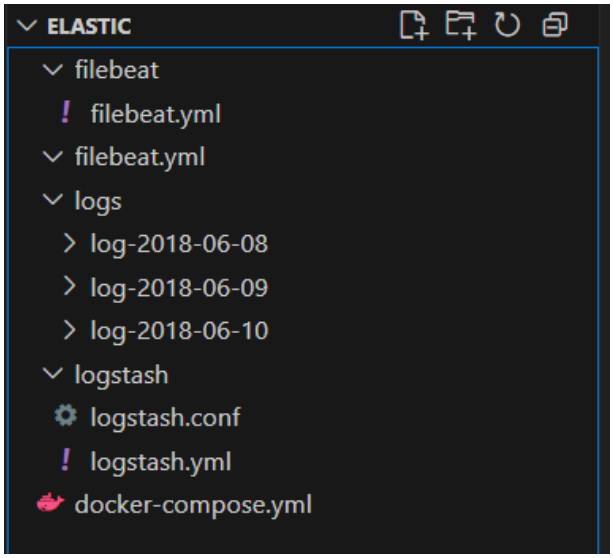

FIGURE 3.1 – Organisation des fichiers du projet ELK.

La Figure 3.1 illustre l'organisation des fichiers utilisée pour le déploiement, incluant Filebeat, Logstash et les logs bruts. Cette structure permet de monter les fichiers de configuration directement dans les conteneurs Docker.

3.1.1 Elasticsearch

Le service Elasticsearch est déployé en mode *single-node* à l'aide de l'image officielle `elasticsearch:7.16.2`. Une allocation mémoire explicite (`-Xms512m -Xmx512m`) limite la consommation des ressources.
Un volume persistant est monté sur `/usr/share/elasticsearch/data` afin de conserver les index en cas de redémarrage. Les ports `9200` (API REST) et `9300` (transport interne) sont exposés pour permettre la communication avec Logstash et Kibana.

3.1.2 Logstash

Logstash sert de couche d'intégration et de transformation. Son pipeline est défini dans un fichier `logstash.conf` monté en lecture seule dans le conteneur. Il écoute les évènements Filebeat sur le port `5044` via l'entrée `beats{}`. Il applique ensuite une chaîne de filtres GROK, de conversions, de normalisation et d'enrichissement, avant d'envoyer les données structurées vers Elasticsearch.

3.1.3 Kibana

Kibana est configuré pour s'interfacer directement avec Elasticsearch via la variable d'environnement `ELASTICSEARCH_HOSTS`. L'interface web est exposée sur le port `5601`. Elle permet de visualiser les index, d'inspecter les documents ingérés et de créer les visualisations utilisées par le tableau de bord final.

3.1.4 Filebeat

Filebeat agit comme collecteur local. Il surveille en continu les fichiers `.txt` situés dans le répertoire `/logs` monté depuis le système hôte. La directive `fields_under_root: true` garantit que les champs personnalisés ajoutés par Filebeat (`log_type`) sont insérés directement à la racine du document JSON, ce qui simplifie le traitement dans Logstash. La redirection de la sortie est effectuée vers Logstash sur le port `5044`.

3.2 Configuration de Filebeat

La configuration de Filebeat assure la collecte des journaux bruts et leur transmission sans transformation à Logstash. Le fichier `filebeat.yml` est volontairement minimaliste , le pattern `/logs//.txt` dans ce fichier permet de capturer tous les fichiers de log générés par les builds, quelle que soit la sous-structure des répertoires. L'agent ne filtre ni n'enrichit les lignes ; il se contente de transmettre les événements de manière fiable et hiérarchisée à Logstash.

3.3 Pipeline Logstash : parsing et enrichissement

Le pipeline Logstash est le composant clé pour transformer les logs bruts en documents structurés exploitables dans Elasticsearch.

3.3.1 Extraction de la date depuis le chemin du fichier

Les fichiers de log sont nommés selon le pattern `log-YYYY-MM-DD`. Une première expression GROK extrait la date directement depuis `log.file.path`, ce qui permet de reconstruire un timestamp complet même pour les lignes ne contenant que l'heure.

3.3.2 Parsing multi-structures via GROK

Les journaux de build présentent une grande diversité de formats. Un bloc GROK multi-pattern est utilisé pour reconnaître et extraire :

— les étapes de build (*step start*, *step finish*),
— les lignes d'état et de résumé,
— les métadonnées (`buildername`, `platform`, etc.),
— les codes d'erreur,
— les niveaux de journalisation,
— les informations de téléchargement.

En cas d'échec de parsing, la ligne est marquée via le tag `_grok_nomatch`.

3.3.3 Construction du timestamp

Lorsque la ligne contient un timestamp complet ISO8601, celui-ci est utilisé comme `@timestamp`. Sinon, Logstash reconstruit un champ intermédiaire `combined_ts` en combinant la date extraite du nom de fichier et l'heure extraite de la ligne. Ce champ est ensuite interprété via le filtre `date`.

3.3.4 Normalisation et conversions

Les champs numériques (`results_count`, `exit_code`, `bytes_downloaded`, etc.) sont convertis en entiers ou flottants. Les champs textuels tels que `loglevel` sont convertis en minuscules pour homogénéiser les analyses dans Elasticsearch.

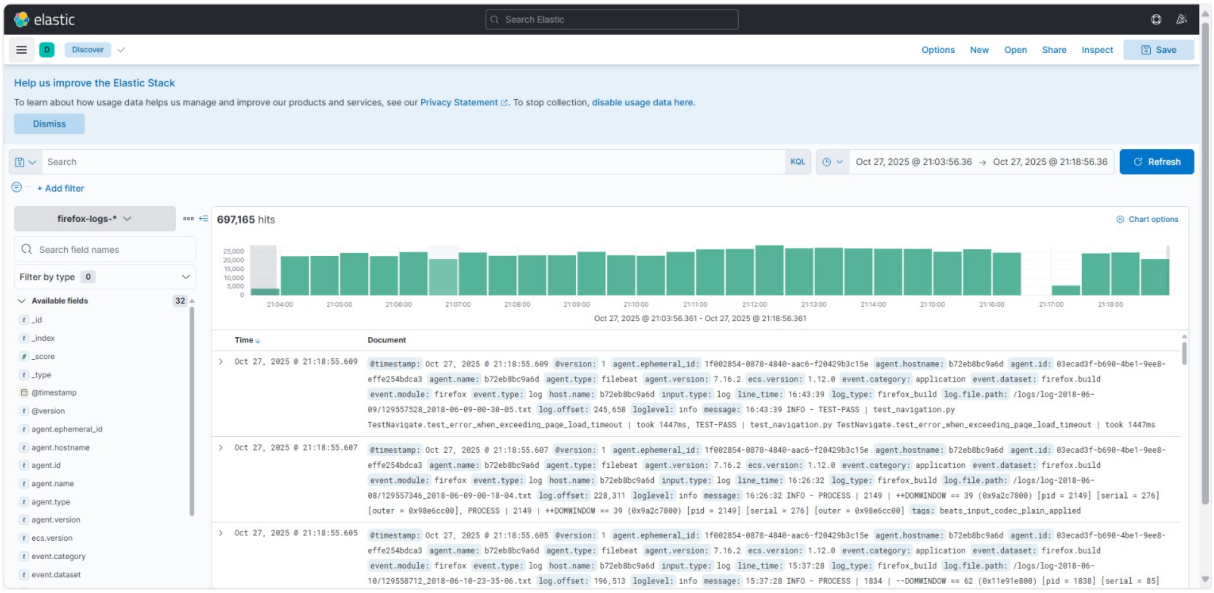

FIGURE 3.2 – Visualisation des documents ingérés dans Kibana Discover après parsing et normalisation via Logstash.

Comme illustré dans la Figure 3.2, les documents ingérés présentent les champs structurés générés par le pipeline Logstash, notamment `loglevel`, `results_text`, `exit_code` et le timestamp reconstruit. Cette étape confirme la bonne application des filtres GROK, des conversions et de la reconstruction temporelle.

3.3.5 Enrichissement via tags et nettoyage

Logstash ajoute automatiquement des tags d'état pour faciliter l'analyse :

— `exit_error` lorsque `exit_code` est non nul,

— `build_warning_or_failure` lorsque `results_text` n'est pas `success`.

Les lignes ne correspondant à aucun motif sont explicitement supprimées via le filtre `drop{}` afin de réduire le bruit dans l'index final.

3.4 Création du template d'index Elasticsearch

Un template d'index statique a été défini pour éviter les mappings dynamiques incohérents. Celui-ci impose explicitement le type de chaque champ, garantit leur agrégabilité et assure leur cohérence à travers tous les index correspondant au motif `firefox-logs-*`.

Les champs critiques tels que `@timestamp`, `log.file.path`, `loglevel`, `results_text`, `exit_code` et les métriques de téléchargement sont définis avec leurs types adaptés (`date`, `keyword`, `integer`, `float`, `long`). Ce template garantit que les visualisations dans Kibana puissent exploiter les agrégations temporelles, les calculs statistiques et les regroupements par catégorie sans erreur de type.

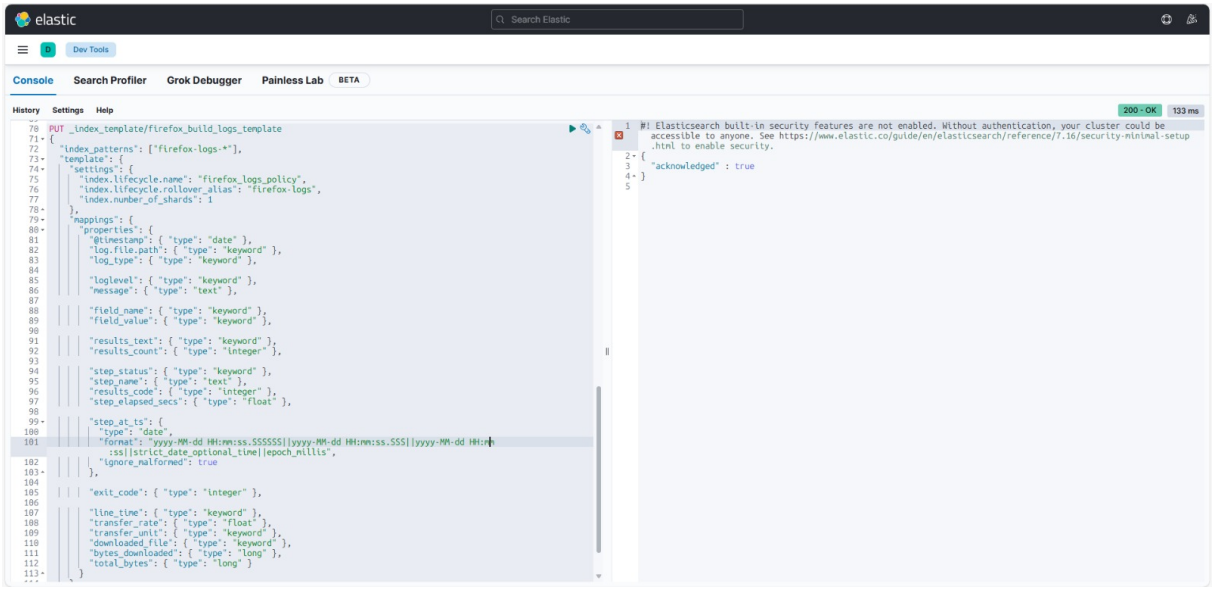

FIGURE 3.3 – Aperçu du mapping de l'index `firefox-logs-*` dans Kibana.

3.5 Construction du Dashboard dans Kibana

3.5.1 Vue d'ensemble du tableau de bord

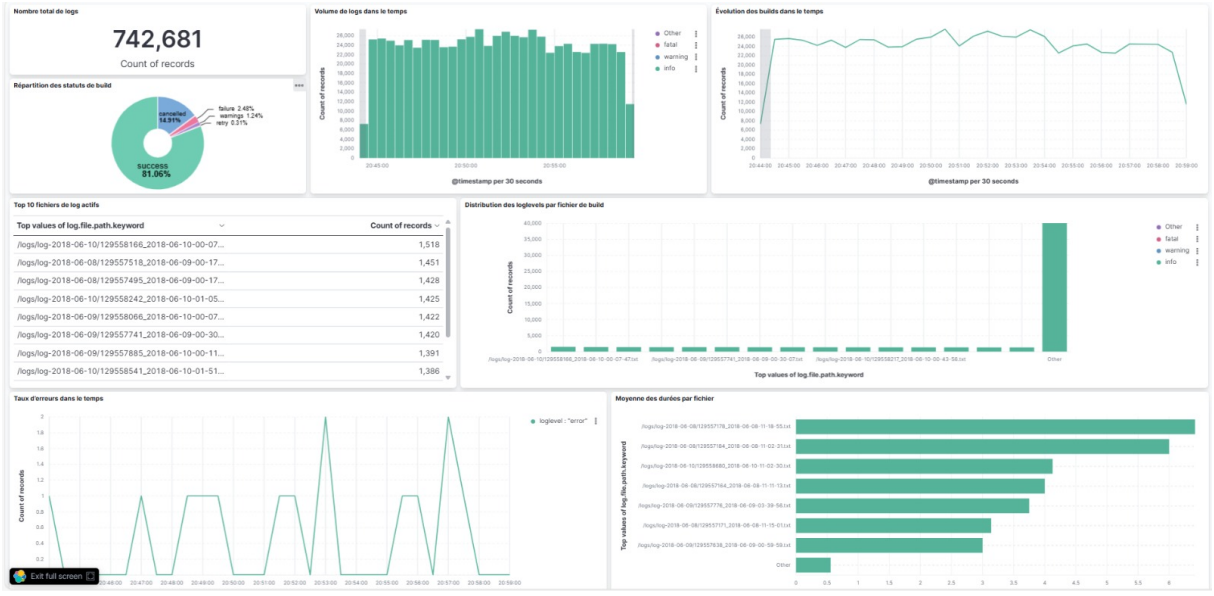

FIGURE 3.4 – Tableau de bord Kibana complet intégrant l'ensemble des métriques de build.

La Figure 3.4 présente la vue globale du tableau de bord conçu dans Kibana. Il regroupe les indicateurs clés nécessaires au suivi des builds Firefox en temps réel : le volume de logs ingérés, la répartition des statuts de build, les variations temporelles, la distribution des niveaux de log ainsi que les statistiques par fichier de build. Cette vue d'ensemble per-

met d'évaluer immédiatement l'état du pipeline d'intégration et de détecter d'éventuelles anomalies (pics d'erreurs, instabilité temporelle, échecs récurrents).

Cette section présente les visualisations du tableau de bord conçu dans Kibana. Les graphiques permettent d'analyser le volume total de logs, la distribution des statuts de build, l'évolution temporelle du flux, la répartition des niveaux de journalisation, la fréquence des erreurs et la durée moyenne des étapes de compilation. L'objectif est d'obtenir une lecture synthétique et diagnostique de l'activité des builds Firefox.

3.5.2 Volume global de données

FIGURE 3.5 – Nombre total de logs indexés.

Le tableau de bord indique un volume total de `739 218` entrées. Ce volume élevé confirme la stabilité du pipeline d'ingestion et l'absence de pertes. L'ordre de grandeur est cohérent avec l'exécution parallèle de plusieurs builds, générant des centaines de milliers de lignes en continu.

3.5.3 Répartition des statuts de build

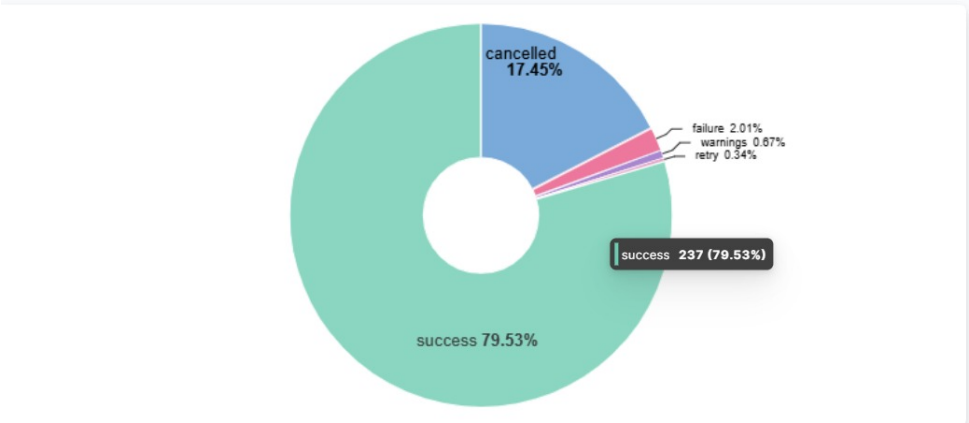

FIGURE 3.6 – Répartition des statuts de build.

La majorité des builds sont marqués `success` (environ 79,5%), ce qui valide la stabilité globale de la chaîne d'intégration. Les builds annulés représentent 17%, tandis que les états *failure*, *warnings* et *retry* restent marginaux. Ce profil est typique d'un environnement CI/CD bien industrialisé, où les annulations sont souvent provoquées par des interruptions manuelles ou des dépendances externes.

3.5.4 Volume de logs dans le temps

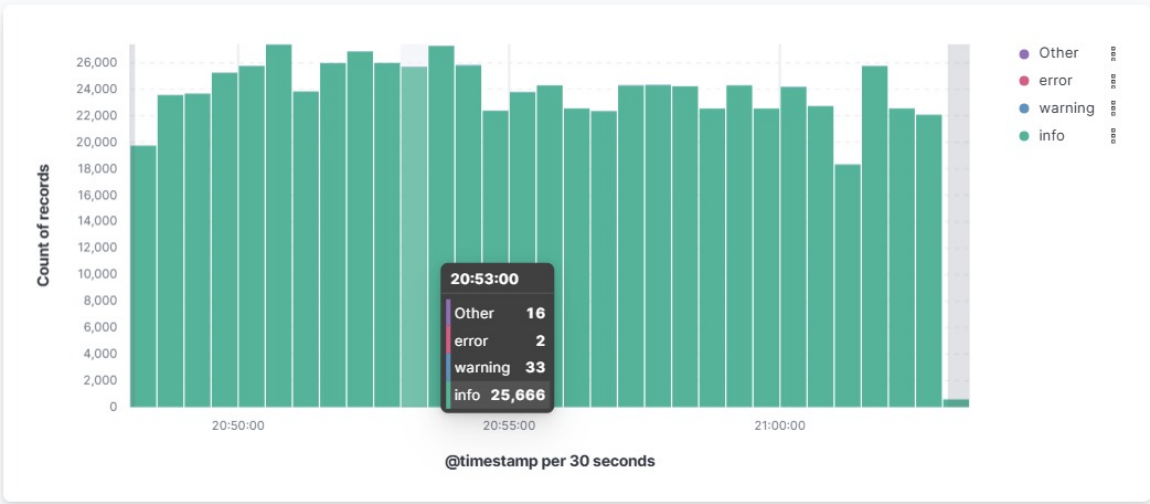

FIGURE 3.7 – Volume de logs par intervalle de 30 secondes.

L'histogramme temporel montre une cadence stable, autour de 24 000 à 26 000 événements par tranche de 30 secondes. Cette régularité confirme que les builds s'exécutent en parallèle de façon homogène. Les variations observées en fin de période correspondent simplement à l'arrêt du flux simulé, et non à un incident.

3.5.5 Évolution du nombre de builds actifs

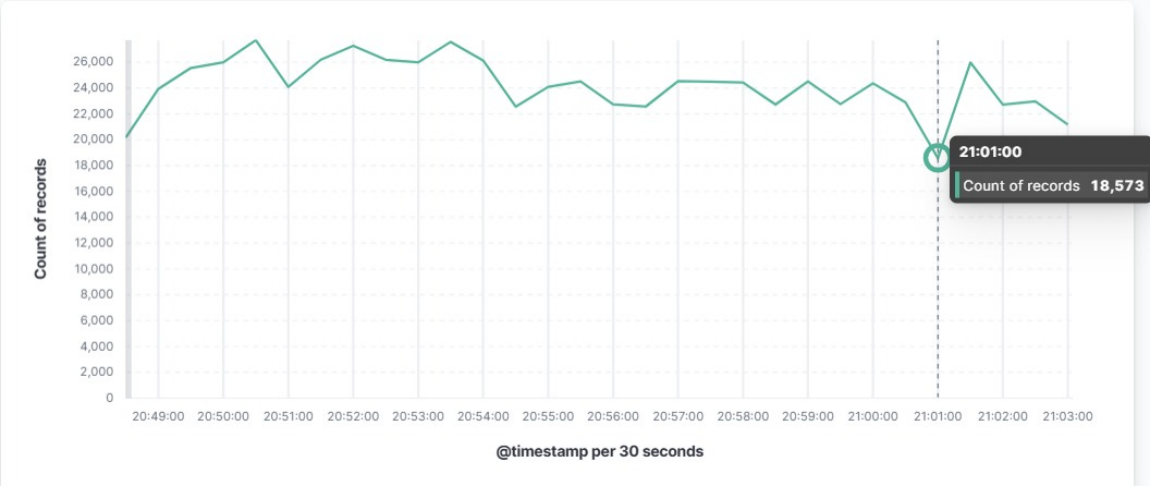

FIGURE 3.8 – Évolution du nombre de builds en cours.

La courbe représentant l'évolution du nombre de builds actifs montre une stabilité globale, avec quelques fluctuations ponctuelles. Les baisses intermédiaires indiquent des fins de build tandis que les hausses signalent le début de nouveaux jobs. L'absence de zones prolongées à zéro confirme que le pipeline ne s'est pas bloqué.

3.5.6 Fichiers de logs les plus actifs

Top values of log.file.path.keyword	Count of records
/logs/log-2018-06-08/129557495_2018-06-09-00-17-45.txt	1,298
/logs/log-2018-06-08/129557361_2018-06-09-00-09-10.txt	1,265
/logs/log-2018-06-10/129558456_2018-06-10-00-50-24.txt	1,231
/logs/log-2018-06-10/129558199_2018-06-10-00-59-33.txt	1,215
/logs/log-2018-06-10/129558687_2018-06-10-11-58-11.txt	1,202
/logs/log-2018-06-10/129558154_2018-06-10-00-53-03.txt	1,163
/logs/log-2018-06-09/129558000_2018-06-10-00-11-44.txt	1,162
/logs/log-2018-06-09/129557732_2018-06-09-00-30-13.txt	1,159
/logs/log-2018-06-09/129557572_2018-06-09-00-52-04.txt	1,158
/logs/log-2018-06-08/129557183_2018-06-08-11-50-06.txt	1,154

FIGURE 3.9 – Fichiers de logs présentant le plus grand volume d'événements.

Les fichiers contenant le plus grand nombre d'événements correspondent principalement aux builds du 10 juin. Cela confirme la cohérence temporelle de l'ingestion et permet d'identifier les builds les plus volumineux. Un volume anormalement élevé dans un fichier peut indiquer un comportement non standard, comme une boucle de log ou un build exceptionnellement verbeux.

3.5.7 Distribution des niveaux de log par fichier

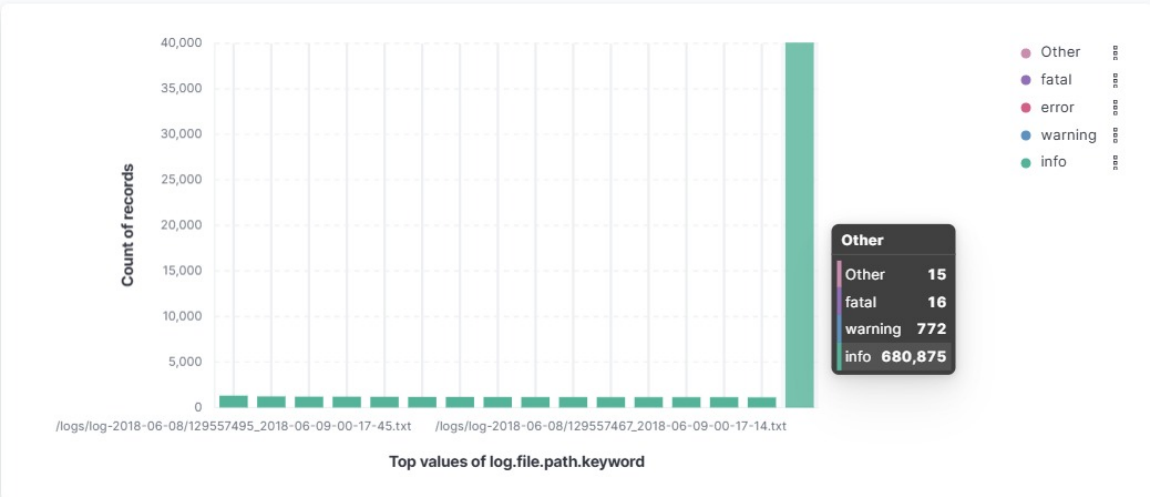

FIGURE 3.10 – Distribution des niveaux de log (`info`, `warning`, `error`, `fatal`) par fichier.

La répartition des niveaux de log est très asymétrique : plus de 680 000 événements sont de type `info`. Les messages `warning`, `error` et `fatal` sont présents mais en quantité faible. Ce comportement est typique d'un système de build massif où les logs d'information

dominent largement. Les pics d'erreurs dans des fichiers spécifiques permettent néanmoins de cibler les builds problématiques.

3.5.8 Taux d'erreurs dans le temps

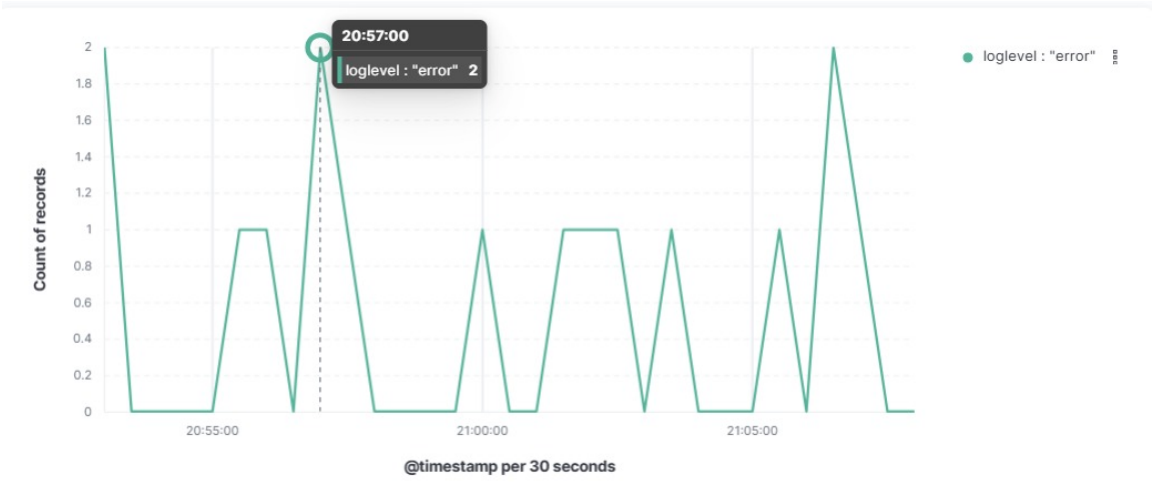

FIGURE 3.11 – Taux d'erreurs (`loglevel = error`) par intervalle de 30 secondes.

La fréquence des erreurs est faible et isolée, rarement plus de deux occurrences par tranche de 30 secondes. Ces anomalies ponctuelles correspondent à des interruptions mineures : timeout, erreurs réseau, ou échecs d'étapes secondaires. L'absence de croissance continue ou de séquence prolongée d'erreurs élimine la possibilité d'une défaillance structurelle.

3.5.9 Durée moyenne des étapes de compilation

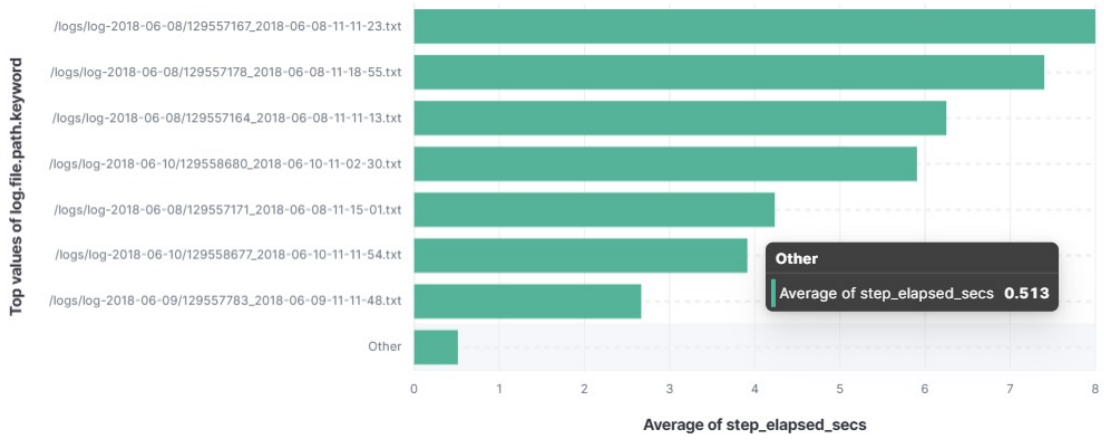

FIGURE 3.12 – Durée moyenne des étapes de build par fichier de log.

Le diagramme horizontal met en évidence une forte variabilité des durées moyennes (`step_elapsed_secs`). Certains builds affichent plus de 7 secondes par étape en moyenne,

tandis que d'autres se situent entre 3 et 5 secondes. Ces différences reflètent des facteurs tels que la plateforme, le cache ou la charge d'exécution. Les outliers (durées très élevées) constituent des candidats évidents pour l'analyse d'anomalies.

3.5.10 Remarque sur le comportement dynamique du tableau de bord

Les indicateurs présentés dans ce tableau de bord sont recalculés à chaque rafraîchissement, car l'ingestion des données simule un flux temps réel. Filebeat rejoue en continu les journaux, Logstash les transmet immédiatement à Elasticsearch, et Kibana interroge l'index à chaque mise à jour. Ainsi, les métriques (nombre total de logs, répartitions, moyennes et histogrammes temporels) évoluent au fil du temps, ce qui reproduit fidèlement le fonctionnement d'un pipeline de supervision en temps réel.

Conslusion

L'architecture ELK déployée permet désormais de traiter l'ensemble du flux de journaux de build de manière complète et fiable. Filebeat assure une collecte continue et hiérarchisée, Logstash transforme et structure précisément les événements, Elasticsearch garantit un stockage cohérent et interrogeable, tandis que Kibana expose les indicateurs clés en temps réel. L'ensemble du pipeline prouve sa robustesse : les logs bruts hétérogènes sont convertis en documents normalisés, typés et enrichis, ce qui permet d'obtenir des visualisations pertinentes et exploitables pour l'analyse opérationnelle. Les mécanismes de parsing, de reconstruction du timestamp, de normalisation et d'enrichissement se traduisent par un tableau de bord capable de suivre les builds, détecter les anomalies et mettre en évidence les tendances critiques. Cette implémentation fournit la base technique solide nécessaire pour les phases suivantes : optimisation, corrélation plus avancée des évènements et extension à de nouveaux scénarios de supervision.

Chapitre 4

Détection et Analyse des Anomalies dans les Logs de Build Firefox

Introduction

Les systèmes d'intégration continue (CI) et de compilation automatisée produisent des volumes considérables de logs, retraçant l'exécution de chaque étape du pipeline. Ces fichiers sont essentiels au diagnostic en cas de défaillance, mais leur inspection manuelle est difficile car les anomalies ne se manifestent pas toujours sous la forme d'erreurs explicites. Dans de nombreux cas, les comportements anormaux se traduisent par des **variations inhabituelles dans le volume ou la répétition des messages**.

Afin d'améliorer la capacité de surveillance et de diagnostic, nous avons mis en place un système de **détection automatique d'anomalies** reposant sur le module **Elastic Machine Learning** intégré à Kibana. Celui-ci permet d'identifier automatiquement les comportements déviants sans avoir à définir manuellement des seuils ou des règles.

4.1 Principe de la Détection d'Anomalies dans Elasticsearch

Elastic ML repose sur un modèle d'apprentissage **non supervisé**, basé sur la théorie de la densité probabiliste : le modèle apprend la distribution *normale* des données à partir de l'historique et détecte toute observation dont la probabilité d'occurrence est faible.

4.1.1 Modélisation du comportement normal

Le modèle construit une estimation dynamique :

$$P(x_t \mid \text{contexte}) \rightarrow \text{score d'anomalie}(x_t)$$

où x_t représente la valeur observée pour un intervalle temporel.
Les anomalies sont détectées lorsque :

$$P(x_t) \ll E[x]$$

ou lorsque le comportement s'écarte d'une tendance apprise (rupture, pic, plateau inhabituel).

4.1.2 Scores d'anomalies

Score d'anomalie	Interprétation
0 – 25	Comportement normal
25 – 50	Déviation légère, à surveiller
50 – 75	Anomalie confirmée
75 – 100	Comportement critique nécessitant investigation immédiate

Cette échelle rend la surveillance intuitive et exploitable même par des non-experts.

4.2 Configuration du Job de Machine Learning

4.2.1 Justification du choix de la métrique

Contrairement à des métriques classiques telles que le temps d'exécution, les logs étudiés ne contiennent pas systématiquement des informations numériques exploitables. En revanche, un comportement anormal génère souvent une **augmentation anormale du nombre de messages** (ex : boucle de retry, avalanche d'avertissements, échecs répétés de dépendances).

Ainsi, la métrique retenue est :

$$\text{count(messages)} \quad \text{par fichier et par intervalle de 15 minutes}$$

4.2.2 Paramétrage du job

- Type de job : **Advanced Job**
- Index source : `firefox-logs-*`
- Fonction d'analyse : **count**
- Partition : `log.file.path.keyword` (modèle indépendant par fichier)
- Influencers : `log.file.path.keyword`, `loglevel.keyword`
- Bucket span (fenêtre temporelle) : **15 minutes**

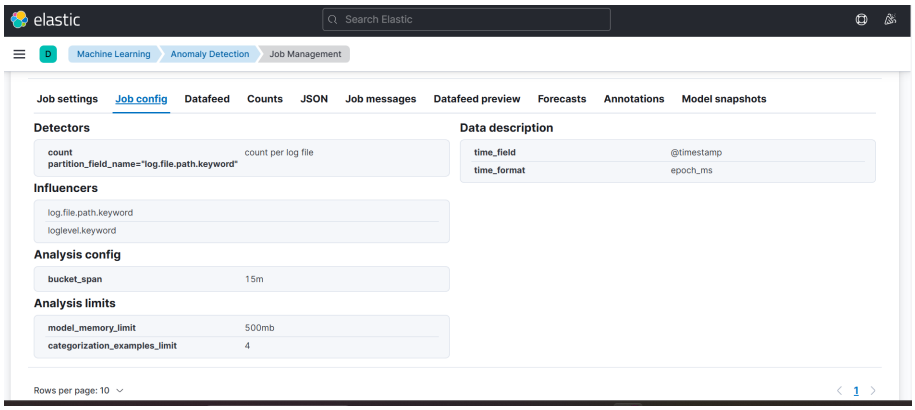

FIGURE 4.1 – Configuration du job de détection d'anomalies dans Kibana.

4.3 Résultats et Interprétation

Le modèle a été exécuté sur **8,59 millions de lignes de logs**. Les visualisations issues de Kibana ont permis de repérer plusieurs anomalies significatives, principalement caractérisées par des pics soudains dans le volume de messages.

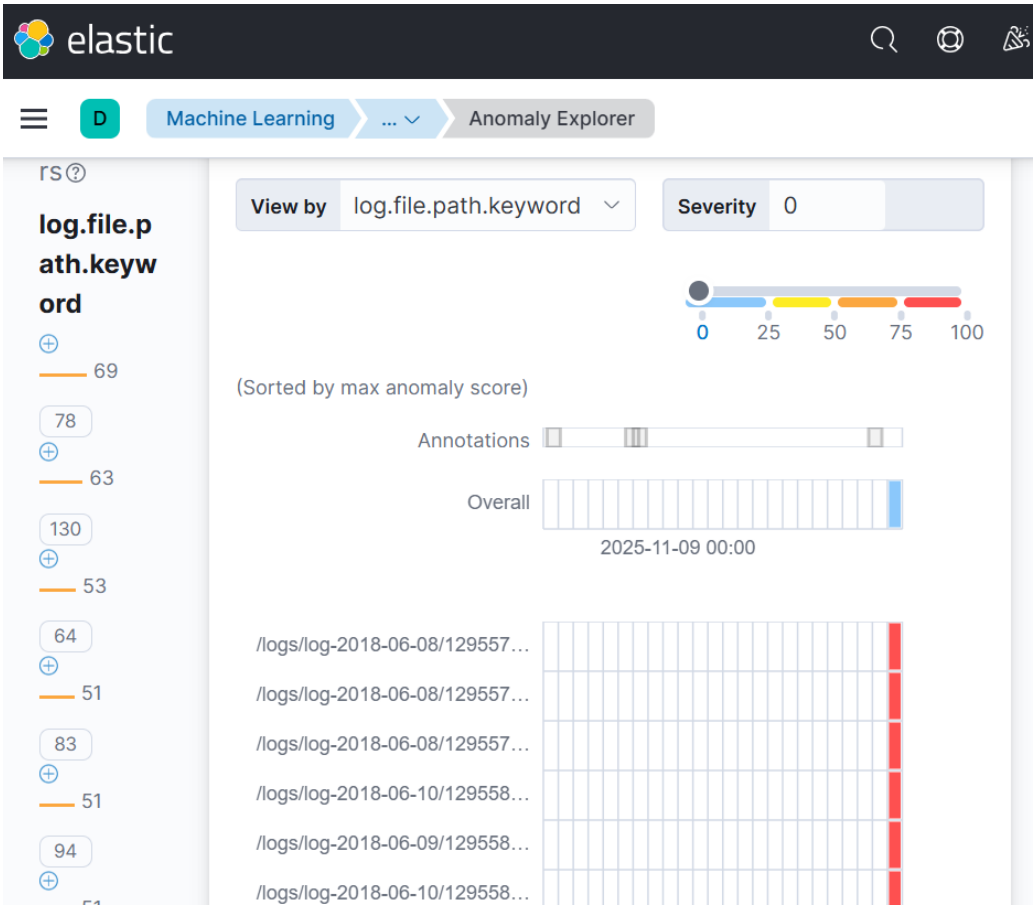

FIGURE 4.2 – Vue globale des anomalies classées par fichier (Anomaly Explorer).

4.3.1 Analyse d'une anomalie représentative

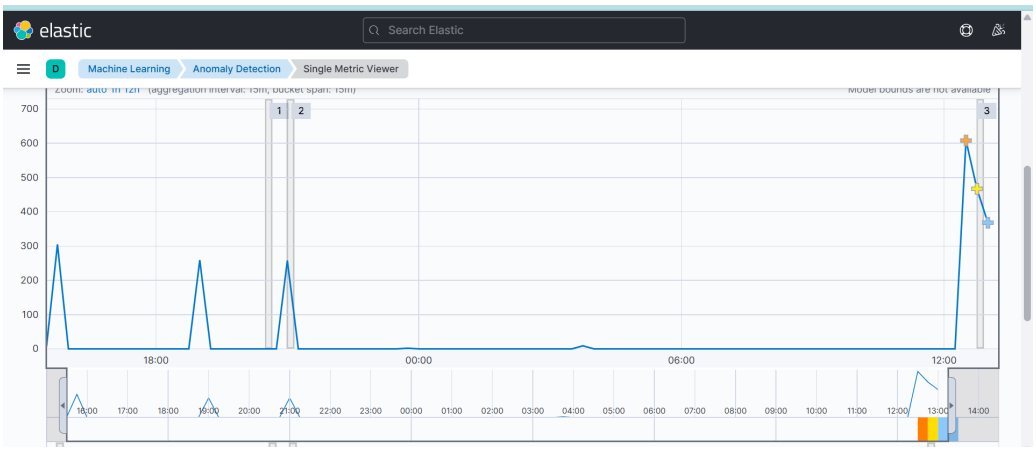

FIGURE 4.3 – Évolution temporelle du volume de logs pour un fichier spécifique avec anomalies annotées.

On observe que certaines périodes présentent une **production de 2 000 à 3 000 messages en 15 minutes**, alors que la valeur typique est située autour de **15 à 40 messages**. Cela représente une augmentation de l'ordre de :

$$\frac{2600}{25} \approx 104\times$$

Ce type d'explosion volumétrique est caractéristique :
— de **boucles d'erreur silencieuses**,
— de **relances de tâches échouées**,
— ou de **dépendances externes instables (ex : artefacts de compilation)**.

4.4 Discussion

4.4.1 Avantages de l'approche

— Aucun seuil manuel à configurer (**auto-adaptatif**).
— Détection **en temps réel** sur de grands volumes.
— Analyse par fichier (**granularité précise**).

4.4.2 Limites observées

— Le modèle identifie des volumes anormaux mais **pas le contenu des messages**.
— L'interprétation finale nécessite encore une analyse humaine.

4.4.3 Améliorations possibles

Une prochaine étape consisterait à intégrer :
— la **catégorisation automatique** des messages (`ml.find_similar`)
— du **clustering sémantique** basé sur BERT ou Word2Vec

— pour détecter non seulement **des volumes** mais aussi **des motifs textuels récurrents**.

Conclusion du Chapitre

La détection automatique d'anomalies via Elastic ML a permis de révéler des comportements anormaux impossibles à détecter efficacement manuellement. Cette intégration améliore non seulement la visibilité sur l'état du pipeline CI, mais réduit également le temps de diagnostic et contribue à la fiabilité globale du système.

Conclusion Générale

Dans ce travail, nous avons conçu et implémenté une solution complète de supervision, d'analyse et de détection d'anomalies appliquée aux logs du processus de build de Firefox. Face à un volume important de données hétérogènes et continuellement générées, l'enjeu était de fournir un système capable non seulement de centraliser et structurer les logs, mais également d'en extraire automatiquement des informations pertinentes pour le diagnostic et la prise de décision.

L'adoption de la suite **ELK** (Elasticsearch, Logstash, Kibana) a permis d'assurer l'ingestion, l'indexation et l'exploration des données de logs de manière fluide et scalable. La construction du **tableau de bord interactif** a permis de visualiser les métriques essentielles : volume de logs, répartition des niveaux de log, évolution temporelle des builds, taux d'erreurs, ainsi que les fichiers les plus actifs. Ce tableau de bord constitue un outil de supervision opérationnelle permettant une compréhension globale et en temps réel de l'état du pipeline de build.

Dans une seconde étape, nous avons intégré le module **Elastic Machine Learning** pour la mise en place d'un mécanisme de **détection automatique d'anomalies**. Ce module, basé sur un modèle non supervisé, a permis d'identifier des comportements atypiques sans définir de seuils manuels. L'analyse a révélé plusieurs pics anormaux de génération de logs, traduisant des boucles silencieuses ou des séquences d'erreurs répétées au sein du pipeline. La visualisation des anomalies via le *Anomaly Explorer* et le *Single Metric Viewer* a facilité leur interprétation, permettant une localisation précise dans le temps et par fichier.

Ainsi, la solution proposée offre une **surveillance proactive**, une **réduction du temps de diagnostic** et une **amélioration de la fiabilité du processus de build**. Elle constitue une base solide pour une stratégie d'observabilité avancée.

Perspectives : Afin d'enrichir davantage la compréhension des anomalies, plusieurs pistes d'amélioration peuvent être envisagées :

- Intégration d'une **analyse sémantique du contenu des logs** (clustering, vectorisation, modèles NLP).
- Détection des patterns d'erreurs récurrents via des modèles de **classification** ou **similarité**.
- Mise en place d'alertes automatiques et intégration avec des outils DevOps (Jenkins, Slack, Jira).

En conclusion, cette approche démontre la pertinence de l'utilisation des technologies Big Data et d'apprentissage automatique pour améliorer la visibilité, la résilience et la performance des pipelines logiciels modernes.