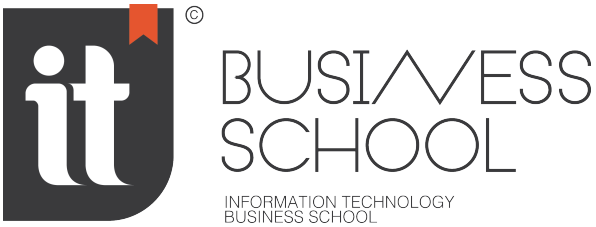
République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Ecole Supérieure Privée des Technologies de l'Information et de Management de Nabeul



Rapport de Mini Projet soumis afin de valider la matière de

Frameworks Big Data

Réalisé par

José Marie Mokeni

Application de Monitoring avec la Stack ELK

Soutenu le 13/12/2024 devant le Jury composé de :

Mr Mohamed Najeh Issaoui Jury

Dedicaces

À ma famille, pour leur amour inconditionnel, leur soutien constant et leurs encouragements infinis. Vous êtes ma source d'inspiration et ma force motrice dans tous mes projets.

À mes amis, pour leur amitié sincère, leurs moments de joie partagés et leur soutien indéfectible. Vos sourires ont illuminé mes journées et vos encouragements m'ont poussés à aller toujours plus loin.

À tous ceux qui croient en moi et m'ont encouragés à poursuivre mes rêves, je vous dédie ce travail. Votre confiance en moi me donne la force de surmonter tous les défis.

À tous ceux qui se battent pour avoir l'avenir qu'ils méritent, que votre détermination et votre courage inspirent ceux qui vous entourent. Votre résilience face aux obstacles est une source d'inspiration pour nous tous.

Enfin, à tous ceux qui ont croisé mon chemin et ont contribué, de près ou de loin, à mon parcours, je vous adresse mes plus sincères remerciements. Votre impact sur ma vie est inestimable, et je vous en suis profondément reconnaissant.

Remerciements

Je tiens tout d'abord à exprimer ma profonde gratitude envers Mr Mohamed Najeh Issaoui pour son soutien continu et son encouragement tout au long de ce projet. Votre expertise et vos conseils précieux ont été d'une importance capitale pour le développement et la réalisation de ce travail. Je tiens également à remercier chaleureusement tous ceux qui ont contribué de près ou de loin à ce projet. Leur collaboration, leurs idées et leur dévouement ont grandement enrichi cette expérience et ont permis d'atteindre les objectifs fixés. Enfin, je souhaite exprimer ma reconnaissance envers ma famille et mes amis pour leur soutien indéfectible et leur encouragement constant tout au long de ce parcours. Merci à tous pour votre contribution précieuse et votre confiance en ce projet.

Tables des matières

Liste des figures

Préambule

La conception et l'implémentation d'un projet étant des aptitudes pré requises chez un ingénieur, il nous a été demandé dans le cadre d'un Mini Projet Framework Big Data, de concevoir, de développer et de déployer une application web de monitoring de logs à l'aide de la stack ELK, de la configuration de la stack à l'exploitation de celle-ci, en passant par la conception des pipeline d'ingestion automatique, en répondant bien entendu à une problématique réelle.

Chapitre 1

Introduction

1.1 Contexte du Projet

Dans un environnement informatique moderne, la gestion et l'analyse des logs système constituent un enjeu majeur pour les entreprises. Face à la multiplication des applications et services, les équipes techniques doivent traiter un volume croissant de données de logs, provenant de sources diverses et présentant des formats hétérogènes.
Ce projet s'inscrit dans une démarche d'amélioration de la surveillance et de l'analyse des performances système au sein d'une entreprise. L'objectif est de centraliser et d'exploiter efficacement les données de logs générées par différentes applications internes.

1.2 Problématique

Les principaux défis auxquels l'entreprise est confrontée sont :

- La dispersion des logs à travers différents systèmes et applications
- L'hétérogénéité des formats de logs (CSV, JSON, formats texte)
- La difficulté d'analyse en temps réel des événements critiques
- Le besoin d'une visualisation claire et personnalisable des données
- La nécessité d'une interface utilisateur intuitive pour la manipulation des logs

1.3 Objectifs du Projet

Le projet vise à mettre en place une solution complète de monitoring des logs reposant sur la stack ELK (Elasticsearch, Logstash, Kibana), complétée par une interface web développée avec Flask. Les objectifs spécifiques sont :

- Centraliser la collecte des logs provenant de différentes sources
- Normaliser le traitement des différents formats de logs
- Proposer des tableaux de bord de visualisation pertinents

- Faciliter l'analyse et la recherche dans les logs via une interface web intuitive
- Permettre l'injection de nouveaux logs dans le système
- Assurer la persistance et la traçabilité des données

1.4 Périmètre du Projet

Le système mis en place traite trois types principaux de logs :

- Les logs d'accès Nginx au format JSON
- Les logs de requêtes lentes MySQL
- Les métriques système (CPU, mémoire, disque, réseau)

L'ensemble de la solution est conteneurisée via Docker, facilitant ainsi son déploiement et sa maintenance. Le projet inclut également un générateur de logs permettant de simuler des données pour les tests et la validation du système.

Chapitre 2

Architecture globale

2.1 Vue d'ensemble

L'architecture du système de monitoring repose sur trois piliers principaux :

- La Stack ELK (Elasticsearch, Logstash, Kibana) pour le traitement et la visualisation des logs
- Une interface web développée avec Flask pour faciliter l'interaction avec le système
- Un système de stockage et de persistance des données

L'ensemble de ces composants est conteneurisé via Docker, assurant ainsi une portabilité et un déploiement simplifié de la solution.

2.2 Composants de l'Architecture

2.2.1 Sources de Logs

Le système prend en charge trois types principaux de logs :

- **Logs Nginx** : Logs d'accès au format JSON contenant les informations sur les requêtes HTTP
- **Logs MySQL** : Logs des requêtes lentes avec les temps d'exécution et les métriques de performance
- **Métriques Système** : Données de performance système (CPU, mémoire, disque, réseau)

2.2.2 Stack ELK

Logstash

Logstash assure le traitement des logs avec les fonctionnalités suivantes :

- Pipeline de traitement configuré pour chaque type de log
- Parsing et transformation des données via des filtres Grok
- Normalisation des timestamps et enrichissement des données
- Acheminement vers Elasticsearch

Elasticsearch

Base de données distribuée assurant :

- Indexation des logs par type et par date
- Recherche full-text et filtrage avancé
- Stockage optimisé pour les requêtes analytiques
- API REST pour l'interaction avec les autres composants

Kibana

Interface de visualisation offrant :

- Tableaux de bord personnalisés par type de log
- Visualisations graphiques des métriques clés
- Interface de requêtage des logs
- Export des données et rapports

2.3 Interface Web Flask

L'application web développée avec Flask propose :

- Upload de fichiers logs via interface drag & drop
- Historique des logs uploadés
- Interface de recherche dans les logs
- Intégration des tableaux de bord Kibana

2.4 Conteneurisation et Déploiement

L'architecture est conteneurisée via Docker avec :

- Un container pour chaque composant de la stack ELK
- Un container pour l'application Flask
- Des volumes persistants pour les données
- Un réseau dédié pour la communication inter-containers

Le fichier `docker-compose.yaml` orchestre l'ensemble des services et définit :

- Les dépendances entre services
- Les volumes et points de montage
- Les variables d'environnement
- Les ports exposés

2.5 Flux de Données

Le flux de données dans le système suit le parcours suivant :

1. Upload des fichiers logs via l'interface web
2. Stockage temporaire dans le système de fichiers
3. Traitement par Logstash (parsing et transformation)
4. Indexation dans Elasticsearch
5. Visualisation via Kibana ou l'interface web

2.6 Diagramme d'Architecture

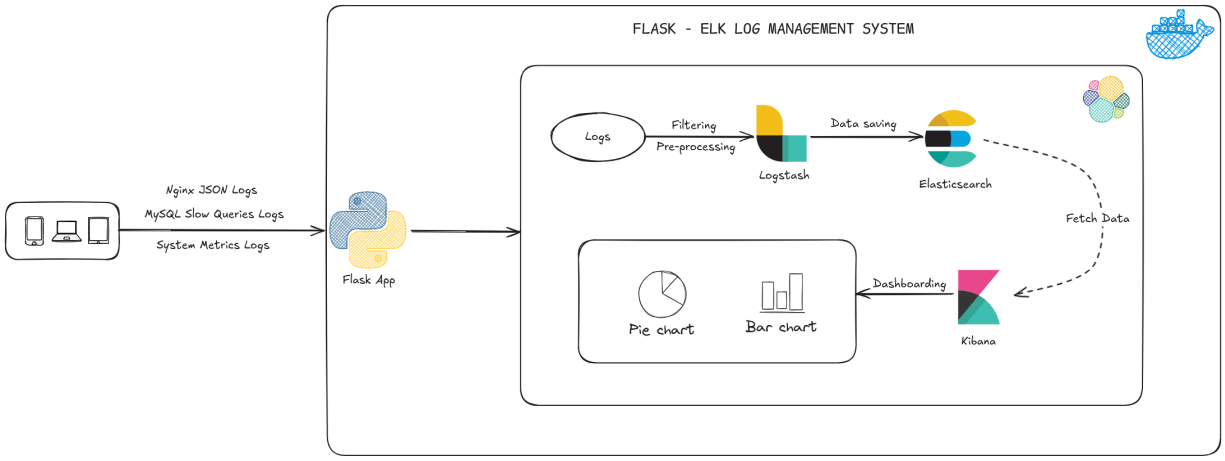


Figure 2.1: Architecture globale de la solution

Chapitre 3

Configuration de la stack ELK

3.1 Configuration d'Elasticsearch

Elasticsearch est configuré comme un nœud unique pour ce projet, optimisé pour un environnement de développement.

3.1.1 Configuration de Base

Le fichier `elasticsearch.yml` contient la configuration principale :

Listing 3.1: elasticsearch.yml

```
cluster.name: "es-docker-cluster"
network.host: 0.0.0.0
xpack.security.enabled: false
discovery.type: single-node
```

Cette configuration :

- Désactive la sécurité X-Pack pour simplifier le développement
- Configure un cluster à nœud unique
- Permet l'accès depuis n'importe quelle adresse IP

3.1.2 Configuration Docker

Dans le fichier `docker-compose.yaml`, Elasticsearch est configuré avec :

- 2GB de mémoire JVM allouée
- Un volume persistant pour les données
- Le port 9200 exposé pour l'API REST
- Des healthchecks pour assurer la disponibilité du service

Listing 3.2: Configuration Elasticsearch dans docker-compose.yaml

```
elasticsearch:
  image: elasticsearch:8.12.0
  environment:
    - node.name=elasticsearch
    - cluster.name=es-docker-cluster
    - discovery.type=single-node
    - bootstrap.memory_lock=true
    - "ES_JAVA_OPTS=-Xms2g -Xmx2g"
    - xpack.security.enabled=false
  volumes:
    - elasticsearch-data:/usr/share/elasticsearch/data
  ports:
    - "9200:9200"
```

3.2 Configuration de Logstash

Logstash est configuré pour traiter trois types de logs différents avec des pipelines spécifiques.

3.2.1 Configuration Principale

Le fichier `logstash.yml` définit les paramètres globaux :

Listing 3.3: logstash.yml

```
http.host: "0.0.0.0"
xpack.monitoring.elasticsearch.hosts: [ "http://elasticsearch:9200" ]
```

3.2.2 Pipeline de Traitement

Le fichier `logstash.conf` définit trois pipelines pour les différents types de logs :

Listing 3.4: Pipeline Logstash - Section Input

```
input {
  # MySQL Slow Query Logs
  file {
    path => "/logs/mysql/*.log"
    type => "mysql-slow"
    start_position => "beginning"
    sincedb_path => "/dev/null"
    mode => "tail"
    codec => multiline {
      pattern => "^#␣Time"
      negate => true
      what => "previous"
    }
  }

```

```
  # Nginx Logs (JSON)
  file {
    path => "/logs/nginx/*.log"
    type => "nginx-access"
    codec => json
  }

  # System Metrics
  file {
    path => "/logs/system/*.log"
    type => "system-metrics"
  }
}
```

3.2.3 Filtres de Traitement

Les filtres sont configurés pour chaque type de log :

Listing 3.5: Pipeline Logstash - Section Filter

```
filter {
  if [type] == "mysql-slow" {
    grok {
      match => {
        "message" => [
          "# Time: %{TIMESTAMP_ISO8601:timestamp}",
          "# User@Host: %{DATA:user}",
          "# Query_time: %{NUMBER:query_time:float}"
        ]
      }
    }
  }

  else if [type] == "nginx-access" {
    date {
      match => [ "timestamp", "ISO8601" ]
      target => "@timestamp"
    }
  }

  else if [type] == "system-metrics" {
    grok {
      match => {
        "message" => "%{WORD:metric_name} %{NUMBER:metric_value:float}"
      }
    }
  }
}
```

3.3 Configuration de Kibana

Kibana est configuré pour se connecter à Elasticsearch et fournir des tableaux de bord personnalisés.

3.3.1 Configuration de Base

Le fichier `kibana.yml` contient :

Listing 3.6: kibana.yml

```
server.name: kibana
server.host: "0.0.0.0"
elasticsearch.hosts: [ "http://elasticsearch:9200" ]
monitoring.ui.container.elasticsearch.enabled: true
```

3.3.2 Tableaux de Bord

Trois tableaux de bord principaux ont été créés :

Dashboard Métriques Système

- Visualisations de type "Jauge" pour CPU, disque, mémoire
- Graphiques d'évolution temporelle
- Métriques réseau entrantes/sortantes

Dashboard Nginx

- Top 5 des adresses IP
- Répartition des codes HTTP
- Temps de réponse moyen
- URIs les plus consultées

Dashboard MySQL

- Temps moyen des requêtes
- Requêtes les plus lentes
- Nombre de lignes examinées
- Distribution des temps de lock

Chapitre 4

Application Web Flask

4.1 Architecture de l'Application

4.1.1 Structure du Projet

L'application Flask suit une architecture modulaire avec la structure suivante :

Listing 4.1: Structure du projet Frontend

```
frontend/
          app.py
          Dockerfile
          requirements.txt
          templates/
              base.html
              dashboard.html
              history.html
              search.html
              upload.html
```

4.1.2 Configuration de l'Application

L'application Flask est configurée avec les paramètres suivants :

Listing 4.2: Configuration Flask

```
app = Flask(__name__)
app.config['SECRET_KEY'] = 'this-is-my-secret-key'
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024  # 16MB max file size

# Configuration des dossiers de logs
LOG_DIRS = {
    'mysql': os.path.join(app.config['UPLOAD_FOLDER'], 'mysql'),
    'nginx': os.path.join(app.config['UPLOAD_FOLDER'], 'nginx'),
    'system': os.path.join(app.config['UPLOAD_FOLDER'], 'system')
}
```

4.2 Fonctionnalités Principales

4.2.1 Gestion des Uploads

Le système d'upload permet de gérer différents types de fichiers logs :

Listing 4.3: Gestion des uploads

```
@app.route('/', methods=['GET', 'POST'])
def upload_files():
    if request.method == 'POST':
        if 'files[]' not in request.files:
            flash('No files selected')
            return redirect(request.url)

        files = request.files.getlist('files[]')

        for file in files:
            if file and allowed_file(file.filename):
                file_content = file.read()
                log_type = detect_log_type(file_content)

                if log_type:
                    save_dir = LOG_DIRS[log_type]
                    timestamp = datetime.now().strftime('%Y%m%d-%H%M%S')
                    new_filename = f"{log_type}-{timestamp}{original_ext}"
                    file_path = os.path.join(save_dir, new_filename)
                    file.save(file_path)
```

4.2.2 Détection Automatique du Type de Log

L'application implémente un système intelligent de détection du type de log :

Listing 4.4: Détection du type de log

```
def detect_log_type(file_content):
    if isinstance(file_content, bytes):
        content = file_content.decode('utf-8', errors='ignore')

    # MySQL slow query patterns
    if re.search(r'# Time:|# User@Host:|# Query_time:', content):
        return 'mysql'

    # Nginx access log patterns (JSON)
    if re.search(r'{"timestamp":.*"request_method":', content):
        return 'nginx'

    # System metrics patterns
    if re.search(r'(cpu_usage|memory_usage|disk_usage)', content):
        return 'system'

```

```
    return None
```

4.2.3 Interface de Recherche

L'application offre une interface de recherche avancée dans Elasticsearch :

Listing 4.5: Interface de recherche

```
@app.route('/search', methods=['GET', 'POST'])
def search_logs():
    if request.method == 'POST':
        query_text = request.form.get('query')
        log_type = request.form.get('log_type')

        index_name = f"{log_type}-*"
        search_body = {
            "query": {
                "query_string": {
                    "query": query_text
                }
            }
        }

        res = es.search(index=index_name, body=search_body, size=50)
        hits = res['hits']['hits']
```

4.3 Interface Utilisateur

4.3.1 Templates HTML

L'application utilise un système de templates avec héritage :

Listing 4.6: Template de base

```
<!-- base.html -->
<!DOCTYPE html>
<html>
<head>
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    <nav>
        <a href="{{ url_for('upload_files') }}">Upload</a>
        <a href="{{ url_for('upload_history') }}">History</a>
        <a href="{{ url_for('search_logs') }}">Search</a>
    </nav>
    {% block content %}{% endblock %}
</body>
</html>
```

4.3.2 Intégration des Tableaux de Bord

Les tableaux de bord Kibana sont intégrés via des iframes :

Listing 4.7: Configuration des dashboards

```
DASHBOARD_URLS = {
    'mysql': "http://localhost:5601/app/dashboards#/view/...",
    'nginx': "http://localhost:5601/app/dashboards#/view/...",
    'system': "http://localhost:5601/app/dashboards#/view/..."
}

@app.route('/dashboard')
def show_dashboard():
    dashboard_type = request.args.get('dashboard')
    if dashboard_type not in DASHBOARD_URLS:
        flash('Invalid dashboard selection')
        return redirect(url_for('upload_files'))

    dashboard_url = DASHBOARD_URLS[dashboard_type]
    return render_template('dashboard.html',
                           dashboard_url=dashboard_url)
```

4.4 Sécurité et Gestion des Erreurs

4.4.1 Validation des Fichiers

Mise en place de contrôles de sécurité pour les uploads :

- Vérification des extensions autorisées
- Limitation de la taille des fichiers
- Nettoyage des noms de fichiers
- Gestion des permissions

4.4.2 Gestion des Erreurs

Implémentation d'un système de gestion des erreurs :

- Messages flash pour les retours utilisateur
- Logging des erreurs
- Redirection en cas d'erreur
- Validation des entrées utilisateur

4.4.3 Conteneurisation

L'application est conteneurisée avec Docker :

Listing 4.8: Dockerfile Frontend

```
FROM python:3.9-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .
ENV FLASK_APP=app.py
ENV FLASK_ENV=development

CMD ["flask", "run", "--host=0.0.0.0"]
```

Chapitre 5

Déploiement

5.1 Approche de la Conteneurisation

5.1.1 Avantages de la Conteneurisation

La conteneurisation de l'application avec Docker offre plusieurs avantages majeurs :

- **Portabilité** : L'application peut être déployée sur n'importe quel système supportant Docker
- **Reproductibilité** : Garantie du même environnement d'exécution partout
- **Isolation** : Chaque service s'exécute dans son propre conteneur
- **Scalabilité** : Possibilité de répliquer facilement les services
- **Facilité de déploiement** : Une seule commande pour déployer l'ensemble de la stack

5.2 Configuration Docker

5.2.1 Architecture des Conteneurs

L'application est composée de plusieurs conteneurs interconnectés :

Listing 5.1: docker-compose.yaml

```
services:
  elasticsearch:
    image: elasticsearch:8.12.0
    environment:
      - discovery.type=single-node
      - "ES_JAVA_OPTS=-Xms2g -Xmx2g"
    volumes:
      - elasticsearch-data:/usr/share/elasticsearch/data
    ports:
      - "9200:9200"

  logstash:
    image: logstash:8.12.0
```

```
    volumes:
      - ./logstash/config:/usr/share/logstash/config
      - ./logstash/pipeline:/usr/share/logstash/pipeline
      - ./logs:/logs
    depends_on:
      elasticsearch:
        condition: service_healthy

  kibana:
    image: kibana:8.12.0
    environment:
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
    ports:
      - "5601:5601"
    depends_on:
      elasticsearch:
        condition: service_healthy

  frontend:
    build:
      context: ./frontend
    ports:
      - "80:5000"
    volumes:
      - ./logs:/logs
```

5.2.2 Gestion des Volumes

Le système utilise plusieurs volumes Docker pour la persistance des données :

- `elasticsearch-data` : Stockage des index Elasticsearch
- `./logs` : Dossier partagé pour les fichiers de logs
- `./logstash/config` et `./logstash/pipeline` : Configurations Logstash

5.2.3 Réseau Docker

Un réseau dédié est créé pour la communication entre les services :

Listing 5.2: Configuration réseau

```
networks:
  elk:
    driver: bridge
```

5.3 Déploiement de l'Application

5.3.1 Prérequis

Pour déployer l'application, seuls les éléments suivants sont nécessaires :

- Docker Engine (version 20.10+)
- Docker Compose (version 2.0+)
- Minimum 4GB de RAM disponible
- 10GB d'espace disque

5.3.2 Procédure de Déploiement

Le déploiement se fait en une seule commande :

Listing 5.3: Commande de déploiement

```
docker-compose up -d
```

Cette commande :

- Télécharge les images nécessaires
- Crée les volumes requis
- Configure le réseau
- Démarre les services dans l'ordre correct
- Active les healthchecks

5.3.3 Vérification du Déploiement

Pour vérifier que tous les services sont opérationnels :

Listing 5.4: Vérification des services

```
# Verifier l'etat des conteneurs
docker-compose ps

# Consulter les logs des services
docker-compose logs -f [service_name]

# Tester les endpoints
curl http://localhost:9200  # Elasticsearch
curl http://localhost:5601  # Kibana
curl http://localhost:80    # Frontend
```

5.4 Maintenance et Opérations

5.4.1 Commandes Utiles

Principales commandes pour la gestion quotidienne :

Listing 5.5: Commandes de maintenance

```
# Arreter les services
docker-compose down

# Redemarrer un service sp cifique
docker-compose restart [service_name]

# Mettre a jour les images
docker-compose pull

# Voir les logs en temps r el
docker-compose logs -f
```

5.4.2 Sauvegarde des Données

Les données importantes sont automatiquement persistées grâce aux volumes Docker :

- Les index Elasticsearch sont stockés dans le volume `elasticsearch-data`
- Les fichiers de logs sont conservés dans le dossier `./logs`
- Les configurations sont versionnées dans le dépôt du projet

5.5 Possibilités de Déploiement Cloud

Bien que l'application n'ait pas été déployée sur le cloud dans le cadre de ce projet, sa conteneurisation permet un déploiement facile sur différentes plateformes :

- AWS ECS (Elastic Container Service)
- Google Cloud Run
- Azure Container Instances
- Kubernetes (AKS, GKE, EKS)
- DigitalOcean App Platform

La seule adaptation nécessaire serait la configuration des variables d'environnement spécifiques à chaque plateforme cloud.

Conclusion

Synthèse du Projet

Ce projet de monitoring avec la stack ELK a permis de mettre en place une solution complète et fonctionnelle de gestion des logs. Les principaux objectifs ont été atteints :

- **Centralisation des Logs** : Mise en place d'un système centralisé capable de collecter et traiter différents types de logs (Nginx, MySQL, métriques système)
- **Traitement Automatisé** : Configuration de pipelines Logstash pour le traitement et la normalisation automatique des logs
- **Visualisation Efficace** : Création de tableaux de bord Kibana adaptés à chaque type de log, permettant une analyse visuelle pertinente
- **Interface Utilisateur** : Développement d'une application web Flask intuitive pour l'upload, la recherche et la visualisation des logs
- **Solution Portable** : Conteneurisation complète de l'application permettant un déploiement simplifié

Points Forts de la Solution

Aspects Techniques

- Architecture modulaire et extensible
- Détection automatique du type de logs
- Traitement en temps réel des données
- Persistance des données via Docker volumes
- Configuration flexible et adaptable

Aspects Fonctionnels

- Interface utilisateur intuitive
- Visualisations personnalisées par type de log
- Recherche avancée dans les logs

- Gestion de l'historique des uploads
- Intégration transparente avec Kibana

Axes d'Amélioration

Sécurité

La solution pourrait être améliorée par l'ajout de :

- Authentification des utilisateurs
- Chiffrement des données sensibles
- HTTPS pour toutes les communications
- Gestion des rôles et permissions
- Audit des accès et actions

Performance

Plusieurs optimisations sont envisageables :

- Mise en place de Redis pour le cache
- Configuration d'un cluster Elasticsearch
- Optimisation des index Elasticsearch
- Compression des logs archivés
- Load balancing pour l'interface web

Fonctionnalités

De nouvelles fonctionnalités pourraient enrichir la solution :

- Système d'alerting
- Export des données et rapports
- API REST complète
- Support de formats de logs additionnels
- Tableaux de bord personnalisables

Perspectives

Ce projet constitue une base solide pour un système de monitoring des logs, avec plusieurs perspectives d'évolution :

1. **Mise à l'échelle** : Évolution vers une architecture distribuée pour gérer de plus grands volumes de données
2. **Intégration Cloud** : Déploiement sur des plateformes cloud pour bénéficier de leur scalabilité et disponibilité
3. **Machine Learning** : Intégration d'algorithmes de détection d'anomalies et de prédiction
4. **Automatisation** : Mise en place de CI/CD pour les mises à jour et le déploiement
5. **Monitoring Avancé** : Ajout de métriques métier et d'indicateurs de performance clés

Ces améliorations permettraient d'enrichir la solution tout en conservant sa simplicité d'utilisation et sa flexibilité, qui sont ses points forts actuels.