Documentation de la Mise en Place des Logs et du Monitoring avec Elasticsearch et Kibana

Introduction

Cette documentation présente comment j'ai mis en place le système de logs unifié pour mon application de **détection d'anomalies** et comment ces logs sont envoyés à **Elasticsearch** pour être visualisés via **Kibana**. Ce guide couvre l'installation, la configuration, l'intégration des logs et l'utilisation des visualisations dans Kibana.

Objectif

L'objectif est de capturer les événements, les performances et les erreurs de l'application pour les analyser et surveiller en temps réel via un tableau de bord Kibana. Cela permet d'identifier les anomalies dans le traitement des données et d'améliorer la performance des modèles.

Pré-requis

- **Elasticsearch** et **Kibana** installés et configurés
- **Python 3.9+**
- **MLflow**, **Streamlit**, **Pandas**, **psutil**
- Compte sur Elasticsearch cloud avec accès à un index.

Étapes de Mise en Place des Logs

1. Configuration de la Connexion à Elasticsearch

La première étape consiste à établir une connexion avec Elasticsearch. Pour ce faire, nous utilisons la bibliothèque **Elasticsearch-py** dans notre code Python.

```
from elasticsearch import Elasticsearch

# Connexion à Elasticsearch
es = Elasticsearch(
    ["https://votre-instance-cloud-elasticsearch"],
    basic_auth=("votre-utilisateur", "votre-mot-de-passe")
)
```

2. Création de la Fonction d'Envoi des Logs

La fonction suivante permet d'envoyer des logs vers un index Elasticsearch spécifique. Cette fonction sera appelée chaque fois qu'un événement important (comme une prédiction ou une détection d'anomalie) se produit dans l'application.

```
def send_log_to_elastic(log_data):
    """
    Envoie les logs à Elasticsearch.

    Args:
        log_data (dict): Dictionnaire contenant les informations de log.
    """
    es.index(index="logs_votre_projet", body=log_data)
```

3. Structuration des Logs Unifiés

Tous les logs sont envoyés dans un format structuré et unifié. Voici un exemple de log pour une exécution réussie d'un modèle de détection d'anomalies :

```
log_data = {
    "timestamp": datetime.utcnow().isoformat(),
    "event": "model_execution",
    "model_name": "IsolationForest",
    "model_version": "1.0.0",
    "application_name": "AnomalyDetectionApp",
    "status": "completed",
    "response_time": response_time,
    "cpu_usage": cpu_usage,
    "memory_usage": memory_usage,
    "details": {
        "anomalies_count": anomalies_count,
        "successful_predictions": successful_predictions,
        "failed_predictions": failed_predictions
    },
    "log_level": "INFO"
}
```

Cette structure de log comprend les informations suivantes :

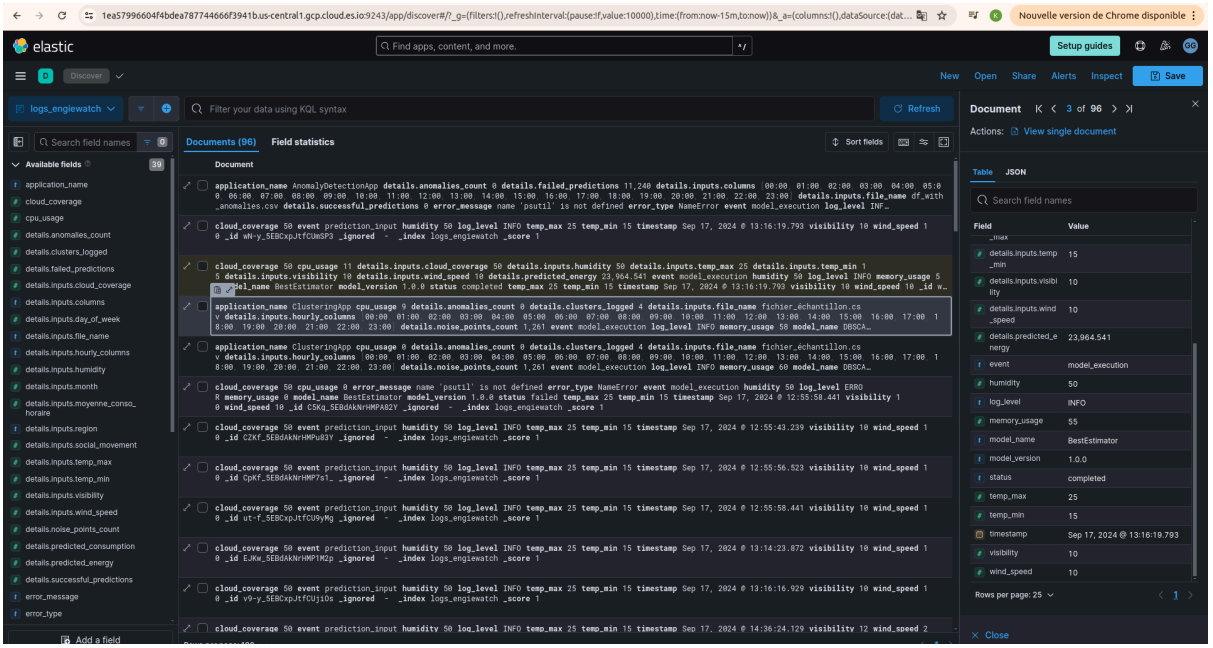
- **event** : Le type d'événement (par exemple, exécution d'un modèle)
- **model_name** : Le nom du modèle utilisé
- **status** : Le statut de l'événement (réussi ou échoué)

- **response_time** : Le temps d'exécution du modèle
- **cpu_usage** et **memory_usage** : Les métriques système pendant l'exécution
- **details** : Des informations spécifiques au modèle et à l'événement (comme le nombre d'anomalies détectées).

4. Implémentation des Logs dans l'Application

Nous avons intégré le logging à différents moments du cycle de vie des modèles dans l'application. Voici un exemple d'intégration dans la fonction de détection d'anomalies :

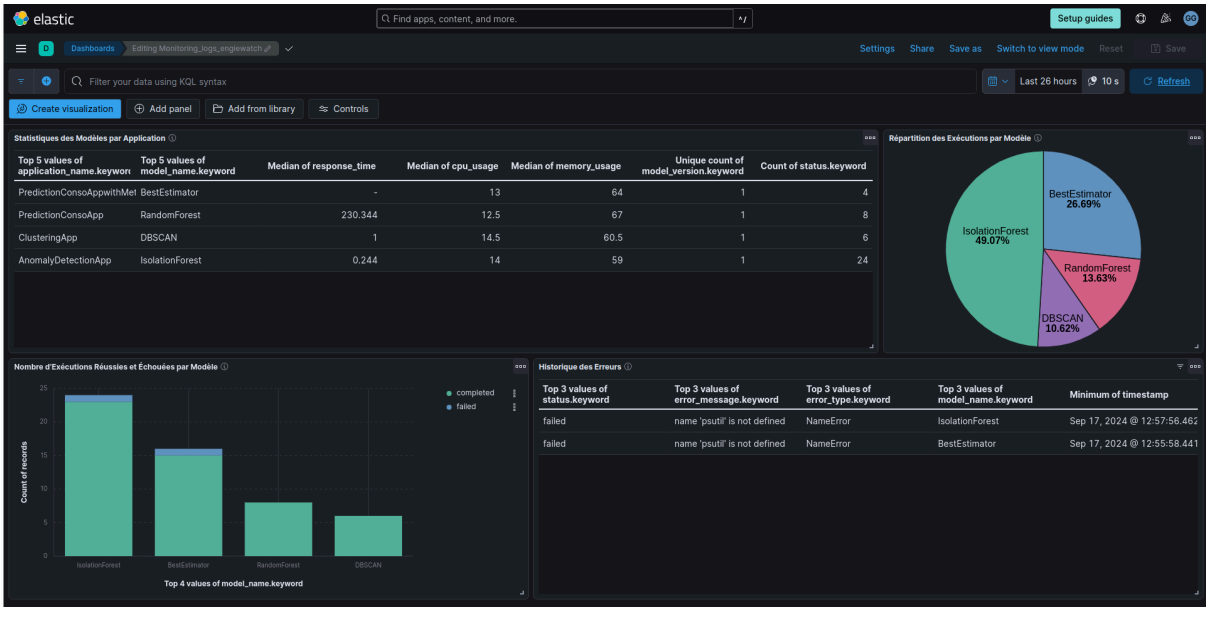
```
def show_anomalie_detection():
    start_time = time.time()
    log_data = {
        "timestamp": datetime.utcnow().isoformat(),
        "event": "model_execution",
        "model_name": "IsolationForest",
        "model_version": "1.0.0",
        "status": "started",
        "log_level": "INFO"
    }

    try:
        # Début de la détection d'anomalies
        data = df[hourly_columns]
        df["anomaly"] = loaded_model.predict(data)
        log_data.update({
            "status": "completed",
            "response_time": time.time() - start_time,
        })
    except Exception as e:
        log_data.update({
            "status": "failed",
            "error_message": str(e),
            "response_time": time.time() - start_time,
        })

    send_log_to_elastic(log_data)
```



5. Utilisation de Kibana pour la Visualisation des Logs

Après avoir envoyé les logs à Elasticsearch, nous pouvons configurer un tableau de bord dans **Kibana** pour visualiser ces logs.

1. **Création de l'Index dans Kibana** : Allez dans la section **Index Patterns** de Kibana et créez un modèle d'index pour vos logs (par exemple, `logs_votre_projet`).
2. **Création de Visualisations** : Utilisez l'outil de visualisation de Kibana pour créer des graphiques pertinents comme :
    - Histogramme des temps de réponse des modèles
    - Graphique des erreurs et des réussites
    - Usage CPU/Mémoire au fil du temps



6. Exemples de Visualisations dans Kibana

- **Histogramme des Temps de Réponse** : Montre la répartition des temps d'exécution des modèles.
- **Graphique des Anomalies Détectées** : Un graphique montrant le nombre d’anomalies détectées par région ou par jour.
- **Graphique des Utilisations CPU et Mémoire** : Pour surveiller l'efficacité du système pendant l'exécution des modèles.

7. Conclusion

L'intégration de **logs unifiés** et l'envoi vers **Elasticsearch** permettent de mieux suivre les performances des modèles en temps réel et d'identifier les goulots d'étranglement potentiels. Avec **Kibana**, il est possible de créer des visualisations puissantes pour les données de logs, offrant ainsi une meilleure vue d'ensemble du fonctionnement de l'application.