*Une image contenant texte, rouge, bus, transport

Description générée automatiquement*

**Cahier des Charges Projet MLOps London Fire Brigade**

|  |  |
| --- | --- |
| Github link | <https://github.com/Malakof/mlops_london_firebrigade> |
| Equipe | Richard De Falco, Vincent Dos Reis, Olfa Arfani, Mathieu Moreira |
| Dernière mise a jour | 09/10/2024 |
| Slack DTS | @juil24cmlops\_pompier |

**PROMOTION MLOPS 2024**

**ARFANI Olfa de falco richard DOS REIS VINCENT MOREIRA mATTHIEU**

**GROUPE JUL24** |**Encadrement : Sébastien - DataScientest**

**2024**

***Remerciements***

*Nous tenons a remercier l’équipe pédagogique DatascienceTest ainsi que notre encadrant Sebastien de nous avoir donne les elements essentiels et nécessaires pour aboutir a ce travail.*

***Abreviations***

***LFB : London Fire Brigade***

***AttendanceTime : Temps d’arrivee en secondes***

***MLOps : Machine learning Ops***

***MVP : Minimum Valuable Product***

# Table des matières

[Table des matières 2](#_Toc2745)

[Introduction 3](#_Toc2112)

[Contexte 3](#_Toc16154)

[Objectifs 4](#_Toc28205)

[Busines canvas 6](#_Toc66)

[Le modele 7](#_Toc23946)

[Objectifs de modèle 7](#_Toc30412)

[Description 7](#_Toc5287)

[Base de données 8](#_Toc18277)

[Introduction 8](#_Toc28335)

[Les données initiales 8](#_Toc19699)

[Les données additionnelles 9](#_Toc15795)

[La fusion des jeux de données 9](#_Toc28925)

[Application Programming Interface 9](#_Toc3565)

[Introduction 9](#_Toc31841)

[Authentification des utilisateurs 9](#_Toc25642)

[Endpoints de l'API 10](#_Toc19090)

[Testing & Monitoring 12](#_Toc22656)

[Introduction 12](#_Toc23543)

[Erreurs possibles 12](#_Toc14266)

[Succès minimum 13](#_Toc26666)

[Conclusion 14](#_Toc26913)

[Deploiement continu 14](#_Toc22297)

[Monitoring 16](#_Toc905)

[Deploiement continu 24](#_Toc97)

[Monitoring 26](#_Toc19039)

[Schéma d’implémentation 35](#_Toc26651)

[Conclusion 36](#_Toc20156)

# Introduction

## Contexte

### **Du point de vue technique :**

**Projet fil rouge** du cursus Data Analyst Orange promotion MLOPS 2024.

### **Du point de vue économique et social :**

La London Fire Brigade (abrégé LFB) est le corps de sapeurs-pompiers de Londres. C'est le **cinquième plus grand corps de sapeurs-pompiers dans le monde** avec environ 6000 employés dont 5000 sapeurs-pompiers professionnels. Comme l’ensemble des brigades dans le monde, la LFB recherche continuellement à s’améliorer afin de porter secours aux londoniens.

Une publication annuelle est produite afin de montrer en toute transparence leurs performances, mais aussi leurs défaillances.

Depuis le 1er Janvier 2023 le plan « Your London Fire Brigade » a vu le jour. Il vise à améliorer leur service en ajoutant une dimension de réflexion collaborative avec les citoyens de Londres.

L’un des points crucial, évoqué dans leur rapport, est le temps de réaction (Attendance Time). C’est le temps passé entre la sollicitation d’une équipe jusqu’au point de secours.

### **Du point de vue scientifique :**

Une image contenant carte, texte, atlas

Description générée automatiquementLa LFB, London Fire Brigade ou Brigade des Sapeurs-Pompiers de Londres, est le service d'incendie et de sauvetage le plus actif du Royaume-Uni et l'une des plus grandes organisations de lutte contre l'incendie et de sauvetage au monde.

Une image contenant carte, texte

Description générée automatiquementSon périmètre d’intervention s’étend sur le « Grand Londres » qui est actuellement d’une **superficie de** **1 572km² pour 9 millions d’habitants**.

*Figure 1 : Carte des boroughs du Grand Londres*

Le « Grand Londres » est divisé en 33 boroughs (circonscriptions administratives).

La **London Fire Brigade possède 102 casernes**.

Au travers de rapports annuels, la LFB met à disposition ses résultats et chiffres clefs. Par l’intermédiaires de ces derniers, elle présente des objectifs à atteindre comme celui du temp de réponse sur un incident.

*Figure 2 : Carte des casernes de la LFB*

Celui-ci est fixé à :

• 6 minutes pour le premier véhicule

• 8 minutes pour le second véhicule

***Sources : [www.ec.europa.eu](http://www.ec.europa.eu) & [www.nifc.gov](http://www.nifc.gov)***

## Objectifs

Une image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquementLa brigade de sapeurs-pompiers de Londres cherche à développer une application de prédiction du temps de réponse à la suite d’un appel pour une intervention. Cette variable s’appelle : **AttendanceTimeSecond**

Ainsi la variable cible « AttendanceTimeSeconds » est la composante de 2 autres variables du jeu de données « Mobilisations » qui sont :

- Le TurnoutTimeSeconds ou le temps de préparation des équipes d’intervention

- Le TravelTimeSeconds ou le temps de trajet jusqu’au lieu de l’incident

Cette application devra s'intégrer à leur infrastructure existante et fonctionner sur plusieurs supports, notamment une interface graphique accessible via un site web.

La solution proposée est une API pour une intégration plus large.

Le contexte d'intégration de l'application inclut le site web de la brigade, des logiciels internes ainsi que des serveurs cloud pour une meilleure gestion des données et des prévisions en temps réel.

Voici les différentes parties prenantes :

- **L'administrateur** de l'application sera désigné par la brigade et comprendra des responsables IT spécialisés dans la gestion des systèmes de gestion d'interventions d'urgence.

- **Les utilisateurs** de l'application seront les opérateurs du centre d'appels d'urgence, ainsi que les chefs de brigade sur le terrain, qui auront besoin de prévisions précises pour optimiser leurs opérations.

- **Le commanditaire** de l'application est la direction générale de la brigade de sapeurs-pompiers de Londres, qui cherche à améliorer l'efficacité et la rapidité des interventions en utilisant des technologies basées sur les données.

L'application doit répondre à la problématique de la prédiction du temps de réponse pour chaque intervention, en prenant en compte divers facteurs tels que l'emplacement de l'incident, le trafic, la disponibilité des équipes et des véhicules, ainsi que les conditions météorologiques.

## Busines canvas

Une image contenant texte, capture d’écran, Police, Impression

Description générée automatiquement

# Le modele

## Objectifs de modèle

* **Prédiction des Temps de Réponse :** Développer un modèle capable de prédire le temps de réponse des équipes d'intervention en fonction de divers facteurs tels que la distance à l'incident, le type d'incident, et la localisation des stations de pompiers.
* **Amélioration des Processus Décisionnels :** Fournir des informations exploitables pour aider à la planification des ressources et à l'optimisation des interventions.

## Description

Dans le contexte actuel due l’exercice, nous utilisons un model simple en raison du temps imparti pour le projet de fin de formation. A ce titre, le nombre d’expériences ainsi que le nombre de variable utilisé est limité.

|  |  |
| --- | --- |
| **Linear Regression**  - Mean Squared Error: 7080.8196  - R2 Score: 0.0414  - Mean Absolute Error: 69.6099  - Max Error: 255.5894 | **Decision Tree Regressor**  - Mean Squared Error: 7080.8197  - R2 Score: 0.0414  - Mean Absolute Error: 69.6099  - Max Error: 255.5894 |
| **Random Forest Regressor**  - Mean Squared Error: 21498.4674  - R2 Score: 0.0259  - Mean Absolute Error: 109.0240  - Max Error: 789.0247 | **K-NN Regressor**  - Mean Squared Error: 24576.7840  - R2 Score: -0.1135  - Mean Absolute Error: 116.6612  - Max Error: 821.4 |

A ce titre, le principe est de s’appuyer sur la distance à parcourir ainsi que la station de départ. Dans un second temps et dans l’hypothèse d’un véritable projet d’entreprise, le modèle pourrait dans un second temps, utiliser par exemple le jour ou bien l’heure de départ pour être plus performant.

Après le test de 4 modèles ( Linear Regression, Decision Tree Regressor, Random Forest Regressor, K-NN Regressor) nous avons retenu le modèle Linear Regression.

À la suite de l’analyse des données, il apparaît que la profondeur nécessaire pour l’entraînement est de 2 ans.

Le **temps d’entraînement** est très rapide avec **moins d’une minute** de traitement.

Le temps de prédiction est pratiquement instantané. Cependant les tests ont été effectués sur une station de travail classique. Il faudrait faire des simulations avec un plus grand nombre de sollicitations.

*Figure 3 : Répartition de l'erreur absolue pour la méthode de régression linéaire*

***Sources : [www.london-fire.gov.uk](http://www.london-fire.gov.uk) & [www.data.london.gov.uk](http://www.data.london.gov.uk)***

# Base de données

## Introduction

Une image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquement

## Les données initiales

Les données initiales se composent de 2 jeux. Elles sont disponibles en libre accès depuis le site London Datastore. Elles sont mises à jour tous les mois.

Pour notre projet nous avons fait le choix d’effectuer notre analyse avec une période comprenant des années complètes du 1er janvier 2009 au 31 Décembre 2023.

Les 2 jeux de données sont :

* **LFB Incident data** - Datastore - with notional cost and UPRN from January 2009

Il s’agit de l’ensemble des **données relatives aux signalisations des incidents** auprès des brigades de sapeurs-pompiers de Londres sur la période de janvier 2009 à décembre 2022 LFB

* **LFB Mobilisation** data from January 2009

Il s’agit de l’ensemble des **données de mobilisation par véhicule** sur la période de janvier 2009 à décembre 2022

Ces 2 jeux de données ont fait l’objet d’une analyse séparée dans un premier temps, puis après un « merge », d’autres analyses ont suivis.

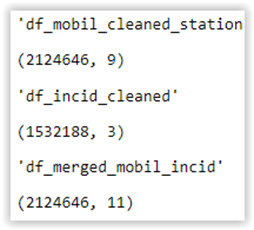
## Les données additionnelles

* **Les stations (Casernes)**

Il s’agit d’une table des coordonnées géographiques des stations de pompiers de Londres

Sources : Les coordonnées géographiques des stations ont été récupérées à l’aide de ce fichier : https://london-fire.labs.theodi.org/data/stations.csv

## La fusion des jeux de données

La jointure est de type 1 à n. C’est-à-dire que pour un incident, nous pouvons avoir 1 à plusieurs mobilisations de moyen.



# Application Programming Interface

## Introduction

L'API (Application Programming Interface) constitue un élément central du projet MLOps pour la London Fire Brigade (LFB). Elle sert d'interface entre le modèle prédictif, la base de données, et les utilisateurs finaux. Bien qu'il ne soit pas nécessaire d'intégrer une interface graphique dans le cadre de ce projet, l'API doit être conçue de manière à garantir une interaction fluide et sécurisée avec les différents composants du système.

## Authentification des utilisateurs

L'API doit inclure un mécanisme d'authentification robuste pour garantir que seuls les utilisateurs autorisés peuvent accéder aux fonctionnalités sensibles. Deux types d'utilisateurs seront définis :

* **Administrateurs :** Utilisateurs ayant des droits d'accès complets, capables de gérer les données, de former le modèle et d'accéder à des fonctionnalités avancées.
* **Utilisateurs Standards :** Utilisateurs ayant des droits d'accès limités, principalement pour effectuer des requêtes de prédiction et consulter les résultats.

L'authentification sera mise en œuvre via un système de base HTTP, utilisant des identifiants codés en dur pour les tests initiaux (par exemple, "admin": "fireforce", "user": "london123"). À long terme, il est recommandé d'intégrer un système d'authentification plus sécurisé, tel que OAuth2.

## Endpoints de l'API

L'API comportera **plusieurs endpoints**, chacun ayant une fonction spécifique. Voici les principaux endpoints que nous souhaitons intégrer :

**Process\_data :**

Méthode : GET

Description : Cet endpoint permet de traiter les données d'incidents et de mobilisation. Il peut être configuré pour traiter un type de données spécifique ou les deux.

Paramètres :

*Incident :* Boolean, par défaut false. Si true, traite les données d'incidents.

*Mobilisation :* Boolean, par défaut false. Si true, traite les données de mobilisation.

*Convert\_to\_pickle :* Boolean, par défaut false. Si true, convertit les données traitées en format pickle.

Appels à la Base de Données : Cet endpoint interagira avec la base de données pour récupérer les données brutes nécessaires au traitement.

Logs : Les actions de traitement et les erreurs éventuelles seront enregistrées dans un fichier de log dédié (data\_preprocessing.log).

**Build\_features :**

Méthode : GET

Description : Cet endpoint déclenche le processus de construction des caractéristiques à partir des données prétraitées.

Appels à la Base de Données : Il chargera les données prétraitées depuis la base de données, construira les caractéristiques nécessaires et les stockera dans la base de données.

Logs : Les étapes de construction des caractéristiques seront enregistrées dans build\_features.log.

**Train\_model :**

Méthode : POST

Description : Cet endpoint permet de former le modèle de prédiction en utilisant les caractéristiques construites.

Request Body :

*data\_path* : Chemin vers le fichier CSV contenant les données d'entraînement.

*model\_path* : Chemin où le modèle entraîné sera sauvegardé.

*encoder\_path* : Chemin où l'encodeur sera sauvegardé.

Appels à la Base de Données : Il peut interagir avec la base de données pour récupérer les caractéristiques nécessaires à l'entraînement.

Logs : Les informations sur le processus d'entraînement et les erreurs seront enregistrées dans train\_model.log.

**Predict**

Méthode : GET

Description : Cet endpoint permet de faire des prédictions sur les temps de réponse en fonction des paramètres fournis.

Paramètres :

*Distance* : Float représentant la distance à l'incident en kilomètres.

*Station* : String représentant le nom de la station de pompiers.

Appels à la Base de Données : Cet endpoint peut charger le modèle pré-entraîné et l'encodeur depuis la base de données pour effectuer la prédiction.

Logs : Les prédictions effectuées et les éventuelles erreurs seront enregistrées dans predict\_model.log.

# Testing & Monitoring

## Introduction

Dans tout projet d'ingénierie logicielle, en particulier ceux intégrant des technologies avancées comme l'intelligence artificielle (IA) et l'apprentissage automatique (ML), il est crucial d'identifier les erreurs potentielles et de définir des critères de succès. Cette section explore les erreurs possibles qui pourraient survenir lors de la mise en œuvre du projet MLOps pour la London Fire Brigade (LFB) et établit les critères de succès minimum nécessaires pour garantir un lancement efficace et durable.

## Erreurs possibles

**L’Inexactitude des Prédictions :**

L'une des erreurs les plus préoccupantes dans un projet de prédiction est l'inexactitude des résultats fournis par le modèle. Plusieurs facteurs peuvent contribuer à cette inexactitude :

**- Qualité des données :** Si les données utilisées pour entraîner le modèle sont incomplètes, biaisées ou de mauvaise qualité, cela peut entraîner des prédictions erronées. Par exemple, des données historiques mal enregistrées ou des incidents non documentés peuvent fausser les résultats.

**- Modèle mal ajusté :** Un modèle mal configuré ou mal entraîné peut ne pas capturer les relations sous-jacentes dans les données. Cela peut résulter d'une sélection inappropriée des caractéristiques, d'un surajustement ou d'un sous-ajustement.

**- Changements dans les conditions :** Les conditions opérationnelles peuvent évoluer, rendant les données historiques moins pertinentes. Par exemple, des changements dans la densité de la circulation, l'urbanisation ou les politiques de la LFB peuvent affecter les temps de réponse.

**Les problèmes d'intégration**

L'intégration du modèle dans les systèmes existants de la LFB peut également poser des défis :

* **Compatibilité des systèmes :** Les systèmes de gestion des incidents et d'autres outils utilisés par la LFB peuvent ne pas être compatibles avec les nouvelles solutions basées sur l'IA. Cela peut nécessiter des ajustements techniques ou des mises à jour coûteuses.
* **Formation des utilisateurs :** Les équipes doivent être formées pour utiliser efficacement les nouveaux outils et comprendre les résultats fournis par le modèle. Un manque de formation peut entraîner une mauvaise utilisation des systèmes et une résistance au changement.

**Les problèmes de sécurité et de confidentialité**

L'utilisation de données sensibles, telles que les informations sur les incidents et les ressources humaines, soulève des préoccupations en matière de sécurité et de confidentialité :

* **Fuites de données :** Les systèmes automatisés peuvent être vulnérables aux cyberattaques, ce qui pourrait entraîner des fuites de données sensibles. Il est essentiel de mettre en place des mesures de sécurité robustes pour protéger ces informations.
* **Conformité aux réglementations :** La LFB doit s'assurer que l'utilisation des données respecte les réglementations en matière de protection des données, telles que le RGPD. Des violations peuvent entraîner des sanctions financières et nuire à la réputation de l'organisation.

## Succès minimum

Pour garantir le succès du projet MLOps, il est essentiel de définir des critères de succès minimum. Ces critères permettront d'évaluer l'efficacité du modèle et son impact sur les opérations de la LFB.

**La Précision du Modèle**est un des critères les plus importants est la précision du modèle. Pour que le projet soit considéré comme un succès, le modèle doit atteindre un niveau de précision d'au moins 75 % dans ses prédictions de temps de réponse. Cela signifie que les prédictions doivent être suffisamment fiables pour être utilisées dans la planification des interventions.

**L’Adoption par les Utilisateurs** est l'acceptation et l'adoption par les utilisateurs finaux sont également des indicateurs clés de succès. Pour que le projet soit considéré comme réussi :

* **Feedback Positif :** Les utilisateurs doivent fournir un retour positif sur l'utilité et la facilité d'utilisation du système. Des enquêtes et des sessions de feedback peuvent être mises en place pour évaluer cette acceptation.
* **Utilisation Active :** Un taux d'utilisation élevé du système par les équipes opérationnelles est un indicateur que le modèle répond à un besoin réel et est intégré dans les processus quotidiens.

**L’amélioration des Temps** **de Réponse** est un autre critère de succès est l'impact mesurable sur les temps de réponse aux incidents. Pour que le projet soit considéré comme un succès :

* **Réduction des Temps de Réponse :** Les données doivent montrer une réduction significative des temps de réponse aux incidents après l'implémentation du modèle. Cela peut être mesuré en comparant les temps de réponse avant et après l'utilisation du modèle.
* **Amélioration de la Satisfaction des Citoyens :** Des enquêtes auprès des citoyens peuvent être menées pour évaluer leur satisfaction concernant la rapidité et l'efficacité des interventions de la LFB.

## Conclusion

L'identification des erreurs possibles et la définition des critères de succès minimum sont des étapes essentielles pour garantir la réussite du projet MLOps de la London Fire Brigade. En anticipant les défis potentiels et en établissant des objectifs clairs, la LFB peut s'assurer que l'intégration de l'IA et du ML dans ses opérations améliore réellement l'efficacité et la sécurité des interventions, tout en répondant aux attentes des utilisateurs et des citoyens. Ce projet représente une avancée significative vers une gestion des interventions plus intelligente et réactive, contribuant ainsi à la sécurité publique à Londres.

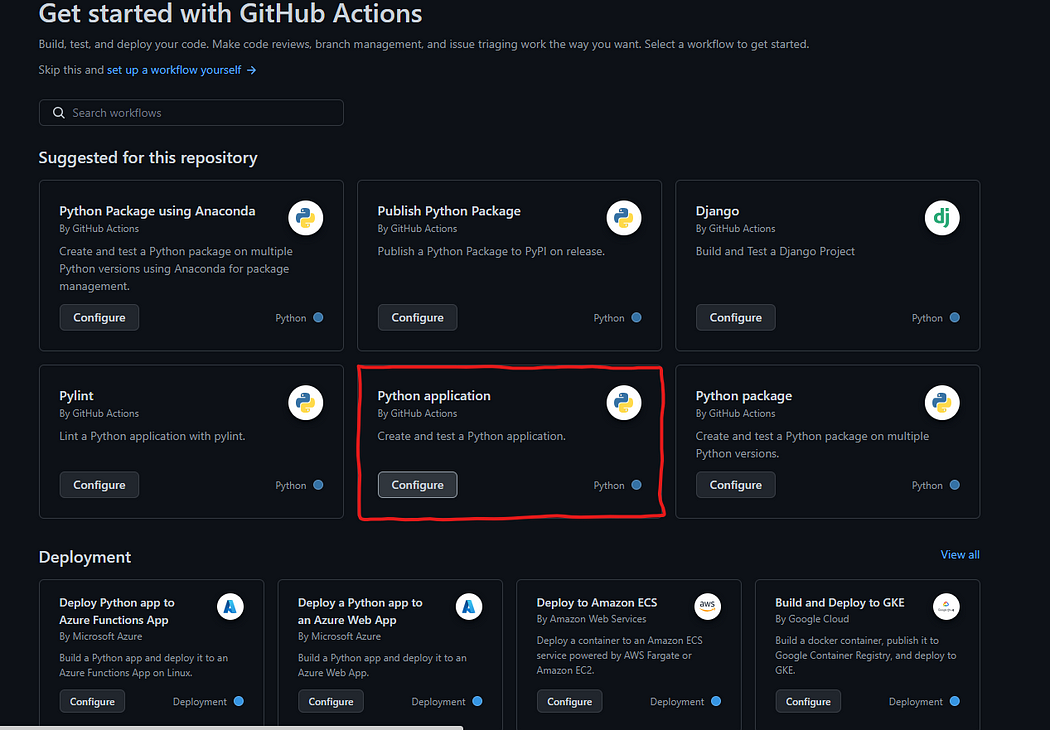
# Deploiement continu

1. Automatisation des tests

Les Tests unitaires et integration sont lances comme décrit dans le diagramme d’activite du projet précédemment après de l’etape de build.

Nous mettons en place une pipeline de Build via GithubAction qui par la suite execute les tests.

Nous choisirons d’abord une configuration de GithubAction pour un projet Python.



Puis par la suite écrire la pipeline YAML  
Comme vous pouvez le constater le projet a besoin de tourner sur une image type ubuntu.

# Monitoring

Nous avons choisi **docker-compose** pour avoir un service prometheus qui va ingérer les logs provenant des api.

Au niveau du compose.yml fichier nous avons besoin d’ajouter ces lignes

# This workflow will install Python dependencies, run tests and lint with a single version of Python

# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python

name: Python application

on:

push:

branches: [ "main" ]

pull\_request:

branches: [ "main" ]

permissions:

contents: read

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Set up Python 3.10

uses: actions/setup-python@v3

with:

python-version: "3.10"

- name: Install dependencies

run: |

if [ -f requirements.txt ]; then pip install -r requirements.txt; fi

- name: Lint with flake8

run: |

# stop the build if there are Python syntax errors or undefined names

flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics

# exit-zero treats all errors as warnings. The GitHub editor is 127 chars wide

flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics

- name: Test with unitest

run: |

python -m unittest discover -s tests

A group of logos on a white background

Description automatically generated

Pour exécuter un script Python, nous avons besoin de deux choses, à savoir un interpréteur, c'est-à-dire une version de Python, et un répertoire dans lequel sont stockées les fichiers sources des librairies utilisées. Utiliser un environnement virtuel consiste à utiliser un répertoire spécifique où sont stockées ces librairies. Ainsi L'environnement virtuel permet d'avoir plusieurs versions d'une même librairie installées sur une même machine et utilisées par différentes applications.

Nous pouvons aussi noter une chose assez intéressante: toutes les solutions proposant d'utiliser ces technologies de virtualisation proposent des systèmes d'images à reproduire (celles-ci seront détaillées dans la prochaine partie). Pour les fournisseurs de Cloud, c'est le catalogue de machines pré-paramétrées (AMIs pour AWS). Pour les containers, ce sont les images Docker pré-existantes ainsi que les DockerFiles. Dans le cas des environnements virtuels, on trouve les fichiers requirements.txt généralement utilisés.

Ainsi la virtualisation permet à la fois d'isoler des processus entre eux, de les isoler de la machine hôte mais aussi d'instancier facilement, rapidement des dispositifs virtuels identiques entre eux: on peut facilement reproduire un environnement virtuel, une machine virtuelle ou un container.

- Demarrer les service

**docker compose up**

- Ajouter Prometheus comme data source

Explications

1. Loki Service

* Image : nous utiliserons la dernière version de Grafana Loki (Grafana/loki: latest)
* Ports : Loki va etre accessible sur le port 3100 localement et mappe au port 3100 dans le container.
* Volumes : Le loki config log-config.yml est mappe au /etc/loki/loki-config.yaml dans le container
* Command : La commande s’assure que Loki utilise le fichier de conf fourni.

1. **Prometheus Service:**

* ****Image:** Nous utilisons Prometheus version 2.46.0** (prom/prometheus:v2.46.0).
* ****Ports:**** Prometheus est accessible pour le port 9090 localement et dans le container
* ****Volumes:**** Le config file prometheus.yml est mappe au /etc/prometheus/prometheus.yml dans le container.

1. **Grafana Service:**

* ****Image:**** We’re using the latest version of Grafana (grafana/grafana:latest).
* ****Ports:**** Grafana will be accessible on port 3000 both locally and inside the container.

This docker-compose.yml file sets up all three services with the necessary configurations. Once everything is set up, you can start the stack with a simple docker-compose up command.

****Step 2: Configuring Prometheus****

Now that we have our Docker Compose file set up, the next step is to configure Prometheus to scrape metrics from various targets, including itself, Loki, and any application you might be running. This configuration will be defined in the prometheus.yml file.

****Prometheus Configuration File (prometheus.yml)****

# Global configuration  
global:  
 scrape\_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.  
 evaluation\_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.  
 # scrape\_timeout is set to the global default (10s).  
  
# Scrape configurations  
scrape\_configs:  
 # Scrape Prometheus itself  
 - job\_name: "prometheus"  
 static\_configs:  
 - targets: ["localhost:9090"] # Assuming Prometheus is running on localhost:9090  
  
 # Scrape Loki metrics  
 - job\_name: "loki"  
 metrics\_path: "/metrics" # Loki's metrics endpoint  
 static\_configs:  
 - targets: ["loki:3100"] # Replace with your Loki's address and port  
  
 # Scrape application metrics  
 - job\_name: "application"  
 metrics\_path: "/api/metrics" # Your application metrics endpoint  
 static\_configs:  
 - targets: ["192.168.100.28:3000"] # Replace with your application's address and port

****Explanation:****

1. ****Global Configuration:****

* ****scrape\_interval:**** Il s’agit d’un intervalle qui permet de faire le scrapping des metrics Prometheus. Nous l’avons sette a 15s.
* ****evaluation\_interval:**** Prometheus va evaluer ces regles toutes les 15 s.

2. ****Scrape Configurations:****

* ****Prometheus:**** Ce job est configure pour scrapper les metrics de Prometheus meme pour tourner sur localhost:9090 dans le container.
* ****Loki:**** tourne sur loki:3100. Toujours dans le meme docker network
* **Applicatio**n:**** This job is configured to scrape metrics from your application.

****Important Notes:****

* ****Metrics Endpoint:**** We’re assuming your application exposes a metrics endpoint at /api/metrics. You can adjust this path if your application exposes metrics at a different endpoint.
* ****IP Address:**** Since Prometheus is running inside a Docker container, it cannot access services running on localhost of your host machine. Instead, you need to provide the actual IP address of the host machine (in this example, 192.168.100.28) where your application is running.

****Step 3: Configuring Loki with S3 Storage****

Now that we have Prometheus configured, the next step is to set up Loki for log aggregation. Loki is a horizontally-scalable, highly-available log aggregation system inspired by Prometheus. In this configuration, we’ll be using S3 as the storage backend for Loki to store logs.

****Loki Configuration File (loki-config.yaml) avec partage vers AWS S3****

auth\_enabled: false  
  
server:  
 http\_listen\_port: 3100  
 grpc\_listen\_port: 9095  
  
ingester:  
 wal:  
 dir: "/tmp/wal"  
 lifecycler:  
 address: 127.0.0.1  
 ring:  
 kvstore:  
 store: inmemory  
 replication\_factor: 1  
 chunk\_idle\_period: 5m  
 chunk\_retain\_period: 30s  
 chunk\_target\_size: 1536000  
 chunk\_block\_size: 262144  
  
schema\_config:  
 configs:  
 - from: 2023-01-01  
 store: tsdb  
 object\_store: s3  
 schema: v13  
 index:  
 prefix: index\_  
 period: 24h  
  
storage\_config:  
 tsdb\_shipper:  
 active\_index\_directory: /loki/index  
 cache\_location: /loki/index\_cache  
 cache\_ttl: 24h  
 aws:  
 s3: s3.eu-west-1.amazonaws.com  
 bucketnames: my-app-logs  
 region: eu-west-1  
 access\_key\_id: <ACCESS\_KEY\_ID>  
 secret\_access\_key: <SECRET\_ACCESS\_KEY>  
 s3forcepathstyle: false  
  
compactor:  
 working\_directory: /loki/compactor  
  
limits\_config:  
 reject\_old\_samples: true  
 reject\_old\_samples\_max\_age: 168h  
  
chunk\_store\_config:  
  
table\_manager:  
 retention\_deletes\_enabled: true  
 retention\_period: 168h  
  
ruler:  
 storage:  
 type: local  
 local:  
 directory: /loki/rules  
 rule\_path: /loki/rules  
 ring:  
 kvstore:  
 store: inmemory  
 enable\_api: true

****Explanation:****

1. ****auth\_enabled: false****

* Disables authentication for Loki, making it easier to set up and test locally. In a production environment, you would likely enable authentication for security reasons.

2. ****Server Configuration:****

* ****http\_listen\_port:**** Loki’s HTTP server will listen on port 3100.
* ****grpc\_listen\_port:**** Loki’s gRPC server will listen on port 9095.

3. ****Ingester Configuration:****

* ****wal:**** Configures the write-ahead log (WAL) directory.
* ****lifecycler:**** Configures the lifecycler for the ingester, including the in-memory key-value store (kvstore) and replication settings.
* ***chunk\_ settings:*** Configure the chunk storage parameters, including idle periods, retention periods, and sizes.

4. ****Schema Configuration:****

* ****schema\_config:**** schema de config de loki.

5.**Configuration de s**tockage :****

* ****tsdb\_shipper:**** configurer TSDB pour avoir un index active et caching.
* ****aws:**** Configuer AWS S3 comme un stockage backend.

6. ****Compactor Configuration:****

* ****working\_directory:**** verifie le working repertoire.

7. ****Limits Configuration:****

* ****reject\_old\_samples****.
* ****reject\_old\_samples\_max\_age:****  (168 heures ou 7 jours).

8. ****Table Manager:****

* ****retention\_deletes\_enabled****
* ****retention\_period****

9. ****Ruler Configuration:****

* ****storage****
* ****rule\_path****
* ****enable\_api****

****Step 4: Demarrer la Stack****

*docker-compose up*

****Acceder aux Services****

* ****Prometheus:**** Access Prometheus at http://localhost:9090.
* ****Grafana:**** Access Grafana at http://localhost:3000.
* ****Loki:**** Loki will be running and accessible at http://localhost:3001, but you’ll primarily interact with it through Grafana.

****Adding Data Sources in Grafana****

1. ****Log in to Grafana**** with your credentials.

2. ****Add Prometheus as a Data Source:****

* Navigate to ****Configuration > Data Sources > Add data source**** and select ****Prometheus****.
* ****Important:**** Since Prometheus is running inside a Docker container, you cannot use localhost:9090 as the address. Instead, use prometheus:9090 as the URL. This is because, within the Docker Compose network, services are reachable by their container names.

3. ****Add Loki as a Data Source:****

* Similarly, add Loki as a data source by selecting ****Loki**** from the list of available data sources.
* ****Important:**** Instead of using localhost:3100, use loki:3100 as the URL. This is the service name within the Docker Compose network.

****Creating/Importing Dashboards****

* ****Create Dashboards:**** You can create custom dashboards to visualize your metrics and logs. Grafana offers a wide range of widgets and options to visualize time-series data and logs.
* ****Import Dashboards:**** Grafana also allows you to import pre-built dashboards from the Grafana community. This can save time and provide valuable insights out of the box.

By correctly configuring the data source URLs using the service names (prometheus:9090 for Prometheus and loki:3100 for Loki), you ensure that Grafana can successfully communicate with these services within the Docker Compose network.

This small adjustment is crucial when working with services running inside Docker containers, as localhost will refer to the Grafana container itself, not the host machine or other containers.

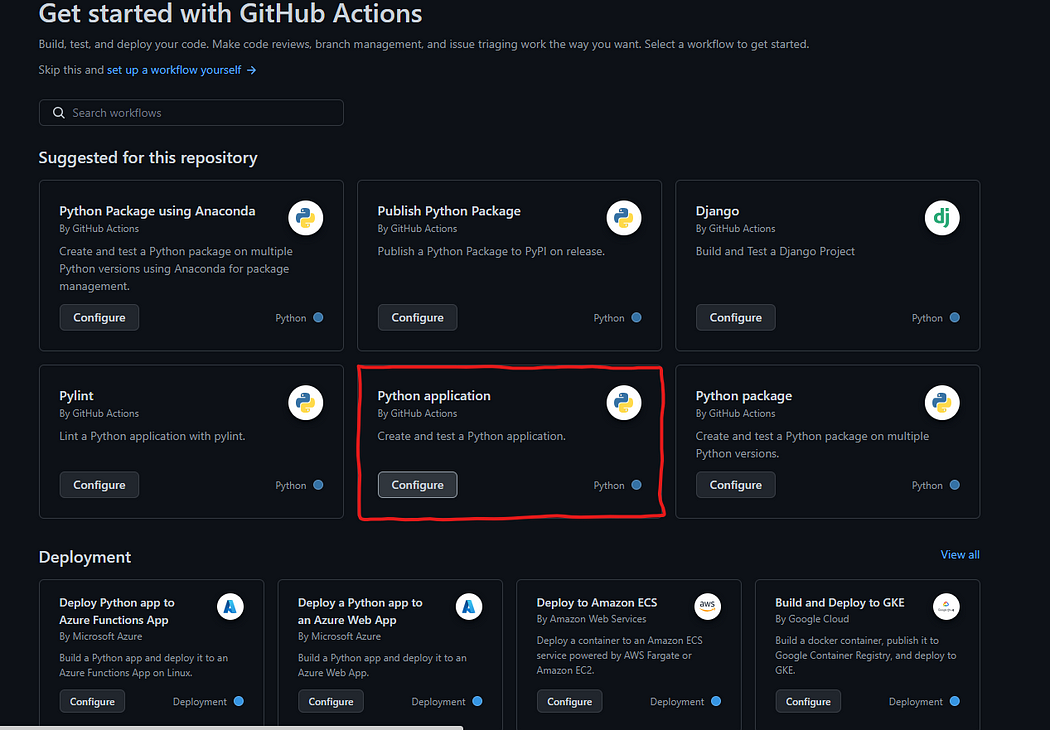
# Deploiement continu

1. Automatisation des tests

Les Tests unitaires et integration sont lances comme décrit dans le diagramme d’activite du projet précédemment après de l’etape de build.

Nous mettons en place une pipeline de Build via GithubAction qui par la suite execute les tests.

Nous choisirons d’abord une configuration de GithubAction pour un projet Python.



Puis par la suite écrire la pipeline YAML  
Comme vous pouvez le constater le projet a besoin de tourner sur une image type ubuntu.

# This workflow will install Python dependencies, run tests and lint with a single version of Python

# For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-python

name: Python application

on:

push:

branches: [ "main" ]

pull\_request:

branches: [ "main" ]

permissions:

contents: read

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Set up Python 3.10

uses: actions/setup-python@v3

with:

python-version: "3.10"

- name: Install dependencies

run: |

if [ -f requirements.txt ]; then pip install -r requirements.txt; fi

- name: Lint with flake8

run: |

# stop the build if there are Python syntax errors or undefined names

flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics

# exit-zero treats all errors as warnings. The GitHub editor is 127 chars wide

flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics

- name: Test with unitest

run: |

python -m unittest discover -s tests

# Monitoring

Nous avons choisi **docker-compose** pour avoir un service prometheus qui va ingérer les logs provenant des api.

Au niveau du compose.yml fichier nous avons besoin d’ajouter ces lignes

A group of logos on a white background

Description automatically generated

Pour exécuter un script Python, nous avons besoin de deux choses, à savoir un interpréteur, c'est-à-dire une version de Python, et un répertoire dans lequel sont stockées les fichiers sources des librairies utilisées. Utiliser un environnement virtuel consiste à utiliser un répertoire spécifique où sont stockées ces librairies. Ainsi L'environnement virtuel permet d'avoir plusieurs versions d'une même librairie installées sur une même machine et utilisées par différentes applications.

Nous pouvons aussi noter une chose assez intéressante: toutes les solutions proposant d'utiliser ces technologies de virtualisation proposent des systèmes d'images à reproduire (celles-ci seront détaillées dans la prochaine partie). Pour les fournisseurs de Cloud, c'est le catalogue de machines pré-paramétrées (AMIs pour AWS). Pour les containers, ce sont les images Docker pré-existantes ainsi que les DockerFiles. Dans le cas des environnements virtuels, on trouve les fichiers requirements.txt généralement utilisés.

Ainsi la virtualisation permet à la fois d'isoler des processus entre eux, de les isoler de la machine hôte mais aussi d'instancier facilement, rapidement des dispositifs virtuels identiques entre eux: on peut facilement reproduire un environnement virtuel, une machine virtuelle ou un container.

- Demarrer les service

**docker compose up**

- Ajouter Prometheus comme data source

Explications

1. Loki Service

* Image : nous utiliserons la dernière version de Grafana Loki (Grafana/loki: latest)
* Ports : Loki va etre accessible sur le port 3100 localement et mappe au port 3100 dans le container.
* Volumes : Le loki config log-config.yml est mappe au /etc/loki/loki-config.yaml dans le container
* Command : La commande s’assure que Loki utilise le fichier de conf fourni.

1. **Prometheus Service:**

* ****Image:** Nous utilisons Prometheus version 2.46.0** (prom/prometheus:v2.46.0).
* ****Ports:**** Prometheus est accessible pour le port 9090 localement et dans le container
* ****Volumes:**** Le config file prometheus.yml est mappe au /etc/prometheus/prometheus.yml dans le container.

1. **Grafana Service:**

* ****Image:**** Nous utilisons la derniere version de Grafana (grafana/grafana:latest).
* ****Ports:**** Grafana va etre accessible sur le port 3000.

Ce docker-compose.yml file sette tous les services avec les configurations nécessaires. Une fois que tout est setter, nous pouvons démarrer les service avec compose command.

****Step 2: Configurer Prometheus****

Maintenant que docker compose fichier est fait, pour la prochaine etape nous pouvons configurer le scrapping des metrics vers des targets differentes incluant loki et autres.

****Prometheus Configuration File (prometheus.yml)****

# Global configuration  
global:  
 scrape\_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.  
 evaluation\_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.  
 # scrape\_timeout is set to the global default (10s).  
  
# Scrape configurations  
scrape\_configs:  
 # Scrape Prometheus itself  
 - job\_name: "prometheus"  
 static\_configs:  
 - targets: ["localhost:9090"] # Assuming Prometheus is running on localhost:9090  
  
 # Scrape Loki metrics  
 - job\_name: "loki"  
 metrics\_path: "/metrics" # Loki's metrics endpoint  
 static\_configs:  
 - targets: ["loki:3100"] # Replace with your Loki's address and port  
  
 # Scrape application metrics  
 - job\_name: "application"  
 metrics\_path: "/api/metrics" # Your application metrics endpoint  
 static\_configs:  
 - targets: ["192.168.100.28:3000"] # Replace with your application's address and port

****Explication:****

1. ****Configuration globale:****

* ****scrape\_interval:**** Il s’agit d’un intervalle qui permet de faire le scrapping des metrics Prometheus. Nous l’avons sette a 15s.
* ****evaluation\_interval:**** Prometheus va évaluer ces regles toutes les 15 s.

2. ****Scrape Configurations:****

* ****Prometheus:**** Ce job est configure pour scrapper les metrics de Prometheus meme pour tourner sur localhost:9090 dans le container.
* ****Loki:**** tourne sur loki:3100. Toujours dans le meme docker network
* **Applicatio**n:**** Ce job va configurer le scrapping des metrics provenant de l’application.

****Notes Importantes :****

* ****Metrics Endpoint:**** Nous supposons que l’application expose une metric endpoint /api/metrics. Vous pouvez ajuster ce path vers different endpoint.
* ****IP Address:**** Depuis que Prometheus tourne sur un docker container, il ne peut plus acceder aux services tournant sur du localhost. Par contre nous avons besoin de fournir l’IP courante de la machine Host exemple 192.168.100.28

****Step 3: Configurer Loki vers le stockage de AWS S3****

Loki est horizontalement scalable. Afin d’avoir un stockage durable dans le temps, nous choisissons de partager les logs vers AWS S3, en ajoutant bien évidemment la sécurité et chiffrement symetric niveau bucket.

****Loki Configuration File (loki-config.yaml) avec partage vers AWS S3****

auth\_enabled: false  
  
server:  
 http\_listen\_port: 3100  
 grpc\_listen\_port: 9095  
  
ingester:  
 wal:  
 dir: "/tmp/wal"  
 lifecycler:  
 address: 127.0.0.1  
 ring:  
 kvstore:  
 store: inmemory  
 replication\_factor: 1  
 chunk\_idle\_period: 5m  
 chunk\_retain\_period: 30s  
 chunk\_target\_size: 1536000  
 chunk\_block\_size: 262144  
  
schema\_config:  
 configs:  
 - from: 2023-01-01  
 store: tsdb  
 object\_store: s3  
 schema: v13  
 index:  
 prefix: index\_  
 period: 24h  
  
storage\_config:  
 tsdb\_shipper:  
 active\_index\_directory: /loki/index  
 cache\_location: /loki/index\_cache  
 cache\_ttl: 24h  
 aws:  
 s3: s3.eu-west-1.amazonaws.com  
 bucketnames: my-app-logs  
 region: eu-west-1  
 access\_key\_id: <ACCESS\_KEY\_ID>  
 secret\_access\_key: <SECRET\_ACCESS\_KEY>  
 s3forcepathstyle: false  
  
compactor:  
 working\_directory: /loki/compactor  
  
limits\_config:  
 reject\_old\_samples: true  
 reject\_old\_samples\_max\_age: 168h  
  
chunk\_store\_config:  
  
table\_manager:  
 retention\_deletes\_enabled: true  
 retention\_period: 168h  
  
ruler:  
 storage:  
 type: local  
 local:  
 directory: /loki/rules  
 rule\_path: /loki/rules  
 ring:  
 kvstore:  
 store: inmemory  
 enable\_api: true

****Explication :****

****auth\_enabled: false****

* Ceci pour désactiver l’authentification dans l’environnement de dev. Il peut etre activé dans un environnement de prod pour ajouter la sécurité.

****Server Configuration:****

* ****http\_listen\_port:**** loki ecoute sur 3100.
* ****grpc\_listen\_port:****  loki gRPC ecoute sur le port 9095.

3. ****Ingester Configuration:****

* ****wal:**** Configure le write-ahead log (WAL) repertoire.
* ****lifecycler:**** Configure the lifecycler for the ingester, inclus le in-memory key-value store (kvstore) et les settings de replication.
* ***chunk\_ settings:*** Configure the chunk parameters de stockage, incluant idle periods, retention periods, et sizes.

4. ****Schema Configuration:****

* ****schema\_config:**** specifie le schema de configuration pour Loki. Il definit le stockage backend (tsdb), object storage (s3), schema version (v13), and index settings. The schema is applied starting from January 1, 2023.

5. ****Configuration de stockage:****

* ****tsdb\_shipper:**** Configure le TSDB shipper to manage active index repertoires et caching.
* ****aws:**** Configure AWS S3 comme un stockage backend pour Loki. Vous avez besoin de fournir access\_key\_id, secret\_access\_key, et d’autres details pertinentes.

6. ****Configuration comparateur :****

* ****working\_directory:**** Specifie the working directory pour le compactor, responsible pour compacting et le retain des logs.

7. ****Configuration limits:****

* ****reject\_old\_samples:**** active le rejet des anciens exemples.
* ****reject\_old\_samples\_max\_age:**** Set l’age maximum pour les samples - 168h ou 17 jours.

8. ****Table Manager:****

* ****retention\_deletes\_enabled:**** Active la suppression des anciennes data sur la base de la retention.
* ****retention\_period:**** Set la retention period des logs a 168 h (7 jours).

9. ****Ruler Configuration:****

* ****storage:**** Configure local stockage pour rules.
* ****rule\_path:**** Specifies ou le directory rules sont stockees.
* ****enable\_api:**** active API pour gerer les rules.

Cette config sette loki pour ingerer les logs stocke sur S3.  
Elle gere la retention de logs efficacement. Elle configure aussi loki pour exposer les metrics que Prometheus peut scrapper. Permettant ainsi de monitorer le health et la supervision.

****Step 4: Demarrer la Stack****

Avec toutes les configurations mises en place nous pouvons deja avoir le flow complet de monitoring, logging.

****Demarrer les services****

*docker-compose up*

Cette command va pull les docker images nécessaires, créer les containers et démarrer les services sur la base des config files décrits précédemment.

****Acceder aux Services****

* ****Prometheus:****  http://localhost:9090.
* ****Grafana:****  http://localhost:3000.
* ****Loki:****  http://localhost:3001

****Ajouter Data Sources dans Grafana****

1. ****Log a Grafana avec les credentials****.

2. Ajouter ****Prometheus as a Data Source:****

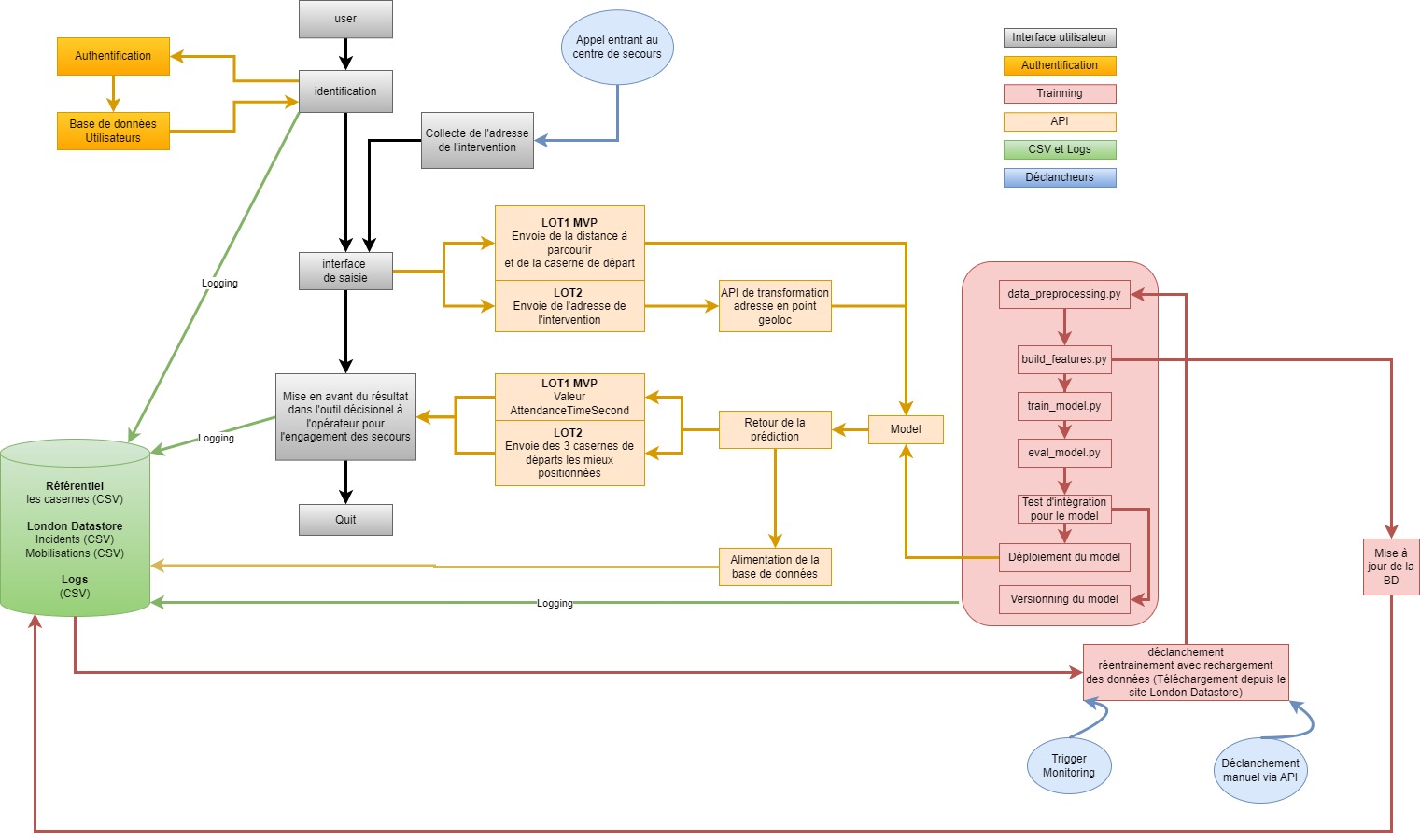
* ****Configuration > Data Sources > Ajout data source**** et ****Prometheus****.
* ****Important:****

Comme Prometheus tourne sur un docker pour y acceder, il faudrait mettre prometheus:9090 a la place de localhost:9090

3. ****Ajouter Loki comme Data source :****

Similairement ajouter loki comme source

# Schéma d’implémentation



# Conclusion

Ce cahier des charges définit les objectifs, le contexte, et les processus clés pour le projet MLOps de la London Fire Brigade. En intégrant des solutions basées sur l'IA et le ML, ce projet vise à améliorer l'efficacité opérationnelle et à optimiser les temps de réponse aux incidents, contribuant ainsi à la sécurité publique à Londres.