	Ministère de l'Enseignement Supérieur et de la Recherche Scientifique Université de Carthage **Institut National des Sciences Appliquées et de Technologie**	Code : DO-PFE-01 Indice de révision : 00 Edition : 08/2022

Rapport de Stage Obligatoire d'été

Filière :Réseaux Informatiques et Télécommunications

Niveau : 3ème Année

Sujet :

Industrialisation des modèles de Machine Learning

Réalisé par : **Mariem Makni**

Entreprise d'accueil :

Add Value by Innovation

Responsable à l'entreprise:	*Avis de la commission des stages*
Mme. Nivine Attoue et M. Baha Rahmouni	

Année Universitaire : 2022-2023

	Ministère de l'Enseignement Supérieur et de la Recherche Scientifique Université de Carthage **Institut National des Sciences Appliquées et de Technologie**	Code : DO-PFE-01 Indice de révision : 00 Edition : 08/2022

Rapport de Stage Obligatoire d'été

Filière :Réseaux Informatiques et Télécommunications

Niveau : 3ème Année

Sujet :

Industrialisation des modèles de Machine Learning

Réalisé par : **Mariem Makni**

Entreprise d'accueil :

Add Value by Innovation

Responsable à l'entreprise: **M. Baha Rahmouni**	***Avis de la commission des stages***

Année Universitaire : 2022-20

1. Introduction:

Les entreprises d'assurance s'efforcent constamment d'améliorer l'interaction avec leurs clients pour favoriser la fidélisation et la satisfaction. Dans cette optique, de nombreuses compagnies d'assurance font désormais appel à **la numérisation et à l'automatisation des processus** pour identifier des opportunités d'interactions plus efficaces et rapides, permettant ainsi d'améliorer globalement l'expérience client.

Dans le domaine de l'assurance automobile, les coûts liés aux sinistres ont augmenté considérablement avec la modernisation des véhicules, ce qui met une pression supplémentaire sur les compagnies d'assurance pour trouver des solutions plus efficaces de gestion des réclamations auto.

Dans le cadre de mon stage au sein de l'équipe Data, j'ai eu l'opportunité de contribuer à l'implémentation de **MLOps**[1] (Machine Learning Operations). L'objectif principal était d'optimiser **le cycle de vie complet des modèles de machine learning,** de leur développement à leur déploiement et leur gestion en production. Ce cycle est presenté par **la figure I.1**

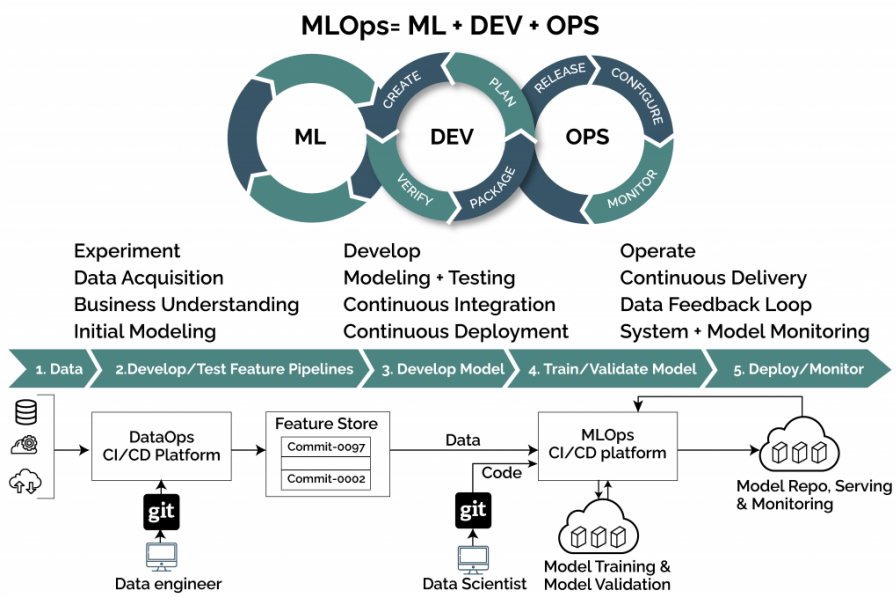

Figure Figure I.1 - Cycle de vie MLOps

Pour atteindre cet objectif, j'ai intégré les modèles de machine learning dans un environnement basé sur les technologies **Docker** et **Kubernetes**. Cette approche nous a permis de créer des conteneurs isolés pour les modèles, facilitant ainsi le déploiement et la gestion à grande échelle.

L'utilisation de **MLflow** a permis d'assurer un suivi et une gestion efficace. En parallèle, j'ai mis en place des pratiques de qualité de code à l'aide de **SonarQube** dès le début du cycle de développement.

Enfin, pour assurer une gestion efficace du code source et favoriser une collaboration transparente avec l'équipe, nous avons utilisé **GitLab** en tant qu'outil de contrôle de version et de gestion des workflows de développement.

Grâce à l'intégration réussie de ces technologies MLOps, notre équipe a gagné en **agilité**, en **qualité des modèles** et en **efficacité opérationnelle**. Ce rapport de stage présentera en détail les différentes étapes du projet ainsi que les résultats obtenus en termes de l'efficacité globale du processus.

2. Présentation de l'entreprise d'accueil:

2.1. Présentation du groupe AddInn

AddInn[2]- Add Value by **Inn**ovation, est un acteur majeur de la transformation digitale, présent sur le marché **euro-méditerranéen** et **africain**. Avec son siège à **Paris** et deux sites en **Tunisie.** Il entretient également des relations avec des clients au Congo, au Gabon et en Belgique, et prévoit l'ouverture prochaine d'un nouveau site en **Belgique.** Il accompagne les entreprises dans leurs projets de transformation en proposant une équipe de 100 consultants et ingénieurs hautement qualifiés. Son chiffre d'affaires a atteint **5 millions d'euros** en **2022**, avec une croissance annuelle moyenne de **25%** depuis **2017**. AddInn s'appuie sur ses différentes filiales pour répondre aux besoins de ses clients, comme illustré dans **la figure II.1.**

Figure II.1 - Les filiales du groupe AddInn

2.2. Secteur d'activité

AddInn propose une gamme complète de services pour répondre aux besoins variés des entreprises privées et publiques. Parmi ses principales missions, la société se consacre à la transformation digitale et l'assistance en matière de design fonctionnel et technique. Elle intervient également dans le domaine de l'assistance IT, offrant des services d'intégration de solutions logicielles, de conseil en architecture informatique et de data engineering, incluant l'architecture Big Data. De plus, AddInn propose des prestations en analyse de données et en data science.

2.3. Clients et références

Grâce à ses produits et la qualité de son travail, a gagné la confiance des grandes entreprises internationales. Son portefeuille client touche plusieurs secteurs d'activité tels que la télécommunication, le transport, les banques et les assurances illustré par **la figure II.2**

Figure II.2 - Les clients du groupe AddInn

2.4. Organisation de la société

ADDINN est organisé en structure matricielle, qui repose sur un découpage par fonction et par projet. Il existe plusieurs fonctions opérationnelles et d'autres techniques qui travaille ensemble en mode projet, où l'hiérarchie est quasi inexistante.

Les départements marketing et ventes, ressources humaines, et administratif sont des départements supports qui opèrent en étroite collaboration avec le directeur général pour fournir de l'assistance à la direction technique et assurer le bon déroulement de leur activité.

Les relations entre les collaborateurs au sein de la direction technique sont des relations horizontales. Ces départements travaillent en synchronisation pour réaliser des projets en commun.**La figure II.3** schématise l'organisation de la société

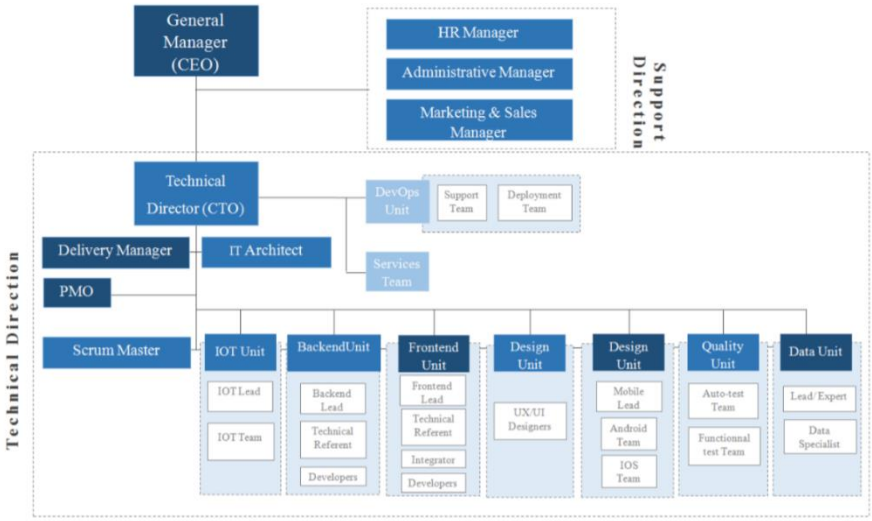

Figure II.3 - Organigramme AddInn Tunisie

3. Objectifs visés:

3.1. Taches à effectuer

• **Création des API (FastAPI) :** Cette tâche consiste à concevoir et développer les interfaces de programmation d'application (API) à l'aide du framework FastAPI.

• **Conteneurisation des API (docker):** Cette tâche implique de créer des conteneurs Docker pour les API développées.

• **Orchestration des conteneurs (Kubernetes) :** Cette tâche concerne l'utilisation de Kubernetes pour orchestrer les conteneurs.

• **CI/ CD Gitlab :**

- **Gestion des versions de code(git) :** Cette tâche concerne la mise en place d'un système de contrôle de version à l'aide de Git

- **Vérification qualité de code (Sonarqube) :** Cette tâche consiste à utiliser **SonarQube** pour analyser le code source, détecter les erreurs, les vulnérabilités et les problèmes de qualité.

- **Monitoring avec MLflow :** Cette tâche consiste à intégrer MLflow pour le monitoring des modèles de machine learning.

- CI/CD pipelines : Cette tâche implique la création de pipelines d'intégration continue et de déploiement continu (CI/CD) dans GitLab.

4. Journal

Figure III.1 - Diagramme de Gantt

5. Travail réalisé:

5.1. API creation

5.1.1. Comparaison Fastapi et Flask

Au cours du projet, nous avons opté pour FastAPI plutôt que Flask pour la création d'API en raison de ses nombreux avantages, notamment une syntaxe concise et moderne, une validation automatique des types de données des requêtes, et une gestion efficace des erreurs grâce à HTTPException.

5.1.2. Personnalisation de Swagger UI

Pour améliorer la fonctionnalité et la convivialité des API, j'ai personnalisé la documentation en utilisant l'interface utilisateur Swagge, présenté dans **la figure III.1** qui offre une interface interactive permettant aux utilisateurs de tester et d'explorer les fonctionnalités sans attendre la finalisation de l'interface utilisateur. L'intégration de Swagger UI à FastAPI et la personnalisation de la documentation permet une collaboration plus étroite avec nos clients. Les utilisateurs peuvent maintenant fournir des commentaires et des ajustements précoces, accélérant ainsi le processus de développement et améliorant la qualité globale de nos API en tant qu'ingénieurs en Machine Learning.

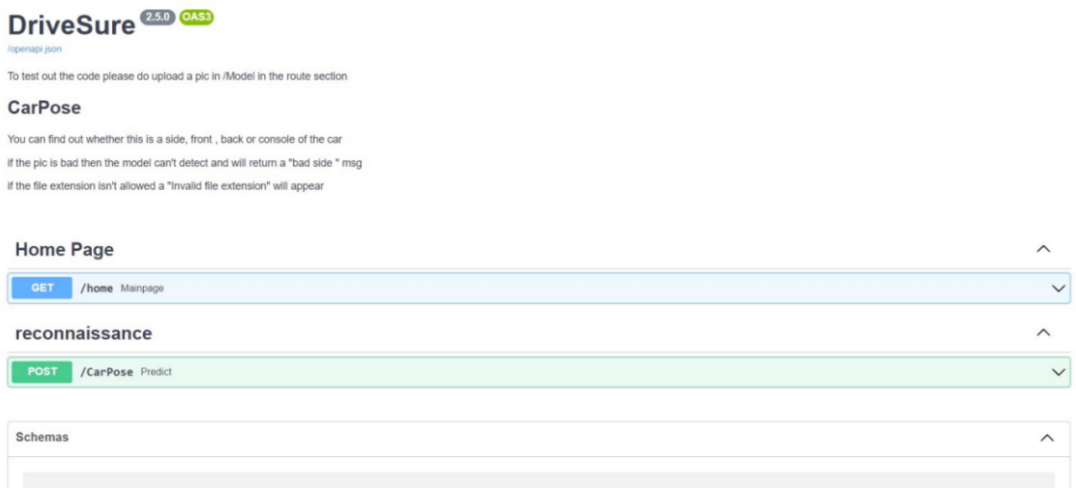

Figure IV.1 - Swagger User Interface

5.2.Conteneurisation des API

Nous avons utilisé une image de base **Debian** car elle offre une **stabilité** et une **compatibilité étendue**. Nous avons installé **pip**, exécuté **un environnement virtuel** et installé le module **pytesseract** qui est une interface pour **tesseract**. nous avons également installé **YOLOv5**. Après les avoir installés, nous avons ensuite installé le reste des **dépendances** requises.

En utilisant le module **pytesseract**, nous avons pu exploiter les fonctionnalités de **tesseract**, un **moteur de reconnaissance optique de caractères (OCR)** réputé. De plus, nous avons intégré **YOLOv5**, le modèle d'identification d'objet le plus rapide et précis du marché. Grâce à ces outils, nous avons réussi à effectuer des tâches d'extraction de texte à partir d'images, ce qui est essentiel pour la reconnaissance de plaques d'immatriculation et l'extraction de données de cartes d'identité. L'installation des dépendances supplémentaires requises a été réalisée à l'aide de **pipreqs**, qui est **un gestionnaire de paquets Python** largement utilisé.

En résumé, nous avons pu créer un environnement de développement solide et adapté à nos besoins spécifiques.

5.3. Orchestration des conteneurs

L'orchestration des conteneurs est devenue essentielle dans le déploiement d'applications modernes, offrant une gestion efficace des conteneurs pour assurer **disponibilité**, **évolutivité** et **fiabilité**. Dans notre projet, nous avons opté pour **Kubernetes**, une plateforme d'orchestration de conteneurs largement adoptée, pour le déploiement et la gestion de nos applications conteneurisées.

Kubernetes suit **une architecture client-serveur**, avec des composants clés tels que le **kube-controller-manager**, le **kube-scheduler** et le **kubelet**. En travaillant en harmonie, ces composants assurent une **gestion** et un **contrôle** optimaux des conteneurs.

La figure III.2 présente **le tableau de bord Minikube**. Minikube[3] est un outil open-source qui permet de déployer et de gérer des clusters Kubernetes localement, sur une machine individuelle. Le tableau de bord Minikube offr**e une interface graphique conviviale** qui permet aux utilisateurs de visualiser et de gérer les ressources et les objets du cluster Kubernetes.

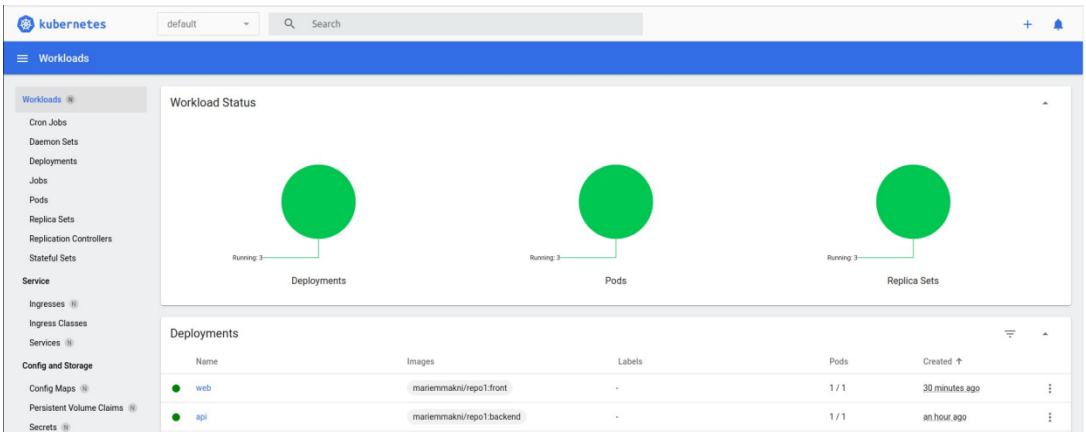

Figure IV.2 - le tableau de bord Minikube

5.3.1. Configuration des déploiements et des services

Pour déployer nos applications sur Kubernetes, nous avons utilisé des fichiers de **déploiement YAML**. Les déploiements décrivent **l'état souhaité de nos applications**, y compris le nombre de réplicas, les images à utiliser et les ports à exposer.

En plus des déploiements, nous avons configuré des **services** pour exposer nos applications aux autres composants de Kubernetes. Les services permettent d'**acheminer le trafic vers les pods associés aux déploiements**.

5.3.2. Configuration de l'Ingress

Dans notre projet, nous avons configuré l'Ingress dans Kubernetes, un mécanisme essentiel pour **gérer les règles de routage HTTP** vers nos services. L'Ingress agit comme une passerelle entre le trafic externe et nos services internes. Pour cela, nous avons créé un fichier app-ingress.yaml spécifiant les règles de routage pour nos applications, comme l'association du sous-domaine à

nos services API. Nous avons utilisé l'**Ingress NGINX Controller[4]** comme notre contrôleur Ingress, offrant une solution fiable pour l'exposition de nos applications. Les tests locaux nous ont assuré du bon fonctionnement de l'Ingress avant le déploiement en environnement de production.

5.3.3. Configuration des Volumes Persistants

Les volumes persistants jouent un rôle essentiel dans **le stockage fiable et durable des données** de nos applications conteneurisées. Dans notre projet, j'ai configuré des volumes persistants pour stocker les images passées à notre application **de manière sécurisée**. Pour mettre en place ces volumes persistants, on a utilisé un fichier de configuration YAML, définissant les caractéristiques du volume, telles que **sa capacité**, **son mode d'accès**, et **sa rétention des données**. Ensuite, on a créé un autre fichier de configuration pour réclamer l'utilisation de ce volume dans nos déploiements. Cela a permis de garantir que les données nécessaires restent accessibles, même en cas de redéploiement des conteneurs ou d'échec temporaire de l'un d'entre eux. Cette approche a grandement amélioré la fiabilité et la stabilité globale de notre environnement conteneurisé.

5.3.4. Erreurs rencontrés et solutions

Lors de la mise en place de Kubernetes, on a rencontré certains défis, notamment en travaillant initialement sur Windows avec **WSL[5]** (Windows Subsystem for Linux), ce qui a posé des problèmes avec les volumes persistants. On a constaté que WSL présentait des incompatibilités avec certaines fonctionnalités requises pour la configuration.

Pour résoudre ce problème, nous avons pris la décision de passer à une machine virtuelle **Ubuntu** dédiée, car elle offrait une meilleure compatibilité avec Kubernetes. De plus, on a initialement fait face à des problèmes pour monter les images du conteneur API sur nos machines locales. Cependant, on a compris que la réservation des ressources sur **le nœud Minikube** empêchait l'accès direct à ces images. Pour résoudre ce problème, on a créé un répertoire **/mnt/data** sur le nœud Kubernetes et avons modifié nos configurations pour y accéder correctement.

5.4. CI/ CD Gitlab

5.4.1. SonarQube

Pour assurer la qualité du code source, nous avons intégré **SonarQube** dans notre processus de développement. SonarQube est une **plateforme d'analyse de code** qui détecte les erreurs, les vulnérabilités et les problèmes de qualité dans le code. Dans notre environnement Kubernetes, nous avons déployé des pods **SonarQube** et **PostgreSQL** pour permettre les vérifications de qualité et l'analyse du code.

Nous avons donc accès au tableau de bord SonarQube présentée sur **la figure III.3**, qui offre une vue détaillée des résultats de l'analyse de code. Cette interface graphique conviviale nous permet de visualiser les principaux problèmes détectés dans le code source, tels que les bugs, les vulnérabilités de sécurité, les duplications de code, les mauvaises pratiques de programmation, ainsi que les mesures de la qualité du code.

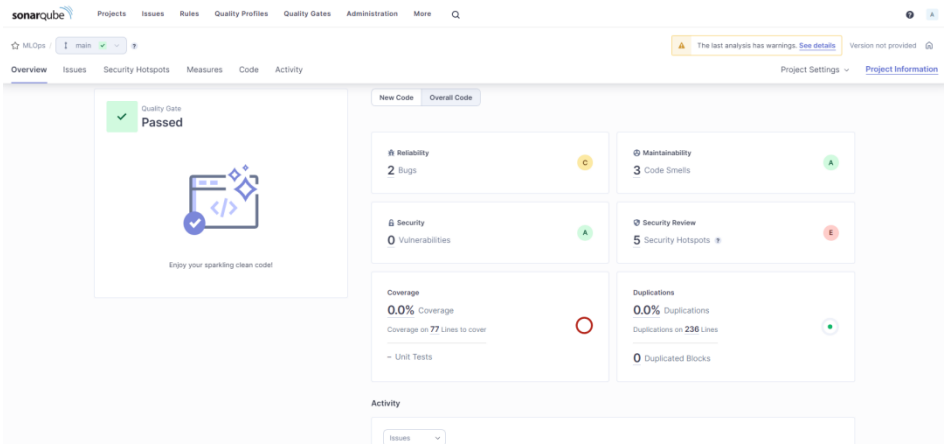

Figure IV.3 - Le tableau de bord SonarQube

5.4.2. MLFlow

MLFlow a été un élément crucial pour le monitoring de nos modèles de machine learning. C'est une **plateforme open-source** qui offre des fonctionnalités de **suivi** et de **gestion** des expérimentations, des versions de modèles et des métriques de performance pour les projets de machine learning. En intégrant MLFlow dans notre pipeline, on peut suivre facilement l'évolution des performances de nos modèles, comparer les différentes versions et optimiser les paramètres pour améliorer leurs résultats. **La figure III.4** met en évidence l'une de ces

expériences spécifiques, où nous avons configuré un processus d'extraction de matricule à partir d'images.

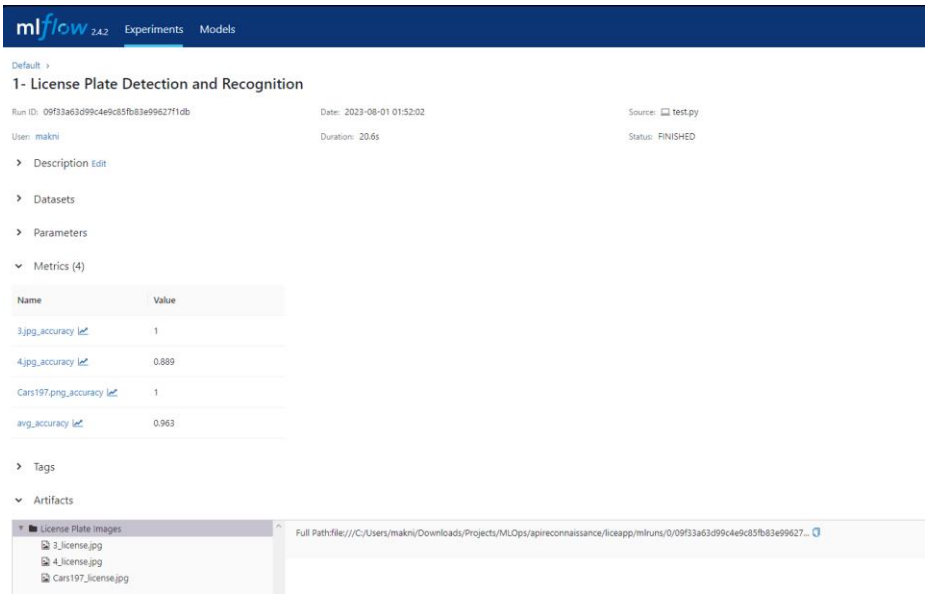

Figure IV.4 - Experience MLflow

5.4.3. GitLab

Nous utilisons GitLab comme système de contrôle de version pour notre projet. GitLab est une **plateforme centralisée** qui nous permet de travailler en collaboration sur nos projets, en offrant **la gestion du code source, les pipelines CI/CD**, et bien plus encore.

Notre pipeline GitLab est organisé en différentes étapes :

- Test

- Revue de code avec SonarQube

- Construction des images Docker et leur mise en ligne sur Docker Hub

- Déploiement avec Kubernetes

- Monitoring avec MLFlow

Cette approche nous a permis d'automatiser et de faciliter le processus de développement, de déploiement et de suivi de nos applications, assurant ainsi une meilleure qualité de code et une gestion efficace de nos modèles de machine learning.

6. Consolidation des acquis:

Au cours de ce projet MLOps, j'ai pu mettre en pratique les enseignements essentiels que j'ai acquis lors de mes cours d'**algèbre**, **statistiques**, **processus stochastiques** et **analyse de données**. Ces connaissances se sont révélées cruciales pour comprendre les modèles développés par mes collègues et ainsi intégrer avec succès les API associées. De plus, ma formation en **réseaux** m'a été d'une grande utilité pour appréhender l'architecture de Kubernetes[6] et configurer les services, les Ingress, et autres composants réseau nécessaires à notre projet. Ce cours m'a également introduit à Ansible, me permettant de développer une solide compréhension des **concepts d'automatisation**, qui s'est avérée inestimable dans la gestion et le déploiement efficaces des applications. Par ailleurs, travaillant sur une machine virtuelle **Ubuntu**, mes connaissances préalables en **UNIX** ont grandement facilité le processus, me permettant de naviguer sans heurts dans l'environnement de développement. De plus, la maîtrise de la programmation en **Python** et la manipulation de fichiers YAML, acquises au cours de ma formation, ont été des atouts clés pour le développement réussi du projet.

L'ensemble de ces compétences et enseignements a joué un rôle fondamental dans ma contribution réussie à ce projet, en me permettant de comprendre, de développer et de déployer des systèmes complexes, d'optimiser les processus et de mettre en œuvre l'automatisation pour une efficacité accrue.

7. Conclusion:

Au terme de mon stage au sein de l'équipe Data, j'ai eu l'opportunité de contribuer à la mise en place de **MLOps** au sein de notre projet d'assurance automobile. L'objectif principal de ce projet était de développer et d'optimiser les processus de développement, de déploiement et de gestion des modèles de machine learning utilisés par notre équipe.

Grâce à l'intégration réussie des technologies **Docker** et **Kubernetes**, nous avons pu créer un environnement robuste et scalable pour nos modèles de machine learning. La containerisation des modèles a facilité leur déploiement et leur gestion à grande échelle, tout en assurant l'isolation des environnements.

L'utilisation de **MLflow** nous a permis de suivre et de gérer efficacement nos expérimentations et versions de modèles. Cela a grandement facilité le processus d'évaluation et d'amélioration continue des modèles, en nous donnant une meilleure visibilité sur leurs performances.

La mise en place de pratiques de qualité de code dès le début du cycle de développement grâce à **SonarQube** a joué un rôle essentiel dans l'assurance de la fiabilité et de la maintenabilité de notre code. Cela nous a aidés à identifier et à corriger rapidement les éventuelles erreurs ou vulnérabilités.

Enfin, l'utilisation de **GitLab** en tant qu'outil de contrôle de version et de gestion des workflows de développement a favorisé une collaboration transparente au sein de l'équipe. Cela a permis d'optimiser notre efficacité opérationnelle et de garantir une gestion efficace du code source.

En conclusion, notre projet MLOps a été couronné de succès, renforçant ainsi notre capacité à déployer et à gérer efficacement les modèles de machine learning dans l'écosystème de l'assurance automobile. Pour des améliorations futures, nous envisageons d'explorer davantage les **Helm charts** pour une gestion plus efficace des déploiements et d'intégrer pleinement le concept de **microservices** pour une architecture encore plus flexible et évolutive.

Je tiens également à remercier toute l'équipe Data pour son soutien et sa collaboration tout au long de ce projet passionnant.

Bibliographies

[1] MLOps. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning?hl=fr

[2] Addinn website. https://addinn.com/

[3] Minikube. https://minikube.sigs.k8s.io/docs/start/

[4] Nginx Ingress Controller. https://kubernetes.github.io/ingress-nginx/

[5] Travailler sur les systèmes de fichiers Windows et Linux. https://learn.microsoft.com/en-us/windows/wsl/filesystems

[6] Présentation de l'architecture kubernetes. https://www.redhat.com/fr/topics/containers/kubernetes-architecture.