

PROJET DEVOPS & MLOPS



**Pipeline DevOps–MLOps pour un modèle
de Machine Learning**



Préparé par :

**DIONO BOUBACAR PEROU
SIE HANS OUATTARA**

Encadré par :

Pr. AZZEDINE KHIAT

Filière : Ingénierie Informatique et Intelligence Artificielle (5ème année)

Table des matières

1	Chapitre 1 : Contexte et architecture du projet	1
1.1	Introduction	1
1.2	Présentation du projet	1
1.3	Architecture du pipeline	1
1.4	Explication du pipeline DevOps	3
1.5	Explication du pipeline MLOps	3
1.6	Technologies utilisées	3
1.7	Conclusion	3
2	Chapitre 2 : Implémentation DevOps et MLOps	4
2.1	Introduction	4
2.2	Mise en place de l'API Flask et intégration du modèle	4
2.3	Conteneurisation du projet avec Docker	4
2.4	Versionnement et structuration du projet sur GitHub	4
2.5	Mise en place du pipeline CI/CD GitHub Actions	5
2.6	Déploiement sur AWS EC2	5
2.7	Création du Dashboard utilisateur	5
2.8	Diagramme de séquence du processus de prédiction	5
2.9	Conclusion	6
3	Chapitre 3 : Résultats, Tests et Validation du Pipeline	7
3.1	Introduction	7
3.2	MLOps côté API	7
3.3	Conteneurisation avec Docker	18
3.4	Gestion du code avec GitHub	20
3.5	Pipeline CI/CD avec GitHub Actions	23
3.6	Déploiement sur AWS EC2 et Docker	26
3.7	Dashboard final	33
4	Chapitre 4 : Conclusion générale et perspectives	35
4.1	Introduction	35
4.2	Synthèse des résultats	35
4.3	Perspectives et améliorations	35
4.4	Conclusion finale	35

Table des figures

1.1	Architecture MLOps intégrant API Flask, Dashboard, CI/CD, Docker et AWS EC2	2
2.1	Diagramme de séquence représentant le flux complet d'une prédiction.	5
3.1	Vérification des prérequis nécessaires à la mise en place de l'API.	7
3.2	Création de l'arborescence du projet MLOps.	8
3.3	Initialisation Git sur le projet.	8
3.4	Lancement du laboratoire AWS pour l'entraînement du modèle.	9
3.5	Recherche et accès au service Amazon SageMaker.	9
3.6	Accès à l'interface Notebook pour préparer l'environnement d'entraînement.	9
3.7	Création d'un nouveau Notebook dédié à l'entraînement du modèle.	10
3.8	Configuration et paramétrage du Notebook avant exécution.	10
3.9	Confirmation de la création réussie du Notebook dans SageMaker.	11
3.10	Organisation des dossiers dans JupyterLab pour une structure MLOps propre. .	11
3.11	Ouverture du Notebook contenant le futur code d'entraînement.	12
3.12	Insertion du code d'entraînement dans le Notebook SageMaker.	13
3.13	Résumé des résultats d'entraînement produit par le Notebook.	14
3.14	Vérification de la sauvegarde du modèle final (pkl + metadonnées).	14
3.15	Téléchargement local du modèle entraîné pour intégration dans l'API.	15
3.16	Création du fichier app.py pour l'API Flask.	15
3.17	Création du fichier modeloader.py pour le chargement du module.	16
3.18	Installation de Flask pour le projet.	16
3.19	Lancement de l'API Flask.	17
3.20	Affichage de la page HTTP une fois l'API lancée.	17
3.21	Test de prédiction avec curl sur l'API Flask.	17
3.22	Création du Dockerfile pour le projet.	18
3.23	Vérification des versions et remplissage du requirements.txt.	18
3.24	Lancement du build Docker pour créer l'image du projet.	19
3.25	Exécution du conteneur Docker et test de l'API.	19
3.26	Connexion à GitHub et création du repository du projet.	20
3.27	Confirmation de la création réussie du repository.	21
3.28	Push du projet local vers le repository GitHub.	21
3.29	Vérification de la réussite de l'envoi du projet sur GitHub.	22
3.30	Écriture du fichier YAML pour le pipeline CI/CD.	23
3.31	Exécution du pipeline CI/CD après commit et push.	24
3.32	Vérification de l'arborescence finale sur GitHub.	25
3.33	Accès à AWS EC2 pour la création d'une instance.	26
3.34	Interface de création de l'instance EC2.	26
3.35	Paramétrage des informations de l'instance EC2.	27

3.36 Confirmation de la création réussie de l’instance EC2.	28
3.37 Configuration du groupe de sécurité pour l’instance EC2.	28
3.38 Connexion SSH à l’instance EC2 pour déployer l’application.	29
3.39 Mise à jour de l’instance EC2 avec update et upgrade.	29
3.40 Installation de Docker sur EC2 pour exécuter l’application.	30
3.41 Configuration des droits Docker sur l’instance.	30
3.42 Clonage du projet depuis GitHub sur l’instance EC2.	30
3.43 Construction de l’image Docker pour l’API.	31
3.44 Exécution du conteneur Docker et vérification de l’API.	31
3.45 Test final de prédiction pour valider le pipeline complet.	31
3.46 Tests complémentaires pour valider la cohérence des prédictions.	32
3.47 Interface principale du dashboard après intégration.	33
3.48 Test de prédiction effectué avec succès depuis le dashboard.	33
3.49 Dashboard final amélioré avec visualisations et tests supplémentaires.	34

Chapitre 1

Contexte et architecture du projet

1.1 Introduction

Ce chapitre présente le contexte général du projet, son intérêt pédagogique ainsi que l'architecture globale mise en place pour le pipeline DevOps–MLOps. Le projet a pour objectif de fournir aux étudiants une approche concrète combinant l'automatisation du développement logiciel et l'intégration de modèles de Machine Learning.

1.2 Présentation du projet

Le projet consiste à construire un pipeline DevOps–MLOps complet comprenant les éléments suivants : - Une API Flask exposant un modèle de Machine Learning (Decision Tree, Random Forest, Logistic Regression) - Un modèle entraîné dans un Notebook AWS - Conteneurisation avec Docker - Déploiement sur une instance EC2 AWS - Pipeline CI/CD basique avec GitHub Actions

Chaque étape est documentée avec des captures d'écran et des explications détaillées.

1.3 Architecture du pipeline

Le diagramme ci-dessous illustre l'architecture globale du projet :

L'architecture comprend les composants suivants :

- Dashboard Web (HTML, CSS, Bootstrap)
- API Flask avec routes de prédiction
- Chargement du modèle ML via `model_loader.py`
- Modèle ML entraîné et sérialisé (`model.pkl`)
- Conteneur Docker pour déploiement
- Instance AWS EC2 pour hébergement

Architecture du Projet DevOps/MLOps

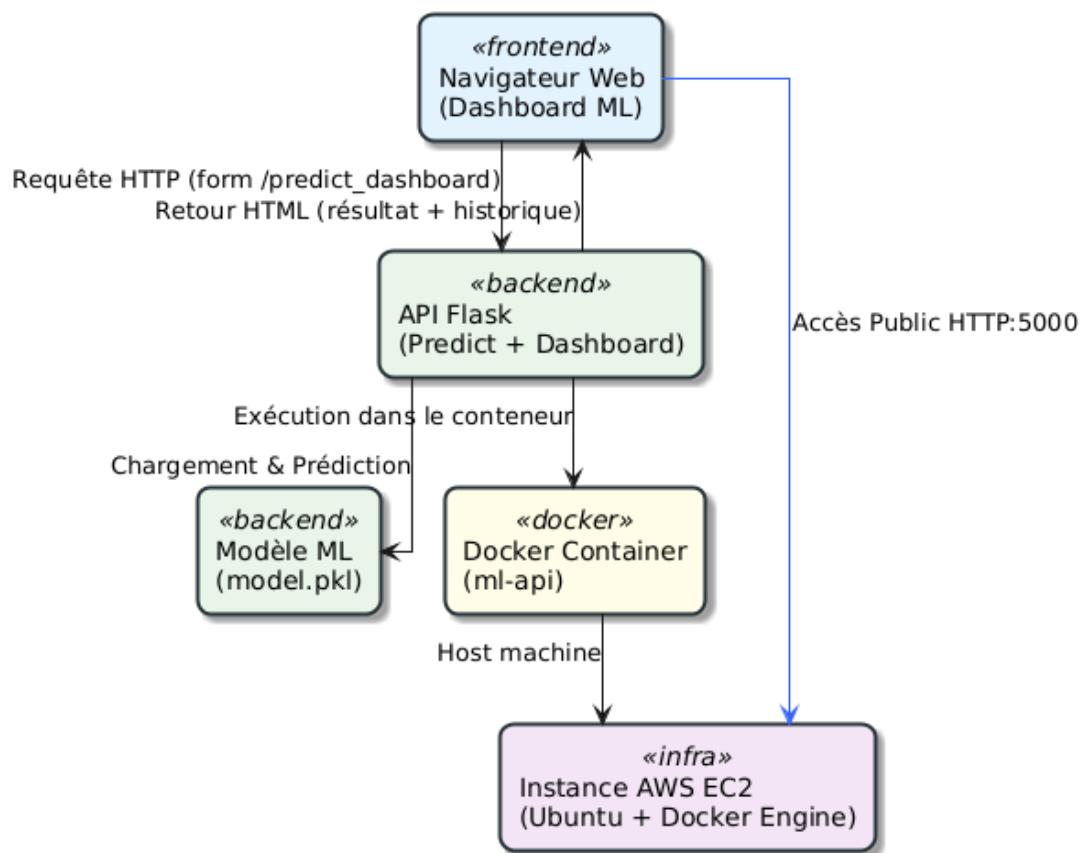


FIGURE 1.1 – Architecture MLOps intégrant API Flask, Dashboard, CI/CD, Docker et AWS EC2

1.4 Explication du pipeline DevOps

Le pipeline DevOps automatise la construction, le test et le déploiement de l'application. Il comprend les étapes suivantes :

1. Gestion du code source via GitHub
2. Intégration continue avec GitHub Actions pour construire et tester l'application
3. Conteneurisation avec Docker pour assurer la portabilité
4. Déploiement automatique sur l'instance AWS EC2

Ce pipeline assure une livraison rapide et fiable du projet tout en minimisant les erreurs humaines.

1.5 Explication du pipeline MLOps

Le pipeline MLOps se concentre sur l'automatisation des modèles de Machine Learning. Les étapes principales sont :

1. Préparation et nettoyage des données dans un Notebook AWS
2. Entraînement et validation du modèle ML (Decision Tree, Random Forest, etc.)
3. Srialisation du modèle avec pickle pour une réutilisation dans l'API
4. Intégration du modèle dans l'API Flask
5. Déploiement et tests des prédictions via Docker et EC2

Ce pipeline permet de reproduire et déployer rapidement des modèles ML fiables dans un environnement cloud.

1.6 Technologies utilisées

Le projet utilise plusieurs technologies clés :



1.7 Conclusion

Ce chapitre a présenté le contexte du projet ainsi que l'architecture globale, en détaillant les pipelines DevOps et MLOps. Il sert de base pour les chapitres suivants qui détaillent l'implémentation du pipeline, l'entraînement du modèle, le déploiement et l'évaluation finale.

Chapitre 2

Implémentation DevOps et MLOps

2.1 Introduction

Ce chapitre décrit la réalisation complète du système DevOps–MLOps mis en place pour ce projet. Le travail a débuté par la préparation de l’API Flask et du modèle d’apprentissage automatique, avant de poursuivre avec la conteneurisation Docker, l’intégration GitHub, la configuration du pipeline CI/CD et le déploiement final sur AWS EC2. Enfin, le tableau de bord utilisateur (Dashboard) a été conçu pour offrir une interface simple et intuitive permettant d’interagir avec le modèle.

2.2 Mise en place de l’API Flask et intégration du modèle

La première étape du projet consiste à développer une API légère basée sur Flask. Cette API charge un modèle pré-entraîné au format `model.pkl`, traite les données reçues au format JSON et renvoie une prédiction. Elle constitue l’élément central du pipeline MLOps, car elle encapsule l’intelligence artificielle dans un service web accessible et réutilisable.

2.3 Conteneurisation du projet avec Docker

Après la mise en place de l’API, l’étape suivante a été la conteneurisation avec Docker. L’objectif est de garantir un environnement cohérent et reproductible, indépendamment de la machine hôte. Le Dockerfile intègre Python, les dépendances nécessaires via `requirements.txt`, ainsi que l’API Flask elle-même. Cette conteneurisation constitue un pilier DevOps essentiel pour assurer la portabilité du projet.

2.4 Versionnement et structuration du projet sur GitHub

Le projet complet a ensuite été poussé sur GitHub. Le dépôt permet d’assurer le suivi des modifications, la collaboration ainsi que l’intégration du pipeline CI/CD. Une arborescence claire a été définie incluant les scripts Python, les fichiers Docker, les dépendances, le modèle, et le Dashboard.

2.5 Mise en place du pipeline CI/CD GitHub Actions

Un pipeline d'intégration continue a été élaboré via GitHub Actions. Celui-ci vérifie automatiquement la structure du projet, la présence du Dockerfile, et tente la construction de l'image. En cas de réussite, les modifications sont validées. Le pipeline permet d'éviter les erreurs, de renforcer la qualité du code et d'automatiser les opérations DevOps.

2.6 Déploiement sur AWS EC2

Une instance AWS EC2 Ubuntu a été configurée pour servir de serveur d'hébergement. Après installation de Docker et de Git, l'image Docker contenant l'API a été reconstruite directement sur l'instance. Le conteneur a ensuite été lancé et exposé au public, permettant d'interroger l'API depuis n'importe quel client ou script. Cette étape représente la partie "Ops" du pipeline DevOps.

2.7 Création du Dashboard utilisateur

Une interface Web a été développée afin de permettre à l'utilisateur d'interagir facilement avec l'API. Le Dashboard permet de saisir les quatre paramètres du modèle et d'afficher instantanément la prédition obtenue. Il relie l'expérience utilisateur à la logique MLOps en consommant directement l'API déployée sur le cloud.

2.8 Diagramme de séquence du processus de prédition

Le diagramme suivant illustre précisément le cheminement d'une prédition : interaction de l'utilisateur avec le Dashboard, envoi des données vers l'API Flask, traitement du modèle et retour du résultat.

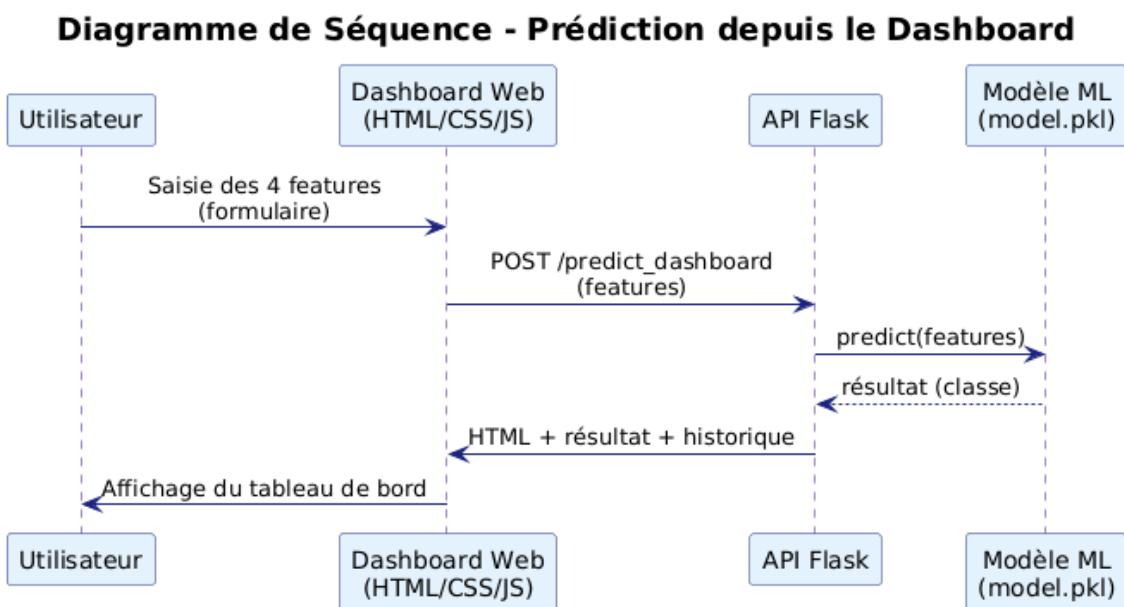


FIGURE 2.1 – Diagramme de séquence représentant le flux complet d'une prédition.

2.9 Conclusion

Ce chapitre a présenté l'ensemble du pipeline DevOps et MLOps, depuis la création de l'API jusqu'au déploiement cloud et à l'interface utilisateur. Chaque étape a été réalisée de manière progressive et cohérente pour garantir un système fiable, automatisé et facile à maintenir. Le chapitre suivant sera consacré aux résultats, aux captures d'écrans ainsi qu'à l'analyse globale du fonctionnement du système.

Chapitre 3

Résultats, Tests et Validation du Pipeline

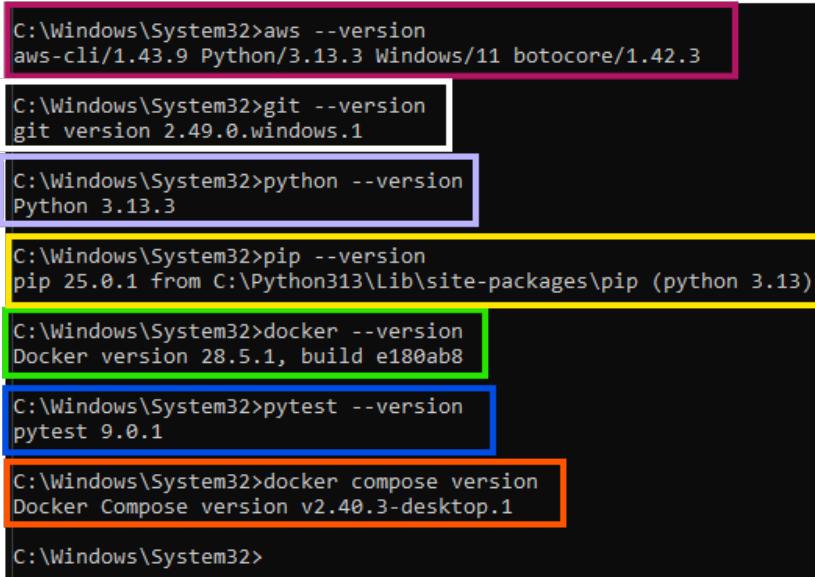
3.1 Introduction

Ce chapitre présente l'ensemble des résultats obtenus durant la réalisation du projet DevOps–MLOps. Chaque étape clé a été documentée à travers des captures issues du développement : création de l'API, exécution du modèle, conteneurisation, mise en place du tableau de bord, ainsi que les phases GitHub et GitHub Actions. L'objectif est de fournir une vision claire, visuelle et structurée du workflow complet.

3.2 MLOps côté API

Cette première section illustre la préparation du projet, la configuration des dossiers, la création du notebook d'entraînement et la génération du modèle machine learning.

Avant de commencer, nous avons vérifié que tous les prérequis nécessaires (Python, bibliothèques, environnement AWS) étaient disponibles et fonctionnels.



```
C:\Windows\System32>aws --version
aws-cli/1.43.9 Python/3.13.3 Windows/11 botocore/1.42.3
C:\Windows\System32>git --version
git version 2.49.0.windows.1
C:\Windows\System32>python --version
Python 3.13.3
C:\Windows\System32>pip --version
pip 25.0.1 from C:\Python313\Lib\site-packages\pip (python 3.13)
C:\Windows\System32>docker --version
Docker version 28.5.1, build e180ab8
C:\Windows\System32>pytest --version
pytest 9.0.1
C:\Windows\System32>docker compose version
Docker Compose version v2.40.3-desktop.1
C:\Windows\System32>
```

FIGURE 3.1 – Vérification des prérequis nécessaires à la mise en place de l'API.

Nous avons ensuite créé l'arborescence du projet afin de structurer clairement les fichiers du futur pipeline MLOps.

```
C:\Windows\System32>cd %USERPROFILE%\Desktop
C:\Users\DELL\Desktop>mkdir devops-mlops-aws-student-project
C:\Users\DELL\Desktop>cd devops-mlops-aws-student-project
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>mkdir notebooks model api docker tests docs
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>mkdir docs\screenshots
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>mkdir .github
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>mkdir .github\workflows

C:\Users\DELL\Desktop\devops-mlops-aws-student-project>dir
Le volume dans le lecteur C n'a pas de nom.
Le numéro de série du volume est 64A4-2F55

Répertoire de C:\Users\DELL\Desktop\devops-mlops-aws-student-project

05/12/2025 13:01    <DIR>          .
05/12/2025 12:59    <DIR>          ..
05/12/2025 13:01    <DIR>          .github
05/12/2025 13:01    <DIR>          api
05/12/2025 13:01    <DIR>          docker
05/12/2025 13:01    <DIR>          docs
05/12/2025 13:01    <DIR>          model
05/12/2025 13:01    <DIR>          notebooks
05/12/2025 13:01    <DIR>          tests
              0 fichier(s)          0 octets
              9 Rép(s)   64 310 730 752 octets libres

C:\Users\DELL\Desktop\devops-mlops-aws-student-project>
```

FIGURE 3.2 – Création de l'arborescence du projet MLOps.

Une initialisation Git a été réalisée afin de versionner correctement tout le projet et préparer la future intégration GitHub.

```
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>git init
Initialized empty Git repository in C:/Users/DELL/Desktop/devops-mlops-aws-student-project/.git/
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

C:\Users\DELL\Desktop\devops-mlops-aws-student-project>
```

FIGURE 3.3 – Initialisation Git sur le projet.

Nous avons ensuite lancé le laboratoire AWS pour accéder à SageMaker et réaliser l'entraînement du modèle directement dans le cloud.

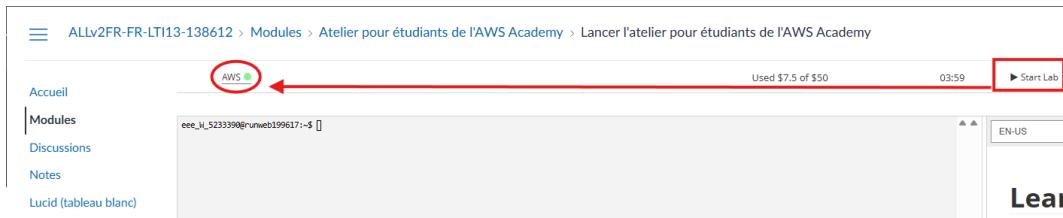


FIGURE 3.4 – Lancement du laboratoire AWS pour l'entraînement du modèle.

Après avoir ouvert l'environnement AWS, nous avons recherché Amazon SageMaker pour accéder à l'interface JupyterLab.

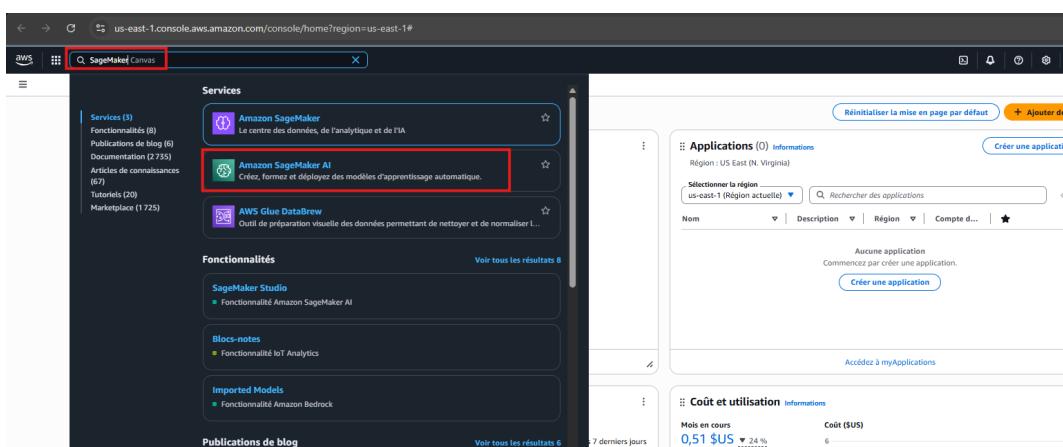


FIGURE 3.5 – Recherche et accès au service Amazon SageMaker.

Cette étape montre que nous avons accédé à l'interface Notebook pour préparer l'environnement d'entraînement.

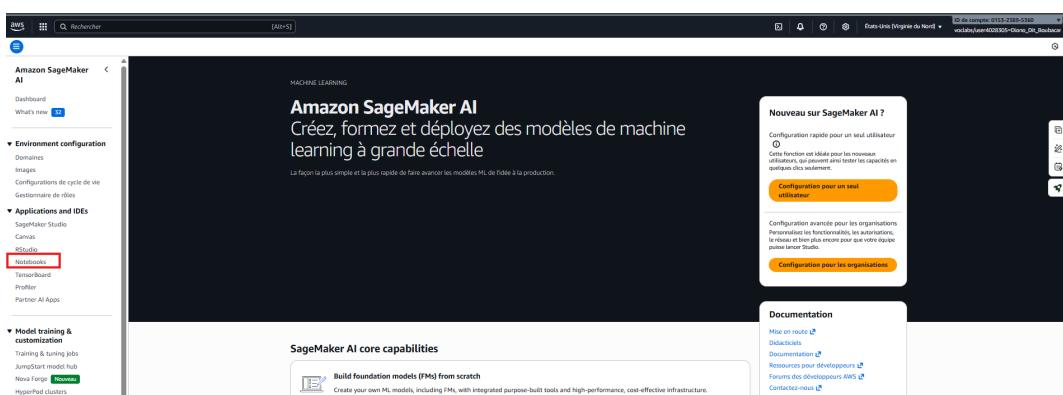


FIGURE 3.6 – Accès à l'interface Notebook pour préparer l'environnement d'entraînement.

Nous avons créé un nouveau Notebook dédié à l'entraînement du modèle afin d'isoler le code.

FIGURE 3.7 – Création d'un nouveau Notebook dédié à l'entraînement du modèle.

Dans cette étape, nous avons configuré et rempli le Notebook avec les bibliothèques nécessaires et les paramètres d'exécution.

FIGURE 3.8 – Configuration et paramétrage du Notebook avant exécution.

Ici, nous voyons que le Notebook a été créé avec succès et est prêt pour l'entraînement du modèle.

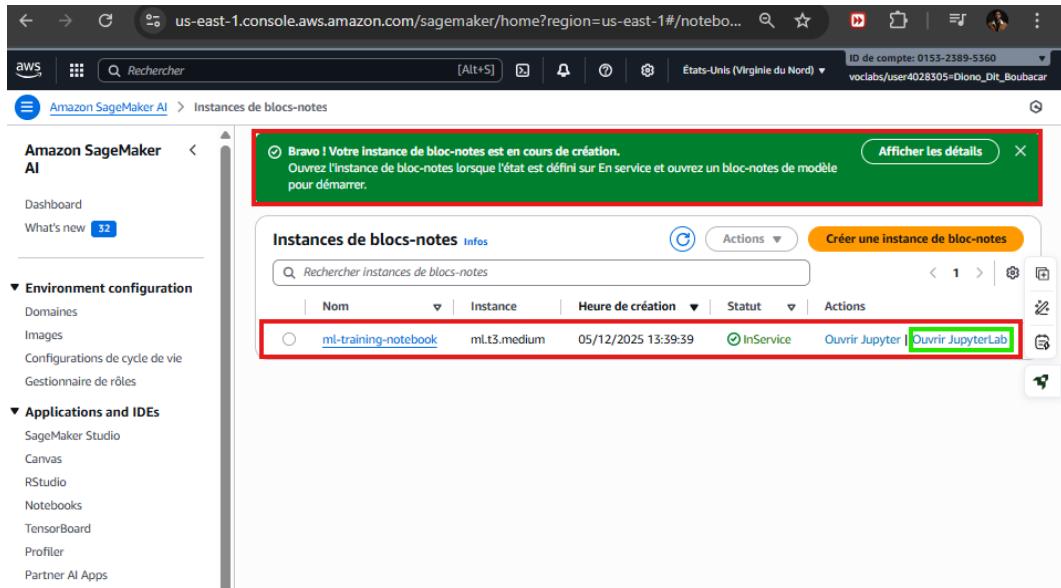


FIGURE 3.9 – Confirmation de la création réussie du Notebook dans SageMaker.

Nous avons organisé les dossiers dans JupyterLab pour maintenir une structure de projet claire et modulable.

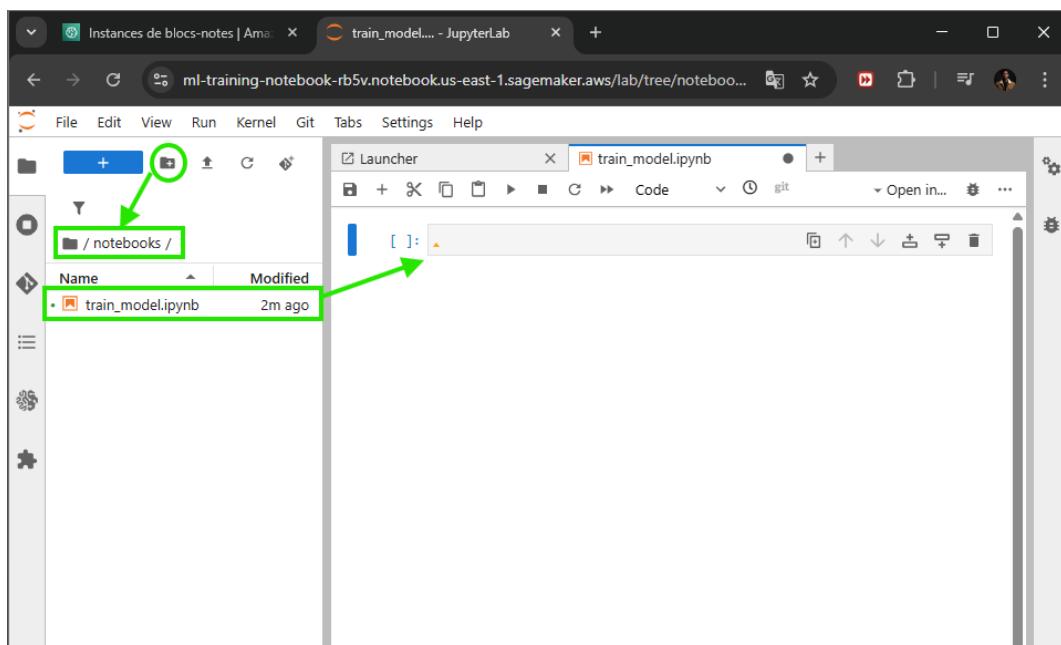


FIGURE 3.10 – Organisation des dossiers dans JupyterLab pour une structure MLOps propre.

Cette étape montre l'ouverture du Notebook contenant le futur code d'entraînement.

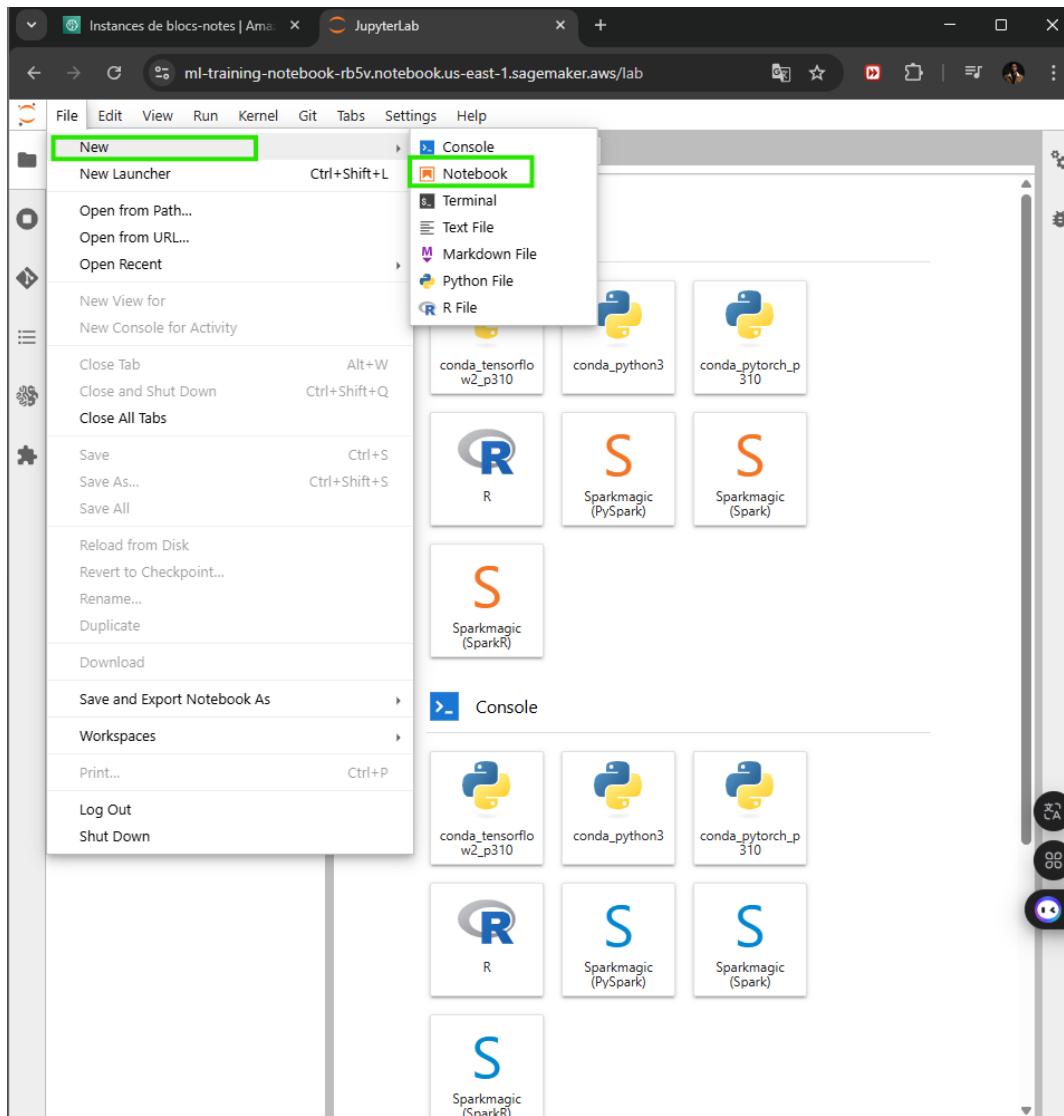


FIGURE 3.11 – Ouverture du Notebook contenant le futur code d'entraînement.

Nous avons collé le code d'entraînement dans le Notebook pour préparer l'exécution du modèle.

The screenshot shows a JupyterLab interface with a red box highlighting the main code editor area. The notebook is titled 'train_model.ipynb'. The code imports various libraries like pandas, numpy, and scikit-learn, and then loads the Iris dataset. It prints the version numbers of the imported packages (Pandas 2.3.3, Scikit-learn 1.7.2). The code then prints the first few rows of the dataset (X and y) and shows the dimensions of the dataset (150 rows by 4 columns).

```
# --- IMPORTATIONS ---
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
import joblib
import json
from datetime import datetime
import os

print("✅ Importations réussies")
print(f"Pandas: {pd.__version__}")

# Pour scikit-learn, faut l'importer d'abord
import sklearn
print(f"Scikit-learn: {sklearn.__version__}")

# --- CHARGEMENT DU DATASET ---
print("=" * 50)
print("📊 CHARGEMENT DU DATASET IRIS")
print("=" * 50)

data = load_iris()
X = pd.DataFrame(data.data, columns=data.feature_names)
y = pd.Series(data.target, name='target')

print(f"✅ Dataset chargé depuis scikit-learn")
print(f"\n💡 Features (variables):")
for i, feat in enumerate(data.feature_names, 1):
    print(f"  {i}. {feat}")

print("\n💡 Classes à prédire:")
for i, cls in enumerate(data.target_names, 1):
    print(f"  {i}. {cls}")

print("\n💡 Dimensions: {X.shape[0]} lignes x {X.shape[1]} colonnes")

# Aperçu des premières lignes
print("\n💡 Aperçu des données (5 premières lignes):")
print(X.head())
=====

📊 CHARGEMENT DU DATASET IRIS
=====
✅ Dataset chargé depuis scikit-learn
```

FIGURE 3.12 – Insertion du code d'entraînement dans le Notebook SageMaker.

Le Notebook a généré un récapitulatif des résultats d'entraînement pour vérifier la cohérence des sorties.

```

PROJET DEVOPS-MLOPS - PIPELINE COMPLET
Entraînement & Sauvegarde du Modèle ML

1. IMPORTATION DES LIBRAIRIES
Pandas, Scikit-learn, Joblib importés

2. CHARGEMENT DU DATASET IRIS
• Source: scikit-learn (load_iris)
• Échantillons: 150, Features: 4
• Features: sepal length (cm), sepal width (cm), petal length (cm), petal width (cm)
• Classes: setosa, versicolor, virginica

3. PRÉPARATION DES DONNÉES (Train/Test Split)
• X_train: (120, 4) (80%)
• X_test: (30, 4) (20%)
• Distribution: {0: 40, 2: 40, 1: 40}

4. ENTRAÎNEMENT DU MODÈLE DECISION TREE
Modèle entraîné

5. ÉVALUATION SUR DONNÉES DE TEST
ACCURACY: 0.9667 (96.67%)

Rapport de classification:
setosa      Precision: 1.000 | Recall: 1.000
versicolor  Precision: 1.000 | Recall: 0.900
virginica   Precision: 0.909 | Recall: 1.000

6. IMPORTANCE DES FEATURES
1. sepal length (cm) → Importance: 0.000
2. sepal width (cm) → Importance: 0.000
3. petal length (cm) → Importance: 0.579
4. petal width (cm) → Importance: 0.421

7. SAUVEGARDE DU MODÈLE
model.pkl sauvegardé (2449 octets)
metadata.json sauvegardé

8. TEST DE PRÉDICTION
Exemple test:
Features: {'sepal_length': 4.4, 'sepal_width': 3.0, 'petal_length': 1.3, 'petal_width': 0.2}
→ Prédiction: setosa

Test API (exemple Setosa):
Input: {'sepal_length': 5.1, 'sepal_width': 3.5, 'petal_length': 1.4, 'petal_width': 0.2}
→ Prédiction: setosa

9. STRUCTURE CRÉÉE
model/
└── model.pkl      # Modèle sérialisé
└── metadata.json  # Métadonnées du modèle
Accuracy finale: 96.7%

```

FIGURE 3.13 – Résumé des résultats d'entraînement produit par le Notebook.

Nous avons vérifié que le modèle final (pkl) et les fichiers de métadonnées ont été correctement enregistrés.

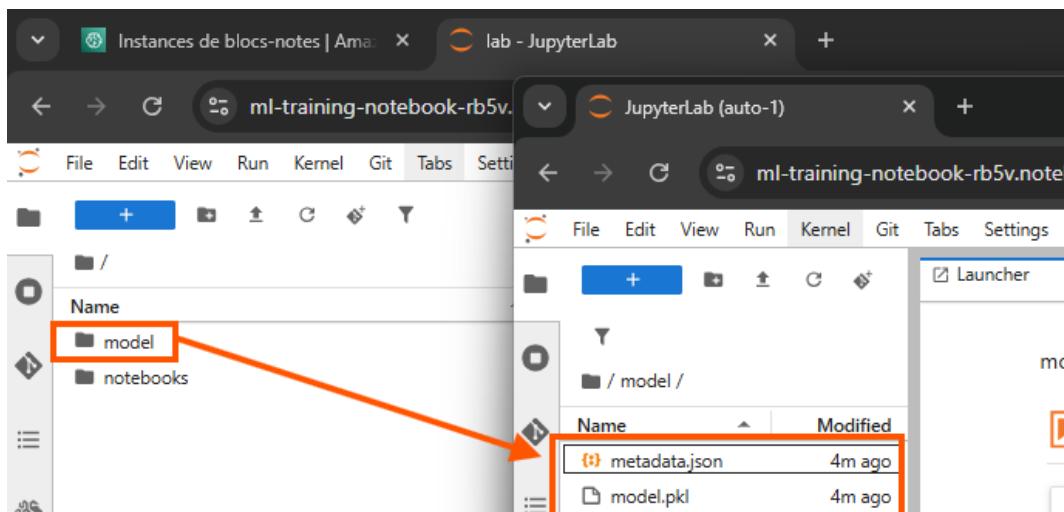


FIGURE 3.14 – Vérification de la sauvegarde du modèle final (pkl + métadonnées).

Enfin, nous avons téléchargé le modèle entraîné sur notre PC pour l'intégrer dans l'API Flask.

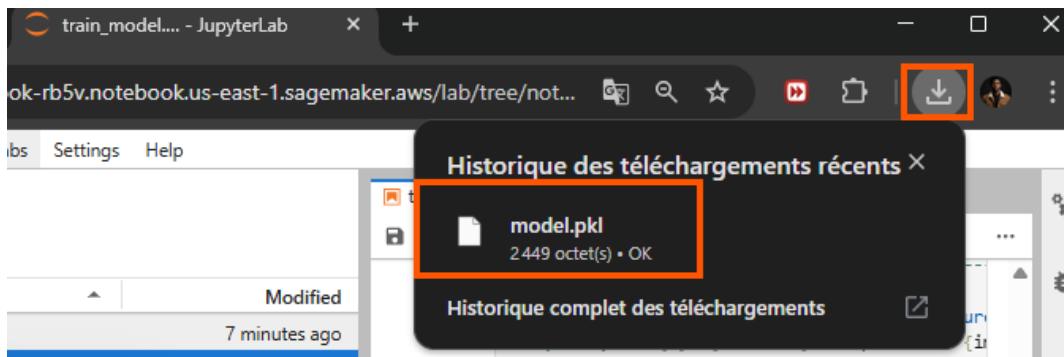


FIGURE 3.15 – Téléchargement local du modèle entraîné pour intégration dans l'API.

Nous avons créé le fichier ‘app.py‘ pour lancer l’API Flask et préparer les endpoints.

A screenshot of a terminal window with several command-line entries. The first two lines show directory navigation and opening a file: "C:\Users\DELL\Desktop\devops-mlops-aws-student-project>cd api" and "C:\Users\DELL\Desktop\devops-mlops-aws-student-project\api>notepad app.py". The third line shows the current directory: "C:\Users\DELL\Desktop\devops-mlops-aws-student-project\api>". Below the terminal is a code editor window with the file "app.py" open. The file path "C:\Users\DELL\Desktop\devops-mlops-aws-student-project\api\app.py" is highlighted with a red box. An orange arrow points from the terminal entry "notepad app.py" to the file tab in the code editor. The code editor shows the following Python script:

```
from flask import Flask, request, jsonify
from model_loader import load_model

app = Flask(__name__)
model = load_model()

@app.route("/")
def home():
    return {"message": "API ML operational"}

@app.route("/predict", methods=["POST"])
def predict():
    try:
        data = request.json["features"] # Ex: [5.1, 3.5, 1.4, 0.2]

        prediction = model.predict([data])[0]

        return jsonify({
            "input": data,
            "prediction": int(prediction)
        })

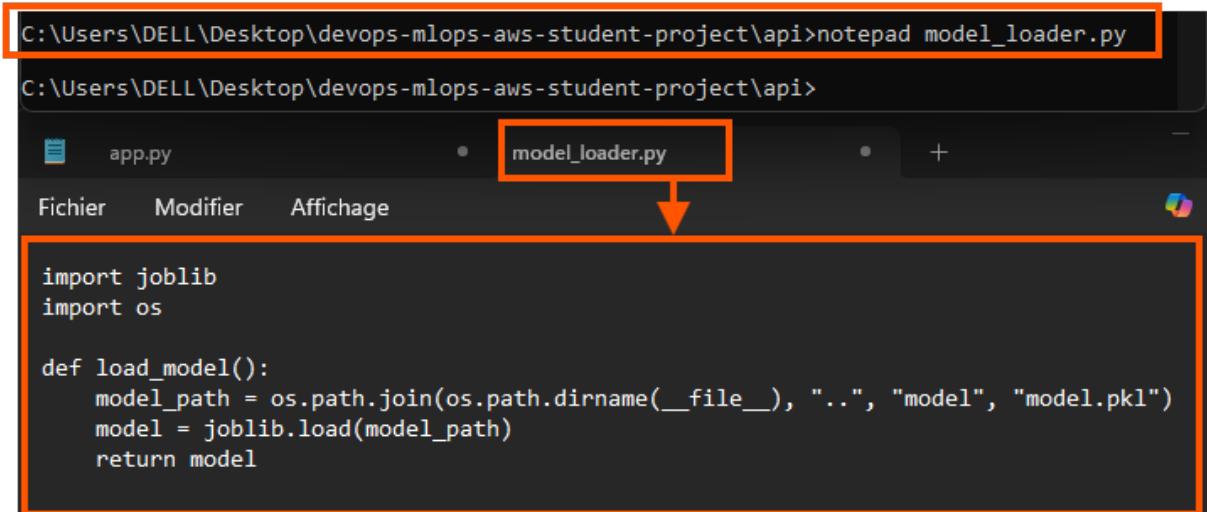
    except Exception as e:
        return jsonify({"error": str(e)}), 400

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

The entire code block is highlighted with a large red box.

FIGURE 3.16 – Crédit du fichier app.py pour l’API Flask.

Cette étape montre la création de model_loader.py pour charger le modèle pré-entraîné dans l'API.

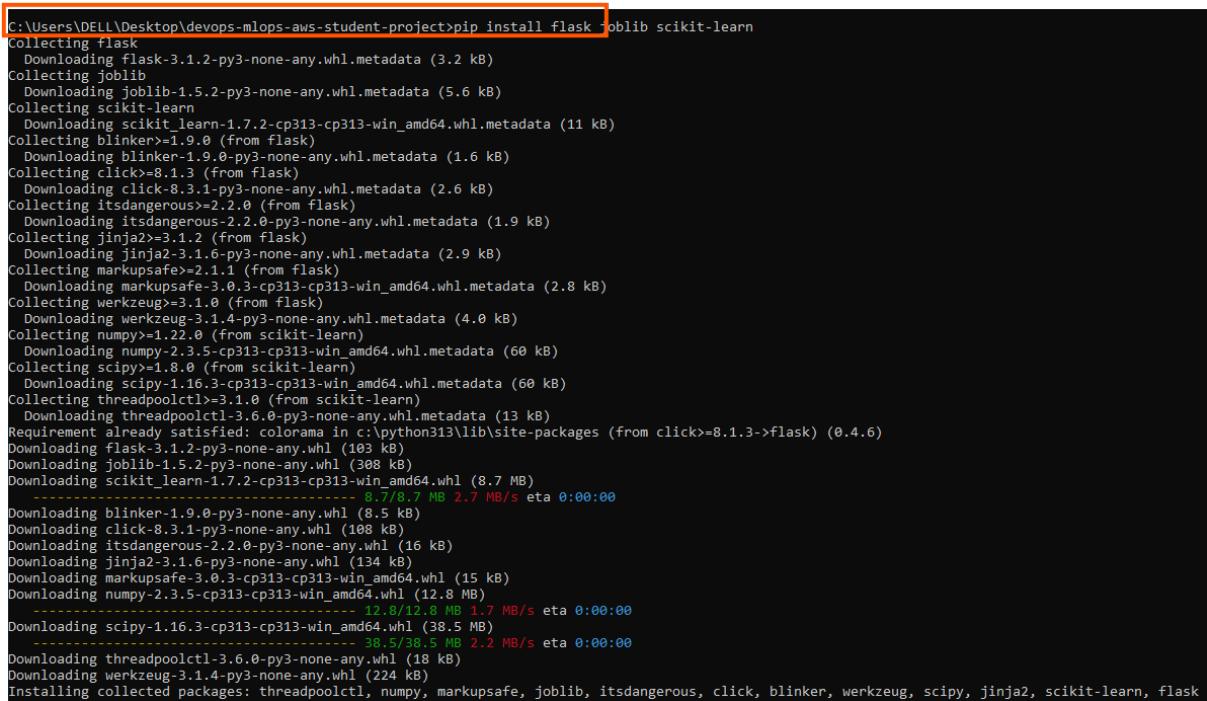


```
C:\Users\DELL\Desktop\devops-mlops-aws-student-project\api>notepad model_loader.py
C:\Users\DELL\Desktop\devops-mlops-aws-student-project\api>
● app.py          • model_loader.py          ● +
Fichier    Modifier    Affichage
↓
import joblib
import os

def load_model():
    model_path = os.path.join(os.path.dirname(__file__), "..", "model", "model.pkl")
    model = joblib.load(model_path)
    return model
```

FIGURE 3.17 – Création du fichier *model_loader.py* pour le chargement du module.

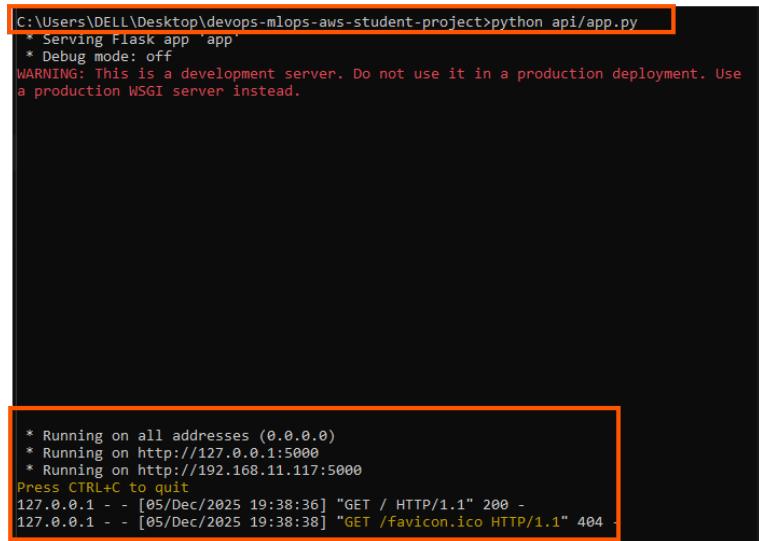
Nous avons installé Flask et vérifié les dépendances nécessaires pour exécuter l'API.



```
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>pip install flask -oblib scikit-learn
Collecting flask
  Downloading flask-3.1.2-py3-none-any.whl.metadata (3.2 kB)
Collecting joblib
  Downloading joblib-1.5.2-py3-none-any.whl.metadata (5.6 kB)
Collecting scikit-learn
  Downloading scikit_learn-1.7.2-cp313-cp313-win_amd64.whl.metadata (11 kB)
Collecting blinker>=1.9.0 (from flask)
  Downloading blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Collecting click>=8.1.3 (from flask)
  Downloading click-8.3.1-py3-none-any.whl.metadata (2.6 kB)
Collecting itsdangerous>=2.2.0 (from flask)
  Downloading itsdangerous-2.2.0-py3-none-any.whl.metadata (1.9 kB)
Collecting jinja2>=3.1.2 (from flask)
  Downloading jinja2-3.1.6-py3-none-any.whl.metadata (2.9 kB)
Collecting markupsafe>=2.1.1 (from flask)
  Downloading markupsafe-3.0.3-cp313-cp313-win_amd64.whl.metadata (2.8 kB)
Collecting werkzeug>=3.1.0 (from flask)
  Downloading werkzeug-3.1.4-py3-none-any.whl.metadata (4.0 kB)
Collecting numpy>=1.22.0 (from scikit-learn)
  Downloading numpy-2.3.5-cp313-cp313-win_amd64.whl.metadata (60 kB)
Collecting scipy>=1.8.0 (from scikit-learn)
  Downloading scipy-1.16.3-cp313-cp313-win_amd64.whl.metadata (60 kB)
Collecting threadpoolctl>=3.1.0 (from scikit-learn)
  Downloading threadpoolctl-3.6.0-py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: colorama in c:\python313\lib\site-packages (from click>=8.1.3->flask) (0.4.6)
Downloading flask-3.1.2-py3-none-any.whl (103 kB)
Downloading joblib-1.5.2-py3-none-any.whl (308 kB)
Downloading scikit_learn-1.7.2-cp313-cp313-win_amd64.whl (8.7 MB)
----- 8.7/8.7 MB 2.7 MB/s eta 0:00:00
Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Downloading click-8.3.1-py3-none-any.whl (108 kB)
Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
Downloading markupsafe-3.0.3-cp313-cp313-win_amd64.whl (15 kB)
Downloading numpy-2.3.5-cp313-cp313-win_amd64.whl (12.8 MB)
----- 12.8/12.8 MB 1.7 MB/s eta 0:00:00
Downloading scipy-1.16.3-cp313-cp313-win_amd64.whl (38.5 MB)
----- 38.5/38.5 MB 2.2 MB/s eta 0:00:00
Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Downloading werkzeug-3.1.4-py3-none-any.whl (224 kB)
Installing collected packages: threadpoolctl, numpy, markupsafe, joblib, itsdangerous, click, blinker, werkzeug, scipy, jinja2, scikit-learn, flask
```

FIGURE 3.18 – Installation de Flask pour le projet.

Cette étape montre le lancement de l'API localement afin de vérifier que tout fonctionne correctement.



```
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>python api/app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use
a production WSGI server instead.

 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.11.117:5000
Press CTRL+C to quit
127.0.0.1 - - [05/Dec/2025 19:38:36] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [05/Dec/2025 19:38:38] "GET /favicon.ico HTTP/1.1" 404 -
```

FIGURE 3.19 – Lancement de l'API Flask.

Nous avons ouvert la page HTTP sur 'localhost :5000' pour confirmer que l'API répond aux requêtes.

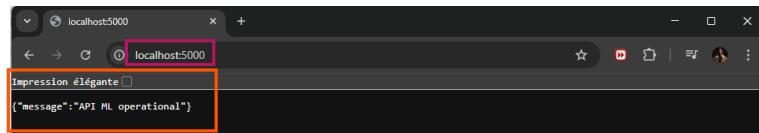
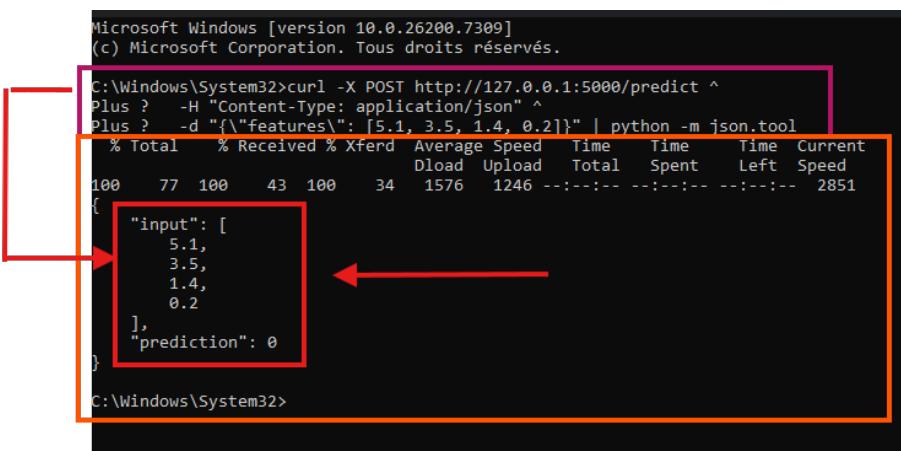


FIGURE 3.20 – Affichage de la page HTTP une fois l'API lancée.

Test de prédiction via curl pour s'assurer que le modèle intégré à l'API retourne les résultats attendus.



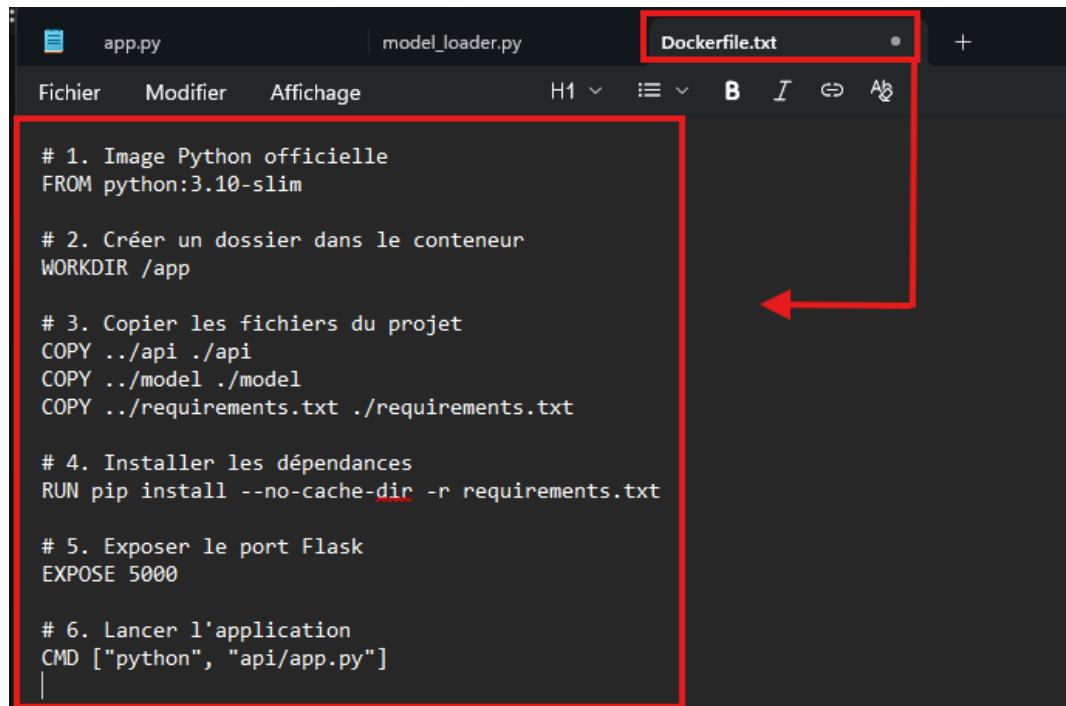
```
Microsoft Windows [version 10.0.26200.7309]
(c) Microsoft Corporation. Tous droits réservés.

C:\Windows\System32>curl -X POST http://127.0.0.1:5000/predict ^
  -H "Content-Type: application/json" ^
  -d "{\"features": [5.1, 3.5, 1.4, 0.2]} | python -m json.tool
% Total    % Received % Xferd  Average Speed   Time   Time  Current
          Dload  Upload Total   Spent    Left  Speed
100      77  100    43  100     34   1576  1246 --:--:-- --:--:-- 2851
{
  "input": [
    5.1,
    3.5,
    1.4,
    0.2
  ],
  "prediction": 0
}
C:\Windows\System32>
```

FIGURE 3.21 – Test de prédiction avec curl sur l'API Flask.

3.3 Conteneurisation avec Docker

Nous avons créé le fichier Dockerfile pour construire l'image contenant l'API Flask et le modèle.



```
# 1. Image Python officielle
FROM python:3.10-slim

# 2. Créer un dossier dans le conteneur
WORKDIR /app

# 3. Copier les fichiers du projet
COPY ../api ./api
COPY ..model ./model
COPY ..requirements.txt ./requirements.txt

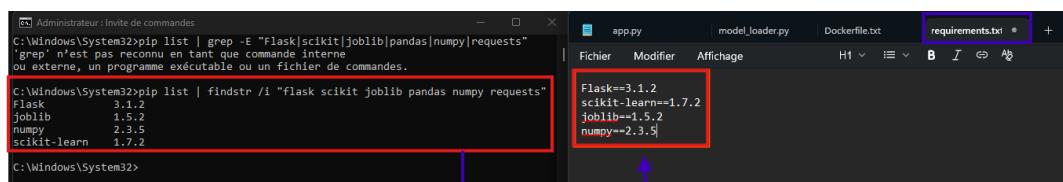
# 4. Installer les dépendances
RUN pip install --no-cache-dir -r requirements.txt

# 5. Exposer le port Flask
EXPOSE 5000

# 6. Lancer l'application
CMD ["python", "api/app.py"]
```

FIGURE 3.22 – Crédit de Dockerfile pour le projet.

Puis nous avons vérifié les versions des bibliothèques et rempli le fichier requirements.txt pour Docker.

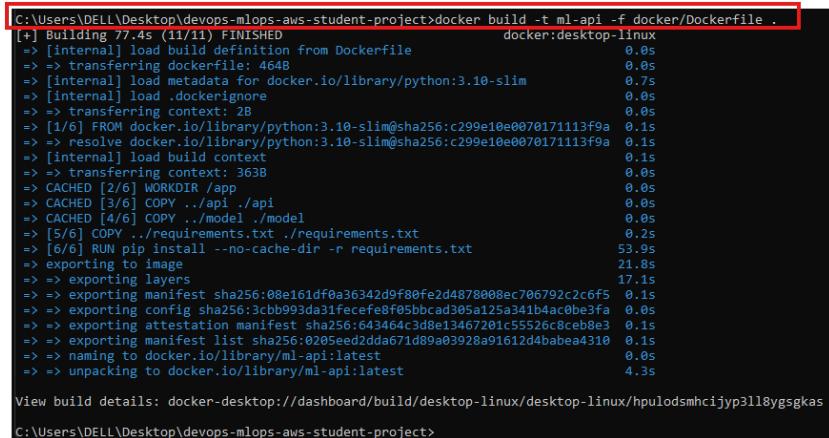


```
C:\Windows\System32>pip list | grep -E "Flask|scikit|joblib|pandas|numpy|requests"
'grep' n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.
C:\Windows\System32>pip list | findstr /i "flask scikit joblib pandas numpy requests"
Flask          2.1.2
joblib        1.5.2
numpy         2.3.5
scikit-learn   1.7.2
```

```
Flask==3.1.2
scikit-learn==1.7.2
joblib==1.5.2
numpy==2.3.5
```

FIGURE 3.23 – Vérification des versions et remplissage du requirements.txt.

Construction de l'image Docker avec tous les composants nécessaires pour exécuter l'API et le Dashboard.



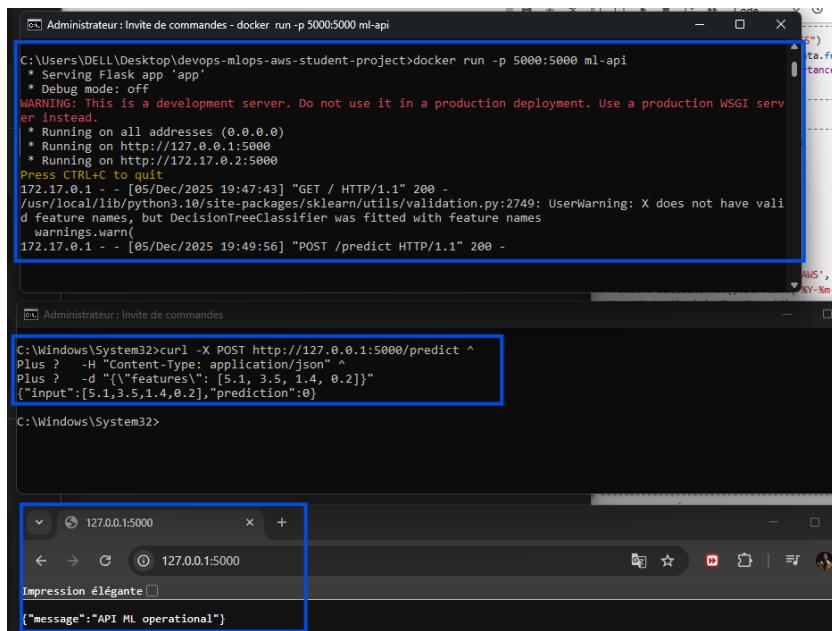
```
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>docker build -t ml-api -f docker/Dockerfile .
[+] Building 77.4s (11/11) FINISHED
   docker:desktop-linux
--> [internal] load build definition from Dockerfile
--> => transferring dockerfile: 464B
--> [internal] load metadata for docker.io/library/python:3.10-slim
--> [internal] load .dockerignore
--> => transferring context: 2B
--> [1/6] FROM docker.io/library/python:3.10-slim@sha256:c299e10e0070171113f9a
--> => resolve docker.io/library/python:3.10-slim@sha256:c299e10e0070171113f9a
--> [internal] load build context
--> => transferring context: 363B
--> CACHED [2/6] WORKDIR /app
--> CACHED [3/6] COPY ./api ./api
--> CACHED [4/6] COPY ./model ./model
--> [5/6] COPY ./requirements.txt ./requirements.txt
--> [6/6] RUN pip install --no-cache-dir -r requirements.txt
--> exporting to image
--> exporting layers
--> exporting manifest sha256:08e161df0a36342d0f80fc2d4878008ec706792c2c6f5
--> => exporting config sha256:3ccb993da31fecefe8f05bbcad305a125a341b4a0be3fa
--> => exporting attestation manifest sha256:643464c3d8e13467201c5526c8ceb8e3
--> => exporting manifest list sha256:0205seed2ddaa671d89a03928a91612d4babae4310
--> => naming to docker.io/library/ml-api:latest
--> => unpacking to docker.io/library/ml-api:latest
--> => 4.3s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/hpulodsmhcijyp3ll8ygsgkas

C:\Users\DELL\Desktop\devops-mlops-aws-student-project>
```

FIGURE 3.24 – Lancement du build Docker pour créer l'image du projet.

Lancement du conteneur Docker et vérification que l'API répond correctement aux requêtes HTTP.



```
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>docker run -p 5000:5000 ml-api
C:\Windows\System32>curl -X POST http://127.0.0.1:5000/predict ^
  Plus ?
  -H "Content-Type: application/json" ^
  Plus ?
  -d "{\"features\": [5.1, 3.5, 1.4, 0.2]}"
  {"input": [5.1, 3.5, 1.4, 0.2], "prediction": 0}

C:\Windows\System32>
```

FIGURE 3.25 – Exécution du conteneur Docker et test de l'API.

3.4 Gestion du code avec GitHub

Pour commencer, nous nous sommes connectés à GitHub afin de créer un nouveau repository dédié au projet.

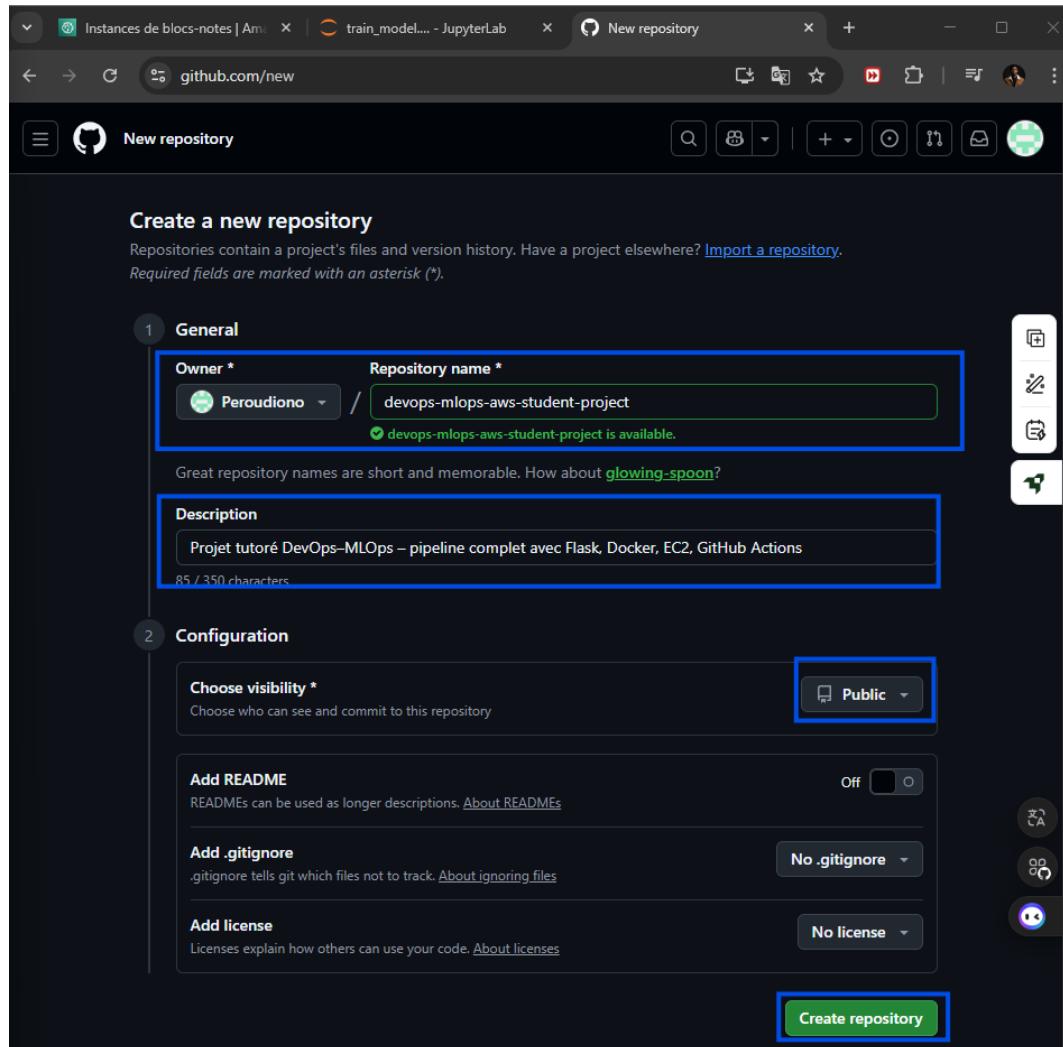


FIGURE 3.26 – Connexion à GitHub et création du repository du projet.

Le repository a été créé avec succès et affiché dans l'interface GitHub.

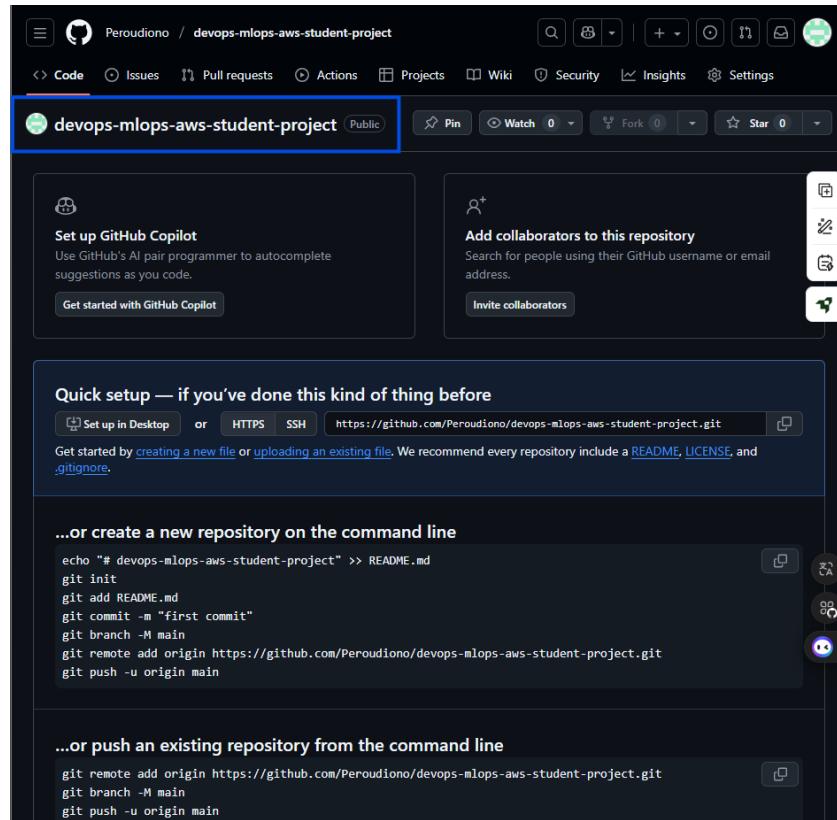


FIGURE 3.27 – Confirmation de la création réussie du repository.

Nous avons poussé le projet local vers GitHub pour centraliser le code.

```
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>git add .
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>git commit -m "Initial project files - DevOps-MLOps pipeline"
[master (root-commit) efe7d9a] initial project files - DevOps-MLOps pipeline
 6 files changed, 57 insertions(+)
create mode 100644 api/_pycache_/_model_loader.cpython-313.pyc
create mode 100644 api/app.py
create mode 100644 api/model_loader.py
create mode 100644 docker/Dockerfile
create mode 100644 model/model.pkl
create mode 100644 requirements.txt
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>git branch -M main
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>git remote add origin https://github.com/Peroudiono/devops-mlops-aws-student-project.git
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>git push -u origin main
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 8 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (12/12), 2.92 KiB | 271.00 KiB/s, done.
Total 12 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Peroudiono/devops-mlops-aws-student-project.git
 * [new branch]    main -> main
branch 'main' set up to track 'origin/main'.
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>
```

FIGURE 3.28 – Push du projet local vers le repository GitHub.

La vérification sur GitHub a confirmé que tous les fichiers ont été correctement envoyés.

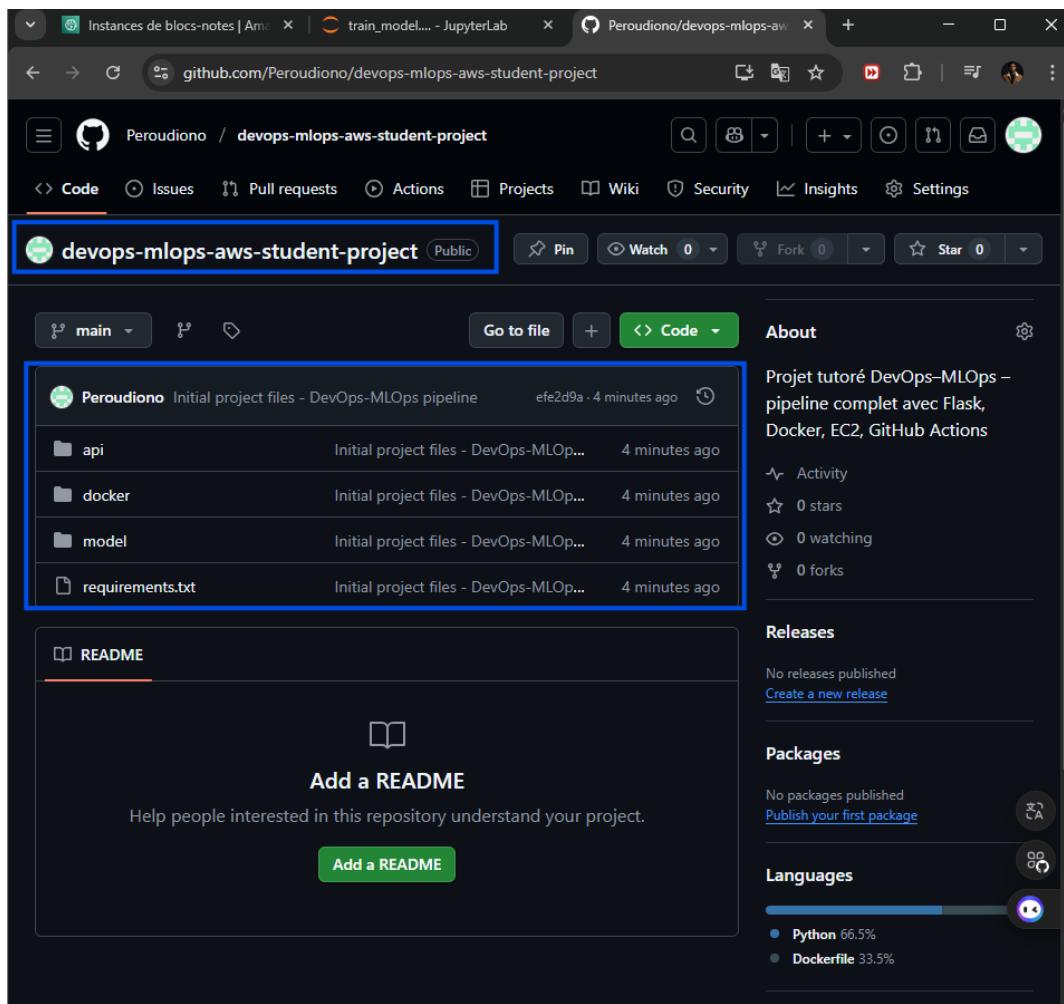
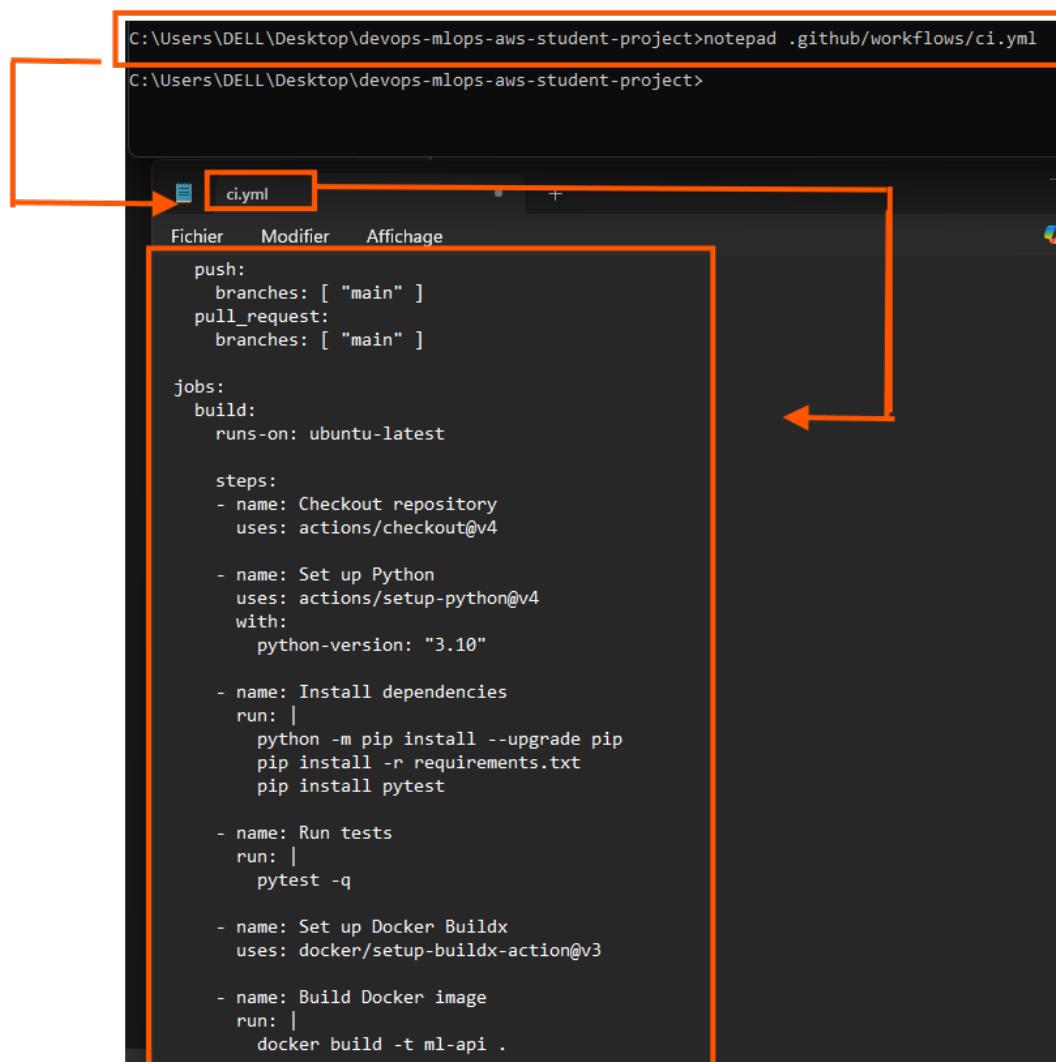


FIGURE 3.29 – Vérification de la réussite de l'envoi du projet sur GitHub.

3.5 Pipeline CI/CD avec GitHub Actions

Nous avons créé le fichier de workflow GitHub Actions pour automatiser les tests et la construction Docker.



```
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>notepad .github/workflows/ci.yml
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>

ci.yml

Fichier Modifier Affichage
push:
  branches: [ "main" ]
pull_request:
  branches: [ "main" ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: "3.10"

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
          pip install pytest

      - name: Run tests
        run: |
          pytest -q

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3

      - name: Build Docker image
        run: |
          docker build -t ml-api .
```

FIGURE 3.30 – Écriture du fichier YAML pour le pipeline CI/CD.

Après avoir configuré le workflow, un commit et un push ont déclenché l'exécution automatique.

The screenshot shows a GitHub repository interface. At the top, a terminal window displays the command-line output of a git commit and push:

```
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>git add .github/workflows/ci.yml
C:\Users\DELL\Desktop\devops-mlops-aws-student-project>git commit -m "Add CI pipeline with GitHub Actions"
[main e7b5a3c] Add CI pipeline with GitHub Actions
 1 file changed, 37 insertions(+)
 create mode 100644 .github/workflows/ci.yml

C:\Users\DELL\Desktop\devops-mlops-aws-student-project>git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 698 bytes | 349.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Peroudiono/devops-mlops-aws-student-project.git
  efe2d9a..e7b5a3c main -> main
```

An orange box highlights the commit message and the pushed file. An orange arrow points from this highlighted area to the GitHub UI below, specifically to the list of files in the repository. The GitHub UI shows the repository details, including the commit history where the CI pipeline run is visible:

Peroudiono / devops-mlops-aws-student-project

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

devops-mlops-aws-student-project Public

main Go to file + <> Code

Peroudiono Add CI pipeline with GitHub Actions · e7b5a3c · 4 minutes ago

.github/workflows Add CI pipeline with GitHub Actions · 4 minutes ago

api Initial project files - DevOps-MLO... · 24 minutes ago

docker Initial project files - DevOps-MLO... · 24 minutes ago

model Initial project files - DevOps-MLO... · 24 minutes ago

requirements.txt Initial project files - DevOps-MLO... · 24 minutes ago

README

About

Projet tutoré DevOps-MLOps – pipeline complet avec Flask, Docker, EC2, GitHub Actions

Activity

0 stars

0 watching

0 forks

Releases

No releases published

Create a new release

FIGURE 3.31 – Exécution du pipeline CI/CD après commit et push.

Nous avons vérifié que l'arborescence finale du projet sur GitHub respecte les exigences du pipeline.

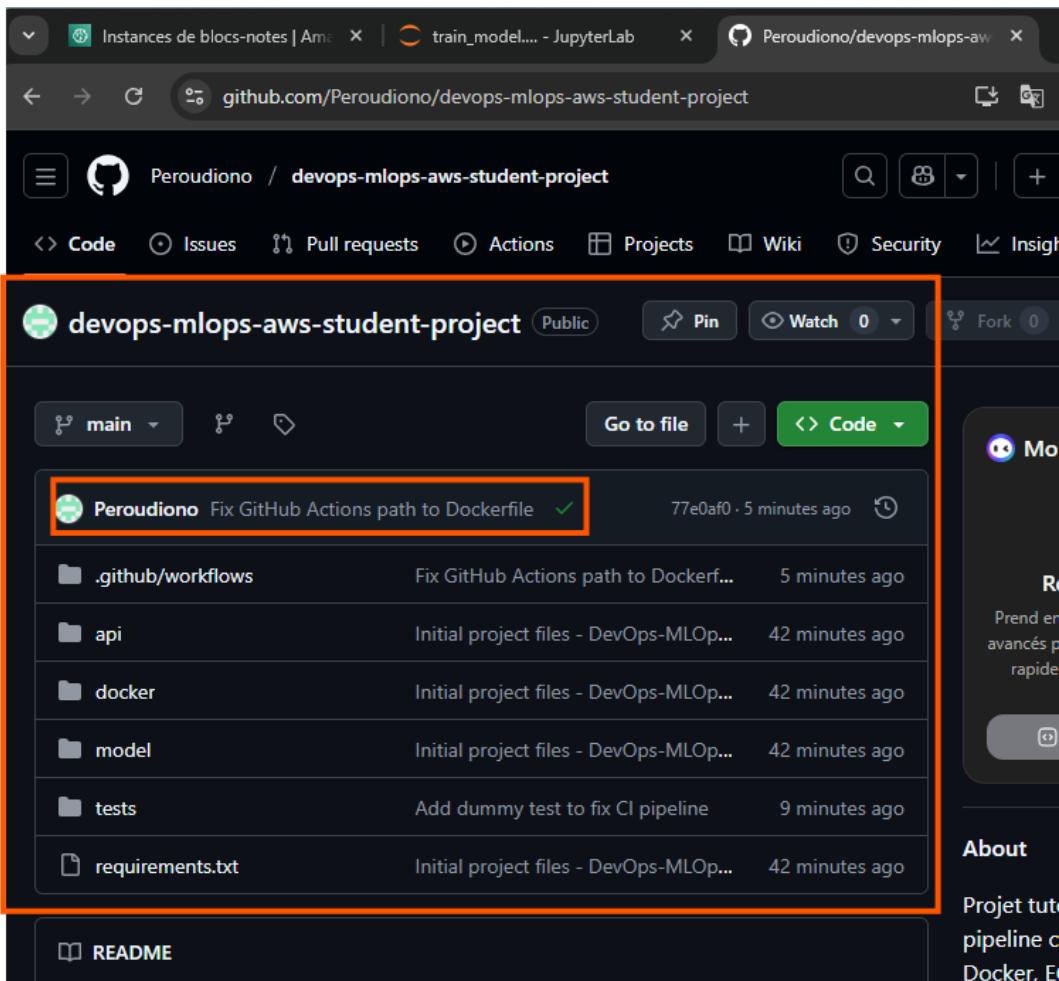


FIGURE 3.32 – Vérification de l'arborescence finale sur GitHub.

3.6 Déploiement sur AWS EC2 et Docker

Pour cette partie, nous avons configuré une instance EC2 afin de déployer l'application Dockerisée et tester les prédictions via l'API.

Avant tout, nous avons recherché et accédé au service EC2 sur AWS.

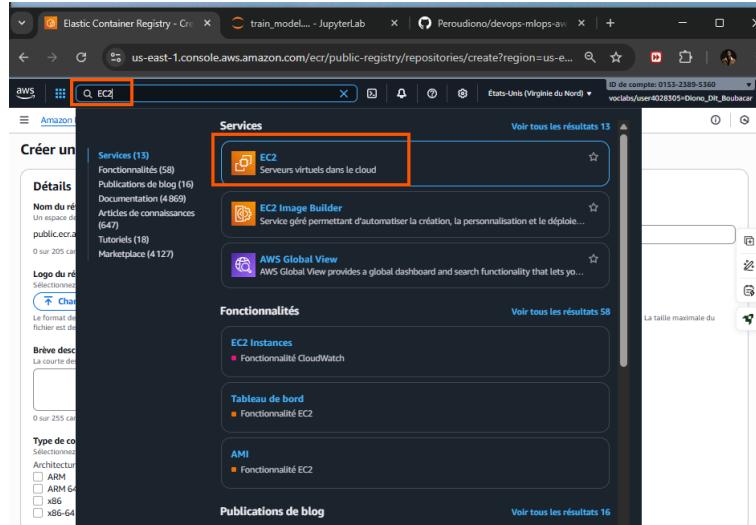


FIGURE 3.33 – Accès à AWS EC2 pour la création d'une instance.

Création de l'instance EC2 avec les configurations de base.

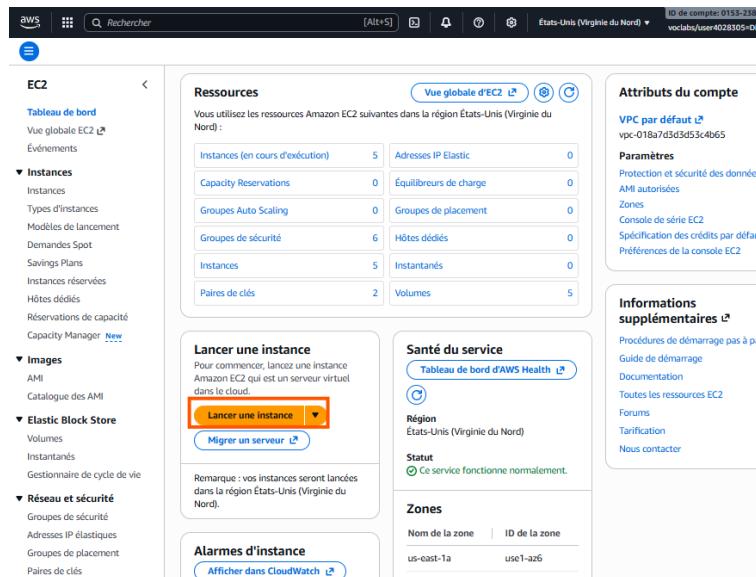


FIGURE 3.34 – Interface de création de l'instance EC2.

Configuration des informations et ressources de l'instance.

The screenshot shows the AWS EC2 'Launch Instance' configuration page. The top navigation bar includes the AWS logo, a search bar with 'Rechercher [Alt+S]', and various status icons. The region is set to 'États-Unis (Virginie du Nord)'. The main content area is titled 'Nom et balises' (Name and tags) and shows a 'Nom' field containing 'ml-api-server' with an orange border. A link 'Ajouter des balises supplémentaires' (Add additional tags) is visible. To the right, a 'Récapitulatif' (Summary) panel shows 'Nombre d'instances' (Number of instances) set to 1. Below this, sections for 'Image logicielle (AMI)' (Software image), 'Type de serveur virtuel' (Virtual server type), 'Pare-feu (groupe de sécurité)' (Firewall (security group)), and 'Stockage (volumes)' (Storage (volumes)) are partially visible. The central area contains tabs for 'Récentes' and 'Démarrage rapide' (Recent and Quick Start). Under 'Amazon Machine Image (AMI)', a specific AMI entry for 'Ubuntu Server 22.04 LTS (HVM), SSD Volume Type' is highlighted with an orange border. This entry includes details like ID: ami-0c398cb65a93047f2, Architecture: 64 bits, and Date de publication: 2025-10-15. The 'Nom d'utilisateur' (User name) field is set to 'ubuntu' with a green 'Fournisseur vérifié' (Verified provider) badge. The 'Type d'instance' (Instance type) section shows 't2.micro' selected, with its details (Family: t2, 1 vCPU, 1 Go Mémoire, Generation actuelle: true, etc.) also highlighted with an orange border. A 'Toutes les générations' (All generations) checkbox and a 'Comparer les types d'instance' (Compare instance types) link are present. The bottom of the page has an 'Annuler' (Cancel) button.

FIGURE 3.35 – Paramétrage des informations de l'instance EC2.

Vérification de la création réussie de l’instance EC2.

The screenshot shows the AWS EC2 Instances page. The left sidebar has 'Instances' selected. The main area displays a table of 6 instances:

Name	ID d'instance	État de l'instance	Type d'insta...	Contrôle des statu...	Statut d'alarme
TP-DevOps	i-06d8d033d49855cd0	En cours d...	t2.micro	2/2 vérifications r	Afficher les alarm
TP2-DEVOPS	i-08a9a27d9b1a07cf1	En cours d...	t2.micro	2/2 vérifications r	Afficher les alarm
TP4-EC2#2	i-083f87ead1ca12594	En cours d...	t2.micro	2/2 vérifications r	Afficher les alarm
TP3-NodeJS-EC2	i-09648c44b6e053a42	En cours d...	t2.micro	2/2 vérifications r	Afficher les alarm
TP4-EC2#1	i-022d62921d19ccdd9d	En cours d...	t2.micro	1/2 vérifications r	Afficher les alarm
ml-api-server	i-07c0c0f07fd9fe3b0	En cours d...	t2.micro	2/2 vérifications r	Afficher les alarm

FIGURE 3.36 – Confirmation de la création réussie de l’instance EC2.

Mise en place du groupe de sécurité pour autoriser le trafic nécessaire.

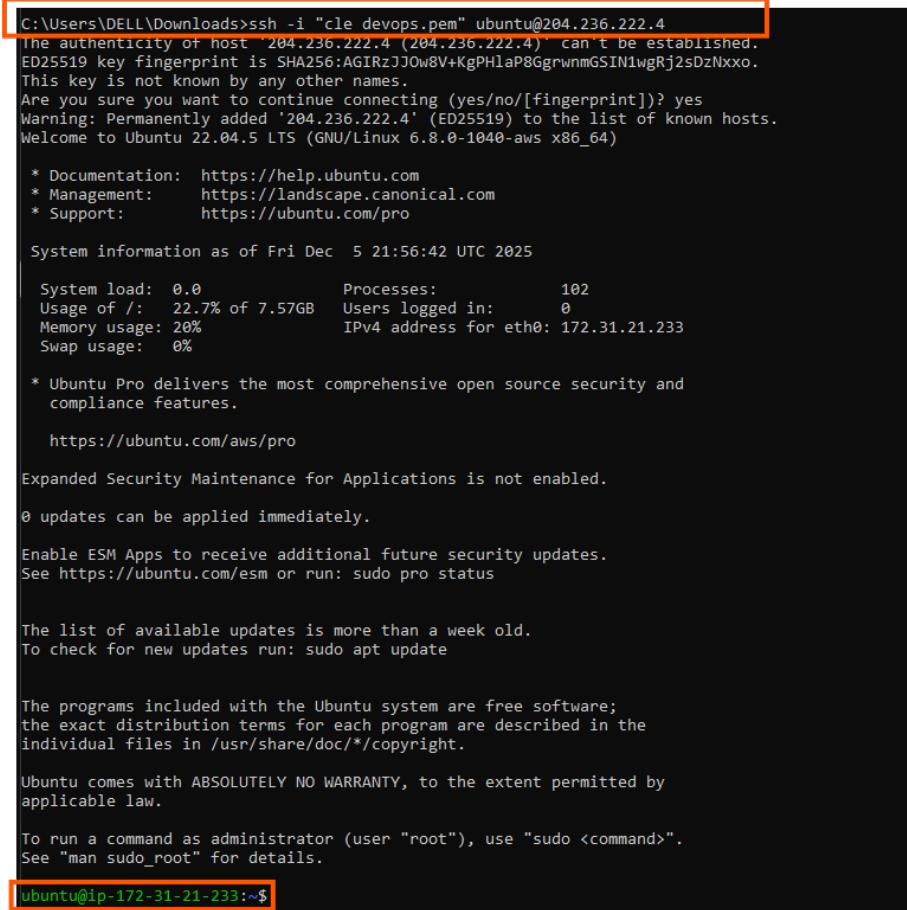
The screenshot shows the 'Modifier les règles entrantes' (Edit incoming rules) page for a security group. It lists two rules:

- ACCEZ SSH**: Type TCP, Port 22, Source 0.0.0.0/0
- Flask API**: Type TCP, Port 5000, Source 0.0.0.0/0

A warning message at the bottom states: "⚠️ Les règles dont la source est 0.0.0.0/0 ou ::/0 permettent à toutes les adresses IP d'accéder à votre instance. Nous vous recommandons de paramétrer les règles du groupe de sécurité de sorte que les accès soient uniquement autorisés depuis des adresses IP connues."

FIGURE 3.37 – Configuration du groupe de sécurité pour l’instance EC2.

Connexion SSH réussie à l'instance pour configuration.



```
C:\Users\DELL\Downloads>ssh -i "cle devops.pem" ubuntu@204.236.222.4
The authenticity of host '204.236.222.4 (204.236.222.4)' can't be established.
ED25519 key fingerprint is SHA256:AGIRzJJ0w8V+KgPHlaP8GgrwnmGSIN1wgRj2sDzNxxo.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '204.236.222.4' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.8.0-1040-aws x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

System information as of Fri Dec 5 21:56:42 UTC 2025

System load: 0.0          Processes:           102
Usage of /: 22.7% of 7.57GB  Users logged in:      0
Memory usage: 20%          IPv4 address for eth0: 172.31.21.233
Swap usage:  0%

* Ubuntu Pro delivers the most comprehensive open source security and
  compliance features.

  https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

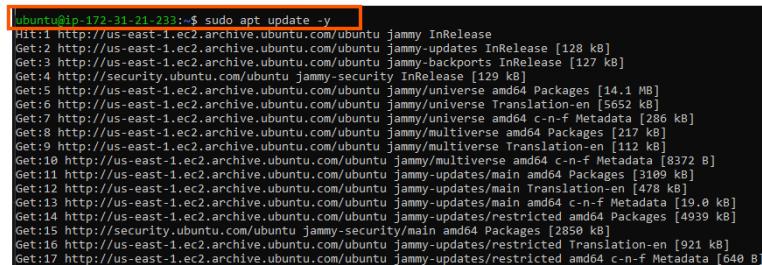
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-21-233:~$
```

FIGURE 3.38 – Connexion SSH à l'instance EC2 pour déployer l'application.

Mise à jour du système avec update et upgrade avant installation des dépendances.



```
ubuntu@ip-172-31-21-233:~$ sudo apt update -y
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 Packages [14.1 MB]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe Translation-en [5652 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 c-n-f Metadata [286 kB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 Packages [217 kB]
Get:9 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse Translation-en [112 kB]
Get:10 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/multiverse amd64 c-n-f Metadata [8372 B]
Get:11 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [3109 kB]
Get:12 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main Translation-en [478 kB]
Get:13 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 c-n-f Metadata [19.0 kB]
Get:14 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages [4939 kB]
Get:15 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [2858 kB]
Get:16 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted Translation-en [921 kB]
Get:17 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 c-n-f Metadata [640 B]
```

FIGURE 3.39 – Mise à jour de l'instance EC2 avec update et upgrade.

Installation de Docker sur l’instance EC2.

```
ubuntu@ip-172-31-21-233:~$ sudo apt install docker.io -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-buildx docker-compose-v2 docker-doc
rinse zfs-fuse | zfsutils
The following NEW packages will be installed:
bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 0 not upgraded.
Need to get 76.3 MB of archives.
After this operation, 289 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/universe amd64 pigz amd64 2.6.1 [63.6 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy/main amd64 bridge-utils amd64 1.7-1ubuntu3 [34.4 kB]
```

FIGURE 3.40 – Installation de Docker sur EC2 pour exécuter l’application.

Activation et attribution des droits Docker à l’utilisateur.

```
ubuntu@ip-172-31-21-233:~$ sudo systemctl start docker
ubuntu@ip-172-31-21-233:~$ sudo systemctl enable docker
ubuntu@ip-172-31-21-233:~$ sudo systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2025-12-05 22:04:27 UTC; 1min 17s ago
     TriggeredBy: • docker.socket
       Docs: https://docs.docker.com
      Main PID: 12667 (dockerd)
        Tasks: 8
       Memory: 31.9M
          CPU: 290ms
        CGroup: /system.slice/docker.service
                └─12667 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock

Dec 05 22:04:27 ip-172-31-21-233 dockerd[12667]: time="2025-12-05T22:04:27.025962103Z" level=info msg="Log file rotated"
Dec 05 22:04:27 ip-172-31-21-233 dockerd[12667]: time="2025-12-05T22:04:27.367846679Z" level=info msg="Log file rotated"
Dec 05 22:04:27 ip-172-31-21-233 dockerd[12667]: time="2025-12-05T22:04:27.388490166Z" level=info msg="Docker daemon has started up"
Dec 05 22:04:27 ip-172-31-21-233 dockerd[12667]: time="2025-12-05T22:04:27.388668937Z" level=info msg="Ingesting configuration from /etc/docker/daemon.json"
Dec 05 22:04:27 ip-172-31-21-233 dockerd[12667]: time="2025-12-05T22:04:27.395613545Z" level=warning msg="Ingesting configuration from /etc/docker/daemon.json"
Dec 05 22:04:27 ip-172-31-21-233 dockerd[12667]: time="2025-12-05T22:04:27.395634020Z" level=warning msg="Ingesting configuration from /etc/docker/daemon.json"
Dec 05 22:04:27 ip-172-31-21-233 dockerd[12667]: time="2025-12-05T22:04:27.423390033Z" level=info msg="Comparing configuration from /etc/docker/daemon.json"
Dec 05 22:04:27 ip-172-31-21-233 dockerd[12667]: time="2025-12-05T22:04:27.432694192Z" level=info msg="Docker daemon is ready to receive API calls"
Dec 05 22:04:27 ip-172-31-21-233 dockerd[12667]: time="2025-12-05T22:04:27.432875853Z" level=info msg="API endpoint listening on /var/run/docker.sock"
Dec 05 22:04:27 ip-172-31-21-233 systemd[1]: Started Docker Application Container Engine.

ubuntu@ip-172-31-21-233:~$ sudo usermod -aG docker ubuntu
ubuntu@ip-172-31-21-233:~$
```

FIGURE 3.41 – Configuration des droits Docker sur l’instance.

Installation de Git et clonage du projet depuis GitHub.

```
ubuntu@ip-172-31-21-233:~$ sudo apt install git -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
git is already the newest version (1:2.34.1-1ubuntu1.15).
git set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
ubuntu@ip-172-31-21-233:~$ git clone https://github.com/Peroudiono/devops-mlops-aws-student-project.git
Cloning into 'devops-mlops-aws-student-project'...
remote: Enumerating objects: 26, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 26 (delta 4), reused 26 (delta 4), pack-reused 0 (from 0)
Receiving objects: 100% (26/26), 4.23 KiB | 1.06 MiB/s, done.
Resolving deltas: 100% (4/4), done.
ubuntu@ip-172-31-21-233:~$ cd devops-mlops-aws-student-project
ubuntu@ip-172-31-21-233:~/devops-mlops-aws-student-project$
```

FIGURE 3.42 – Clonage du projet depuis GitHub sur l’instance EC2.

Construction de l'image Docker contenant l'API et le modèle.

```
ubuntu@ip-172-31-21-233:~/devops-mlops-aws-student-project$ docker build -t ml-api -f docker/Dockerfile .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
Install the buildx component to build images with BuildKit:
https://docs.docker.com/go/buildx/
Sending build context to Docker daemon 77.82kB
Step 1/8 : FROM python:3.10-slim
--> 6f924957e3d2
Step 2/8 : WORKDIR /app
--> Using cache
--> 872412b70785
Step 3/8 : COPY api/ ./api/
--> 9de8d4357b21
Step 4/8 : COPY model/ ./model/
--> a8c204cd896a
Step 5/8 : COPY requirements.txt .
--> a0dad4fe4617c
Step 6/8 : RUN pip install --no-cache-dir -r requirements.txt
--> Running in 10409613985d
Collecting Flask==3.1.2
  Downloading flask-3.1.2-py3-none-any.whl (103 kB)
  103.3/103.3 kB 25.7 MB/s eta 0:00:00
Collecting scikit-learn==1.7.2
  Downloading scikit_learn-1.7.2-cp310-cp310-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (9.7 MB)
  9.7/9.7 kB 116.9 MB/s eta 0:00:00
Collecting joblib==1.5.2
  Downloading joblib-1.5.2-py3-none-any.whl (308 kB)
  308.4/308.4 kB 307.3 MB/s eta 0:00:00
Collecting numpy==1.24.3
  Downloading numpy-1.24.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.3 MB)
  17.3/17.3 kB 137.6 MB/s eta 0:00:00
Collecting blinker>=1.9.0
  Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Collecting jinja2>=3.1.2
  Downloading jinja2-3.1.6-py3-none-any.whl (134 kB)
  134.9/134.9 kB 338.1 MB/s eta 0:00:00
Collecting itsdangerous>=2.2.0
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting markupsafe>=2.1.1
  Downloading markupsafe-2.0.3-cp310-cp310-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (20 kB)
Collecting click>=8.1.3
  Downloading click-8.3.1-py3-none-any.whl (108 kB)
  108.3/108.3 kB 326.6 MB/s eta 0:00:00
Collecting werkzeug>=3.1.0
  Downloading werkzeug-3.1.4-py3-none-any.whl (224 kB)
  225.0/225.0 kB 382.0 MB/s eta 0:00:00
Collecting threadpoolctl>=3.1.0
  Downloading threadpoolctl-3.6.0-py3-none-any.whl (18 kB)
Collecting scipy>=1.8.0
```

FIGURE 3.43 – Construction de l'image Docker pour l'API.

Lancement du conteneur Docker et vérification de son fonctionnement.

```
ubuntu@ip-172-31-21-233:~/devops-mlops-aws-student-project$ docker run -d -p 5000:5000 ml-api
27d5674c0ec7a49d4949cc2f5731b097e8e083fc73b54f7ef9604aae17bf295
ubuntu@ip-172-31-21-233:~/devops-mlops-aws-student-project$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
27d5674c0ec7        ml-api              "python api/app.py"   10 seconds ago    Up 9 seconds      0.0.0.0:5000->5000/tcp, [::]:5000->5000/tcp   vigorous_montalcini
ubuntu@ip-172-31-21-233:~/devops-mlops-aws-student-project$
```

FIGURE 3.44 – Exécution du conteneur Docker et vérification de l'API.

Test final des prédictions via l'API depuis l'instance.

```
ubuntu@ip-172-31-21-233:~/devops-mlops-aws-student-project$ curl -X POST http://204.236.222.4:5000/predict
-H "Content-Type: application/json" -d "{\"features\": [5.1, 3.5, 1.4, 0.2]}"
{"input": [5.1, 3.5, 1.4, 0.2], "prediction": 0}
```

FIGURE 3.45 – Test final de prédiction pour valider le pipeline complet.

Exécution de prédictions supplémentaires pour s'assurer de la stabilité.

```
ubuntu@ip-172-31-21-233:~/devops-mlops-aws-student-project$ # Test 1 : Iris-setosa (classe 0)
curl -X POST http://204.236.222.4:5000/predict \
-H "Content-Type: application/json" \
-d '{"features": [5.1, 3.5, 1.4, 0.2]}'

echo ""

# Test 2 : Iris-versicolor (classe 1)
curl -X POST http://204.236.222.4:5000/predict \
-H "Content-Type: application/json" \
-d '{"features": [6.0, 2.7, 5.1, 1.6]}'

echo ""

# Test 3 : Iris-virginica (classe 2)
curl -X POST http://204.236.222.4:5000/predict \
-H "Content-Type: application/json" \
-d '{"features": [7.7, 3.0, 6.1, 2.3]}'
{"input": [5.1, 3.5, 1.4, 0.2], "prediction": 0} ←
{"input": [6.0, 2.7, 5.1, 1.6], "prediction": 1} ←
{"input": [7.7, 3.0, 6.1, 2.3], "prediction": 2} ←
ubuntu@ip-172-31-21-233:~/devops-mlops-aws-student-project$
```

FIGURE 3.46 – Tests complémentaires pour valider la cohérence des prédictions.

3.7 Dashboard final

Cette section illustre le résultat final du tableau de bord après intégration complète avec l'API Flask et le modèle ML.

Pour commencer, nous avons ouvert l'interface principale du dashboard pour vérifier la mise en place des fonctionnalités.

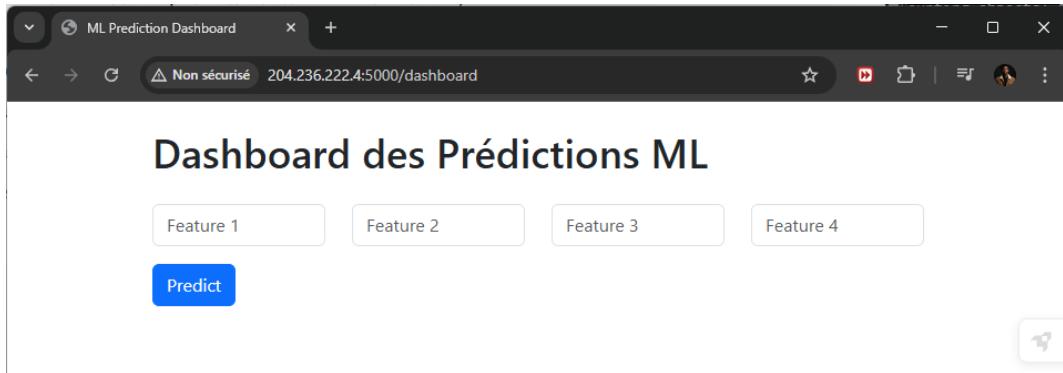


FIGURE 3.47 – Interface principale du dashboard après intégration.

Nous avons ensuite réalisé un test avec des données d'exemple afin de confirmer le bon fonctionnement des prédictions.

A screenshot of a web browser window titled "ML Prediction Dashboard". The address bar shows "Non sécurisé 204.236.222.4:5000/predict_dashboard". The page content is titled "Dashboard des Prédictions ML". It features four input fields labeled "Feature 1", "Feature 2", "Feature 3", and "Feature 4", followed by a blue "Predict" button. Below the input fields, there is a section titled "Historique des prédictions" containing a table with two rows of data. The table has columns for Feature 1, Feature 2, Feature 3, Feature 4, and Prediction. The first row has values 5.1, 3.5, 1.4, 0.2, and 0 respectively. The second row has values 6.0, 2.7, 5.1, 1.6, and 2 respectively.

FIGURE 3.48 – Test de prédiction effectué avec succès depuis le dashboard.

Enfin, nous avons amélioré le dashboard avec des visualisations supplémentaires pour rendre l'analyse plus intuitive.

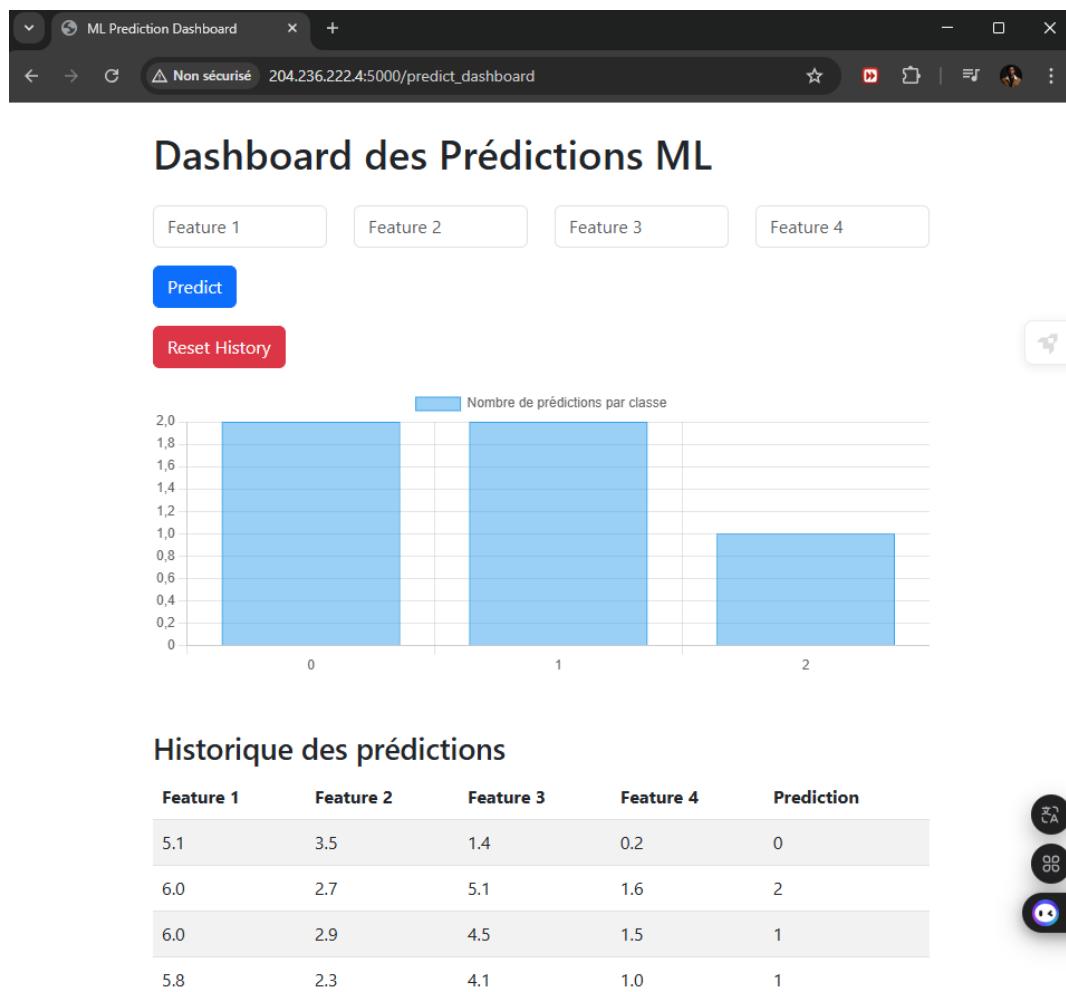


FIGURE 3.49 – Dashboard final amélioré avec visualisations et tests supplémentaires.

Chapitre 4

Conclusion générale et perspectives

4.1 Introduction

Dans ce dernier chapitre, nous récapitulons les objectifs atteints, les résultats obtenus et les apprentissages tirés du projet DevOps-MLOps. Nous proposons également des perspectives d'amélioration et des recommandations pour de futurs travaux.

4.2 Synthèse des résultats

Le projet a permis de mettre en place un pipeline DevOps et MLOps complet :

- Développement d'une API Flask pour exposer le modèle ML.
- Création d'un dashboard interactif pour tester les prédictions.
- Conteneurisation avec Docker pour garantir la portabilité.
- Déploiement sur AWS EC2 et intégration d'un pipeline CI/CD via GitHub Actions.

4.3 Perspectives et améliorations

Pour aller plus loin, plusieurs axes peuvent être envisagés :

- Ajout d'un système de monitoring en temps réel des prédictions.
- Mise en place d'un pipeline de mise à jour automatique du modèle ML.
- Intégration d'outils de visualisation avancée pour le dashboard.
- Sécurisation renforcée de l'API et du serveur cloud.

4.4 Conclusion finale

Le projet a démontré l'importance de l'automatisation et de la reproductibilité dans le développement et le déploiement de modèles ML. Grâce à une approche DevOps-MLOps structurée, nous avons obtenu un système fonctionnel, maintenable et évolutif, tout en respectant les bonnes pratiques industrielles.