

Construire et Déployer une API de Prédiction de Sentiment sur Twitter avec TF-IDF et Régression Logistique

Introduction :

Cet article détaille pas à pas la création d'une API de prédiction des sentiments des tweets, ainsi que son déploiement sur Microsoft Azure en s'appuyant sur GitHub Actions et MLflow, dans le cadre d'un processus MLOps. Le projet repose sur l'utilisation de TF-IDF (Term Frequency-Inverse Document Frequency) pour l'extraction des caractéristiques textuelles et d'un modèle de régression logistique, choisi pour sa simplicité et sa portabilité. En effet, la régression logistique est une méthode efficace pour ce type de tâche, tout en étant légère et facile à déployer, ce qui est particulièrement important étant donné l'utilisation de l'offre Azure gratuite qui impose certaines limites de ressources. L'objectif de ce projet est de prédire les sentiments (positif ou négatif) des tweets de manière rapide et fiable, tout en offrant une solution facilement intégrable dans des environnements de production. Le projet intègre également un suivi avec MLflow, permettant d'assurer une gestion continue du modèle en production et l'enregistrement du modèle en local pour faciliter son traçage et sa réutilisation dans des contextes futurs.

I. Détail du modèle et approche

Dans le cadre de ce projet, l'objectif était de prédire le sentiment des tweets de manière efficace et rapide. J'ai exploré plusieurs approches, notamment des modèles avancés comme BERT et LSTM, mais j'ai privilégié la régression logistique. Ce choix s'explique par sa simplicité, sa rapidité d'entraînement et son efficacité, ce qui le rend particulièrement adapté à un premier déploiement d'API.

Bien que des architectures plus complexes, comme un LSTM couplé aux embeddings GloVe, puissent mieux capturer les structures linguistiques des tweets, elles demandent davantage de ressources et un temps d'inférence plus long. De même, BERT, bien que très performant, alourdit significativement le déploiement. Ainsi, la régression logistique constitue un bon compromis, offrant des performances solides avec un coût computationnel réduit.

Mesure	BERT	LSTM GloVe	LSTM Word2Vec	Logistic Regression
Test Accuracy	0.766	0.785	0.766	0.767
Test Loss	0.482	0.453	0.482	0.482
Train Accuracy	0.974	0.778	0.774	0.778
Train Loss	0.071	0.471	0.473	0.471
Validation Accuracy	0.816	0.788	0.788	0.770
Validation Loss	0.706	0.45	0.46	0.484

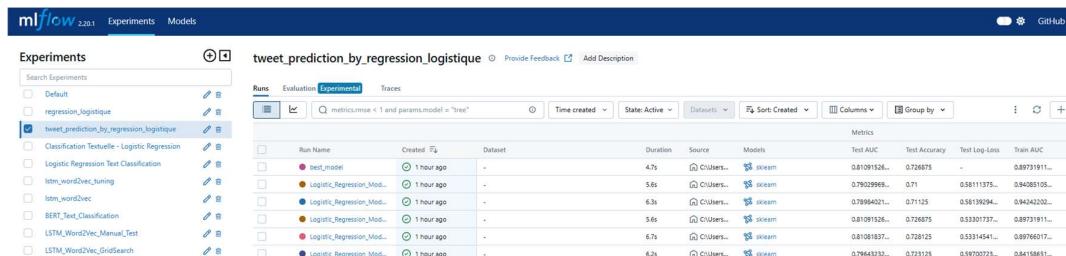
Le modèle de régression logistique atteint des résultats encourageants :

- ➡ **Test Accuracy : 76%**
- ➡ **Train Accuracy : 77%**
- ➡ **Validation Accuracy : 77 %**

Ces performances témoignent de la capacité du modèle à généraliser correctement, tout en restant optimisé pour une mise en production rapide.

Suivi avec MLflow et Enregistrement du Modèle en Local

MLflow permet de suivre les expériences d'entraînement de modèles, de stocker les résultats (métriques, modèles) et de visualiser les logs.



Run Name	Created	Dataset	Duration	Source	Models	Test AUC	Test Accuracy	Test Log-Loss	Train AUC
best_model	1 hour ago	-	4.7s	CI Users...	sklearn	0.81091326...	0.726875	-	0.89731911...
Logistic_Regression_M...	1 hour ago	-	5.6s	CI Users...	sklearn	0.79022996...	0.71	0.58111375...	0.84085105...
Logistic_Regression_M...	1 hour ago	-	6.3s	CI Users...	sklearn	0.79964021...	0.71125	0.58139094...	0.94242202...
Logistic_Regression_M...	1 hour ago	-	5.6s	CI Users...	sklearn	0.81091326...	0.726875	0.53301737...	0.89731911...
Logistic_Regression_M...	1 hour ago	-	6.7s	CI Users...	sklearn	0.81081837...	0.728125	0.53314541...	0.89766017...
Logistic_Regression_M...	1 hour ago	-	6.2s	CI Users...	sklearn	0.7964332...	0.723125	0.59700723...	0.84158681...

➤ Installation de MLflow en local

Tout d'abord, j'ai installé **MLflow** dans mon environnement local.

```
C:\Users\attia\P7_Réalisez_une_analyse_de_sentiments>myenv\scripts\activate  
(myenv) C:\Users\attia\P7_Réalisez_une_analyse_de_sentiments>pip install mlflow
```

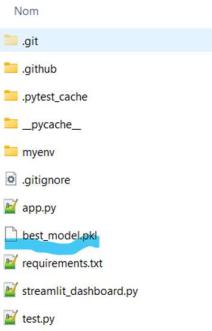
➤ Créer un dossier pour stocker les runs MLflow

Par défaut, MLflow stocke les informations des runs dans un répertoire **mlruns/** situé dans le répertoire courant du projet. Cependant, si on souhaite spécifier un autre emplacement pour enregistrer les résultats, on peut utiliser la méthode `mlflow.set_tracking_uri()` pour définir l'URI de suivi des expériences.

⚠ Note importante :

Les fichiers contenus dans **mlruns/** servent uniquement au suivi local des expériences. **Ils ne doivent pas être versionnés sur GitHub.**

Seul le modèle final (`best_model.pkl`) sera **versionné et déployé via GitHub Actions et Azure**.



➤ Enregistrer un modèle avec MLflow

✓ Lancement de l'expérience MLflow :

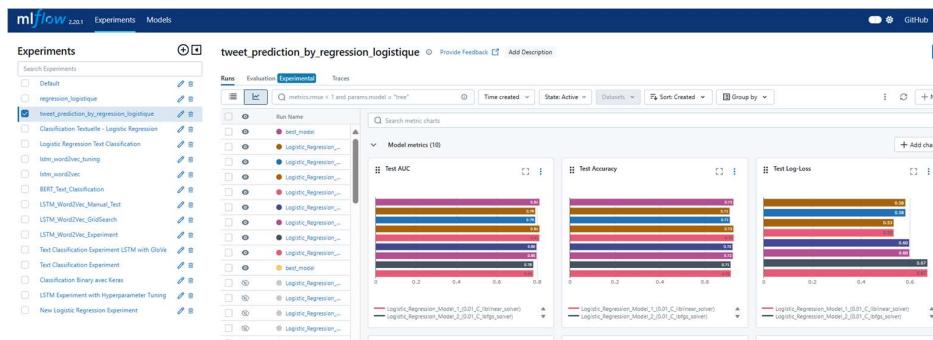
Nous ouvrons une session `mlflow.start_run()` pour capturer les hyperparamètres, métriques et modèles entraînés.

✓ Recherche du meilleur modèle avec GridSearchCV :

Nous testons différentes valeurs de `C` et `solver` pour `LogisticRegression` et enregistrons les **meilleurs hyperparamètres** obtenus.

✓ Suivi des métriques :

Nous enregistrons la précision du meilleur modèle (`best_accuracy`).



✓ Sauvegarde du modèle :

Le meilleur modèle est sérialisé en local avec `joblib.dump()` pour le déploiement.

✓ Enregistrement avec MLflow :

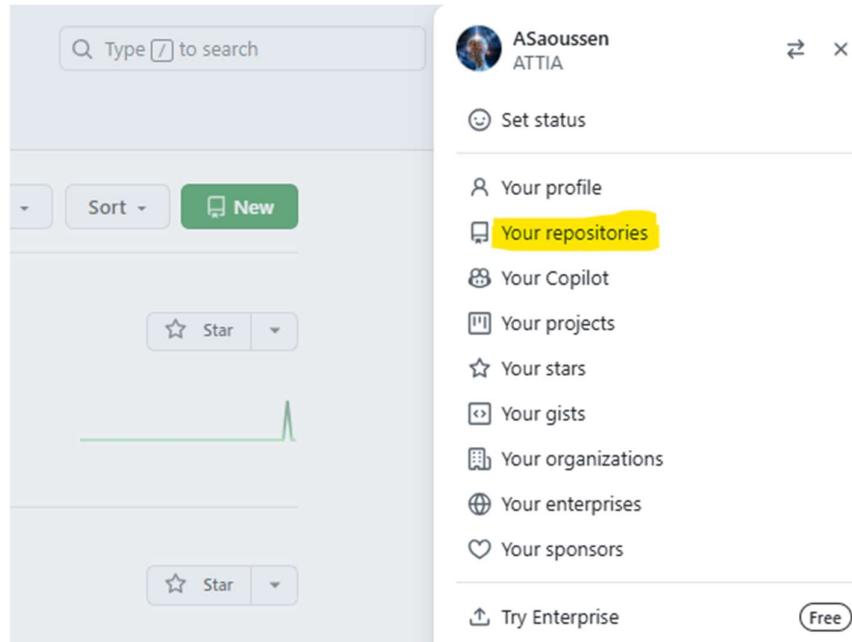
MLflow stocke le modèle dans un **format structuré** pour qu'il puisse être réutilisé facilement dans d'autres applications.

II. Création du repo sur github

La première étape consiste à créer un dépôt GitHub pour héberger le code source de l'API. Pour ce faire :

1. Connectez-vous à votre compte [GitHub](#).
2. Cliquez sur l'onglet "**Your repositories**".

3. Sélectionnez "**New**" pour créer un nouveau dépôt.
4. Renseignez un nom pertinent pour votre projet.
5. Ajoutez une brève description du projet.
6. Cochez l'option "**Add a README file**" pour initialiser le dépôt avec un fichier README.
7. Cliquez sur "**Create repository**" pour finaliser la création.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Required fields are marked with an asterisk (*).

Owner *	Repository name *
 A.Saoussen	/ P7_Réalisez_une_analyse_de_sentiments

Your new repository will be created as P7_Réalisez_une_analyse_de_sentiments-.
The repository name can only contain ASCII letters, digits, and the characters ., -, and _.

Great repository names are short and memorable. Need inspiration? How about [literate-engine](#) ?

Description (optional)
une description

 Public
Anyone on the internet can see this repository. You choose who can commit.

 Private
You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file
This is where you can write a long description for your project. [Learn more about READMEs](#).

Add .gitignore
.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license
License: None

A license tells others what they can and can't do with your code. [Learn more about licenses](#).

 You are creating a public repository in your personal account.

Create repository

III. Archivage du projet sur GitHub

Avant de procéder au déploiement, nous devons organiser et archiver notre projet sur [GitHub](#). Ce projet repose sur plusieurs fichiers et dossiers essentiels :

Structure du projet

Notre projet comprend les fichiers suivants :

- **app.py** : Contient le code source de l'API permettant la prédiction des sentiments des tweets.
- **best_model.pkl** : Fichier contenant le modèle de machine learning sérialisé, prêt à être utilisé pour l'inférence.
- **test.py** : Regroupe les tests unitaires pour valider le bon fonctionnement de l'API et du modèle.
- **requirements.txt** : Liste les dépendances nécessaires au bon fonctionnement du projet (Flask, Scikit-learn, etc.).

- **myenv/** : Dossier contenant l'environnement virtuel utilisé pour isoler les dépendances du projet.

.git	31/01/2025 16:46	Dossier de fichiers
.github	28/01/2025 19:39	Dossier de fichiers
.pytest_cache	29/01/2025 22:26	Dossier de fichiers
__pycache__	29/01/2025 22:33	Dossier de fichiers
myenv	30/01/2025 13:07	Dossier de fichiers
.gitignore	31/01/2025 16:41	Fichier source Git Ignore
app.py	31/01/2025 16:59	Fichier PY
best_model.pkl	23/01/2025 17:52	Fichier PKL
requirements.txt	29/01/2025 21:19	Fichier TXT
streamlit_dashboard.py	30/01/2025 23:57	Fichier PY
test.py	29/01/2025 22:33	Fichier PY

Étapes pour archiver le projet sur GitHub

- **Initialiser un dépôt Git localement**

Assurez-vous d'être dans le répertoire de votre projet, puis exécutez les commandes suivantes dans un terminal

```
C:\Users\attia\P7_Réalisez_une_analyse_de_sentiments>git init
```

- **Ajouter des fichiers au suivi Git**

```
C:\Users\attia\P7_Réalisez_une_analyse_de_sentiments>git add .
```

```
C:\Users\attia\P7_Réalisez_une_analyse_de_sentiments>git commit -m "Initial commit"
```

- **Lier le dépôt local au dépôt distant**

```
C:\Users\attia\P7_Réalisez_une_analyse_de_sentiments>git remote add origin https://github.com/ASauussen/P7_Réalisez_une_analyse_de_sentiments.git
```

- **Pousser les modifications**

```
C:\Users\attia\P7_Réalisez_une_analyse_de_sentiments>git push -u origin main
```

Remarque : Il est recommandé d'ajouter un fichier **.gitignore** pour éviter de versionner des fichiers inutiles comme l'environnement virtuel (myenv/) ou les fichiers temporaires. Vous pouvez créer un **.gitignore** avec le contenu suivant :

```

# Ignorer les répertoires spécifiques
./pytest_cache
/mlartifacts
/mlruns
/myenv
/path_to_save_model
/__pycache__/

#les fichiers
.gitignore
.gitignore.bak
mlflow.db
*.bak
streamlit_dashboard.py

```

Résultat dans github

 ASaussen	supression streamlitapp de github ✓	e5419e4 · 2 weeks ago	 63 Commits
 .github/workflows	Intégration des tests unitaires	2 weeks ago	
 .gitignore	supression streamlitapp de github	2 weeks ago	
 app.py	framework de validation de données	2 weeks ago	
 best_model.pkl	Ajout API	2 weeks ago	
 requirements.txt	Intégration du httpx	2 weeks ago	
 test.py	Modification tests	2 weeks ago	

Ce repository est accessible à l'adresse :

https://github.com/ASaussen/P7_Réalisez_une_analyse_de_sentiments

- **Mise à jour du dépôt GitHub :**

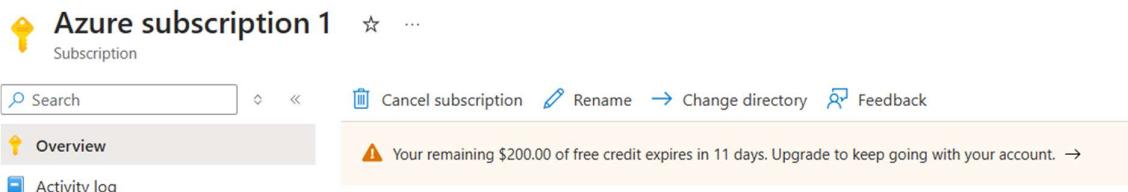
À chaque modification du projet, il est essentiel de mettre à jour le dépôt GitHub pour conserver un historique propre et assurer la collaboration. Pour cela, exécutez les étapes b et d.

 Remarque : Si plusieurs personnes travaillent sur le projet, il est recommandé d'exécuter git pull origin main avant d'envoyer vos modifications afin de synchroniser votre copie locale avec les mises à jour distantes.

IV. Création de l'application dans Azure

1. Activation de mon abonnement Azure

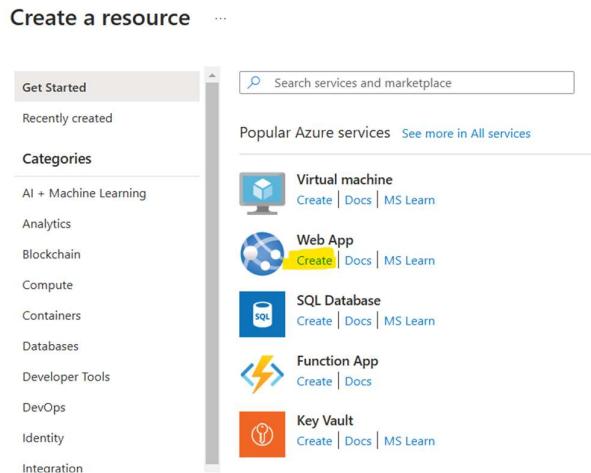
J'ai d'abord activé mon abonnement Azure gratuit, que j'ai obtenu via le programme d'essai d'un mois d'Azure.



The screenshot shows the 'Azure subscription 1' overview page. At the top, there's a search bar, a 'Cancel subscription' button, a 'Rename' button, a 'Change directory' button, and a 'Feedback' button. Below the header, there are two tabs: 'Overview' (which is selected) and 'Activity log'. A prominent yellow warning box at the bottom states: 'Your remaining \$200.00 of free credit expires in 11 days. Upgrade to keep going with your account.' with a link to upgrade.

2. Crédit de l'application Web (Azure App Service)

- **Accéder au portail Azure** : Je me suis connecté à mon compte sur le [portail Azure](#).
- **Créer une nouvelle ressource** : J'ai cliqué sur "Create a resource" en haut à gauche de mon tableau de bord Azure.
- **Choisir Web App** : Dans la barre de recherche, j'ai tapé "**Web App**" et sélectionné l'option correspondante.



The screenshot shows the 'Create a resource' page. On the left, there's a sidebar with 'Get Started' and 'Recently created' sections, followed by a 'Categories' section listing various Azure services like AI + Machine Learning, Analytics, Blockchain, Compute, Containers, Databases, Developer Tools, DevOps, Identity, and Integration. On the right, there's a search bar and a list of 'Popular Azure services' with their respective icons and 'Create' buttons. The 'Web App' service is highlighted with a yellow background and a green 'Create' button.

3. Configurer mon application

- **Abonnement** : J'ai choisi mon abonnement Azure gratuit.
- **Groupe de ressources** : J'ai créé un nouveau groupe de ressources appelé "AzureS1".
- **Nom de l'application** : J'ai sélectionné un nom unique pour mon application, tel que "pythonsentimentsapp".

all your resources.

Subscription * ⓘ

Azure subscription 1

Resource Group * ⓘ

AzureS1

Create new

Instance Details

Name

pythonsentimentsapp

.azurewebsites.net

- **Système d'exploitation** : J'ai choisi **Linux**, car c'est souvent recommandé pour un projet Python.
- **Runtime Stack** : J'ai opté pour **Python 3.12** (ou la version appropriée).
- **Région** : J'ai choisi la région "**West Europe**" pour l'application.
- Activer **l'authentification de base**

Authentication settings

Choose if you would like to allow basic authentication to deploy code to your app. [Learn more ↗](#)

Basic authentication

Disable Enable

○

- **Finalisation** : Après avoir vérifié les informations, j'ai cliqué sur "**Review + Create**", puis sur "**Create**". Cela a pris quelques minutes pour que mon application soit créée.

Une nouvelle application est créée.

 **pythonsentimentsapp** ⚡ ☆ ...

Web App

Search

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Browse Stop Swap Restart Delete Refresh

Essentials

Resource group ([move](#)) : [AzureS1](#)

Status : Running

Location ([move](#)) : West Europe

Subscription ([move](#)) : [Azure subscription 1](#)

V. Configurer le déploiement automatique avec GitHub Actions

- **Accéder au Deployment Center** : J'ai ouvert le **Deployment Center** sous l'application que j'avais créée.
- **Choisir GitHub comme source de déploiement** : J'ai sélectionné **GitHub** pour connecter mon dépôt. Je me suis connecté à mon compte GitHub et j'ai choisi le

dépôt où mon code était hébergé, tel que "**P7_Réalisez_une_analyse_de_sentiments**".

- **Configurer le déploiement** : J'ai sélectionné la branche à déployer. Après cela, j'ai cliqué sur "**Save**".

VI. Configurer Gunicorn et Uvicorn

Pour pouvoir démarrer l'API au chargement de l'application « **pythonsentimentsapp** », on doit utiliser la commande `gunicorn -k uvicorn.workers.UvicornWorker app:app` qui sert à l'exécuter en utilisant **Gunicorn** comme gestionnaire de processus et **Uvicorn** comme serveur ASGI pour gérer les requêtes de manière asynchrone.

Pour ce faire, aller dans « **Settings** », puis « **Configuration** » et modifier « **Startup Command** » comme suit :

Startup Command

```
gunicorn -k  
uvicorn.workers.UvicornWorker app:app
```

VII. V. Création du fichier .yml de déploiement

a. Initialisation du fichier.yml de déploiement

Un fichier de déploiement peut être généré automatiquement à partir de **GitHub Actions**, ce qui facilite grandement l'automatisation du processus.

Étapes pour générer le fichier de déploiement

1. Se rendre dans le **repository GitHub** du projet.
2. Cliquer sur l'onglet "**Actions**".
3. Sélectionner un **workflow proposé** parmi les suggestions.
4. GitHub va générer un fichier **azure-webapps-python.yml** dans le dossier `.github/workflows/`.
5. **Modifier et enregistrer** le fichier si nécessaire (dans mon cas, j'ai adapté certaines commandes pour une compatibilité avec l'OS Windows).

b. Création des étapes de déploiement

Le fichier `.yml` contient différentes **étapes essentielles** pour garantir un déploiement fluide :

- **Checkout du code** : Récupération des fichiers du repository.
- **Installation des dépendances** : Installation des packages nécessaires à l'application.
- **Configuration de l'environnement** : Définition des variables d'environnement requises.
- **Déploiement sur Azure** : Connexion et transfert du projet vers **Azure Web Apps**.
- **Vérification et tests** : Exécution de tests pour s'assurer que l'application fonctionne correctement après déploiement.

```
# Nom du workflow GitHub Actions
name: Build and Deploy Python App to Azure Web App

# Définition des variables d'environnement globales
env:
  AZURE_WEBAPP_NAME: pythonsentimentsapp # Nom de l'application Web sur Azure
  PYTHON_VERSION: '3.12' # Version de Python utilisée

# Déclenchement du workflow
on:
  push:
    branches: ["main"] # Exécuter ce workflow à chaque push sur la branche principale
  workflow_dispatch: # Permet de déclencher manuellement le workflow depuis GitHub

# Définition des permissions requises pour le workflow
permissions:
  id-token: write # Autorise l'utilisation d'un jeton OIDC pour l'authentification
  contents: read # Autorise la lecture des fichiers du repo
```

```

# Définition des tâches du workflow
jobs:
  build:
    runs-on: windows-latest # Spécifie que le job tourne sur un runner Windows

    steps:
      # Étape 1 : Cloner le dépôt GitHub
      - name: Checkout repository
        uses: actions/checkout@v4

      # Étape 2 : Installer Python et mettre en cache les dépendances
      - name: Set up Python
        uses: actions/setup-python@v3
        with:
          python-version: ${{ env.PYTHON_VERSION }} # Utiliser la version définie dans env
          cache: 'pip' # Mettre en cache les dépendances installées avec pip

      # Étape 3 : Créer et activer un environnement virtuel Python
      - name: Create and activate virtual environment
        run: |
          python -m venv myenv # Création de l'environnement virtuel
          myenv\Scripts\activate # Activation de l'environnement (Windows)

      # Étape 4 : Installer les dépendances du projet
      - name: Install dependencies
        run: pip install -r requirements.txt

      # Étape 5 : Exécuter les tests unitaires de l'API
      - name: Run API unit tests
        shell: cmd # Spécifie l'utilisation de l'invite de commandes Windows
        run: |
          pytest test.py --disable-warnings # Exécute les tests avec Pytest en supprimant les warnings

      # Étape 6 : Sauvegarder l'application sous forme d'artefact pour le déploiement
      - name: Upload artifact for deployment
        uses: actions/upload-artifact@v4
        with:
          name: python-app # Nom de l'artefact
          path: |
            . # Inclure tout le projet
            !venv/ # Exclure l'environnement virtuel pour réduire la taille de l'artefact

      # △ Étape en double : ce test est déjà exécuté plus haut
      - name: Run API unit tests
        shell: cmd
        run: |
          pytest test.py --disable-warnings

  deploy:
    runs-on: ubuntu-latest # Utilisation d'un runner Linux pour le déploiement
    needs: build # Exécuter ce job uniquement si le job "build" a réussi

    steps:
      # Étape 7 : Télécharger l'artefact généré lors du build
      - name: Download artifact
        uses: actions/download-artifact@v4
        with:
          name: python-app # Nom de l'artefact à récupérer
          path: . # Le fichier est récupéré dans le répertoire actuel

      # Étape 8 : Se connecter à Azure avec les identifiants stockés dans les secrets GitHub
      - name: Login to Azure
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }} # Clés d'authentification Azure

      # Étape 9 : Déployer l'application sur Azure Web App
      - name: Deploy to Azure Web App
        uses: azure/webapps-deploy@v2
        with:
          app-name: ${{ env.AZURE_WEBAPP_NAME }} # Utilisation du nom de l'application défini en variable
          slot-name: "production" # Déploiement sur l'environnement de production
          publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }} # Utilisation du profil de publication stocké dans les secrets GitHub

```

c. Initialisation des variables d'environnement

Pour que **GitHub Actions** puisse s'authentifier à **Azure**, il est nécessaire de configurer certaines **variables d'environnement** dans le repository GitHub.

i. Création de la variable AZURE_CREDENTIALS

Cette variable stocke les informations d'authentification Azure. Pour la générer, j'ai utilisé la **commande Azure CLI** suivante :

```
C:\Program Files\Microsoft SDKs\Azure\.NET SDK\v2.9>az ad sp create-for-rbac --name "streamlitsentimentsapp" --role contributor --scopes /subscriptions/[REDACTED]/resourceGroups/AzureS1 --json-auth
```

Le texte barré en rouge correspond à l'identifiant de l'abonnement.

Le résultat obtenu est un **JSON** contenant les informations d'authentification, qui doit être ajouté dans GitHub:

```
{  
    "clientId": "[REDACTED]",  
    "clientSecret": "[REDACTED]",  
    "subscriptionId": "[REDACTED]",  
    "tenantId": "[REDACTED]",  
    "activeDirectoryEndpointUrl": "https://login.microsoftonline.com",  
    "resourceManagerEndpointUrl": "https://management.azure.com/",  
    "activeDirectoryGraphResourceId": "https://graph.windows.net/",  
    "sqlManagementEndpointUrl": "https://management.core.windows.net:8443/",  
    "galleryEndpointUrl": "https://gallery.azure.com/",  
    "managementEndpointUrl": "https://management.core.windows.net/"  
}
```

1. Aller dans "**Settings**" du repository.
2. Sélectionner "**Secrets and variables**" → "Actions".
3. Cliquer sur "**New repository secret**".
4. Nommer la variable **AZURE_CREDENTIALS**

ii. Création de la variable AZURE_WEBAPP_PUBLISH_PROFILE

Cette variable permet de déployer l'application sur **Azure Web Apps**.

1. Ouvrir **Azure Portal** et accéder à l'application Web créée.
2. Cliquer sur le bouton "**Download publish profile**".
3. Copier le contenu du fichier téléchargé.
4. Dans **GitHub**, aller dans "**Settings**" → "**Secrets and variables**" → "Actions".
5. Créer une nouvelle variable **AZURE_WEBAPP_PUBLISH_PROFILE** et y coller le contenu copié.

Repository secrets		New repository secret
Name	Last updated	
AZURE_CLIENT_ID	3 weeks ago	
AZURE_CLIENT_SECRET	3 weeks ago	
AZURE_CREDENTIALS	3 weeks ago	
AZURE_SUBSCRIPTION_ID	3 weeks ago	
AZURE_TENANT_ID	3 weeks ago	
AZURE_WEBAPP_PUBLISH_PROFILE	3 weeks ago	

🚀 Une fois ces étapes complétées, l'application est prête pour un déploiement automatisé via **GitHub Actions** !

VIII. Déploiement

A chaque archivage d'une modification, le pipeline de déploiement github Actions se déclenche automatiquement exécutant les étapes de build et de déploiement.

← Build and Deploy Python App to Azure Web App
✓ suppression streamlitapp de github #62

The screenshot shows the GitHub Actions interface. On the left, there's a sidebar with links: 'Summary', 'Jobs' (with 'build' and 'deploy' listed), 'Run details', 'Usage', and 'Workflow file'. The main area is titled 'build' and shows a successful run from two weeks ago. It lists the following steps:

- > ✓ Set up job
- > ✓ Checkout repository
- > ✓ Set up Python
- > ✓ Create and activate virtual environment
- > ✓ Install dependencies
- > ✓ Run API unit tests
- > ✓ Upload artifact for deployment
- > ✓ Run API unit tests
- > ✓ Post Set up Python
- > ✓ Post Checkout repository
- > ✓ Complete job

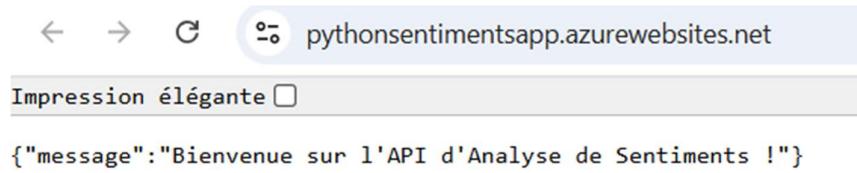
← Build and Deploy Python App to Azure Web App
✓ suppression streamlitapp de github #62

The screenshot shows the GitHub Actions interface. On the left, there's a sidebar with links: 'Summary', 'Jobs' (with 'build' and 'deploy' listed), 'Run details', 'Usage', and 'Workflow file'. The main area is titled 'deploy' and shows a successful run from two weeks ago. It lists the following steps:

- > ✓ Set up job
- > ✓ Pre Login to Azure
- > ✓ Download artifact
- > ✓ Login to Azure
- > ✓ Deploy to Azure Web App
- > ✓ Post Login to Azure
- > ✓ Complete job

Accès à l'API et tests

Pour accéder à mon api il faut aller à l'adresse :
<https://pythonsentimentsapp.azurewebsites.net/>



A screenshot of a web browser window. The address bar shows the URL "pythonsentimentsapp.azurewebsites.net". Below the address bar is a button labeled "Impression élégante". The main content area displays a JSON response: {"message": "Bienvenue sur l'API d'Analyse de Sentiments !"}.

Pour accéder à l'interface de test des méthodes de l'API, il faut taper
<https://pythonsentimentsapp.azurewebsites.net/docs>

FastAPI 0.1.0 OAS 3.1

</openapi.json>

default

GET	/ Root	▼
POST	/predict/ Predict	▼

Schemas

HTTPValidationError > <small>Expand all object</small>	^
InputData > <small>Expand all object</small>	
ValidationError > <small>Expand all object</small>	

Suivi de la performance de mon modèle en production avec Azure Application Insights et Streamlit

Dans le cadre de mon projet de prédiction de sentiment pour des tweets d'une compagnie aérienne, j'ai intégré Azure Application Insights pour suivre la performance de mon modèle en production. Cet outil m'a permis de collecter des données détaillées sur les performances, les erreurs et l'usage de mon API, tout en offrant des visualisations et alertes pour une compréhension en temps réel du comportement de l'application.

L'objectif était de suivre les performances du modèle en temps réel et d'avoir une vue d'ensemble sur son comportement. Grâce à Azure Application Insights, j'ai pu récupérer des logs et des données sur les événements générés par mon API ou application. Dans ce blog, je vais détailler comment j'ai intégré cet outil dans mon application Streamlit pour tester l'API localement avant de passer à la production.

1. Création d'un Compte Azure et Configuration de Application Insights

La première étape a été de créer un compte sur Azure, puis de configurer Application Insights dans le portail Azure. Voici les étapes principales que j'ai suivies :

- **Création d'une ressource Application Insights :**

- Dans le portail Azure, j'ai créé une nouvelle ressource **Application Insights** en sélectionnant le type "Application" et en choisissant la région la plus proche de mon serveur.

- **Obtenir la clé d'instrumentation :**

- Une fois la ressource créée, j'ai récupéré la **clé d'instrumentation**, qui me permet d'envoyer les données de performance de mon API vers **Application Insights**.

2. Préparation de l'environnement

Avant d'intégrer Azure Application Insights, j'ai préparé un environnement local pour tester l'application. J'ai utilisé Streamlit pour la partie front-end et une API locale pour effectuer les prédictions sur les sentiments des tweets. Voici les prérequis nécessaires pour cette intégration :

- Une Instrumentation Key pour Azure Application Insights.
- Le package opencensus installé pour exporter les logs vers Azure.

```

import streamlit as st
import requests
import logging
from opencensus.ext.azure.log_exporter import AzureLogHandler

```

- Une API servant à effectuer les prédictions de sentiment.

3. Configuration d'Azure Application Insights pour le logging

La première étape de l'intégration consistait à configurer un **logger** pour envoyer les informations de journalisation à **Azure**. J'ai utilisé la bibliothèque **opencensus** pour exporter les logs vers **Azure Application Insights**. Voici le code de configuration que j'ai utilisé dans mon application :

```

import streamlit as st
import requests
import logging
from opencensus.ext.azure.log_exporter import AzureLogHandler

# Remplacez par votre Instrumentation Key
instrumentation_key = "████████████████████████████████████████"

# Configuration du logger pour Application Insights
logger = logging.getLogger(__name__)
logger.addHandler(AzureLogHandler(connection_string=f'InstrumentationKey={instrumentation_key}'))
logger.setLevel(logging.INFO)

```

- Cette configuration m'a permis de journaliser les événements importants de mon application et de les envoyer à Azure Application Insights pour un suivi en temps réel.

2. Création de l'interface utilisateur avec Streamlit

Pour collecter les textes à analyser et afficher les résultats des prédictions, j'ai conçu une interface utilisateur simple avec **Streamlit**. L'utilisateur pouvait saisir du texte, obtenir la prédiction du sentiment, et recevoir des retours en temps réel. Voici un extrait du code de l'interface :

```

# Saisie du texte utilisateur
user_input = st.text_area("✍ Entrer le texte que vous souhaitez analyser")

if st.button("🔮 Prédire"):
    if not user_input.strip():
        st.error("✖ Veuillez entrer un texte valide avant de cliquer sur 'Prédire'.")
        logger.warning("Utilisateur n'a pas fourni de texte valide.")
    else:

```

- Dans cette partie, l'utilisateur peut entrer un texte, et si le texte est validé, il soumet sa demande pour obtenir une prédiction. Si un sentiment négatif est détecté, un log est envoyé à Azure Application Insights.

```

# Envoi des logs uniquement si le sentiment est négatif
if sentiment.lower() == "negative":
    logger.warning(f"⚠ Sentiment négatif détecté : {user_input}")

```

```
(myenv) C:\Users\attia\P7_Réalisez_une_analyse_de_sentiments>streamlit run streamlit_dashboard.py
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.1.23:8501
```

Analyse de Sentiments - Air Paradis

Entrez votre prénom

Entrez votre nom

Entrez le texte que vous souhaitez analyser

Prédire

"Vous pouvez trouver le code de mon API « app.py » et de mon interface streamlit « streamlit_dashboard » sur mon repository GitHub : [P7_Réalisez_une_analyse_de_sentiments](https://github.com/ASauussen/P7_Réalisez_une_analyse_de_sentiments) https://github.com/ASauussen/P7_Réalisez_une_analyse_de_sentiments

5. Paramétrage d'une Alerte basée sur les Traces

Afin de suivre de manière proactive les événements liés aux **sentiments négatifs**, j'ai configuré une **alerte** dans **Azure Application Insights** pour déclencher une notification dès que **trois "traces"** sont transmises dans un intervalle de **5 minutes**. Cette alerte permet de surveiller les anomalies en temps réel et d'agir rapidement si le nombre d'événements dépasse le seuil défini. Voici les étapes détaillées pour configurer cette alerte :

1. Accéder au portail Azure et à Application Insights :

- J'ai ouvert le **portail Azure** et accédé à la ressource **Application Insights** liée à mon application de prédiction de sentiment.

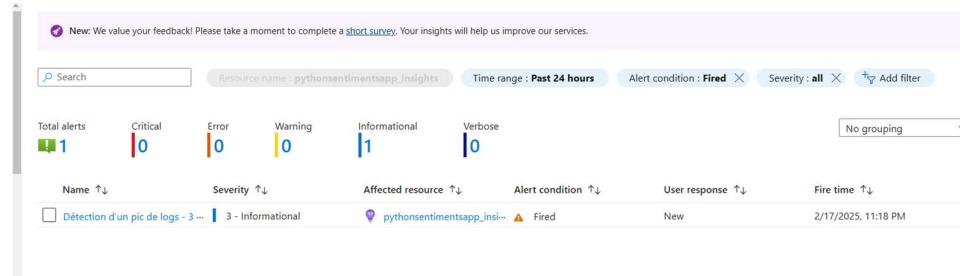
2. Création de l'alerte basée sur les traces :

- Sous la section "Alertes" dans **Application Insights**, j'ai créé une nouvelle règle d'alerte.

- Pour le type de **condition**, j'ai choisi **Traces** comme critère de déclenchement.

The screenshot shows the 'Create an alert rule' interface in Azure Sentinel. The 'Condition' tab is active. On the left, there's a sidebar with a search bar and a list of signals: Page views, Process CPU, Process IO rate, Process private bytes, Processor time, Receiving response time, Send request time, Server exceptions, Server request rate, Server requests, Server response time, Traces, and Activity log. The 'Traces' option is highlighted.

- J'ai défini une **condition personnalisée** pour que l'alerte se déclenche si le nombre de traces envoyées au service atteint **3** dans une fenêtre de **5 minutes**. Ce paramètre garantit que l'alerte sera activée uniquement lorsqu'un certain nombre d'événements significatifs, comme un sentiment négatif, se produisent dans une période courte.



○

3. Définition de l'action d'alerte :

J'ai configuré une **notification par e-mail** pour être alerté immédiatement en cas de dépassement du seuil de traces. Cette notification inclut des détails sur les traces et m'informe des événements qui pourraient nécessiter une action rapide (comme la modification du modèle de prédiction ou une vérification du système).

The screenshot shows the 'Actions' tab of the 'Create an alert rule' page. It displays a 'Use quick actions (preview)' section. In the 'Details' section, an action group name '10 Traces Négatives 5mn' and a display name 'Sentiments' are specified. In the 'Actions' section, an 'Email' action is selected, and the recipient email address 'alia.aoussem@gmail.com' is entered.

○

Une fois cette alerte configurée, j'ai pu garantir une surveillance en temps réel et intervenir immédiatement si un problème émergeait.

6. Collecte des logs et suivi des erreurs

Chaque fois qu'un sentiment négatif est détecté, un log d'alerte est envoyé à Azure

Application Insights, ce qui me permet de suivre ces événements dans le portail Azure. Ce processus est particulièrement utile pour identifier les moments où des sentiments négatifs apparaissent fréquemment et ajuster le modèle si nécessaire. Par exemple, voici le code pour envoyer un log en cas de sentiment négatif :

```
# Envoi des logs uniquement si le sentiment est négatif
if sentiment.lower() == "negative":
    logger.warning(f"⚠️ Sentiment négatif détecté : {user_input}")
```

The screenshot shows the Azure Application Insights interface. On the left, there's a sidebar with 'Investigate' selected, followed by 'Logs'. The main area has a search bar and a 'New Query 1*' button. Below it, a table displays log entries with columns: timestamp [UTC], message, severityLevel, itemType, and customDimensions. The table shows multiple entries from 2025-02-17 at 8:29:17.066 PM, each with a red warning icon and the message 'Sentiment négatif détecté : ugly' or 'Prédiction reçue : 0, Sentiment : Negative'. The 'severityLevel' column shows values like 2 (warning) and 1 (information). The 'itemType' column shows 'trace'. The 'customDimensions' column contains JSON objects like {"process": "MainProcess", "module": "streamlit"}. The table has 1 trace and 2 rows.

Ces logs permettent de garder une trace de la performance de l'API en production et d'intervenir si des anomalies sont détectées.

7. Visualisation et monitoring via Azure

Après avoir mis mon application en production, j'ai pu surveiller l'ensemble des **logs**, des **performances**, et des **erreurs** via le portail **Azure Application Insights**. J'ai pu observer plusieurs éléments clés, notamment :

- Le nombre de **sentiments négatifs** détectés.
- Les **erreurs de connexion à l'API**.
- Le **comportement de l'API** et les retours des utilisateurs en temps réel.

Ces données m'ont aidé à améliorer la qualité de mon modèle et à ajuster les prédictions en fonction des retours en production.

Conclusion

En intégrant Azure Application Insights à mon application Streamlit, j'ai pu suivre en temps réel la performance de mon modèle de prédiction de sentiment et obtenir des informations précieuses sur les retours utilisateurs. Ce processus m'a permis d'améliorer la robustesse de mon modèle, d'identifier rapidement les erreurs et de garantir une meilleure expérience utilisateur.

Si vous avez des questions ou des retours sur ce processus, n'hésitez pas à les partager dans les commentaires.