🧠 Challenge DevOps – Mets en ligne ton assistant IA sur une instance EC2

🎯 Objectif

Tu vas **déployer en ligne ton assistant IA** (ou un projet IA/ML de ton choix) en utilisant une **instance EC2 d'AWS**, accessible publiquement via une URL.
L'objectif est de découvrir **comment passer d'un prototype local à une application déployée en production**.

📚 Contexte

Tu as déjà développé des projets IA localement (chatbot, assistant de lecture de document, etc.).
Mais en entreprise, un projet n'a d'utilité que **s'il est accessible par d'autres** : collègues, clients ou intégrations externes.
Ce challenge t'apprend les bases de la **mise en production** via un **serveur cloud distant**.

🧱 Ce que tu dois construire

1. Une **instance EC2** sous Ubuntu (gratuite si t2.micro)
2. Une **connexion SSH fonctionnelle** depuis ton ordinateur
3. Un environnement propre : installation de `Python`, `pip`, `virtualenv`, `git`, etc.
4. Le **code de ton assistant IA cloné sur le serveur**
5. Un serveur API en ligne, par exemple avec **FastAPI + Uvicorn**, qui expose un endpoint comme `/ask`
6. Le **pare-feu AWS configuré** pour ouvrir le port de ton app (8000 ou 443/80 si reverse proxy)

✅ Critères de validation

- ✔️ Ton assistant est accessible via une URL publique (IP ou domaine)
- ✔️ Le serveur est stable et tourne même après déconnexion SSH (via `screen`, `tmux`, ou `systemd`)
- ✔️ Un utilisateur externe peut poser une question via un appel à l'API (ex : Postman, curl ou site web)
- ✔️ Le README décrit les étapes pour reproduire le déploiement

---

Bonus 1

🔐 Sécurité – Protéger l'accès avec un token

Pour éviter que n'importe qui n'utilise ton API, tu vas ajouter un **système d'authentification par token**. Cela signifie que chaque requête devra inclure un **token secret** (type Bearer Token) pour être acceptée.

🧱 Ce que tu dois faire

1. Créer un fichier `.env` contenant un token secret (`API_TOKEN=...`)
2. Ajouter une vérification dans ton code : si le token n'est pas présent ou incorrect, la requête est rejetée
3. Intégrer cette vérification sur toutes les routes sensibles (`/ask`, etc.)
4. Documenter dans ton README comment utiliser ce token (via `curl`, Postman, ou un client React)

✅ Résultat attendu

- Si le token est correct → la requête est acceptée

- Si le token est absent ou invalide → la requête est bloquée avec un message d'erreur
- Ton API est **sécurisée** : seul quelqu'un ayant le bon token peut l'utiliser

---

Bonus 2

🐳 Docker – Emballe ton API dans un conteneur

Déployer ton application avec Docker te permet d'avoir un environnement **propre, reproductible et portable**. Tu vas créer une **image Docker** qui contient tout ton assistant IA, et la lancer dans un conteneur sur ton EC2.

🧱 Ce que tu dois faire

1. Créer un fichier `Dockerfile` qui :
    - installe les dépendances
    - copie ton code
    - lance ton API (ex: avec `uvicorn`)
2. (Optionnel mais recommandé) Créer un `docker-compose.yml` pour :
    - lancer ton app facilement
    - gérer plusieurs services (ex: API + base vectorielle locale)
3. Sur ton EC2 :
    - installer Docker et Docker Compose
    - cloner ton dépôt

- lancer l'application avec `docker build` puis `docker run`, ou avec `docker-compose up`

✅ Résultat attendu

- Ton assistant IA tourne dans un **conteneur isolé**
- Tu peux le redémarrer en une ligne (`docker-compose up -d`)
- Il est **reproductible** sur n'importe quel serveur

🧠 Avantages

- Aucune dépendance à installer sur l'EC2 (tout est dans le conteneur)
- Plus facile à maintenir, déboguer, migrer
- Préparation à des workflows DevOps pro (CI/CD, déploiement automatisé…)

🚀 Bonus

- Ajouter un **volume Docker** pour stocker tes données (logs, embeddings…)
- Utiliser `restart: always` dans `docker-compose.yml` pour garder l'app active même après un redémarrage
- Pousser ton image sur Docker Hub ou GitHub Container Registry