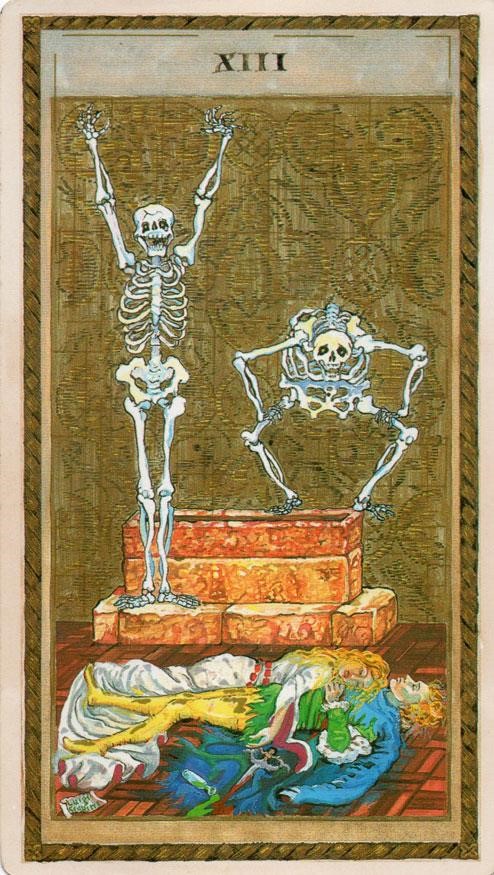
## Menu

# TP fin de formationAdministrateur systèmes DevOps



No non c’est pas la mort…

1. [**Introduction**](#_Introduction)
2. [**Cahier des charges**](#_Cahier_des_charges)

• Objectif

• Fonctionnalités principales

• Technologies utilisées

• Exigences techniques

• Architecture de déploiement sur AWS avec Amazon EKS

• Gestion du code source avec GitHub

• Contraintes de développement

• Livrables attendus

• Calendrier de développement

1. [**Installation**](#_Installation)
   * Configuration de l'environnement de développement
   * Installation des dépendances
   * Configuration de la base de données
   * Installation Cluster EKS et toutes les dépendances AWS
   * Mise en place d’ArgoCD, afin de déployer toute l’infrastructure au sein du cluster
   * Setup des namespaces et systèmes de fichiers
2. [**Architecture du Projet**](#_Architecture_du_projet)
   * Vue d'ensemble de l'architecture
   * Explication des composants principaux
3. [**Fonctionnalités Principales**](#_Fonctionnalités_principales)
   * Authentification utilisateur
   * Gestion des utilisateurs
   * Interface utilisateur
   * Intégration de la base de données PostgreSQL
4. [**Conclusion**](#_Conclusion)
   * Résumé du projet
   * Perspectives d'amélioration

## [Introduction](#_Menu)

Cette documentation accompagne le projet de fin de TP et vise à présenter en détail le respect du cahier des charges fourni. Elle fournit des informations sur l'installation du projet, son architecture, ses fonctionnalités principales, ainsi que des captures d'écran pour illustrer le processus.

L’application et les fichiers de déploiement de l'infrastructure sont hébergés sur deux dépôts GitHub distincts :

1. Pour l'application et l'infrastructure interne au cluster à déployer :
   * [Dépôt pour l'infrastructure et l'application](https://github.com/NunoMars/tp)
2. Pour l'infrastructure AWS et la documentation associée :
   * [Dépôt pour l'infrastructure AWS et la documentation](https://github.com/NunoMars/tp-infra)

L’application possède deux environnements distincts :

* Dev pour le développement.
* Prod pour la production.

## [Cahier des charges](#_Menu)

**Projet** :

Développement et Déploiement d'une Application Web avec Django

**Objectif :**

Créer, développer et déployer une application web utilisant le framework Django en Python avec une base de données PostgreSQL hébergée sur AWS, en utilisant Amazon EKS (Elastic Kubernetes Service) pour l'orchestration des conteneurs, tout en intégrant la gestion du code source via un repository GitHub et le déploiement avec Terraform.

**Fonctionnalités principales :**

1. Authentification utilisateur (inscription, connexion, déconnexion)
2. Gestion des utilisateurs (profil utilisateur, modification du mot de passe)
3. Interface utilisateur conviviale
4. Intégration de la base de données PostgreSQL pour stocker et gérer les données de l'application

**Technologies utilisées :**

* Python
* Django
* PostgreSQL
* AWS (Amazon Web Services, EC2, S3, IAM, EKS)
* Prometheus, Alertmanager, Grafana, Jeager
* Docker
* GitHub
* GitHub Actions
* Terraform
* ArgoCD
* Kubernetes

**Exigences techniques :**

1. Développement de l'application en utilisant Django pour le back-end et HTML/CSS/JavaScript pour le front-end.
2. Utilisation de PostgreSQL comme système de gestion de base de données.
3. Utilisation d'Amazon EKS pour l'orchestration des conteneurs.
4. Intégration de la gestion du code source via un repository GitHub pour le suivi des modifications, la collaboration et le déploiement automatisé.
5. Utilisation de Docker pour la gestion des conteneurs et l'assurance de la portabilité.
6. Utilisation de Terraform pour la configuration et le déploiement de l'infrastructure AWS.

**Architecture de déploiement sur AWS avec Amazon EKS :**

1. Configuration d'un cluster Amazon EKS pour héberger les conteneurs de l'application et de la base de données.
2. Déploiement de l'application Django dans des pods Amazon EKS.
3. Déploiement de PostgreSQL dans des pods Amazon EKS.
4. Configuration des services Amazon EKS pour l'accès à l'application et à la base de données.
5. Utilisation de secrets Kubernetes pour la gestion des informations sensibles telles que les clés d'API et les mots de passe.

**Gestion du code source avec GitHub :**

1. Création d'un repository GitHub pour héberger le code source de l'application.
2. Utilisation de branches Git pour le développement de fonctionnalités et la gestion des versions.
3. Utilisation de GitHub Actions pour automatiser les tests et le déploiement de l'application sur Amazon EKS.
4. Configuration des webhooks GitHub pour déclencher des actions spécifiques (tests, déploiement) lors de la validation du code.

**Déploiement de l'infrastructure AWS avec Terraform :**

1. Utilisation de Terraform pour déployer l'infrastructure requise pour Amazon EKS (VPC, sous-réseaux, groupes de sécurité, etc.).
2. Configuration des ressources AWS nécessaires pour le déploiement de l'application et de la base de données PostgreSQL.
3. Automatisation du processus de déploiement avec Terraform pour assurer la reproductibilité et la gestion efficace de l'infrastructure.

**Contraintes de développement :**

1. Respecter les bonnes pratiques de développement en Python et Django.
2. Assurer la compatibilité multi-navigateurs pour l'interface utilisateur.
3. Suivre les normes de sécurité recommandées pour la protection des données utilisateurs.
4. Configurer des sondes de santé et de préparation pour garantir la haute disponibilité de l'application.

**Livrables attendus :**

1. Configuration Amazon EKS pour le déploiement de l'application et de la base de données.
2. Code source de l'application hébergée sur GitHub avec une documentation détaillée sur son fonctionnement et son déploiement.
3. Configuration Terraform pour le déploiement de l'infrastructure AWS.
4. Accès aux ressources AWS, Amazon EKS et GitHub pour vérification et maintenance éventuelle.
5. Formation et support pour l'administration de l'application.

**Calendrier de développement :**

* Phase 1 : Conception et développement de l'application (4 semaines)
* Phase 2 : Configuration de l'environnement Amazon EKS et déploiement initial avec Terraform (2 semaines)
* Phase 3 : Tests, correction des bugs et optimisation (2 semaines)
* Phase 4 : Formation et remise du projet final (1 semaine)

## [Mise en place et Installation](#_Menu)

Pour garantir une mise en place efficace de notre application, nous suivons plusieurs étapes de mise en place et d'installation.

**Configuration de mon environnement :**

Avant de commencer, je m'assure d'avoir configuré mon environnement de développement avec les outils nécessaires tels que Python, Django, Docker, et les outils de développement AWS. Je m'assure également d'avoir accès à mon compte AWS pour configurer les services cloud.

**Je clone mon repo d’application :**

[Dépôt pour l'infrastructure et l'application](https://github.com/NunoMars/tp)

Je commence par créer un repo pour l’infrastructure à l’intérieur du cluster EKS, selon les spécificités de l’application, sur lequel, j’insère tous les Charts.

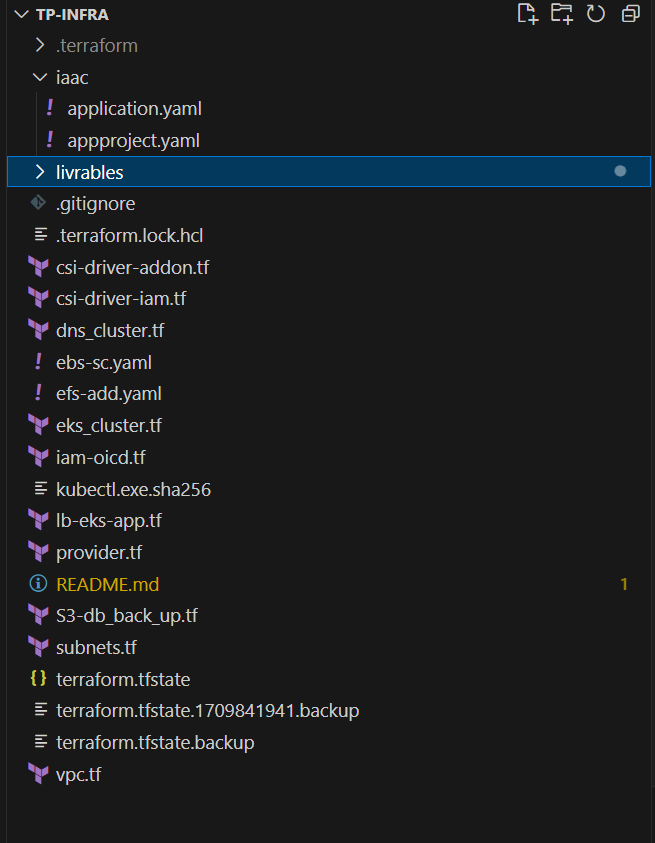
Ensuite s’agissant d’une application Django, je regarde le fichier settings.py afin de constater les variables et les spécificités de l’application.

Je télécharge et setup, AWS Cli, afin d’interagir directement avec aws sans l’interface graphique, ainsi que Terraform et Kubernetes avec Kubectl.

**Je mets en place tous les fichiers Terraform au sein d’un autre repo Github :**

[Dépôt pour l'infrastructure AWS et la documentation](https://github.com/NunoMars/tp-infra)

Je mets en place tous les scripts Terraform afin de constituer mon infra sous aws.

* Les providers, Aws, Terraform et Acme (pour le certificat SSL/TLS)
* La création Du cluster EKS avec ses droits
* La création d’un loadbalancer pour le cluster
* La route DNS avec route53
* Les node-groups
* Les vpc et subnets
* Les roles Aim et Policys pour la securité
* Le script pour le driver CSIEt deux scripts Kubernettes afin d’insérer deux Storageclass efs et ebs

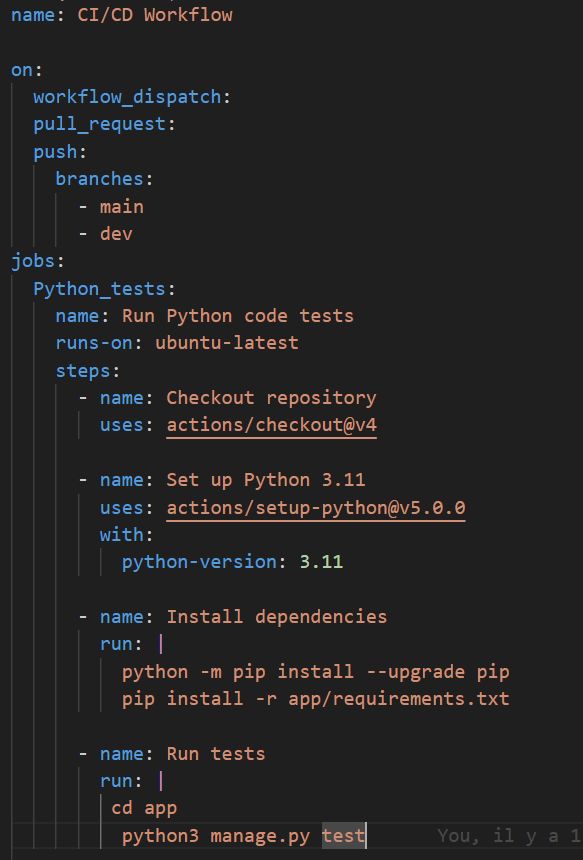
**Installation des dépendances :**

Je commence par installer toutes les dépendances requises pour mon application Django en utilisant pip ou tout autre gestionnaire de paquets Python. Je m'assure d'installer les versions spécifiques des bibliothèques requises pour éviter tout conflit ou incompatibilité.

Ensuite je mets en place le Script de CI/CD sous GuitHub, Sur lequel, je mets les étapes de la chaine CI/CD.

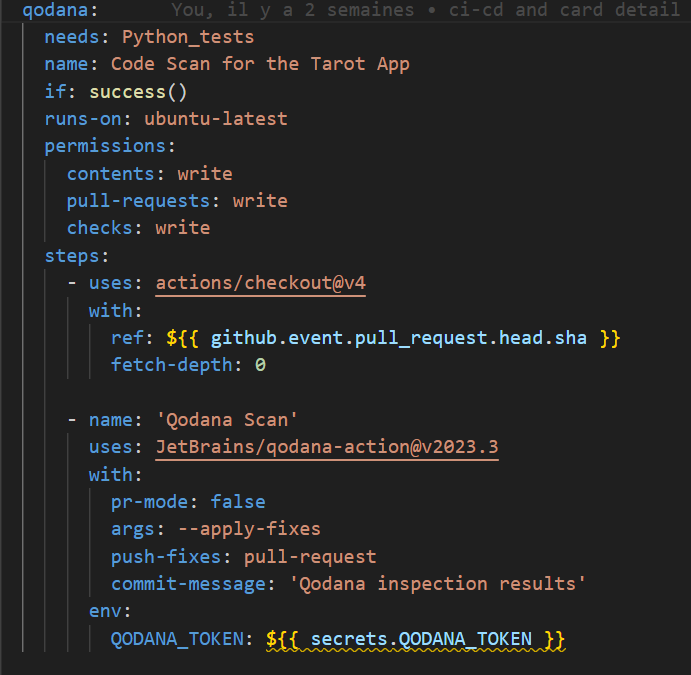
J’ai aussi mis en place des secrets et variables sur le dépôt pour garantir une « cofidentialité » afin de pouvoir interagir avec des dependances.

1. **Les tests de l’application mis en place par les Développeurs, (donc moi self), pratique quand je veux by-pass.. non je blague!**



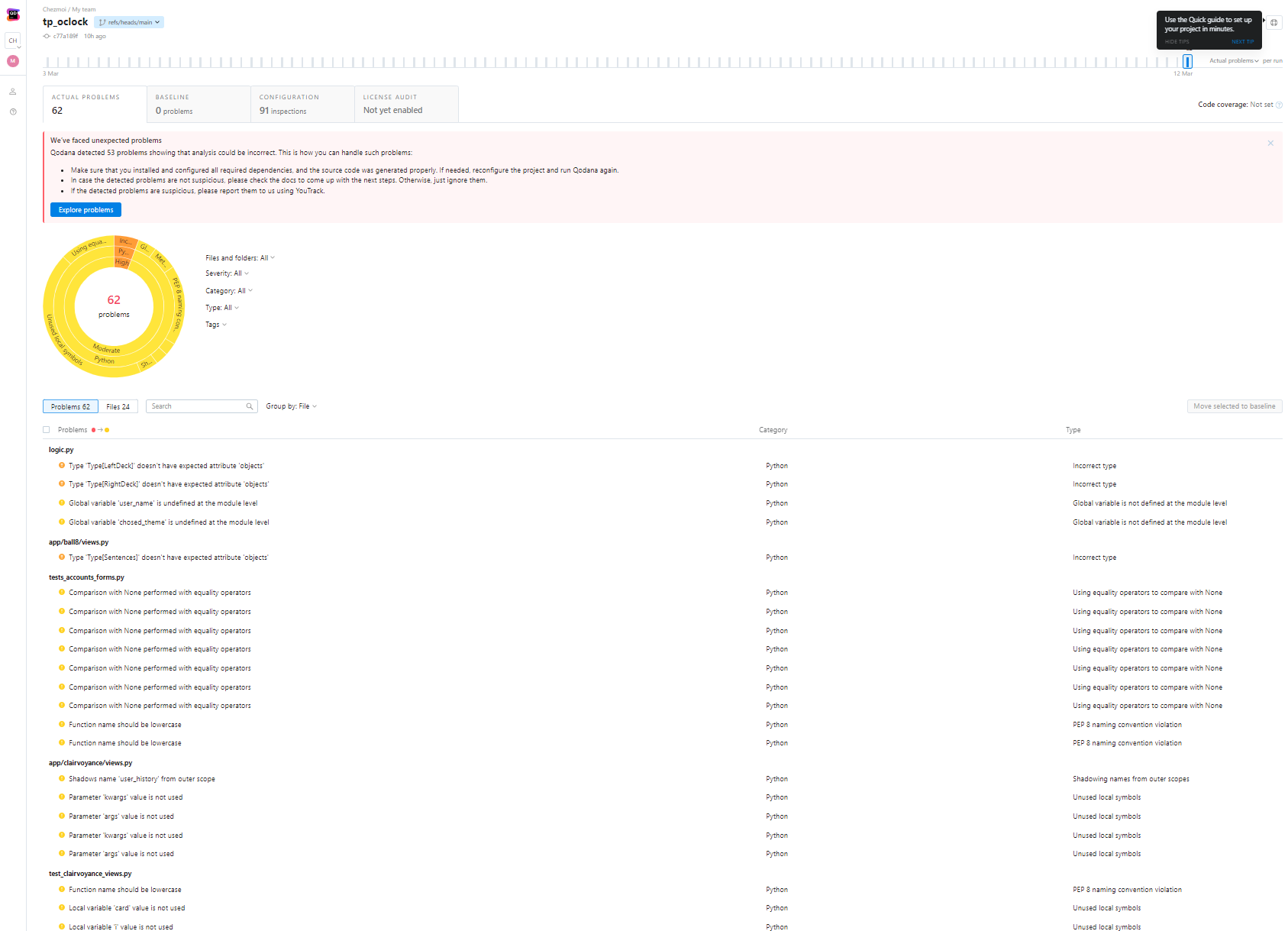
Sur cette étape on installe et on lance tous les tests mis en place, si ok on continue la chaine CI/CD si Fail, la chaine s’arrête ! J’ai mis en place un badge sur le Readme du dépôt afin d’avoir une visibilité supplémentaire sur l’état de la CI/CD.

1. **Le code qualité :**

****

J’ai décidé d’intégrer un outil de qualité de code, car un code propre et sans bugs de versions etc, c’est toujours mieux non… ?

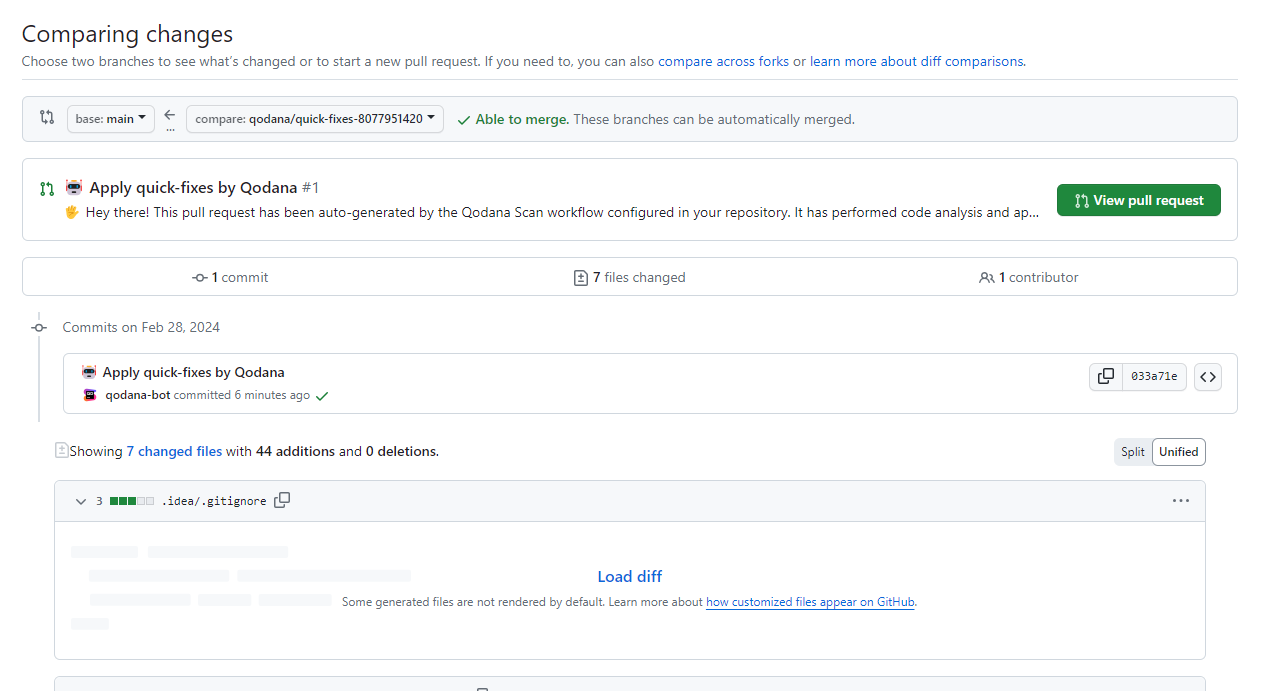
J’ai pris Quadana, il suggère et propose des pull-requests avec les améliorations directement, et envoi un rapport sous la fenêtre de la Ci/CD ainsi que sur le site.



Comme on peut vérifier, je suis un super développeur…

J’ai volontairement laissé les correctifs afin d’avoir quelque chose à l’écran, (faut bien sauver la face hein ? ).

9a donne une visibilité total suer le code Python et propose des corrections, améliorations de code, il peut aussi le faire via un IA et demander une « pull-request » de la branche.

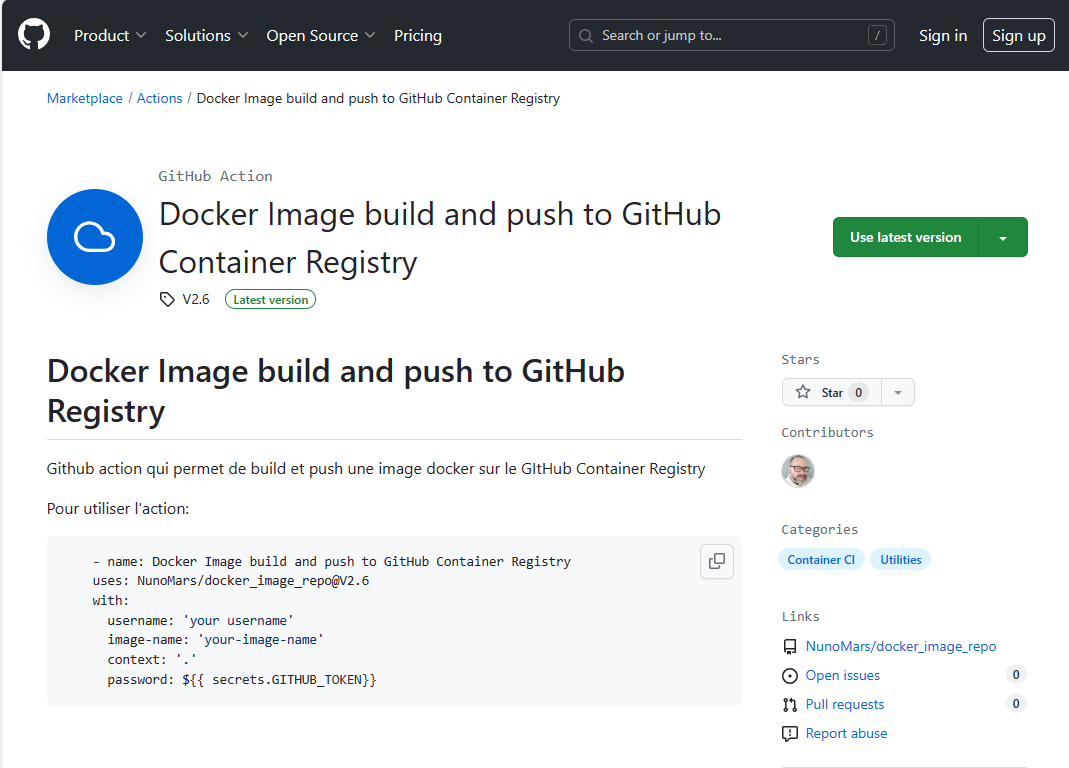


Je n’ai pas autorisé le « Pull-request » automatique car, je suis méfiant et par expérience des IA, je sais que ça peut vite dégénérer, et ça ne conviens pas toujours à l’attendu.

Si Quodana détecte une erreur importante il envoi Fail et la CI/CD s’arrête. Cas contraire on passe à la suite.

1. **La Build and Push l’image :**

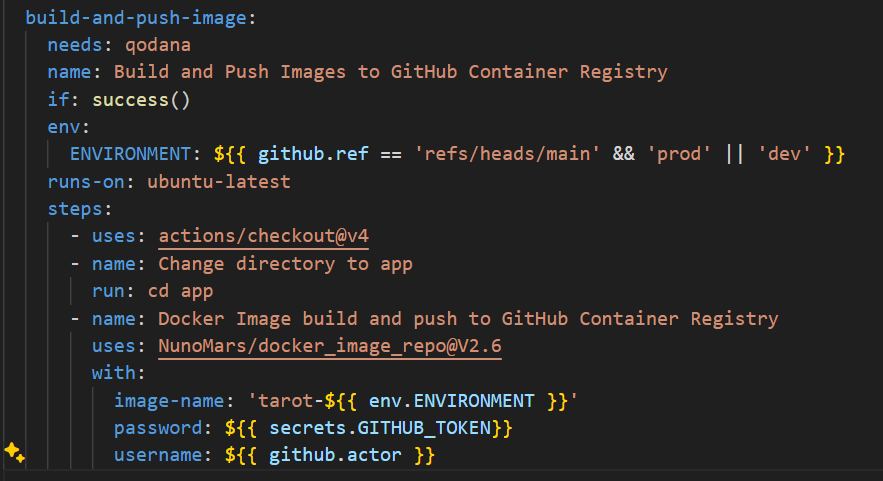
Avant d’en arriver là, on doit bien sur valider les deux étapes précédentes, et afin de faciliter la lecture du fichier de CI/CD j’ai codé une action personnalisé afin de builder et push l’image suer le registre de Github.



Le code de l’action :



Sur la CI/CD ça donne ça :



**Configuration de la base de données :**

Après avoir installé les dépendances, je configure la base de données PostgreSQL en créant une nouvelle base de données et en configurant les paramètres de connexion dans les fichiers de configuration Django.

J’ai donc Mis une base de données différente selon l’environnement en local c’est de l’Splitte et en Production et Développement ce sera donc PostgreSQL installé via Argocd que je vais détailler plus tard sur ce document.

**Installation du Cluster EKS et toutes les dépendances AWS :**

Je suis les instructions fournies par AWS pour créer et configurer mon cluster Amazon EKS. Cela implique de créer un cluster EKS, de configurer les autorisations IAM appropriées, de définir les rôles et les politiques nécessaires, ainsi que de provisionner les ressources supplémentaires requises telles que les instances EC2, les groupes de sécurité, etc.

**Mise en place d’ArgoCD :**

Une fois le cluster EKS configuré, j'installe et configure ArgoCD pour la gestion et le déploiement automatisé de l'infrastructure au sein du cluster. ArgoCD me permettra de définir et de suivre les états des ressources Kubernetes déployées dans le cluster.

**Setup des namespaces et systèmes de fichiers :**

J'organise mon cluster EKS en créant des namespaces pour isoler les différentes parties de mon application. Je définis également les systèmes de fichiers appropriés pour le stockage persistant des données et des médias de mon application.

## [Architecture du projet](#_Menu)

Dans cette partie, l'architecture globale du projet est présentée, avec une explication détaillée de ses différents composants et de leur interaction.

## [Fonctionnalités principales](#_Menu)

Cette section décrit en détail les principales fonctionnalités de l'application, telles que l'authentification utilisateur, la gestion des utilisateurs, l'interface utilisateur conviviale, et l'intégration de la base de données PostgreSQL..

## [Conclusion](#_Menu)

Enfin, la section de conclusion résume les points clés du projet, met en évidence ses succès et ses réalisations, et propose des pistes pour de futures améliorations et développements.