# Documentation de la Chaîne d'Intégration Continue Django CI

Cette documentation détaille la procédure d'installation, de configuration et de test de la chaîne d'intégration continue pour un projet Django, en utilisant GitHub Actions.

## Pré-requis

* Un dépôt GitHub contenant un projet Django.
* Un fichier requirements.txt à la racine du projet listant toutes les dépendances nécessaires.
* Secrets GitHub configurés pour les clés et identifiants nécessaires (Dans notre cas :PINEKEY, OPENKEY, CODECOV\_TOKEN, DOCKER\_USERNAME, DOCKER\_PASSWORD, AZURE\_CREDENTIALS).
* Un compte Docker Hub.
* Un compte Azure avec un groupe de ressources pour créer et gérer des instances de conteneurs.

### Configuration des Secrets GitHub

Dans le dépôt GitHub, aller dans Settings > Secrets and variables > Actions, puis ajouter les secrets suivants :

* PINEKEY
* OPENKEY
* CODECOV\_TOKEN
* DOCKER\_USERNAME
* DOCKER\_PASSWORD
* AZURE\_CREDENTIALS
* AZURE\_CONTAINER\_REGISTRY

## Pipeline Ci/Cd

### Configuration du dépôt GitHub

Création du fichier de workflow: création du fichier .github/workflows/django-ci.yml dans le dépôt GitHub avec un pipeline qui sera chargé de :

Configuration de Python:

* + - Installation de Python 3.11.5.
    - Installation des dépendances listées dans requirements.txt.

Exécution des tests:

* + - Migration de la base de données avec python manage.py migrate.
    - Exécution des tests avec pytest.

Upload de la couverture de code:

* + - Les résultats de couverture de code sont envoyés à Codecov.

### Déploiement

Si les tests passent, les étapes de déploiement suivantes sont exécutées :

Connexion à Docker Hub:

* + - Connexion en utilisant les identifiants fournis dans les secrets GitHub.

Construction et Push de l'image Docker:

* + - Construction de l'image Docker avec le tag spécifié.
    - Push de l'image vers Docker Hub.

Déploiement sur Azure:

* + - Suppression de l'instance de conteneur existante si elle existe.
    - Attente de la suppression complète de l'instance.
    - Création d'une nouvelle instance de conteneur avec l'image Docker mise à jour et les variables d'environnement nécessaires.

### Tests

Lorsque vous poussez du code ou créez une pull request, le pipeline CI/CD est exécuté.

Voici le détaille des tests :

#### TestViews

* Configuration (setUp)
  + Initialise le client de test Django.
  + Définit les URLs pour les différentes vues.
  + Crée un utilisateur de test.
* Test de la vue de connexion (test\_login\_view)
  + Vérifie que la vue de connexion retourne un statut 200 et utilise le template login.html.
* Test de la vue de la page d'accueil (test\_home\_view)
  + Vérifie que la vue de la page d'accueil retourne un statut 200 et utilise le template home.html lorsque l'utilisateur est authentifié.
* Test de la vue de création de conversation (test\_create\_conversation\_view)
  + Vérifie que la vue de création de conversation redirige correctement après avoir créé une nouvelle conversation.
* Test de la vue de détail de conversation (test\_conversation\_detail\_view)
  + Vérifie que la vue de détail de conversation retourne un statut 200 et utilise le template conversations/conversation\_detail.html.

#### TestUrls

* Test de l'URL de connexion (test\_login\_url)
  + Vérifie que l'URL de connexion (/login/) est correctement résolue vers la vue LoginView de Django.
* Test de l'URL de la page d'accueil (test\_home\_url)
  + Vérifie que l'URL de la page d'accueil (/home/) est correctement résolue vers la vue home de l'application conversations.
* Test de l'URL de création de conversation (test\_create\_conversation\_url)
  + Vérifie que l'URL pour créer une nouvelle conversation (/create\_conversation/) est correctement résolue vers la vue create\_conversation.
* Test de l'URL de détail de conversation (test\_conversation\_detail\_url)
  + Vérifie que l'URL de détail de conversation, avec un UUID valide, est correctement résolue vers la vue conversation\_detail.