GITEA

## Berrebi David-Hai

**Sommaire:**

1. **Introduction**
   * Présentation du projet Gitea et de son objectif.
   * Explication de l'installation de Gitea en tant qu'alternative à GitHub sur un serveur local, utilisant Docker pour la gestion des conteneurs.
2. **Configuration Docker Compose**
   * Détails des services configurés pour les environnements de développement, de production et pour Gitea.
   * Description des réseaux Docker utilisés pour connecter les différents services.
3. **Configuration des Serveurs**
   * Explication des Dockerfiles utilisés pour créer les images des serveurs de développement et de production.
   * Détail des étapes de configuration pour assurer la stabilité et la fonctionnalité des serveurs.
4. **Configuration de Gitea**
   * Processus de création d'utilisateurs administrateurs, ainsi que des utilisateurs pour les environnements de développement et de production.
   * Étape par étape pour la création d'un dépôt sur le compte de développement.
5. **Correction de l'URL Gitea**
   * Identification du problème d'URL incorrecte dans Git.
   * Méthodes utilisées pour résoudre ce problème, notamment l'utilisation de nmap pour trouver l'URL par défaut et la modification du fichier de configuration Git.
6. **Ajout du Projet sur Gitea**
   * Procédure détaillée pour ajouter le projet de développement à Gitea, y compris l'initialisation de Git, l'ajout et le commit des fichiers, et le push vers Gitea.
7. **Clonage du Projet sur le Serveur de Production**
   * Étapes pour cloner le projet depuis Gitea sur le serveur de production, assurant ainsi la cohérence entre les environnements de développement et de production.

## Objectif du project

**Objectif du Projet :** Le projet consistait à installer Gitea, une alternative à GitHub, sur un serveur. L'objectif principal était de relier notre serveur de production (prod\_) à notre serveur de développement (dev\_server) en passant par le serveur Gitea (gitea\_serv), le tout en local et utilisant Docker.

**Configuration Docker Compose :**

Un fichier docker-compose.yaml a été utilisé pour configurer et orchestrer les différents conteneurs nécessaires pour les environnements de développement, de production et pour Gitea. Voici les détails de cette configuration :

### Services Développement

* **dev-server :**
  + Construction à partir d'un Dockerfile local.
  + Montages de volumes pour le code et les configurations Caddy.
  + Ports exposés : 8080 pour HTTP, 8003 pour HTTPS.
  + Réseaux : gitea-network, dev-network.
* **dev-db :**
  + Image : MySQL 8.0.
  + Commande de démarrage spécifique pour des modes SQL stricts.
  + Variables d'environnement pour la base de données.
  + Volume monté pour les données MySQL.
  + Réseaux : gitea-network, dev-network.
* **dev-pma :**
  + Image : phpMyAdmin.
  + Dépendance sur dev-db.
  + Port exposé : 8081.
  + Variables d'environnement pour la connexion à la base de données.
  + Réseau : dev-network.

### Services Gitea

* **gitea :**
  + Image : Gitea 1.22.0.
  + Variables d'environnement pour la configuration de la base de données.
  + Volumes montés pour les données Gitea et les configurations de fuseau horaire.
  + Port exposé : 3000.
  + Dépendance sur gitea-db.
  + Réseau : gitea-network.
* **gitea-db :**
  + Image : MySQL 8.4.0
  + Variables d'environnement pour la configuration de la base de données.
  + Volume monté pour les données MySQL.
  + Réseau : gitea-network.

### Services Production

* **prod-server :**
  + Construction à partir d'un Dockerfile local.
  + Montages de volumes pour les configurations Caddy.
  + Ports exposés : 8082 pour HTTP, 8004 pour HTTPS.
  + Réseaux : gitea-network, prod-network.
* **prod-db :**
  + Image : MySQL 8.0.
  + Commande de démarrage spécifique pour des modes SQL stricts.
  + Variables d'environnement pour la base de données.
  + Volume monté pour les données MySQL.
  + Réseaux : gitea-network, prod-network.
* **prod-pma :**
  + Image : phpMyAdmin.
  + Dépendance sur prod-db.
  + Port exposé : 8087.
  + Variables d'environnement pour la connexion à la base de données.
  + Réseau : prod-network.

### Réseaux

* Trois réseaux Docker sont définis :
  + dev-network
  + gitea-network
  + prod-network

une erreur mais survenu car a la base le fichier var était le même de la prod que pour le dev donc jai du changer

### Dockerfile pour le Serveur de Dev

Le Dockerfile dockerfile-dev-serv est utilisé pour définir la configuration et l'environnement d'un conteneur Docker destiné au serveur de développement dans le projet. Voici une explication détaillée de chaque instruction dans ce fichier :

### Instructions du Dockerfile :

1. **FROM dunglas/frankenphp :**
   * Cette instruction spécifie l'image de base à partir de laquelle construire ce conteneur. Dans ce cas, l'image dunglas/frankenphp est utilisée, qui contient un ensemble prédéfini d'outils et de dépendances pour le développement PHP.
2. **RUN apt-get update :**
   * Cette commande met à jour les paquets disponibles dans les dépôts APT. Cela garantit que les dernières versions des paquets sont utilisées lors de l'installation des dépendances.
3. **Installation de Composer :**
   * La prochaine série d'instructions installe Composer, l'outil de gestion des dépendances PHP. Cela se fait en téléchargeant le script d'installation de Composer et en l'exécutant avec PHP, puis en déplaçant l'exécutable résultant vers /usr/local/bin/composer pour qu'il soit accessible globalement.
4. **WORKDIR '/app' :**
   * Cette instruction définit le répertoire de travail dans le conteneur. Lorsque le conteneur démarre, il sera positionné dans le répertoire /app.
5. **Ouverture des droits de lecture, modification et exécution :**
   * La commande chmod 777 -R /app accorde des permissions étendues (lecture, écriture et exécution) à tous les utilisateurs pour le répertoire /app. Cela peut être nécessaire pour garantir que tous les utilisateurs peuvent manipuler les fichiers dans ce répertoire, mais cela peut aussi poser des problèmes de sécurité s'il est utilisé de manière indiscriminée.
6. **Installation d'Extensions PHP supplémentaires :**
   * L'instruction install-php-extensions installe les extensions PHP supplémentaires requises pour le projet. Dans cet exemple, les extensions suivantes sont installées : pdo\_mysql, gd, intl, zip, opcache. Ces extensions sont souvent nécessaires pour exécuter des applications PHP modernes.

### Dockerfile pour le Serveur de Production

Le fichier Dockerfile-prod-serv est utilisé pour créer l'image Docker du serveur de production. Voici une explication détaillée des principales instructions incluses dans ce Dockerfile :

1. **Directive FROM :**
   * Cette directive spécifie l'image de base à partir de laquelle construire cette nouvelle image. Dans ce cas, l'image dunglas/frankenphp est utilisée comme base. Il s'agit probablement d'une image contenant un environnement PHP prêt à l'emploi, adapté pour les applications web.
2. **Mise à Jour du Système :**
   * L'instruction apt-get update est utilisée pour mettre à jour les informations sur les paquets disponibles dans les dépôts APT. Cela garantit que le système est à jour avant d'installer de nouveaux paquets.
3. **Installation de Git :**
   * L'instruction apt-get install -y git installe Git sur l'image. Git est un système de contrôle de version largement utilisé pour le suivi des modifications apportées au code source d'un projet. Son installation est essentielle pour le déploiement et la gestion de projets logiciels.
4. **Répertoire de Travail :**
   * Enfin, l'instruction WORKDIR /var/www définit le répertoire de travail à /var/www dans l'image. Cela signifie que tous les futurs commandes RUN, CMD, ENTRYPOINT, COPY et ADD seront exécutées à partir de ce répertoire.

### Explication Supplémentaire :

Il est important de noter qu'il aurait été bénéfique d'ajouter plus de commentaires ou d'explications dans ce Dockerfile pour clarifier le but de chaque instruction. Par exemple, des commentaires expliquant pourquoi Git est installé sur l'image et la raison du choix de dunglas/frankenphp comme image de base pourraient aider les futurs développeurs à comprendre plus facilement la configuration de l'image Docker.

### Configuration de Gitea

Pour configurer Gitea, plusieurs étapes ont été entreprises, notamment la création d'utilisateurs administrateurs et d'utilisateurs pour les environnements de développement et de production :

1. **Création de l'utilisateur administrateur :**
   * Un utilisateur administrateur nommé adm\_berrebi a été créé en premier lieu.
   * Cet utilisateur a été attribué des privilèges administratifs pour la gestion globale de Gitea.
2. **Création des utilisateurs de développement et de production :**
   * En plus de l'utilisateur administrateur, des utilisateurs spécifiques pour les environnements de développement (dev) et de production (prod) ont été créés.
   * Ces utilisateurs ont été configurés avec les autorisations appropriées en fonction de leurs besoins respectifs dans les environnements de développement et de production.

Ces étapes de configuration ont permis d'établir un environnement Gitea fonctionnel et adapté aux besoins du projet.

### Création d'un Dépôt sur le Compte de Développement

Dans le cadre de la configuration de Gitea, une étape essentielle a été la création d'un dépôt sur le compte de développement :

* **Création du Dépôt :**
  + Sur le compte de l'utilisateur de développement (dev), un dépôt a été créé pour héberger le code source du projet.
  + Ce dépôt a été configuré avec les paramètres appropriés, y compris le nom du dépôt, la description et les options de visibilité et de collaboration selon les besoins du projet.

Cette création de dépôt sur le compte de développement a permis d'établir un espace centralisé pour le développement et la gestion du code source du projet.

### Correction de l'URL Gitea

Après la création de l'URL Git, qui initialement contenait "localhost" et posait un problème pour la connectivité, une correction a été apportée pour résoudre ce problème :

* **Identification de l'Adresse IP :**
  + Pour trouver une adresse IP stable pour le serveur Gitea, la commande docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' gitea a été utilisée initialement.
  + Cependant, cette méthode n'a pas donné une adresse IP fixe, ce qui a nécessité une approche alternative.
* **Utilisation de nmap pour Trouver l'URL par Défaut :**
  + Pour résoudre le problème, la commande nmap -sP a été utilisée pour scanner le réseau et trouver l'URL par défaut associée au serveur Gitea dans le réseau Docker (gitea.dossier\_gitea-network).
* **Modification de l'URL dans le Fichier de Configuration :**
  + Ensuite, dans le dossier .git, le fichier de configuration a été modifié pour remplacer "localhost" par l'URL par défaut trouvée (gitea.dossier\_gitea-network).
  + Cette modification a permis de corriger l'URL utilisée dans Git et de garantir une connectivité appropriée avec le serveur Gitea.

Cette correction a permis de résoudre le problème d'URL incorrecte dans Git et d'établir une connexion stable avec le serveur Gitea.

### Ajout du Projet sur Gitea

Après avoir corrigé l'URL dans le fichier de configuration Git et établi une connexion stable avec le serveur Gitea, le projet de développement a été ajouté à Gitea en suivant les étapes suivantes :

1. **Connexion au Serveur de Développement :**
   * Pour accéder au serveur de développement, la commande docker exec -it dev-server /bin/bash a été utilisée pour ouvrir un shell interactif dans le conteneur.
2. **Initialisation de Git :**
   * Dans le répertoire du projet, la commande git init a été utilisée pour initialiser un dépôt Git local.
   * Ensuite, la commande git checkout -b main a été utilisée pour créer une nouvelle branche principale nommée "main".
3. **Ajout et Commit des Fichiers :**
   * Tous les fichiers du projet ont été ajoutés à l'index Git à l'aide de la commande git add ..
   * Ensuite, un commit a été créé avec la commande git commit -m "nom du commit" pour valider les modifications.
4. **Configuration de l'Origine :**
   * L'URL de l'origine a été configurée avec la commande git remote add origin http://gitea.dossier\_gitea-network/dev/nomduprojet.git.
5. **Pousser les Modifications sur Gitea :**
   * Enfin, les modifications locales ont été poussées vers le dépôt Gitea sur le serveur avec la commande git push -u origin main.

Ces étapes ont permis de mettre en ligne le projet de développement sur le serveur Gitea, fournissant ainsi un espace centralisé pour la collaboration et la gestion du code source

### Clonage du Projet sur le Serveur de Production

Après avoir ajouté avec succès le projet sur le serveur Gitea, une étape supplémentaire consistait à cloner le projet sur le serveur de production. Voici comment cette opération a été réalisée :

1. **Connexion au Serveur de Production :**
   * Pour accéder au serveur de production, via la commande docker exec -it prod-server
2. **Clonage du Projet :**
   * À l'aide de la commande git clone http://gitea.dossier\_gitea-network/dev/nomduprojet.git, le projet a été cloné depuis le dépôt Gitea sur le serveur de production.
   * Cette commande a copié tous les fichiers et l'historique de version du projet dans un répertoire local sur le serveur de production.

Avec le projet cloné sur le serveur de production, l'équipe peut désormais travailler sur le projet dans cet environnement, en garantissant la cohérence entre les environnements de développement et de production.

**Conclusion :**

La mise en place réussie de Gitea comme alternative à GitHub sur un serveur local, orchestrée via Docker en s’aidant d’un Framework qui a permis de connecter efficacement les environnements de développement et de production. La configuration des serveurs, la création des utilisateurs et des dépôts, ainsi que la résolution des problèmes d'URL ont été réalisées avec succès, facilitant ainsi la collaboration et la gestion du code source du projet.