DevOps CI/CD

Project Description:

Le projet est une démonstration de base de CI / CD utilisant **Laravel** comme Framework pour le backend et **Angular** pour le front-end et un serveur **mysql** pour notre base de données, cette application est un **blog** pour présenter les bases de l'API Rest, GET, POST, PUT, DELETE.

(pour simplement exécuter les conteneurs sans configurer go ici)

VCS repository (GitHub) :

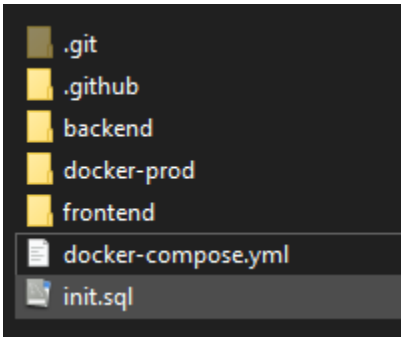
Nous avons choisi de travailler avec **github** à cause des ***github actions***, ce qui nous donne la possibilité de l'utiliser comme plateforme ci sans **Jenkins**, voici le lien pour le repo du projet Yahya-Regragui/blogops-fullstack (github.com)

Environnement :

1. ***Git clone https://github.com/Yahya-Regragui/blogops-fullstack.git***

   pour copier le dépôt sur la machine locale

2. Voici la structure du projet
   a. Le dossier .github contient un fichier .yaml qui a un ensemble d'instructions à exécuter dans gihub actions
   b. Le dossier backend contient l'API rest et backend et son dockerfile , le dossier frontend contient le projet Angular et son dockerfile.
   c. Docker-prod contient ini.sql qui contient un ensemble d'instructions pour créer les tables nécessaires et docker-compose-prod.yml à utiliser dans l'environnement de production.
   d. Fichier docker-compose.yml pour construire nos conteneurs et notre image mysql et pour exécuter tous les conteneurs simultanément avec les paramètres appropriés
   e. Init.sql même fichier dans le dossier docker-prod



3. Pour construire nos images, nous devons :
    a. Télécharger et installer composer ici
    b. Télécharger et installer node.js et npm ici
    c. Ouvrir le dossier du frontend, ou dans linux / cd frontend
        i. Dans le terminal : ***npm install***
    d. Ouvrir le dossier du backend, ou dans linux / cd backend
        i. Dans le terminal : ***composer install***

Docker & docker-compose

1. docker build
    1. Pour créer des images individuellement
        - Ouvrir le dossier du frontend, ou dans linux / cd frontend
        - Dans le terminal : ***docker build -t frontend***

nous allons exécuter le frontend sur le node: 12-alpine car il est optimisé pour cela, après, nous avons défini le répertoire de travail sur l'image, puis nous avons copié le package.json dans le répertoire de travail, puis nous avons configuré node_modules avec npm install, puis nous copions notre projet afin de coller les fichiers de projet dans l'image afin que nous puissions l'utiliser dans la production, enfin nous avons exposé le port 4200 afin que nous puissions y accéder depuis notre machine locale puis nous démarrons le service avec npm run start.

```
FROM node:12-alpine as build-step

WORKDIR /app

COPY package.json ./

RUN npm install

COPY . .

EXPOSE 4200

CMD npm run start
```

- Ouvrir le dossier du backend, ou dans linux / cd backend
- Dans le terminal : ***docker build -t backend***

```
FROM php:7.3

RUN apt-get update -y && apt-get install -y openssl zip unzip git
RUN curl -sS https://getcomposer.org/installer | php -- --install-dir=/usr/local/bin/ --filename=composer
RUN docker-php-ext-install pdo mbstring pdo_mysql

WORKDIR /app/backend
COPY . .
RUN composer install

EXPOSE 8000
CMD php artisan migrate
CMD php artisan serve --host=0.0.0.0
```

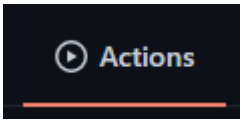
nous utilisons l'image php: 7.3 puisque nous utilisons le framework laravel, nous configurons l'environnement avec composer et pdo, puis nous définissons le répertoire de travail, puis nous copions notre dossier backend sur l'image, enfin nous exposons le port 8000 afin d'y accéder il à partir de la machine locale, puis nous démarrons le service avec php artisan

2. docker-compose
    1. Dans le dossier racine dans le terminal, exécutez ***docker-compose up*** Cette commande construira tous les conteneurs, y compris la base de données mysql

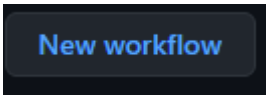
3. docker-compose-prod
    1. Ouvrir le dossier du docker-prod, ou dans linux / cd docker-prod
        - ***docker-compose -f docker-compose-prod.yml up***
    2. Cette commande vous permettra de sauter toutes ces étapes, elle récupérera les conteneurs du hub docker sans avoir besoin de construire, ni de configurer l'environnement

Github Actions CI/CD pipeline

1. Github répertoire
    - Après avoir poussé notre projet vers github nous cliquons sur l'onglet actions



    - Créer un nouveau workflow en cliquant sur ce bouton



    - Maintenant nous pouvons créer notre fichier .yml pour configurer le pipeline
    - La syntaxe est similaire à celle du fichier docker-compose, ce sont les étapes qui seront exécutées une fois que quelque chose est poussé vers ce git repo
        1. Install l'image Ubuntu
        2. Construire tous les conteneurs avec docker-compose up -d –build
        3. timeout pendant 20s donc attendre que la base de données s'initialise
        4. Exécuter des tests unitaires pour notre backend, et s'ils réussissent, nous fermons les conteneurs
        5. Github se connecte en utilisant mes informations d'identification au hub docker, puis transmet les images

```yaml
name: Laravel

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  laravel-tests:

    runs-on: ubuntu-latest

    steps:
    - uses: shivammathur/setup-php@b7d1d9c9a92d8d8463ce36d7f60da34d461724f8
      with:
        php-version: '8.0'
    - uses: actions/checkout@v2

    - name: build app
      run: docker-compose up -d --build

    - name: running db
      uses: jakejarvis/wait-action@master
      with:
        time: '20s'
    #- name: docker ps
     # run: docker ps -a

    - name: unit testing
      run: |
        docker-compose exec -T backend php artisan test
        docker-compose down


    - name: push to docker hub
      env:
        DOCKER_USER: ${{ secrets.DOCKER_USER }}
        DOCKER_PASSWORD: ${{ secrets.DOCKER_PASSWORD}}
      run: |
        docker login -u $DOCKER_USER -p $DOCKER_PASSWORD
        docker-compose push
```

C'est la fin de notre pipeline