CI/CD TP

CI

Nous allons voir maintenant la mise en place d'un pipeline plus avancé utilisant un vrai projet django. Il vous faut dans ce projet deux branches, une qui utilise le pseudo serveur web intégré à django avec une base de donnée sqlite3 que nous appelerons dev, et une branche qui utilise un serveur web nginx et une base de données mariadb qui sera la branche master. Vos deux branches doivent contenir à la racine les fichiers Dockerfile manage.py. Le dépôt que j'utilise s'appel devops, tout comme le projet Django, et l'application du projet Django s'appel presentation.

Vous allez devoir récupérer le dépôt et le push sur un dépôt que vous créerez sur votre Gitlab.

Créez donc sur gitlab un nouveau projet vide, donnez lui un nom et décochez la case 'initialize repository with a README".
Ouvrez ensuite votre terminal et lancez les commandes suivantes (en replacant <adresse de votre depot> par l'adresse le liens vers votre nouveau repository gitlab):

```
git clone https://gitlab.com/m.przybylo/tp_devops.git
cd tp_devops
rm -rf .git
git init
git remote add origin <adresse de votre depot gitlab>
git add *
git commit -m "Initial commit"
git push origin master
git branch dev
git checkout dev
git add *
git commit -m "initial commit dev"
git push origin dev
```

Sur la page de votre dépôt Gitlab, sur la branche dev, ouvrez le web IDE et créez un nouveau fichier .gitlab-ci.yml. Nous allons pour l'instant définir 2 stages : un stage build et un stage test.

Nous allons définir un premier job "build app" qui va utiliser l'image python:3 et se placer dans le stage build.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**			
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

CI/CD TP

Le script du job devra installer curl, installer les dépendances python nécessaires et figurants dans un fichier requirements.txt à la racine du dépot, migrer la base de données, démarrer le serveur web django et vérifier via curl qu'on accède bien à la page et qu'on y trouve bien ce qu'on veut.

curl http://localhost:8000/presentation | grep "Awesome"

Avec la requête curl on essaye d'accéder à notre site et grace à la commande grep on cherche si on y trouve bien le mot "Awesome" qui figure sur mon index.html. Voici donc à quoi doit ressembler la partie script de votre job :

```
script:
  - apt update –y && apt install curl
  - python -m pip install -r requirements.txt
  - python manage.py migrate
  - python manage.py runserver 0.0.0.0:8000
  - curl http://localhost:8000/presentation/ |grep "Awesome"
```

Effectuez votre commit et regardez votre pipeline. Vous devriez voir votre pipeline tourner mais rester bloqué à l'étape "python manage.py runserver 0.0.0.0:8000". En effet la commande runserver n'est pas sensé terminer. Votre pipeline tournera donc indéfiniement, ce qui pose problème étant donné que vous n'avez que 400 minutes offertes par gitlab par mois. Commencez donc par arréter votre job. Il va donc falloir indiquer de faire tourner le processus en arrière-plan. Pour se faire il suffit de rajouter le symbole & a la fin de la commande. Il va également falloir indiquer au processus d'attendre un petit peu pour lancer la commande curl. Nous utiliserons l'instruction sleep pour ça.

```
script:
  - apt update –y && apt install curl
  - python -m pip install -r requirements.txt
  - python manage.py migrate
  - python manage.py runserver 0.0.0.0:8000 &
  - sleep 10
  - curl http://localhost:8000/presentation/ |grep "Awesome"
```

Après avoir commit vos changements vous devriez voir votre pipeline se dérouler sans problèmes.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

CI/CD TP

Nous allons maintenant définir notre job de test qui s'appellera "test app" , qui utilisera également l'image python:3 et qui se placera dans le stage "test".

Le script du job devra installer les dépendances python du fichier requirement.txt, effectuer les migrations et lancer les tests en utilisant la fonctionnalité de test de django.

Faites un commit et passez à la partie suivante quand votre pipeline fonctionne.

Nous allons maintenant rédiger un autre job de test que nous nommerons "lint-job". Celui-ci sera fait pour vérifier les warning et erreurs de votre code. Pour ça nous aurons besoin d'un outil appelé flake8. Sur le registry de gitlab il y a une image flake8 disponible. Nous allons donc utiliser cette image et faire la commande

flake8 .

Cela va lancer les test de syntaxe sur le répertoire courant (.) et ses sous dossiers. Voici le job :

```
lint-job:
  image: registry.gitlab.com/pipeline-components/flake8:latest
  stage: test
  script:
    - flake8 .
```

En faisant votre commit vous devriez voir que le pipeline démarre et lance les deux tests du stage simultanément mais le job lint-job échoue. C'est parceque flake rapporte tous les warnings possibles et est très peu permissif. Nous pouvons dire à flake de ne sélectionner que les alertes de type F (les erreurs de code) en ignorant également les erreurs d'import. Remplacez donc la commande flake par `flake8 --select F --ignore F4 .` . Nous allons également changer le stage de notre job. Gitlab fourni des noms de stage, que vous n'avez pas besoin de déclarer dans la section stages, dont notamment le stage ".pre" qui se déclenche en premier dans votre pipeline. C'est celui-ci que nous indiquerons lint-job.

Faites un commit et passez à la partie suivante quand votre pipeline fonctionne.

Nous allons optimiser notre pipeline. Nous pouvons voir déjà que nous avons deux jobs très similaires : le job "build app" et le job "test app". Ils utilisent la même image, installent les mêmes dépendances et lancent la même commande de migration de base de données. Nous allons donc supprimer le job "test app" et rajouter la commande de tests django dans le job "build app", juste avant la commande runserver.

Faites un commit et passez à la partie suivante quand votre pipeline fonctionne.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

CI/CD TP

CD

Pour mettre en place il faut pour déployer son application sur un serveur de production. N'ayant pas de serveur de production disponible nous allons effectuer notre phase de déploiement en publiant l'image docker contenant notre application sur notre registry du dépôt Gitlab. C'est aussi l'intérêt de Gitlab, vous offir un endroit en ligne ou stocker vos images docker. Nous ferons également la publication de notre image nginx personnalisée.

Pour pouvoir exécuter des commandes docker au sein d'un container il va falloir deux choses : utiliser l'image qui s'appel "docker:latest" et déclarer un service utilisant l'image "docker:dind" (dind voulant dire docker in docker).
Nous allons donc rédiger un job "deploy app" que nous mettrons dans un stage "deploy" que nous déclarerons dans nos stages, qui utilisera l'image et le service nécessaire pour efféctuer des commandes docker.
Nous devrons ensuite déclarer une phase "before_script" dans laquelle nous donnerons au container les identifiant pour notre registry Gitlab.
Nous indiquerons ensuite notre script qui devra définir le tag de l'image en fonction de la branche depuis lequel il est démarré puis build et push l'image généré sur notre registry.
Le job devra avoir une règle indiquant qu'il doit démarrer uniquement si il y a un fichier Dockerfile présent sur la branche qui commit.

Pour faire tout ca nous aurons besoin de plusieurs variables par défaut de Gitlab :

- CI_REGISTRY_USER : l'identifiant de l'utilisateur autorisé à push des images
- CI_REGISTRY_PASSWORD : le mot de passe de l'utilisateur autorisé à push des images
- CI_REGISTRY : l'adresse du registry sur lequel déposer nos images
- CI_REGISTRY_IMAGE : le nom de l'espace sur notre registry réservé à nos images
- CI_COMMIT_BRANCH : le nom de la branche depuis lequel est lancé le commit
- CI_DEFAULT_BRANCH : le nom de la branche par défaut du dépôt Gitlab

Voici le script en question :

```
deploy app:
  image: docker:latest
  stage: deploy
  services:
    - docker:dind
  before_script:
```


Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

CI/CD TP

```
    - docker login -u "$CI_REGISTRY_USER" -p "$CI_REGISTRY_PASSWORD" $CI_REGISTRY
  script:
    - |
      if [[ "$CI_COMMIT_BRANCH" == "$CI_DEFAULT_BRANCH" ]]; then
        tag=""
        echo "Running on default branch '$CI_DEFAULT_BRANCH': tag = 'latest'"
      else
        tag=":$CI_COMMIT_REF_SLUG"
        echo "Running on branch '$CI_COMMIT_BRANCH': tag = $tag"
      fi
    - echo $CI_REGISTRY_IMAGE
    - docker build --pull -t "$CI_REGISTRY_IMAGE"/django"${tag}" .
    - docker push "$CI_REGISTRY_IMAGE"/django"${tag}"
    - docker build --pull -t "$CI_REGISTRY_IMAGE"/nginx"${tag}" .
    - docker push "$CI_REGISTRY_IMAGE"/nginx"${tag}"
  rules:
    - if: $CI_COMMIT_BRANCH
      exists:
          - Dockerfile
```

Faites un commit et passez à la partie suivante quand votre pipeline fonctionne.

Vous disposez maintenant sur votre registry personnel vos images django et nginx nécessaires pour faire tourner votre projet.

Vous allez maintenant pouvoir rédiger le docker-compose de la branche dev. Ce compose devra utiliser les images django et nginx disponibles sur votre registry.

Le services de l'image django devra lancer la commande " gunicorn devops.wsgi:application --bind 0.0.0.0:8000 " et attribuer le volume docker static_volume au repertoire /code/static du container. Le service de l'image nginx devra exposer le 80 de l'hôte au port 80 du container et attribuer également attribuer le volume docker static_volume au repertoire /code/static du container et dépendra du premier container. N'oublier de déclarer static_volume.

Avec ce docker-compose nous pouvons déclarer notre job "test deploy" qui sera dans la nouvelle phase "post deploy". Nous utilserons encore une fois le service "docker:dind" mais cette fois via l'image "docker/compose:1.29.1" pour fonctionner il faudra indiquer l'entrypoint " [""] " pour l'image. Il faudra délcarer deux variables pour la configuration : DOCKER_HOST qui sera

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

CI/CD TP

"tcp://docker:2375" et DOCKER_DRIVER qui sera overlay2. Dans notre scripte nous vérifierons si notre compose fonctionne en de démarrant de manière détaché.

Faites un commit et passez à la partie suivante quand votre pipeline fonctionne.

Branche master

Nous allons nous occuper de la branche master. Notre branche master aura la différence d'utiliser une base de données mariadb. Pour faire ca vous aurez besoin de modifier le docker-compose pour y rajouter la base de donnée et également modifier le fichier settings.py du projet. Nous pourrions également créer un fichier .gitlab-ci.yml sur la branche master et changer le job de build pour qu'il inclue la base de donnée. Nous allons plustot procéder différemment. Nous allons faire en sorte que les deux branches soient équivalents. Il faudra pour ça que nous ayons 2 fichiers docker-compose et que les deux possibilitées soient couvertes dans le fichier settings.py ainsi que dans le fichier .gitlab-ci.

Les fichiers docker-compose

Vous allez commencer par renommer le fichier "docker-compose.yml" en "docker-compose-dev.yml'. Rédigez un fichier "docker-compose-master.yml" dans lequel vous rajouterez le service de base de données. Ce service utilisera l'image mariadb, attribuera le volume docker mysl_data au repertoire /var/lib/mysql/data et utilisera le fichier .env pour ses variables d'environnement. Le service de l'image django devra également utiliser ce fichier de variable d'environnement et devra dépendre du service de base de données.
Il faudra indiquer dans le service utilisant l'image django du fichier "docker-compose-dev.yml" la variable d'environnement "DATABASE=sqlite3". Cette variable a la valeur "mariadb" dans le fichier de variables d'environnement du docker-compose du master.

Le fichier settings.py

Dans le fichier "settigns.py" vous allez devoir indiquer les deux bases de données. La configuration mariadb devra s'appliquer uniquement si la variable d'environnement "DATABASE" est égale à "mariadb" et la configuration sqlite3 devra s'appliquer si elle est égale à "sqlite3".

Le fichier .gitlab-ci.yml

Modifiez le job "build app" du manifeste pour y ajouter le service "mariadb". Vous devrez, pour que cela marche, déclarer dans le job les variables présentes dans le fichier .env et modifier le script. Vous devrez décrire les deux cas en début du script :

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**			
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

CI/CD TP

Si la branche de commit est la branche par défaut alors il faudra declarer la variable d'environnement DATABASE (avec la commande "export DATABASE=mariadb"), installer le paquet mysql-default-client et accorder tous les privilèges sur la base de donnée à l'utilisateur créé pour le service (le nom de l'hôte pour le services mariadb est "mariadb").
Sinon il faudra exporter la variable DATABASE avec la bonne valeur.

Servez-vous du job "deploy app" pour écrire ce script.

Vous devez également modifier le script du job "test deploy" pour qu'il démarre le docker-compose correspondant à la branche sur laquelle le commit est effectué. Pour ca vous avez besoin d'utiliser l'option -f de la commande "docker-compose up" qui vous permet de spécifier le nom du docker-compose à utiliser ainsi que la variable CI_COMMIT_REF_SLUG dans laquelle se trouve le nom littéral de la branche depuis laquelle le commit est issue.

Faites un commit et passez à la partie suivante quand votre pipeline fonctionne.

Sécurisation de la branche master

Nous allons éffectuer une derniere modification pour notre branche master. Nous allons faire en sorte qu'il ne soit pas possible de push directement dessus. Cette branche ne sera utiliser que pour merge les branches de développement qui auront passés leurs pipelines. Pour faire ca il faut que vous alliez dans la catégorie "repository" du menu settings de votre dépôt Gitlab et puis dans la catégorie "Protected branche". Dans cette catégorie il faut que vous n'autorisiez personne à push quoi que se soit sur la branche master.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			