TP CI/CD

L'intégration continue (CI) et le déploiement continu (CD) sont devenus des pratiques incontournables dans le développement logiciel moderne. Ensemble, elles permettent d'automatiser non seulement l'intégration du code, mais aussi son déploiement, garantissant ainsi un flux de travail fluide et efficace.

L'intégration continue (CI) automatise l'intégration fréquente du code dans un projet, permettant de détecter et résoudre les problèmes dès qu'ils apparaissent. Chaque fois qu'une modification de code est apportée (commit), des étapes essentielles comme la compilation, les tests unitaires et fonctionnels sont exécutées automatiquement.

Le déploiement continu (CD), quant à lui, va un pas plus loin en automatisant la mise en production du code validé. Cela signifie que chaque modification validée peut être automatiquement déployée dans un environnement de production, réduisant ainsi le temps entre le développement et la livraison aux utilisateurs finaux.

Ce TP explore les principes fondamentaux de l'intégration continue et du déploiement continu, en se concentrant sur leur mise en œuvre pratique avec GitLab. Vous découvrirez les concepts clés de ces pratiques, ainsi que les bonnes pratiques pour optimiser votre flux de travail avec CI/CD.

Compte rendu sur 5 points à rendre sur Moodle en PDF avant la fin du TP. Merci de ne pas demander à ChatGPT (ou autre) de rédiger votre CR à votre place, faites le TP, justifiez de screenshots et écrivez vos propres phrases !

Commencez ce TP directement sur https://im2ag-gitlab.univ-grenoble-alpes.fr

Votre premier pipeline

Dans ce premier chapitre vous allez créer un container Docker qui exécute un script Python qui affiche « Hello IM2AG !! » :

Créez un nouveau projet. Pour ce projet 3 fichiers vont être nécessaires :

- Un script Python **hello.py**
- Un **Dockerfile**
- Un fichier **.gitlab-ci.yml** pour configurer votre Pipeline

Contenu de **hello.py** :

```
print("Hello, IM2AG !!")
```

Contenu du **Dockerfile** :

```
FROM python

WORKDIR /app
COPY hello.py .
CMD ["python", "hello.py"]
```

Remarque: lorsque vous allez créer ce fichier en lui donnant comme nom Dockerfile, vous verrez automatiquement apparaître une liste déroulante de template. Intéressant...

Vous allez maintenant créer le fichier **.gitlab-ci.yml** à la racine de votre projet. Ce fichier sert à définir les étapes automatisées (build, test, deploy, etc) que Gitlab doit suivre pour chaque commit dans un projet, assurant ainsi un processus de CI/CD. Vous pouvez éditer ce fichier en allant dans `Build > Pipeline editor > Configure pipeline`. Comme vous pouvez le voir, des templates sont aussi disponibles ici.

PS : bien mettre votre login à la place de <votrelogin>.

Contenu de **.gitlab-ci.yml** :

```
stages:
  - build
  - run

build_image:
  stage: build
  script:
    - docker build -t hello-world-<votrelogin> .
  tags:
    - shell

run_container:
  stage: run
  script:
    - docker run hello-world-<votrelogin>
  tags:
    - shell
```

Explications :

`stages` : déclare les étapes du pipeline
`build_image` et `run_container` : les noms des deux tâches
`stage` : indique que cette tâche appartient à l'étape `build` ou `run`
`script` : les commandes à exécuter
`tags` : définit les tags des **runners** Gitlab qui peuvent exécuter cette tâche

Il est important de comprendre les runners. Les runners sont des agents logiciels qui exécutent les jobs définis dans les pipelines. Ils sont généralement installés sur des systèmes différents du système Gitlab. Chaque runner peut être étiqueté avec des tags spécifiques, comme "shell", "docker" ou autre, pour spécifier les types d'environnements ou de configurations nécessaires à l'exécution des jobs. Il est donc nécessaire d'avoir au préalable des runners configurés pour pouvoir exécuter les jobs. Sur im2ag-gitlab, un runner a été partagé et a été configuré pour permettre cela. Vous pouvez les voir en allant dans `Settings > CI/CD > Runners > Expand`. Vous pouvez retrouver le tag `shell` ainsi que deux autres tags.

Au moment où vous avez commit ce fichier .gitlab-ci.yml, **votre premier pipeline a été lancé !** Il sera dorénavant lancé à chacun des commit que vous effectuerez dans ce projet. Allez dans `Build > Pipelines`, cliquez sur le status de votre pipeline (Passed).

Vous retrouvez vos deux étapes (build_image et run_container). Cliquez sur `Jobs` puis le nom du job. Vous devez voir que le pipeline a bien été exécuté sur `im2ag-runner-01` et que tout est OK !

Vous pouvez modifier votre script, commitez et vous verrez que votre image Docker « hello-world-<votrelogin> » se met directement à jour.

Modifiez le hello.py, commitez et expliquez les différentes étapes de l'intégration continue et les modifications prises en compte. Justifiez avec des screenshots.

Publiez cette image

Vous savez maintenant comment build une image, à jour, à chaque fois que vous allez commit votre code. Ce qui serait maintenant intéressant, ce serait de publier automatiquement cette image sur un registre. Dans un environnement professionnel, l'idéal serait que votre entreprise ait son propre dépôt/registre pour déposer les images Docker interne à la boite (Docker Registry ou Harbor par exemple). Pour ce TP nous allons utiliser Docker Hub, le dépôt officiel de Docker, **créez vous un compte directement sur leur site.**

Maintenant, nous allons ajouter au projet existant une étape pour publier l'image sur Docker Hub.

Il va falloir dans un premier temps créer deux variables DOCKER_HUB_USERNAME et DOCKER_HUB_PASSWORD.

- Créez ces deux variables en allant dans `Settings > CI/CD > Variables`. Pour raison de simplicité, décocher la case « Protected ».
- Éditez votre fichier `.gitlab-ci.yml` pour y ajouter une nouvelle étape « publish ». Dans cette étape vous allez :
    - taguer votre image avec le nom hello-$DOCKER_HUB_USERNAME (docker image tag)
    - vous loguer à Docker Hub (docker login)
    - pousser l'image (docker push)

Modifiez le fichier `.gitlab-ci.yml` en utilisant bien les variables.

PS : vous pouvez commenter l'étape run_container, qui n'a plus d'intérêt dans cet exercice.

Une fois votre Pipeline ok, vérifiez sur Docker Hub que votre image a bien été publiée.

Justifiez avec des screenshots du pipeline et que l'image a bien été poussée sur votre Docker Hub.

Prod & PreProd

Tout ceci est fort sympathique, cependant dans la vraie vie, il manque un aspect fondamental... **Il ne faut JAMAIS coder directement sur la branche « main » !!!**

Vous allez donc :

- Créer une nouvelle branche « preprod »
- Puis modifier votre pipeline pour que l'image construite et publiée, soit distincte en deux images (prod et preprod) en fonction de la branche modifiée.

Pour cela utilisez la condition « only » dans vos différentes étapes :

```
only:
   - preprod
# ou
only:
   - main
```

A vous de jouer !

Dans un premier temps, votre commit sur la branche « preprod » devra publier une nouvelle image « preprod » sur le Docker Hub,

Puis dans un second temps, vous devrez faire un merge de la branche « preprod » vers la branche « main » et automatiquement le pipeline devra publier une image « prod » sur le Docker Hub.

Justifiez avec des screenshots.

Pour l'exercice suivant, rendez publique vos images prod et preprod directement depuis le site https://hub.docker.com/

Watchtower

Ce qui serait maintenant vraiment top, ce serait d'avoir un site en PREPROD et un site en PROD, qui tournent en continu et qui prennent automatiquement l'image Docker publiée sur le Docker Hub lorsque la version de l'image est incrémentée. Et là, vous aurez un environnement optimum pour votre travail !

Pour cet exercice il va falloir modifier votre projet :

- Dans un premier temps, modifiez le contenu de votre **hello.py** sur chacune de vos branches :

```
# hello.py
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello, IM2AG Prod !!" # Ou PreProd en fonction de la branche

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

- Puis ajoutez un **requirements.txt** contenant le paquet Flask et assurez-vous que ce fichier soit pris en compte dans votre application via le Dockerfile.

Depuis un système Docker (votre VM sur eCloud), clonez le projet **docker-prod-preprod** :

- https://im2ag-gitlab.univ-grenoble-alpes.fr/rapacchd/**docker-prod-preprod**

Sur le projet **docker-prod-preprod**, modifiez la variable d'environnement dans le fichier ".env" et complétez le hello-compose.yml.

Puis lisez et comprenez le fichier watchtower-compose.yml.

Lancez la stack, vous devez maintenant avoir trois containers lancés. Votre site de PROD, votre site de PREPROD et WATCHTOWER, vérifiez vos deux sites sur votre navigateur.

Justifiez avec des screenshots et expliquez ce que fait Watchtower.

Sur la branche PREPPROD, faites une modifications du fichier « hello.py » et un commit.

Que se passe-t-il ? Expliquez la situation et en quoi c'est trop cool !??

Il faut bien que vous vous soyez logué avec la commande « **docker login** » au préalable, mais il est possible que ça ne fonctionne pas du premier coup. Pensez à regarder les logs du container Watchtower. S'il y a une erreur d'authentification, il faut alors remplacer https://index.docker.io/v1/ par https://docker.io/v1/ dans le fichier /home/ecloud/.docker/config.json.

Tests unitaires

Il est possible d'intégrer vos tests unitaires dans des pipelines. L'automatisation de l'exécution de tests unitaires sur chaque push de code vous permettra de gagner grandement en efficacité.

Ici vous n'avez pas grand chose à faire, vous allez simplement récupérer le projet suivant : https://im2ag-gitlab.univ-grenoble-alpes.fr/rapacchd/testunitim2ag

Comprendre ce projet et l'expliquez.

On remarque la présence de la ligne « image: python ». Expliquez la.

Les artefacts, qu'est-ce donc !?

Conclusion

Vous savez maintenant comment versionner, tester, construire, publier et mettre en service des applications de manières automatisées. Les pratiques CI/CD, ainsi que les outils Docker et GitLab, sont essentiels (voir même indispensable) dans le développement moderne. Si ces méthodes ne sont toujours pas mises en application dans vos entreprises... alors poussez pour qu'elles le soient ! Elles amélioreront grandement votre quotidien et celui de vos collègues. Quoiqu'il en soit, toutes ces compétences vous seront très utiles dans vos futurs projets.