Exercices supplémentaires docker

Table des matières

I Exercice : Docker++ : Docker Compose

Docker Compose

Déployer des images manuellement c'est très drôle, mais dans la pratique on préfère automatiser tout cela. Pour cela, plusieurs technologies existent (ansible & co) mais on conseillera `docker-compose`. C'est un outil qui décrit l'architecture docker, quelles images déployer et comment elles s'interconnectent.

Ce tuto[1] est une bonne ressource pour se familiariser avec la philosophie et la syntaxe. Pour plus d'inspiration, les docker compose de Picasoft sont disponibles sur Gitlab[2] et le wiki de Picasoft[3] contient des instructions très complètes.

Le but de cet exercice est de redéployer l'entièreté du serveur par du docker-compose de telle manière à ce que l'on puisse le re-déployer sur un autre serveur en une seule commande.

1. https://www.educative.io/blog/docker-compose-tutorial
2. https://gitlab.utc.fr/picasoft/projets/dockerfiles
3. https://wiki.picasoft.net/doku.php?id=technique:docker:start

II Exercice : Docker ++ : Gitlab CI

Gitlab CI

Dans cet exercice, nous allons étudier l'utilisation de docker dans le cadre de l'intégration continue de Gitlab. Le principe de l'intégration continue est d'exécuter une série d'opérations (compilation, tests, documentation, publication) lorsqu'un commit Git est poussé sur Gitlab. Pour cela on va utiliser l'instance Gitlab de l'UTC[4].

La CI fonctionne grâce à deux aspects :

- Un fichier `.gitlab-ci.yml` qui décrit la série d'opérations (aussi appelée `pipeline`) et comment la lancer
- Un *runner*, c'est-à-dire un container sur lequel tournera la suite d'opération

Description de la CI

La structure d'un fichier `.gitlab-ci.yml` est en YAML et ressemble à

```yaml
image: debian

build-job:
  stage: build
  tags: [docker]
  script:
    - echo "Hello, $GITLAB_USER_LOGIN!"

test-job1:
  stage: test
  tags: [docker]
  script:
    - echo "This job tests something"

test-job2:
  stage: test
  tags: [docker]
  script:
    - echo "This job tests something, but takes more time than test-job1."
    - echo "After the echo commands complete, it runs the sleep command for 20 seconds"
    - echo "which simulates a test that runs 20 seconds longer than test-job1"
    - sleep 20

deploy-prod:
  stage: deploy
  tags: [docker]
  script:
    - echo "This job deploys something from the $CI_COMMIT_BRANCH branch."
```

Pour aller plus loin sur ce fichier, on peut se référer à la documentation en ligne[5], et s'inspirer de ce qui est fait à Picasoft[6].

Il est possible d'exporter des fichiers à chaque étape de la pipeline dans ce qu'on appelle des *artifacts*. Pour cela, il suffit d'énumérer les fichiers/dossiers à exporter dans un champ artifacts de la forme

---

4. https://gitlab.utc.fr
5. https://docs.gitlab.com/ee/ci/yaml/index.html
6. https://gitlab.utc.fr/picasoft/apis/init/pr-sentation/-/blob/master/.gitlab-ci.yml

```yaml
projet:
  stage: build
  tags:
    - docker
    - latex
  script:
    - cd projet && latexmk -shell-escape -pdf
  artifacts:
    paths:
      - projet/projet.pdf
    expire_in: 1 hour
```

Il existe finalement un espace spécial sur Gitlab nommé *Gitlab Pages*. Il s'agit d'un espace associé à chaque repository capable de servir des pages statiques (html, pdf, etc), qui est concrètement un mini-serveur statique hébergé sur les serveurs de Gitlab (ou de l'UTC dans notre cas). Il est possible d'y mettre des fichiers à l'aide de la CI avec quatre conditions :

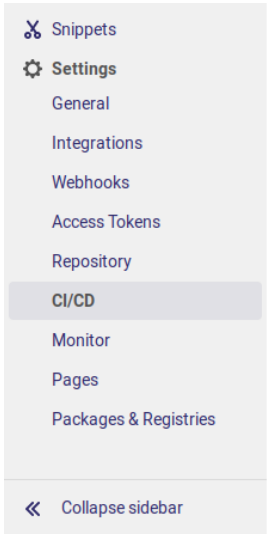
- Avoir un job nommé pages dans le fichier `.gitlab-ci.yml`
- Qu'il soit à l'étape de déploiement
- Qu'il mette lesdits fichiers dans un dossier *public*
- Qu'il exporte ce dossier *public*

Si ces conditions sont réunies, un site de la forme `<username>.gitlab.utc.fr/<projet>` sera créé. La première fois qu'un *Gitlab Page* est crée, cela peut prendre jusqu'à 30 minutes.

```yaml
pages:
  stage: deploy
  tags:
    - linux
  script:
    - mkdir -p public
    - cp projet/projet.pdf public/
  artifacts:
    paths:
      - public
    expire_in: 1 year
```

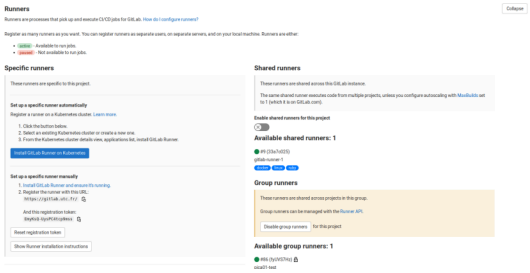
Utilisation de Runners

Un runner est un programme qui met à disposition la puissance de calcul d'une machine réelle (un VPS, votre machine personnelle, n'importe quel ordinateur) à Gitlab. Installer un runner sur une machine Linux revient simplement à exécuter la bonne image docker en suivant ce tuto[7]. Il existe d'autres façons de faire mais celle ci est de loin la plus simple.

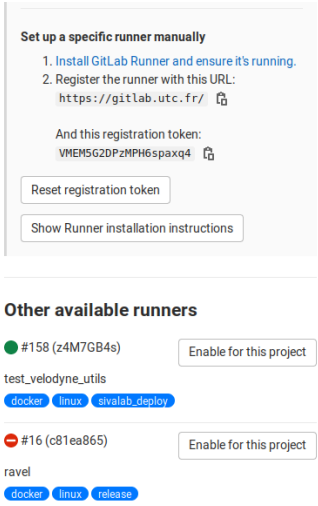


7. https://docs.gitlab.com/runner/install/docker.html

Pour éviter que tout le monde ai à déployer son propre VPS, l'UTC propose un Shared Runner, un runner accessible à n'importe qui à l'UTC mais qui peut être un peu lent car partagé entre tous les utilisateurs.



On peut également spécifier son propre runner, en théorie bien plus rapide et disponible juste à côté. Pour cela, on peut suivre le tuto[8] et récupérer le token spécifique au projet dans la configuration du repository.



Jobs

Lorsqu'un commit est poussé sur Gitlab et que le repository contient un `.gitlab-ci.yml`, un *pipeline* va automatique se lancer. On peut retrouver les pipelines courants et passés dans le menu CI/CD.

Question 1

[solution n°1 p. 8]

Créer un repository sur gitlab.utc.fr dans lequel vous ajouterez un fichier `.gitlab-ci.yml` qui reprend l'exemple précédent. Ajouter dans les paramètres du repository le shared runner afin que la CI puisse se dérouler dessus.

Question 2

[solution n°2 p. 8]

Déployer un runner sur votre VPS, le sélectionner dans les paramètres du repository et vérifier que la CI se déroule bien dessus.

Question 3

[solution n°3 p. 8]

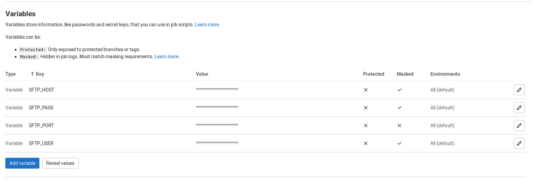
Utilisez les Gitlab Pages pour mettre en ligne une page html ou un pdf de votre choix.

8. https://docs.gitlab.com/runner/register/index.html#docker

Question 4

[solution n°4 p. 8]

Il est de bonne pratique de ne pas mettre en clair ses identifiants et mots de passe dans les scripts. Pour cela Gitlab propose un système de variables. En vous aidant de la documentation[9], remplacer dans le fichier `.gitlab-ci.yml` précédent les identifiants, hôte et mot de passes par des variables que vous aurez préalablement créé dans les paramètres du repository.



9. https://docs.gitlab.com/ee/ci/variables/

Solutions des exercices

Solution n°1

[exercice p. 6]

Solution n°2

[exercice p. 6]

Solution n°3

[exercice p. 6]

Solution n°4

[exercice p. 7]