

TP: Configuration d'un Pipeline CI/CD pour une Application Web PHP

1 – Compréhension et configuration de base

1.1 – Analyse du projet PHP

Il s'agit d'une application PHP simple, avec principalement `index.php`, qui va afficher deux rectangles contenant chacun du texte, et ce en appelant l'app' dédiée `ImageCreator.php`.

- `ImageCreator.php` définit la classe `ImageCreator` qui va prendre en paramètre deux couleurs et deux chaînes de caractères pour ainsi remplir l'objectif énoncé à l'instant.
- L'utilisation du gestionnaire de packages PHP « Composer » est ici nécessaire pour l'appel à la bibliothèque de gestion de date & heure « Carbon », dont l'unique but est de joindre au texte du premier rectangle, la date et l'heure à laquelle celui-ci est généré et donc en l'occurrence à peu près la date et l'heure de l'affichage de la page.

Étant donné le peu d'éléments soulignés, d'autres plus anecdotiques peuvent être évoqués :

- Une page `info.php` est présente et génère la page d'information standard de php grâce à `phpinfo()`.
- Une police d'écriture présente via le fichier `consolas.ttf` est utilisée par `ImageCreator`

Concernant la configuration et le déploiement de l'application, celle-ci est containerisée, avec Docker, via `docker/Dockerfile`.

1.2 – Configuration de l'environnement Docker

Commençons par construire l'image

```
cd "php-devops-tp"

# Commandes docker lancées en tant qu'utilisateur root

# Construction de l'image à partir de la racine du projet
docker build --tag php-devops-tp --file docker/Dockerfile .
# L'image a bien été créée
docker images

# Instanciation d'un container qui va tourner en arrière-plan
docker run --detach --interactive --tty \
  --publish target=80,published=127.0.0.1:9852,protocol=tcp \
  --add-host host.docker.internal:host-gateway \
  --name php-devops-tp_container php-devops-tp
# Il nous est possible d'accéder à la page via le navigateur, à l'adresse "http://localhost:9852"

# Le container est lancé
docker ps

# Possibilité d'entrer dans le container via tty avec `bash`
docker exec --interactive --tty php-devops-tp_container /bin/bash
```

Code 1. – Construction de l'image Docker,
instanciation d'un container et accès au terminal du container

Nous avons à présent



Fig. 1. – Accès à la page info.php du container fonctionnel et accessible

Un problème apparaît cependant avec `index.php` comme le montre l'image ci-dessous

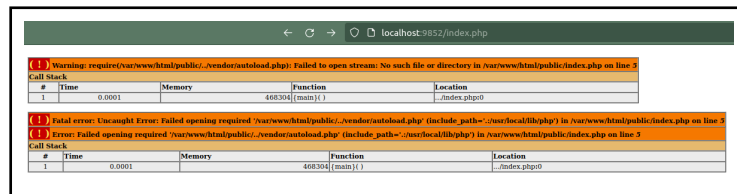


Fig. 2. – Problèmes avec index.php

Pour le corriger, il reste à importer les dépendances avec « Composer » (`composer update`), soit manuellement avec le tty interactif une fois le container lancé, soit en ajoutant l'instruction dans le `Dockerfile`.

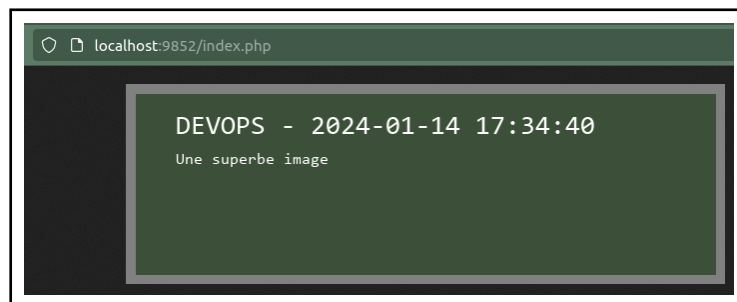


Fig. 3. – index.php fonctionnel après l'ajout des dépendances

Il sera choisi de modifier le `Dockerfile` pour que les dépendances soient déjà correctes lors de l'instanciation d'un container issu de l'image Docker.

2 – Mise en place du pipeline CI/CD

2.1 – Configuration GitHub et CircleCI

Le dépôt étant à présent sur GitHub, nous allons configurer CircleCI, en nous aidant notamment de [ce blog post sur le site de CircleCI](#). Nous nous connectons sur CircleCI avec notre compte GitHub, et il nous est proposé de lier un dépôt.

2.2 — Création du pipeline CI/CD minimal

Certaines des jobs listés dans `.circleci/config.yaml` font appel à des variables d'environnement, certaines n'étant pas encore définies, comme `$GHCR_USERNAME` et `$GHCR_PAT` pour [GitHub Container Registry](#).

Nous avons donc commencé par retirer certains job (`build-docker-image`, `deploy-ssh-staging`, `deploy-ssh-production`) pour s'assurer que les autres fonctionnaient correctement.

Grâce aux informations données par l'étape `Install dependencies` du job `build-setup` qui a échoué sur CircleCI, nous avons pu corriger les versions incohérentes de `php` entre `composer.json` qui nécessitait une version de PHP correspondant à `">=8.2.0"`, `composer.lock` qui n'avait pas été mis à jour avec `composer` à partir de `composer.json`, et `.circleci/config.yaml` qui attendait `php:8.1`.

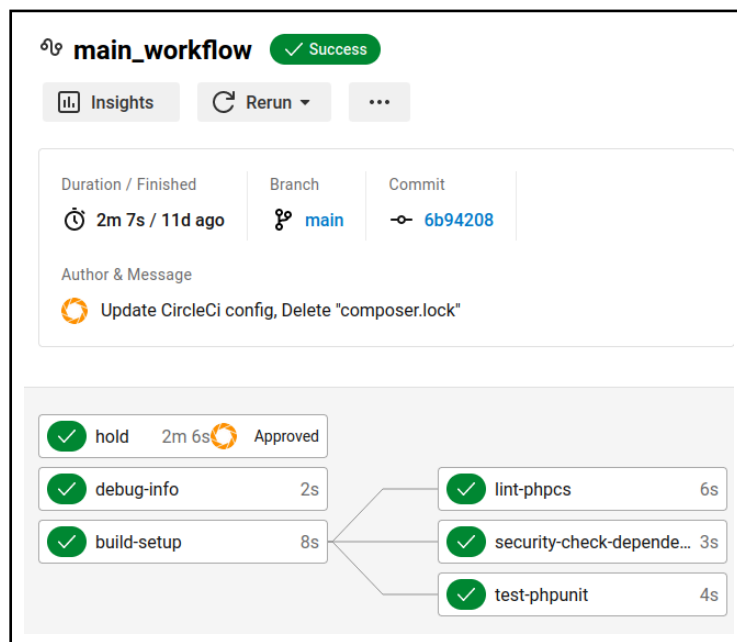


Fig. 4. – Jobs du workflow `main_workflow` se terminant tous avec succès

2.3 — Ajout des variables d'environnement nécessaires

Les variables d'environnement `$CIRCLE_PROJECT_USERNAME`, `$CIRCLE_PROJECT_REPONAME`, `$CIRCLE_BRANCH` et `$CIRCLE_REPOSITORY_URL` étant utilisées, notamment pour en définir d'autres, il convenait de les définir dans la section des variables d'environnement du projet sur CircleCI. Les paramètres du projet sur CircleCI indiquaient qu'aucune variable d'environnement par défaut n'était déclaré. Or, nous avons pu confirmer via les logs de la step `Preparing environment` variables du job `debug-info` des précédents builds, que ces variables (avec d'autres), avaient été définies par défaut, certaines lors de la liaison du projet sur GitHub à CircleCI, d'autres (comme les branches) en fonction du contexte.

2.4 – Gestion des secrets avec Infisical

Nous pensions au début qu'Infisical nous servirait à stocker les secrets utilisés lors des build dans le pipeline dans CircleCI. La suite de cette sous-partie illustre nos réflexions pour parvenir à configurer et stocker dans Infisical les secrets déclarés dans `.circleci/conig.yml` et donc utilisés dans CircleCI, comme `$GHCR_USERNAME` et `$GHCR_PAT` servant aux jobs depuis lesquels sont construits et publiés les images Docker du projet.

2.4.1 – Secrets liés au build dans le pipeline

2.4.1.1 – Idée de base

Le compte Infisical ayant été associé à GitHub, l'intégration fut assez simple. Avec l'aide de l'article d'intégration de CircleCI dans Infisical, le projet a pu être lié. Il restait à lier Infisical dans CircleCI, en utilisant la CLI d'Infisical, ce que [cet article](#) a pu décrire.

L'idée se décomposait comme suit :

1. **Stockage des secrets `$GHCR_USERNAME` et `$GHCR_PAT` dans Infisical.**

`$GHCR_PAT` a été obtenu depuis les *personal access token* dans les paramètres de compte GitHub, et `$GHCR_USERNAME` correspond au nom de l'organisation : ici le dépôt a été publié sous un utilisateur, donc la valeur est le pseudonyme de l'utilisateur.

2. **Génération d'un token de service dans Infisical.**

Nous avons généré un « service token » dans Infisical avec accès en mode lecture seule, avec comme scope « Development », et chemin `/`, sans date d'expiration (mais on pourrait en mettre une et définir des politiques pour les remplacer), qui serait utilisé avec la CLI d'Infisical dans CircleCI avec l'option `--token`.

3. **Stockage sécurisé du token créé dans CircleCI.**

Nous avons pu utiliser les *Contextes* de CircleCI, qui servent à partager des variables d'environnement de manière sécurisée entre les projets, mais que l'on a restraint ici au projet `php-devops-tp` pour définir des variables d'environnement traitées comme des secrets. Définition de cette valeur sous dans un contexte nommé `api_tokens-context`, avec comme nom de variable `INFISICAL_API_TOKEN`.

4. **Invocation des secrets avec l'API d'Infisical dans CircleCI, en utilisant le token pour s'authentifier.**

Les secrets allaient être utiles pour le job `build-docker-image`.

```
if [ -z "$INFISICAL_API_TOKEN" ]; then
  echo "You forgot to set INFISICAL_API_TOKEN variable!"
else
  echo "INFISICAL_API_TOKEN variable is set!"
  echo "Leaking INFISICAL_API_TOKEN value through stdout: '$INFISICAL_API_TOKEN'"
fi
```

Code 2. – Ajout d'un script dans le job `debug info` pour vérifier l'accès à des variables d'environnement protégées dans CircleCI

```
8 INFISICAL_API_TOKEN variable is set!
9 Leaking INFISICAL_API_TOKEN value through stdout: '*****'
```

Fig. 5. – Variable issue du contexte `api_tokens-context` accessible, et masquée automatiquement dans les logs

```
function docker_login() {
  echo "$GHCR_PAT" | docker login ghcr.io -u "$GHCR_USERNAME" --password-stdin
}
infisical run --token "$INFISICAL_API_TOKEN" --env=dev --path=/ -- docker_login
```

Code 3. – Utilisation d'Infisical en mode CLI pour accéder aux secrets dans CircleCI

2.4.1.2 – Exigences liées à l'outil dans l'environnement d'exécution des jobs

Cependant, nous nous sommes rendus compte que la CLI d'Infisical n'était pas présente sur les machines de CircleCI, qui exécutaient les jobs. Il était possible de le télécharger à chaque build de l'image Docker du projet (donc le job où l'outil serait nécessaire pour injecter les secrets), en ajoutant les lignes suivantes dans `.circleci/congify.yml`, issue de [la documentation d'Infisical](#) :

```
curl -Lsf 'https://dl.cloudsmith.io/public/infisical/infisical-cli/setup.deb.sh' | sudo -E bash
sudo apt update
sudo apt install infisical -y
```

Code 4. – Installation de la CLI d'Infisical

Mais cela a un coût en ressources, certes faible, mais existant. Si on devait faire cela pour chaque outil non-natif au système sur lequel le build tourne, les curl/wget ainsi que les temps d'installation cumulés consommeraient beaucoup de ressources.

Il serait plus intéressant de faire tourner le build à partir d'une machine qui contiendrait déjà ces outils, par exemple un conteneur Docker. Les executors déclarés dans `.circleci/congify.yml` sont justement de cette utilité, et une [liste des images utilisables pour différents langages de programmation, etc.](#) peut être trouvée sur le site d'Infisical. Seulement, aucun ne contient infisical, et il faudrait donc en construire une en local, depuis une image DockerInDocker (pour que le job puisse ensuite utiliser docker dedans pour construire l'image de notre projet), image à laquelle il faudrait ajouter les outils liés à PHP (comme on utilise l'exécuteur builder-executor, issu de l'image cimg/php:8.2-node), puis la publier sur *GitHub Container Registry* pour pouvoir éventuellement l'utiliser en tant que contexte de build pour le job build-docker-image.

2.4.1.3 – Réalisation et correction

Mais... tout cela paraissait être beaucoup comparées aux instructions précédentes dans les consignes. Nous avons finalement compris qu'Infisical allait nous servir pour les secrets de l'application en elle-même, ce qui a d'ailleurs été confirmé avec la ligne `RUN ... && apt-get install -y infisical` dans le `Dockerfile`.

Ayant déjà compris l'usage des contextes dans CircleCI, nous avons simplement créé `$GHCR_USERNAME` et `$GHCR_PAT` dans un contexte que nous avons nommé `api_tokens-context`, et invoqué le job `build-docker-image` dans le workflow prévu à cet effet (`container_workflow`) avec ce contexte.

En lançant un build de debug (avant toutes ces réflexions dans Chapitre 2.4.1 —), nous avons pu vérifier que les accès aux secrets étaient effectifs.

```
8 | GHCR_PAT variable is set!
9 | GHCR_USERNAME variable is set!
```

Fig. 6. – Vérification des accès aux secrets d'Infisical depuis un job dans CircleCI

2.4.2 — Secrets liés à l'application

TO[*Mais quels secrets ?*]DO

2.5 — Ajout du job pour construire l'image Docker du projet

2.5.1 — Version de Docker Engine

Nous avons essayé de modifier la version de Docker nécessaire dans `.circleci/config.yaml`, passant de 20.10.23 à 25.0.1, la dernière stable à ce moment, mais celle-ci n'est pas prise en charge dans CircleCI, donc nous avons annulé ce changement.

En remarquant cet avertissement de discontinuation de Docker Engine 20 sur CircleCI, nous avons utilisé le tag default à la place, désignant la dernière version supportée, à savoir Docker Engine 24.

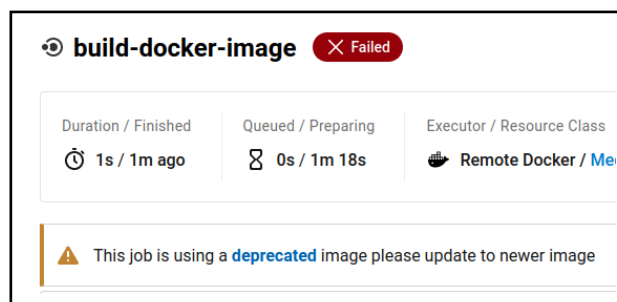


Fig. 7. – Discontinuation de la version 20.10.23 de docker engine par CircleCI

2.5.2 — Nom de répertoire sensible à la casse sous EXT4

Outre cette erreur, lors de l'exécution du job `build-docker-image`, l'étape définie « *Build and Push Docker Image to GHCR (GitHub Container Registry)* » a soulevé l'alerte ci-dessous, et ce, alors que la commande `echo "$GHCR_PAT" | docker login ghcr.io -u "$GHCR_USERNAME" --password-stdin` était utilisée.

WARNING! Your password will be stored unencrypted in `/home/circleci/.docker/config.json`.

TO[*S'occuper de cette erreur*]DO

Cela est dû au fait que Docker va conserver les credentials dans `$HOME/.docker/config.json`, avec un simple encodage en base 64, c'est-à-dire aucune protection concrète.

De même, dans le job `build-docker-image` était spécifié un répertoire au nom erroné pour le `Dockerfile`, à savoir `docker/`, alors que sa recherche est sensible à la casse sous le système de fichiers ext4 (utilisé par la plupart des distributions Linux). Comme il était référencé à quelques endroits dans le projet par `docker/` et qu'il est courant de voir des noms de répertoire en minuscules, le répertoire a été renommé. Cette remarque est importante, car toutes les commandes exécutées sur le répertoire préalablement inexistant `docker/` vont échouer.

2.5.3 — Scope des tokens d'autorisation GitHub

Une autre erreur est survenue

denied: permission_denied: The token provided does not match expected scopes.

La scope du token donné manquait effectivement de la permission d'écriture : sans ça, pas de possibilité de publier des images Docker via ce token sur *GitHub Container Registry*.

Une fois un token avec les bonnes permissions régénéré depuis GitHub, et configuré sous CircleCI, le job pouvait s'exécuter correctement jusqu'au bout.

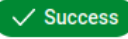
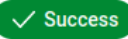
Status	Workflow
▶  Success	container_workflow
▶  Success	main_workflow

Fig. 8. – Workflow `container_workflow` qui fonctionne enfin

Après avoir observé la présence d'un package sur <https://ghcr.io/stanleydinne/php-devops-tp>, nous avons pu le lier au dépôt `php-devops-tp`.

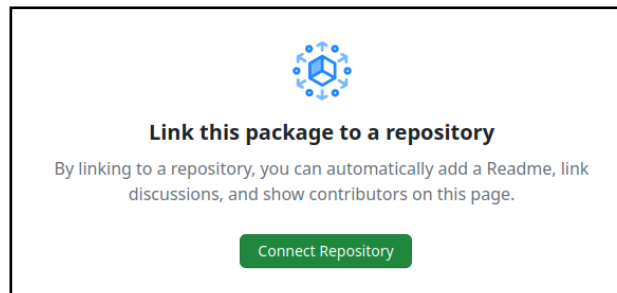


Fig. 9. – Proposition de liaison de l'artefact au dépôt qui lui correspond sur GitHub

3 – Extension du Pipeline

Certains espaces de discussion sur [CircleCI Discuss](#) comme [un système de build et tests basé autour de PHP](#) nous ont permis d'avoir un aperçu de ce qui avait déjà été fait comme configuration autour de PHP dans CircleCI.

3.1 – Ajout de jobs dévaluation de code

Nous avons suivi le [README.md](#) du dépôt `phpmetrics/PhpMetrics`, et configuré `PhpMetrics` dans le job `TO[quel job]DO`.

`TO[https://phpmetrics.org/]DO`, `TO[https://github.com/phpmetrics/PhpMetrics]DO`,
`TO[https://phpqa.io/projects/phploc.html]DO`, `TO[https://discuss.circleci.com/t/my-php-based-build-and-test-systems-on-circleci/17584]DO`

3.2 – Intégration de la qualité du code

3.3 – Déploiement automatisé sur AWS EC2