

# 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

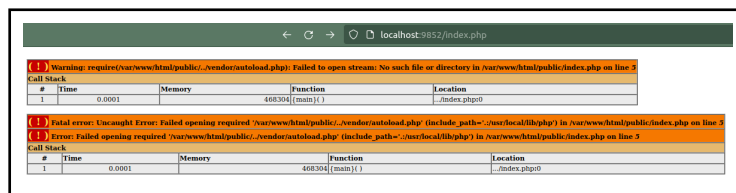


Fig. 2. – Problèmes avec index.php

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`.

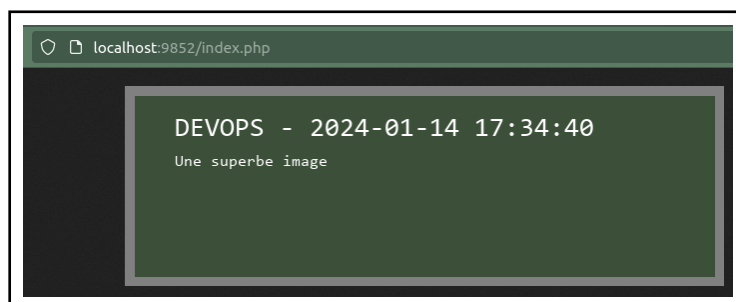


Fig. 3. – index.php fonctionnel après l'ajout des dépendances

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`.

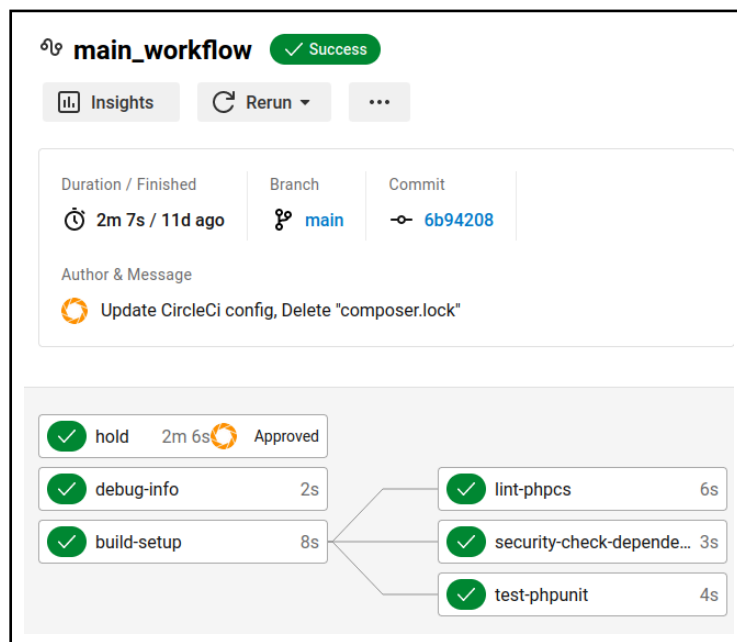


Fig. 4. – Jobs du workflow `main_workflow` se terminant tous avec succès

## 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 déclarée. 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 -sLf '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.

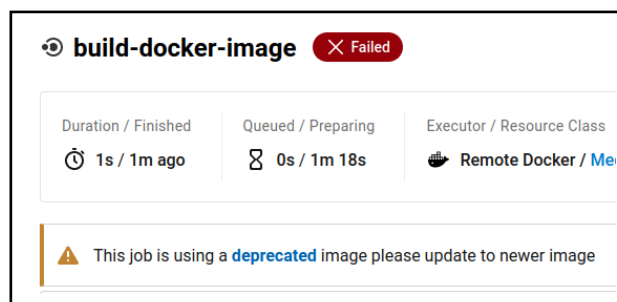


Fig. 7. – Discontinuation de la version 20.10.23 de docker engine par CircleCI

### 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.

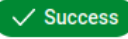
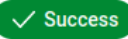
Status	Workflow
▶  Success	container_workflow
▶  Success	main_workflow

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.

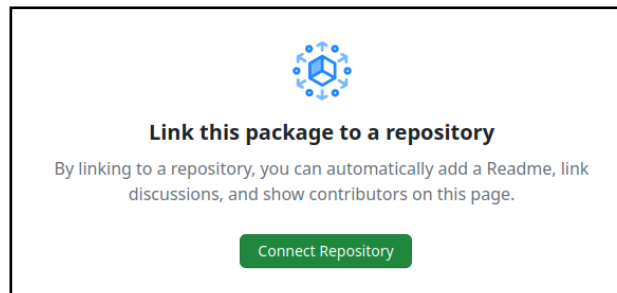


Fig. 9. – Proposition de liaison de l'artefact au dépôt qui lui correspond sur GitHub

#### 2.5.4 – Job pour construire les images Docker optimisé

Lorsque de la documentation seule est ajoutée au dépôt, les jobs se lancent : ce n'est pas souhaitable **TO**[ Vérifier comment changer ça ]**DO**

#### 2.5.5 – Tag des images pour un gestion des version basique

Utilisation du tag du commit comme prefix du nom de l'image envoyée sur GHCR : comme il est unique au sein du dépôt, aucun soucis. Si aucun tag n'est utilisé, le nom par défaut donné devient "\$ {branche}\_\${date}", comme main\_1970-01-01.

```

TAG="$CIRCLE_TAG"
if [[ -z "$TAG" ]]; then
  echo "Tag not set on commit: branch name + today's date used for tag. E.g. \"main_1970-01-01\""
  TAG="$(echo $CIRCLE_BRANCH | tr '[:upper:]' '[:lower:]' | tr '/' '-' | tr -cd '[:alnum:]._-' | cut -c 1-128)"
  TAG="$TAG_$(date +%Y-%m-%d)"
fi

```

Code 5. – Définition du préfixe du nom de l'image pour un versionning basique

### 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.

On va tester les extensions sur le container en local avant de mettre les commandes dans la config qui va être utilisée dans le pipeline. On rentre dans le container local avec `docker start php-devops-tp_container`; `docker exec --interactive --tty php-devops-tp_container /bin/bash`

Selon ChatGPT : **TO** [ Les termes « évaluation de code » et « qualité de code » sont souvent utilisés dans le contexte du développement logiciel pour désigner différents aspects de l'analyse du code source. Voici une explication de la différence entre ces deux concepts :

1. *Évaluation de code* : L'évaluation de code se concentre généralement sur des métriques et des mesures quantitatives pour évaluer différents aspects du code source. Cela peut inclure des mesures telles que la complexité cyclomatique, le nombre de lignes de code, le nombre de fonctions ou de méthodes, la profondeur d'héritage, etc. L'objectif de l'évaluation de code est souvent de fournir une vue d'ensemble de la structure et de la taille du code, ainsi que d'identifier les zones potentielles de complexité ou de problèmes de performance.
2. *Qualité de code* : La qualité de code se réfère à la mesure de la « qualité » globale du code source en termes de lisibilité, maintenabilité, extensibilité et robustesse. Cela peut inclure des aspects tels que la conformité aux bonnes pratiques de programmation, la cohérence du style de code, la gestion des erreurs, la documentation, la facilité de compréhension du code par d'autres développeurs, etc. L'objectif de l'évaluation de la qualité du code est souvent d'identifier les problèmes potentiels qui pourraient affecter la maintenabilité à long terme et la robustesse du logiciel.

En résumé, l'évaluation de code se concentre sur des mesures quantitatives et des métriques spécifiques liées à la structure et à la taille du code, tandis que la qualité de code se concentre sur des aspects plus qualitatifs liés à la lisibilité, à la maintenabilité et à la robustesse du code. Les outils d'évaluation de code et de qualité de code peuvent fournir différents types de analyses pour aider à améliorer différents aspects du développement logiciel. ]**DO**

### 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://discuss.circleci.com/t/my-php-based-build-and-test-systems-on-circleci/17584> ] **DO**

#### 3.1.1 — phpmetrics

**TO** [ <https://phpmetrics.org/> ] **DO** , **TO** [ <https://github.com/phpmetrics/PhpMetrics> ] **DO**

Les instructions sur quelle commande `composer` utiliser étaient présentes sur le [GitHub de phpmetrics](#).

Avec `composer` require --dev "`phpmetrics/phpmetrics=*`" (option --with-all-dependencies pas utilisée ici), on update le fichiers de dépendances `composer.json`, ainsi que `composer.lock` et installe les dépendances demandées. Avec `composer` update, on peut installer les autres dépendances de `composer.json` qui manquent.

Maintenant `./vendor/bin/phpmetrics` est créé, on peut faire

```
php ./vendor/bin/phpmetrics --report-html=myreport src/
chown www-data myreport
mv myreport public/myreport
```

, puis dans le navigateur de la machine hôte, accéder à <http://127.0.0.1:9852/myreport/index.html>.



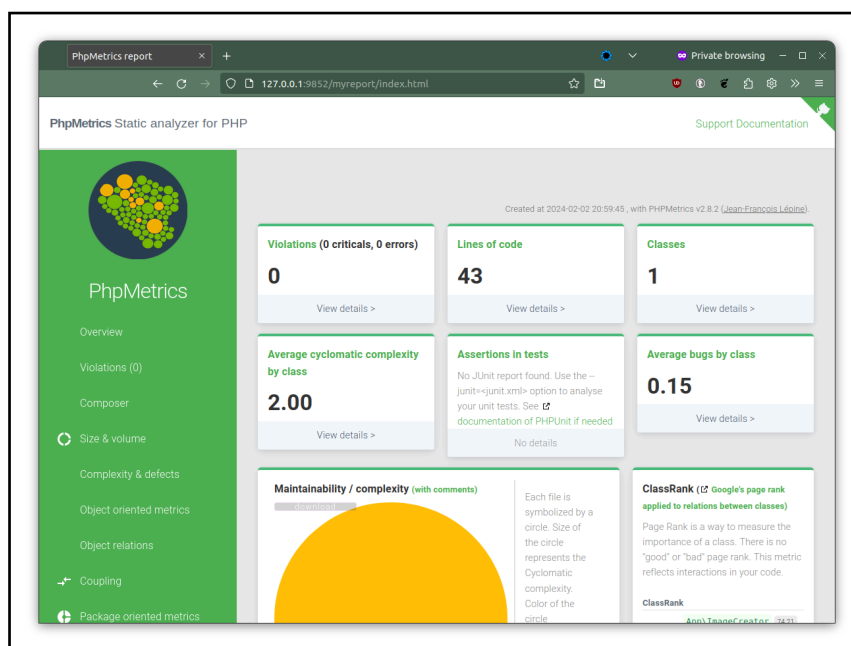


Fig. 10. – Accès à la page du rapport de phpmetrics, depuis un container local

On refait le rapport en analysant tous les fichiers `rm -r public/myreport/ && php ./vendor/bin/phpmetrics --report-html=public/myreport . && chown www-data public/myreport` Ça change rien.

**TO** [ Peut-être mettre une configuration comme ça: <https://phpmetrics.github.io/website/configuration/> ] **DO**

### 3.1.2 – phploc

Comme le suggère le [README.md](#) de `phploc`, l'installation ne se fera pas avec `composer`, mais plutôt en installant le `.phar` (PHP Archive) de l'outil :

```
composer require --dev phploc/phploc
php vendor/bin/phploc
wget https://phar.phpunit.de/phploc.phar
php phploc.phar src/
```

Ça va faire un rapport statique sur la taille des fichiers, les dépendances, la complexité, etc. et ça va l'afficher dans `stdout`.

## 3.2 – Intégration de la qualité du code

De même, toujours en local.

### 3.2.1 – phpmd

<https://github.com/phpmd/phpmd> <https://phpmd.org/>

En extrapolant un peu depuis <https://phpmd.org/download/index.html>, on utilise cette commande pour installer l'outil.

```
composer require --dev "phpmd/phpmd=@stable"
```

Puis l'utilisation



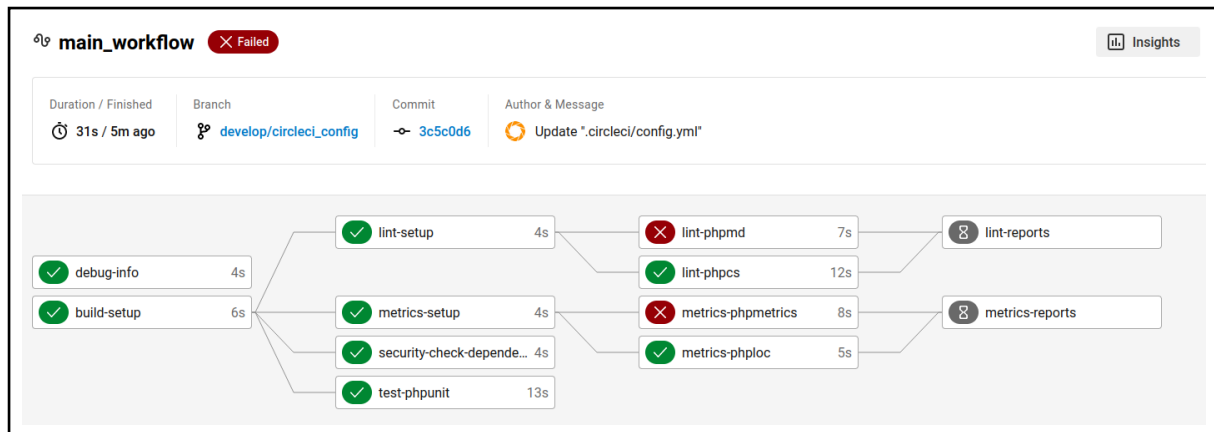


Fig. 12. – Jobs centralisant les rapports, qui se sont pas atteints si leur prédécesseurs ne réussissent pas

```
- lint-reports:
  requires:
    - lint-phpcs
    - lint-phpmd
```

Code 6. – Configuration des dépendances d'un job dans le workflow main\_workflow

On peut le voir dans la figure Fig. 12 ci-dessus, mais les jobs configurés comme en nécessitant d'autres avec la directive `requires`, ne vont pas être atteints si au moins un des jobs desquels il dépend échoue. Cela est problématique dans notre cas, étant donné que l'on voudrait pouvoir stocker les rapports, qu'ils proviennent de job ayant échoués aux attentes ou les ayant respectées.

Grâce à [cette discussion sur les forum de CircleCI](#), nous avons pu trouver une alternative et configurer ainsi nos pipelines. En utilisant l'[API de CircleCI](#), on vérifie quand un certain job est terminé, et tant qu'il ne l'est pas, le job boucle. Cette méthode n'est pas optimale car elle consomme du temps d'exécution en plus dans CircleCI.

### 3.4 – Déploiement automatisé sur AWS EC2

La commande ci-dessous est utilisée dans le job `deploy-ssh-staging` pour mettre à jour le code source de l'application sur l'instance de la machine AWS distante, installer les dépendances PHP et redémarrer le serveur FPM PHP pour appliquer les changements.

```
ssh -o StrictHostKeyChecking=no $STAGING_SSH_USER@$STAGING_SSH_HOST \<< EOF
PHP_FPM_VERSION=$(php -v | head -n 1 | cut -d ' ' -f 2 | cut -d '.' -f 1-2)
cd $STAGING_DEPLOY_DIRECTORY
git pull origin $CIRCLE_BRANCH
composer install --optimize-autoloader --no-interaction --prefer-dist
(flock -w 10 9 || exit 1; sudo -S service php${PHP_FPM_VERSION}-fpm restart ) 9>/tmp/
fpm.lock
EOF
```

**TO** [ Suppression du job `hold` car le but est d'automatiser, alors si on doit se connecter sur CircleCi pour approuver à chaque fois les déploiements... Il faut cependant faire quelque chose de sécurisé ] **DO**

#### 3.4.1 – Configuration sur AWS

L'étape 2 de cette section est laissée à titre informatif pour retracer nos essais, mais elle n'a aucune utilité finalement. L'objectif était de créer un autre compte `updater_agent` qui n'aurait que le droit de

se connecter à l'instance. Mais après avoir lancé l'instance à l'étape d'après (Cf Chapitre 3.4.2 —), il se trouve qu'il suffit de créer un utilisateur sur la machine une fois accédée via SSH.

1. Premièrement, nous allons nous créer un compte AWS en utilisant les free tiers proposés.

1. **TO** *Mettre les screenshot des steps* **DO**

2. On se log en tant que Root user

1. Grâce à [https://docs.aws.amazon.com/organizations/latest/userguide/orgs\\_introduction.html](https://docs.aws.amazon.com/organizations/latest/userguide/orgs_introduction.html), on crée une organisation

2. Ensuite, en s'aidant de <https://circleci.com/docs/deploy-to-aws/#create-iam-user> et de <https://aws.amazon.com/iam/features/manage-users/> et de <https://docs.aws.amazon.com/signin/latest/userguide/introduction-to-iam-user-sign-in-tutorial.html>, on crée un « compte IAM » sans accès root, qui va mettre d'effectuer les updates mentionnés ci-dessus.

- Identifiant updater\_agent

3. Activation des « Service Control Policies » (SCP) Définition de politiques « Service Control Policies » (<https://us-east-1.console.aws.amazon.com/organizations/v2/home/policies/service-control-policy>)

- Définition d'une politique Simple machine connection Policy

```
{ "Effect": "Allow",
  "Action": [
    "ec2:Connect",
    "ec2:DescribeInstances",
    "ec2:DescribeInstanceStatus",
    "ec2:DescribeKeyPairs"
  ],
  "Resource": "*" }
```

- Définition d'une politique DenyEverything

```
{ "Sid": "DenyEverything",
  "Effect": "Deny",
  "Action": "*",
  "Resource": "*" }
```

4. (Liaison automatique des politiques à l'organisation)

5. Liaison manuelle de la politique Simple machine connection Policy au compte updater\_agent

6. Liaison manuelle de la politique pré-existante FullAWSAccess au compte par défaut php-devops-tp\_account

7. Liaison manuelle de la politique DenyEverything au groupe Root

8. Révocation de la politique pré-existante FullAWSAccess du groupe Root

Maintenant updater\_agent peut accéder aux machine et s'y connecter (sous réserve de configuration), et php-devops-tp\_account est toujours administrateur.

3. On reset le mot de passe de l'utilisateur updater\_agent car on ne sait pas comment avoir les mots de passe.. **TO** *IDK ??? Ok on va juste* **DO**

### 3.4.2 — Création des deux machines staging et production

#### 3.4.2.1 — Côté AWS

1. On se rend sur <https://eu-north-1.console.aws.amazon.com/ec2/home>

2. On crée une instance

- Ubuntu Server 22.04
- type « t3.micro » (dans le free tier)

- avec une création d'un nouveau « security group » qui autorise les connexions SSH (justement pour que l'agent puisse faire des updates du software)
  - Il faut aussi autoriser les connexions HTTPS entrantes, pour que l'on puisse accéder au service.
  - 15 Go de SSD
  - une keypair temporaire pour se connecter une fois à l'instance
3. On télécharge le fichier `.pem` à mettre dans le `~/.ssh` de notre machine personnelle
    - `chmod 400 AWS_DevSecOps_staging.pem`
  4. Sur l'onglet de connexion à l'instance sur la console AWS, une commande nous est proposée pour se connecter à la machine (modifiée légèrement pour indiquer qu'elle est dans `~/.ssh`)
    1. `ssh -i "~/.ssh/AWS_DevSecOps2_default.pem" ubuntu@ec2-51-20-87-132.eu-north-1.compute.amazonaws.com`
  5. Une fois dans la machine, on crée un utilisateur avoir des droits réduits grâce à <https://www.digitalocean.com/community/tutorials/how-to-add-and-delete-users-on-ubuntu-20-04> et <https://linuxize.com/post/how-to-delete-group-in-linux/> : `sudo adduser updater_agent`
    - Suivre les étapes en ne définissant que le mot de passe
    - En fait non, il faut lui donner les droits d'administrateur pour qu'il puisse recharger le service php8.2 après avoir chargé le code.
    - `sudo usermod -aG sudo updater_agent`
  6. Changement d'utilisateur pour impersonner `updater_agent` : `su updater_agent`
  7. Création d'une paire de clefs ssh (pour une connexion depuis le pipeline sur CircleCI) : `mkdir -p ~/.ssh && cd ~/.ssh && ssh-keygen -t ed25519 -C "updater_agent"` (Fichier nommé `circleci.key`)
  8. Autorisation de connexion pour éviter l'erreur `updater_agent@51.20.92.240: Permission denied (publickey)`.
    - `cat ~/.ssh/circleci.key.pub > ~/.ssh/authorized_keys`
  9. Autorisation de l'utilisateur à utiliser certaines commandes sans mot de passe (lors du script d'update en ssh automatisé)
    - Pour le serveur de staging

`sudo visudo`

`updater_agent ALL=(ALL) NOPASSWD: /usr/bin/rm`

`updater_agent ALL=(ALL) NOPASSWD: /usr/bin/cp`

`updater_agent ALL=(ALL) NOPASSWD: /usr/sbin/service`

- Pour le serveur de production `updater_agent ALL=(ALL) NOPASSWD: /usr/bin/docker`

1. Copie de la clef privée qui se trouve dans `~/.ssh/circleci.key`

2.

```
sudo apt update
sudo apt upgrade -y
[ -f /var/run/reboot-required ] && sudo reboot -f

# After reboot
sudo add-apt-repository ppa:ondrej/php # To install php8.2
sudo apt install -y curl git php8.2 libapache2-mod-php8.2 php8.2-fpm
sudo apt install -y php8.2-gd php8.2-xml php8.2-mbstring # pour composer
curl -sS https://getcomposer.org/installer | sudo php -- --install-dir=/usr/local/bin --filename=composer
sudo a2enmod rewrite
sudo mkdir -p /var/www/html && cd /var/www
sudo rm -r html/
sudo git clone https://github.com/StanleyDINNE/php-devops-tp
sudo mv php-devops-tp html && cd html
sudo sed -i 's!/var/www/html!/var/www/html/public!g' /etc/apache2/sites-available/000-default.conf
# git checkout develop # TODO
sudo systemctl restart apache2
composer install --optimize-autoloader --no-interaction --prefer-dist
(flock -w 10 9 || exit 1; sudo -S service php8.2-fpm restart ) 9>/tmp/fpm.lock
# sudo rm -r .circleci .idea .vscode docker documents .dockerignore .env.example
```

Code 7. – Installation des services nécessaires, notamment ce qui est fait dans `docker/Dockerfile`

## TO[ *Créer une branche develop* ]DO

- Si on le faisait avec Docker (pour la prod), on aurait juste à réutiliser l'image construite et l'instancier en un container

Cette configuration manuelle est sujette à être bancale si des packets manquent, etc. L'un des but de la containerisation avec Docker est d'avoir des images déjà toutes configurées, et que l'exécution soit reproductible.

Ainsi, pour le serveur de production, toujours sur Ubuntu 22.04, nous avons installé Docker engine grâce [au tutoriel pour Ubuntu sur docker.com](#). Après l'installation, et le setup de l'utilisateur updater\_agent comme pour la machine Staging (avec mot de passe différent, setup de la keypair en mettant la clef privée dans les contextes de CircleCI), il suffisait de récupérer l'image depuis notre dépôt GitHub avec

```
sudo docker pull ghcr.io/stanleydinne/php-devops-tp:latest
sudo docker run --detach --publish 80:80 --name "php-devops-tp_latest_container"
"ghcr.io/stanleydinne/php-devops-tp:latest"
```

Il fallait aussi faire attention que le package soit public sur GHCR.io.

Nous n'allons pas essayer de configurer un certificat pour exposer notre service en HTTPS, mais il faudrait. Si on l'avait fait, il aurait fallu configurer le firewall iptables en ajoutant ça : `sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT`, ce qui va permettre de rendre accessible le service provenant du conteneur Docker sur le port 443 de la machine AWS.

## TO[ *Faire la branche de production* ]DO

### 3.4.2.2 – Côté CircleCI

Grâce à <https://circleci.com/docs/add-ssh-key/>

1. On se rend sur les paramètres du projet
2. Section « SSH Keys »
3. Ajout d'une clef SSH
  - On met comme hostname `staging.aws` (juste pour distinguer)
  - On colle la clef privée copiée de la section précédente Chapitre 3.4.2.1 –
4. On obtient la signature (SHA256: ... )
  - On peut utiliser ça en tant que valeur pour `$STAGING_SSH_FINGERPRINT`, qui sera utilisé dans `.circleci/config.yml`
5. On stocke `$STAGING_SSH_FINGERPRINT="SHA256:..."` en tant que variable d'environnement simple
  - On aurait même pu l'hardcoder dans le `.circleci/config.yml`, mais autant centraliser ce genre de données directement dans CircleCI, et ne pas laisser ça public
6. On fait de même avec `$STAGING_SSH_USER` et `$STAGING_SSH_HOST`, respectivement définis comme `updater_agent` et `staging.aws`

## 3.5 – Modification des flows pour définir des politiques

TO[ *L'idée est de faire des jobs pour agglomérer les reports des différents jobs de metrics, et de lint* ]DO

TO[ *On modifie les flows pour établir différentes politiques:*

*On veut par exemple que les déploiements ne se fassent que sur depuis la branche main, et pour les tags « release\* »*

On veut également ne déployer sur AWS que les containers pour la prod et le staging, issus des images générés du code respectivement présent sur les branches main et develop ]DO

4 — ++

TO[ Dire que le stockage: « 33.4 MB of 2 GB used » est dû aux reports, et qu'il aurait peut-être fallu les gérer dans un répertoire /tmp, et établir une procédure de gestion des reports, d'ailleurs, ceux-ci sont stockés sur AWS, mais de CircleCI (en cliquant sur un report, on est redirigé vers une des machines dédiées sur AWS pour CircleCI)]DO

TO[ Mettre des screenshots des flows)]DO

Désactivation de la branche master dans les condition de création d'images docker

```
filters:
branches:
  only:
    # - master
```

Car vu que la branche principale est main, si quelqu'un fork le repo', crée un branche master, mets du code contaminé, et merge ça sur le dépôt principal, l'image sera créée avec le job.

TO[ Suppression de l'utilisateur de l'option -o StrictHostKeyChecking=no avec ssh, au vu des commentaires dans <https://www.howtouselinux.com/post/ssh-strict-host-key-checking-option>)]DO Mais ça permet de skip

```
The authenticity of host '***** (*****)' can't be established.
ECDSA key fingerprint is ....
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

--

TO[ <https://github.com/orgs/community/discussions/24963> : Deleting a package version for a container on ghcr.io)]DO

D'ailleurs sur la config' SSH par défaut des machines AWS, les connexions SSH où l'utilisateur s'authentifie manuellement avec un mot de passe sont bloquées. Il faut que les utilisateurs utilisent leur clef privée. C'est bien, ça évite de donner la possibilité d'exploiter une attaque en SSH enumeration + brute force le mot de passe.

TO[ docker -e (pour mettre la variable d'environnement INFISICAL

Définir soit-même un secret (regarder le .env.example)]DO