



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

Le dépôt se trouve à l'adresse : <https://github.com/StanleyDINNE/php-devops-tp>

## Sommaire

1 Compréhension et configuration de base .....	2
1.1 Analyse du projet PHP .....	2
1.2 Configuration de l'environnement Docker .....	2
2 Mise en place du pipeline CI/CD .....	3
2.1 Configuration GitHub et CircleCI .....	3
2.2 Création du pipeline CI/CD minimal .....	4
2.3 Ajout des variables d'environnement nécessaires .....	4
2.4 Gestion des secrets avec Infisical .....	5
2.4.1 Secrets liés au build dans le pipeline .....	5
2.4.2 Secrets liés à l'application .....	7
2.5 Ajout du job pour construire l'image Docker du projet .....	7
2.5.1 Gestion des détails lors de la configuration du fichier <code>.circleci/config.yml</code> .....	7
2.5.2 Scope des tokens d'autorisation GitHub .....	8
2.5.3 Tag des images pour une gestion des version basique .....	8
3 Extension du Pipeline .....	9
3.1 Ajout de jobs dévaluation de code .....	9
3.1.1 phpmetrics .....	9
3.1.2 phplloc .....	10
3.2 Intégration de la qualité du code .....	10
3.2.1 phpmnd .....	10
3.2.2 niels-de-blaauw/php-doc-check .....	11
3.3 Intégration des outils dans le pipeline .....	11
3.4 Déploiement automatisé sur AWS EC2 .....	14
3.4.1 Configuration sur AWS : premiers essais .....	14
3.4.2 Instanciation des deux machines côté AWS .....	14
3.4.3 Création d'un utilisateur dédié aux mises à jour de l'app' sur chaque machine .....	15
3.4.4 Configuration des deux machines staging et production .....	16
3.4.5 Côté CircleCI .....	17
3.5 Modification des flows pour définir des politiques .....	18
3.6 Quelques figures du déploiement .....	18
4 Sécurité à tous les étages : idées pour plus de sécurité tout au long du processus de déploiement ..	20
5 Annexes .....	21
5.1 Configuration d'un « Utilisateur IAM » sur AWS .....	21

# 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 librairie 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 la 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

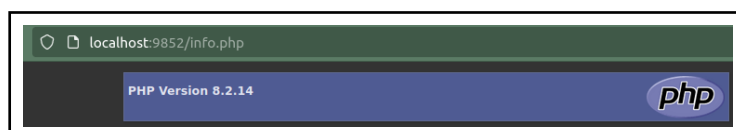


Fig. 1. – Accès à la page `info.php` du container fonctionnel et accessible

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

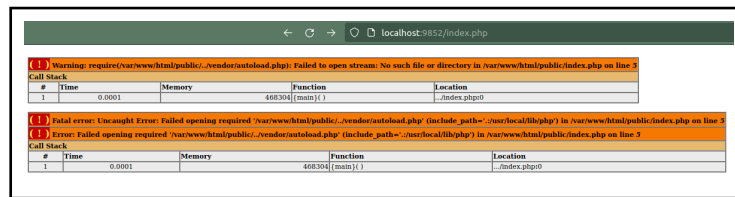


Fig. 2. – Problèmes avec index.php

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`.

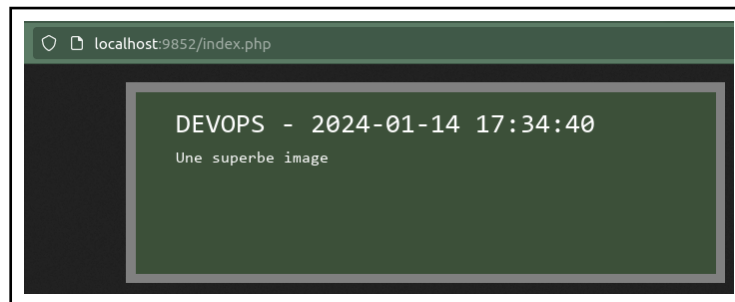


Fig. 3. – index.php fonctionnel après l'ajout des dépendances

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`.

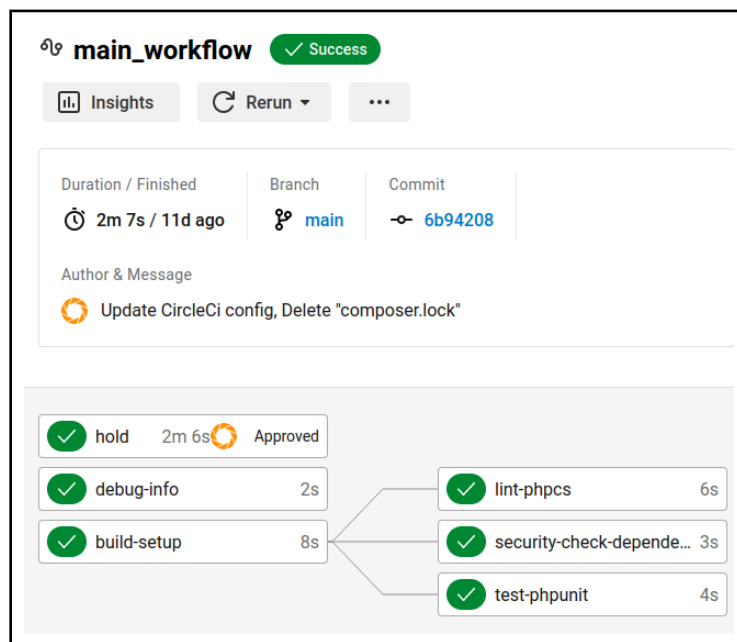


Fig. 4. – Jobs du workflow `main_workflow` se terminant tous avec succès

## 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.

À la fin de toutes les configurations de ce rapport, voici un accès au token, qui peut être utilisé pour l'injection de commandes dans l'application

```
updater_agent@ip-172-31-42-172:~$ sudo docker exec --interactive --tty php-devops-tpplatestcontainer /bin/bash
root@b81e6061615d:/var/www/html# echo $INFISICAL_API_TOKEN
st.65[REDACTED]9f
root@b81e6061615d:/var/www/html# exit
```

Fig. 5. – Récupération réussie du token Infisical depuis le container

### 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. 6. – 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 comparé 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.

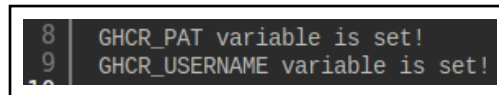


Fig. 7. – Vérification des accès aux secrets d'Infisical depuis un job dans CircleCI

(Avec le recul lors des dernières modification de ce rapport, il a paru beaucoup plus évident qu'il aurait été possible de gérer les secrets du build avec Infisical. Le choix de ne pas le faire expliqué par les raisons ci-dessus dans le Chapitre 2.4.1.2 et Chapitre 2.4.1.3, sont des choix faits relativement tôt (début janvier 2024) et par soucis de ne pas rajouter de complexité inutile à peu de temps du rendu, celui-ci persistera.)

## 2.4.2 – Secrets liés à l'application

### TO[ Configurer le secret avec Infisical pour une injection dans le build ]DO

Considérons que notre application (qui là n'est qu'une application *placeholder*) utilise un secret `$APP_SECRET`. On peut configurer l'injection du token d'Infisical `$INFISICAL_API_TOKEN` en tant que variable d'environnement au moment du lancement du container, avec `sudo docker run -e INFISICAL_API_TOKEN=""$INFISICAL_API_TOKEN" ...`, pour qu'à l'intérieur du container, l'application `a_given_app` puisse utiliser Infisical grâce à `infisical run --token "$INFISICAL_API_TOKEN" --env=dev --path=/ -- a_given_app`. L'application n'utilise pas de secret particulier pour l'instant, mais les tests sur l'injection de secrets avec Infisical en ligne de commande ont été concluants.

## 2.5 – Ajout du job pour construire l'image Docker du projet

### 2.5.1 – Gestion des détails lors de la configuration du fichier `.circleci/config.yml`

Nous avons essayé de modifier la version de Docker nécessaire dans `.circleci/config.yml`, 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.

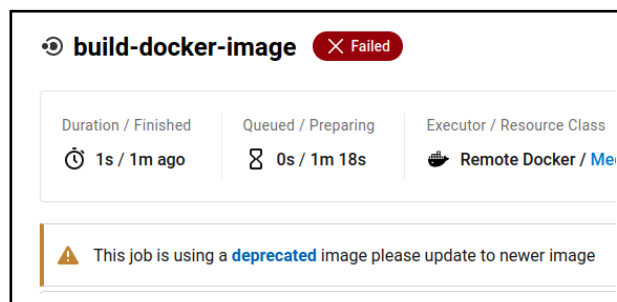


Fig. 8. – Discontinuation de la version 20.10.23 de docker engine par CircleCI

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`.

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. Comme ces fichiers ne sont pas persistants sur les machines dédiées à l'exécution des pipelines sur CircleCI, nous n'avons pas

pris de mesure supplémentaire pour gérer cette alerte, mais dans l'idéal, il faudrait trouver un moyen régler ça.

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.

### 2.5.2 – 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	<code>container_workflow</code>
▶  Success	<code>main_workflow</code>

Fig. 9. – 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`.

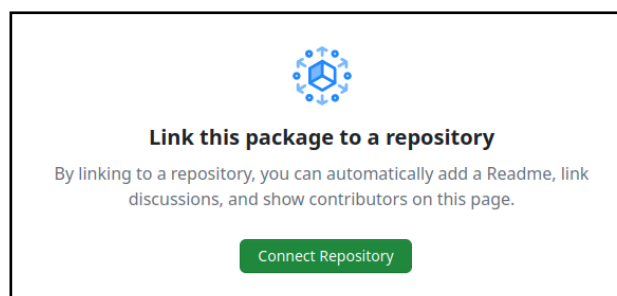


Fig. 10. – Proposition de liaison de l'artefact au dépôt qui lui correspond sur GitHub

### 2.5.3 – Tag des images pour une gestion des version basique

Nous avons configuré une gestion basiques des versions des images docker, avec l'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

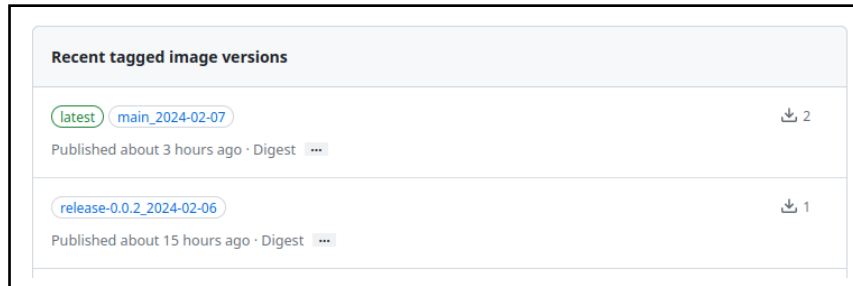


Fig. 11. – Tag et versionning des images docker sur GHCR

### 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.

Nous avons commencer par 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-devoops-tp_container`; `docker exec --interactive --tty php-devops-tp_container /bin/bash`.

Pour être clair, les termes « *évaluation de code* » et « *qualité de code* » sont respectivement des évaluations quantitatives et des évaluations qualitatives du code.

Il y a eu beaucoup de problèmes d'installation avec `composer` sur le pipeline, comme c'était la première fois que nous l'utilisions, et nous avons finalement réussi à régler les incompatibilités, comprendre les packages systèmes dont certaines extensions dépendent, etc.

#### 3.1 – Ajout de jobs dévaluation de code

##### 3.1.1 – phpmetrics

Nous avons suivi le [README.md](#) du dépôt [phpmetrics/PhpMetrics](#), et configuré PhpMetrics dans le job `metrics-phpmetrics` : Leur site donnait un aperçu du fonctionnement et de la logique de l'outil. 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 ins-`

talle les dépendances demandées. Avec `composer update && composer install`, on peut installer phpmetrics ainsi que 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>.

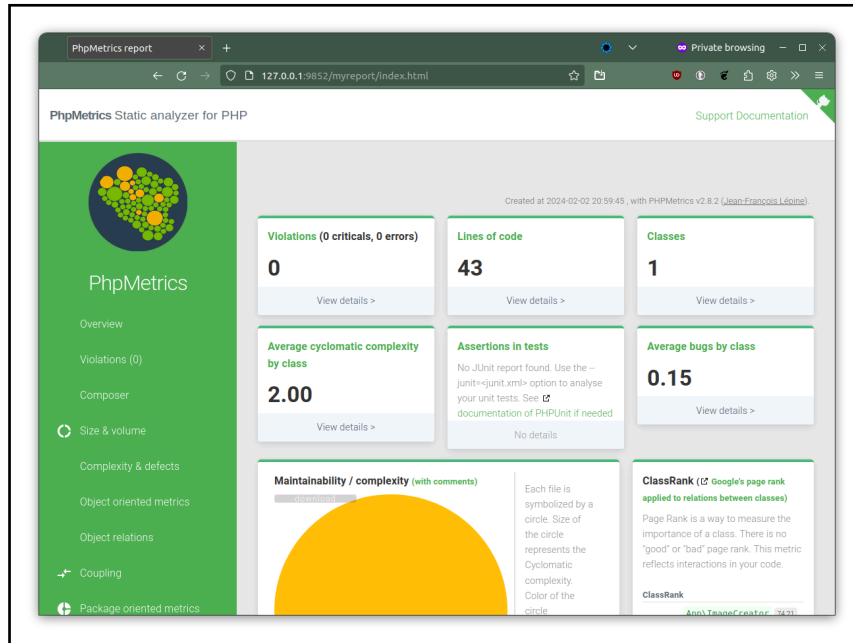


Fig. 12. – Accès à la page du rapport de phpmetrics, depuis un container local

Il existait des configurations mais nous ne nous en sommes pas servis.

### 3.1.2 — phplloc

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

```
composer require --dev phplloc/phplloc && php vendor/bin/phplloc src/
```

```
wget https://phar.phpunit.de/phplloc.phar
```

```
php phplloc.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

### 3.2.1 — phpmd

En extrapolant un peu depuis [la page d'installation](#), on utilise ces commandes pour installer puis utiliser l'outil.

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

```
php ./vendor/bin/phpmd src/ html .circleci/rulesets.xml > phpmd-report.html
```

Nous avons pris le `.circleci/plugins/phpmd_ruleset.xml` depuis [le GitHub de PHPMD](#)

Le job `lint-phpmd` échoue, mais cela est « contrôlé » : en effet, nous avons configuré le job pour échouer en réutilisant les codes d'erreur de la commande. Dans le cas actuel, il existe certaines violations correspondantes au fichier de configuration `.circleci/plugins/phpmd_ruleset.xml` qui persistent dans le code PHP de l'application, comme montré dans la Fig. 13.

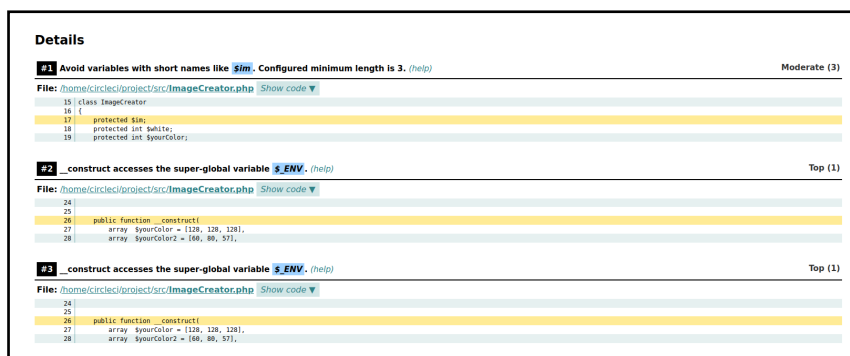


Fig. 13. – Problèmes relevés par PHPMD qui font échouer le job

Nous n'avons pas cherché à corriger le code, car ce n'était pas l'objectif premier de ce travail, et nous préférons allouer plus de temps à la configuration du pipeline, et des connexions aux machines AWS.

### 3.2.2 — niels-de-blaauw/php-doc-check

Même en suivant les instructions du [dépôt GitHub de php-doc-check](#) ou du [site](#), aucune des installations avec les commandes suivantes ne semblait fonctionner :

```
composer require --dev "niels-de-blaauw/php-doc-check=*"
php ./vendor/bin/php-doc-check src
# Ou alors
curl -sSL https://github.com/NielsdeBlaauw/php-doc-check/releases/download/v0.2.2/php-doc-check.phar -o php-doc-check.phar
php php-doc-check.phar src
```

Code 6. – Essais d'installation de php-doc-check

```
Deprecated: Return type of GetOpt\GetOpt::getIterator() should be Traversable, or the #[\ReturnTypeWillChange] attribute should be used to suppress the notice in phar:///var/www/html/php-doc-check.phar on line 503

Call Stack:
 0.0022 1617472 1. {main}() /var/www/html/php-doc-check.phar:0
 0.0069 1951416 2. require('phar:///var/www/html/php-doc-check.phar') /var/www/html/php-doc-check.phar:33
 0.0077 2116560 3. NdB\PhpDocCheck\ApplicationArgumentsProvider->getArguments() /var/www/html/php-doc-check.phar/bin/php-doc-check:10
 0.0077 2116560 4. Composer\Autoload\ClassLoader->loadClass() /var/www/html/php-doc-check.phar/src/ApplicationArgumentsProvider.php:10
 0.0077 2116560 5. Composer\Autoload\includeFile() /var/www/html/php-doc-check.phar/src/GetOpt.php:10
```

Fig. 14. – Problèmes et conflits d'installation soulevés avec l'utilisation de php-doc-check

Comme le dernier commit sur le dépôt date de septembre 2022, et que l'*issue* la plus récente a exactement la même date, les problèmes doivent certainement venir de l'absence de maintenance. Nous n'allons pas creuser plus loin pour l'instant.

## 3.3 — Intégration des outils dans le pipeline

En utilisant la directive `store_artifacts` en tant que step de chaque job rajouté (`metrics-phpmetrics`, `metrics-phploc`, `lint-phpmd`), on peut stocker temporairement (pendant 15 jours selon la [documentation d'InFisical](#)) les rapports dans l'onglet « Artifacts » de chaque job.

L'idée à présent, était de créer deux jobs en plus qui permettraient de centraliser les rapports de *metrics* pour l'un, et les rapports de linting pour l'autre.

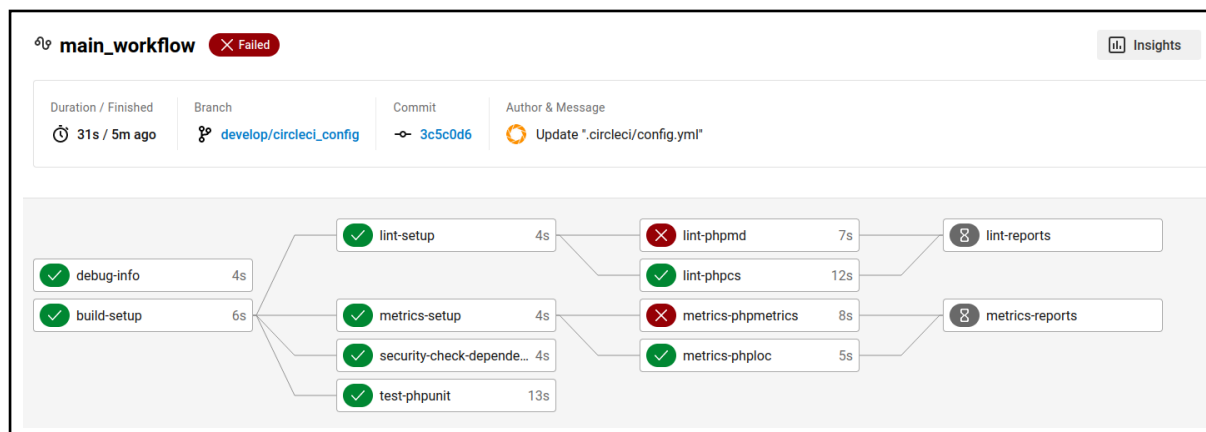


Fig. 15. – Jobs centralisant les rapports, qui se sont pas atteints si leur prédécesseurs ne réussissent pas

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

Code 7. – Configuration des dépendances d'un job dans le workflow main\_workflow

On peut le voir dans la Fig. 15 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. L'idée sous-jacente étant qu'en utilisant l'[API de CircleCI](#) (exemple en Fig. 16), on peut vérifier depuis un job `waiter` qu'un certain job est terminé ou non, et tant qu'il ne l'est pas, le job `waiter` continue d'attendre, en faisant un appel à l'API passé un certain temps.

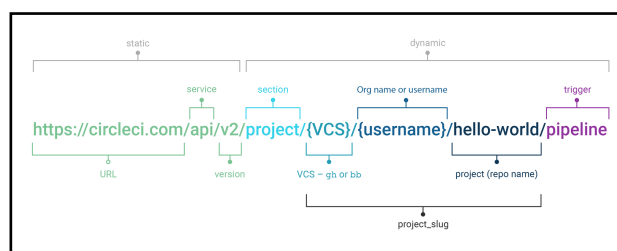


Fig. 16. – Structure de l'API CircleCI

Cette méthode n'est pas forcément optimale car elle consomme du temps d'exécution en plus dans CircleCI, mais elle comble la limitation de la directive `requires`.

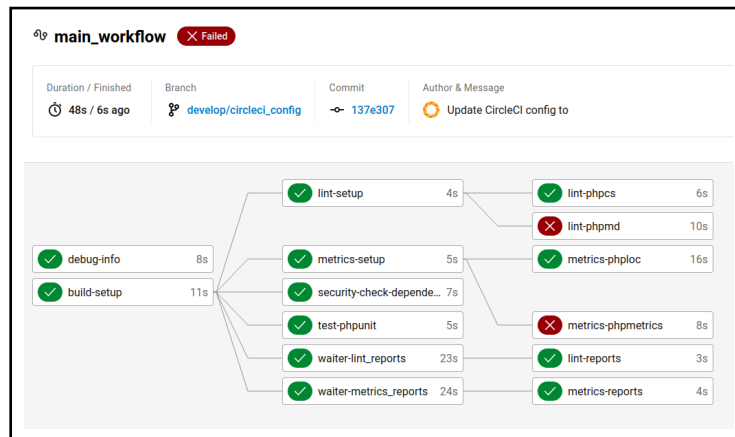


Fig. 17. – Les rapports de linting & metrics pourraient être disponibles et centralisés malgré l'échec de certains jobs

Cependant, malgré beaucoup d'essais de configuration des jobs standards de *CircleCI* persist\_to\_workspace et attach\_workspace, en en créant d'autres comme report\_persist (Cf. [circleci/config.yml](#)), les rapports ne se trouvaient pas regroupés dans la section « Artifacts » des jobs lint-reports et metrics-reports, prévus en tant que jobs agrégateurs de rapports. Nous l'avons compris très peu de temps avant le rendu de ce rapport, mais toujours selon [la même documentation](#), il n'est pas possible de transférer un *workspace* entre d'un job awaited\_job à un job waiter si awaited\_job n'est pas parent/ascendant de waiter.

- Each job can only see content added to the workspace by the jobs that are upstream of it.

Fig. 18. – Impossible pour un job d'utiliser un workspace d'un job qui n'est pas son parent

Assez tardivement donc, nous nous sommes rétractés sur la première configuration par défaut, à savoir publier le rapport d'un job sur son onglet « Artifacts ».

**PHPMD Report**

Generated at 2024-02-07 13:04 with [PHP Mess Detector](#) on PHP 8.2.15 on 5032230bcb4f

**3 problems found**

**Summary**

**By priority**

Count	%	Priority
2	66.7 %	Top (1)
1	33.3 %	Moderate (3)

**By namespace**

Count	%	PHP Namespace
2	100.0 %	App

**By rule set**

Count	%	Rule set
2	66.7 %	Controversial Rules
1	33.3 %	Naming Rules

**By name**

Count	%	Rule name
2	66.7 %	Superglobals
1	33.3 %	ShortVariable

**Details**

[Show details](#)

Fig. 19. – Même si le job échoue, le report est quand même généré et accessible

### 3.4 — Déploiement automatisé sur AWS EC2

Bien que nous comprenions l'intérêt du job hold, Nous avons commencé par désactiver l'utilisation de ce job dans les workflow, car le but du pipeline selon nous, était d'automatiser l'intégration et le déploiement de code ; ainsi, si on doit se connecter sur *CircleCI* pour approuver à chaque release, c'est que l'on n'a pas vraiment confiance en notre pipeline.

La commande tronquée 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 service PHP-FPM pour appliquer les changements.

```
ssh -o StrictHostKeyChecking=no $STAGING_SSH_USER@$STAGING_SSH_HOST \<< EOF
# ...
EOF
```

Nous verrons ci-après (*Cf* Chapitre 3.4.4.1 & Chapitre 3.4.4.2) les spécificités des deux environnements, à savoir sur le serveur staging, et sur le serveur de production.

#### 3.4.1 — Configuration sur AWS : premiers essais

1. Premièrement, nous avons commencé par nous créer un compte AWS en utilisant les free tiers proposés.
2. On se log en tant que Root user

Les étapes suivantes effectuées dans cette section sont laissées en annexe (*Cf* Chapitre 5.1) à titre informatif pour retracer nos essais, mais elles n'ont pas d'utilité directe dans notre configuration finalement. L'objectif était de créer un autre compte `updater_agent` qui n'aurait que le droit de se connecter à l'instance, et ce, au travers de la configuration d'utilisateur IAM. À partir de là, nous cherchions où trouver le mot de passe de l'utilisateur `updater_agent`, ou comment le réinitialiser, mais nous n'arrivions pas à comprendre où chercher. Nous avons ainsi compris que ces « utilisateurs IAM » étaient des comptes AWS aux droits que l'on pouvait restreindre. On peut par exemple les configurer pour qu'ils aient accès à la console ou non, et de manière générale, pour qu'ils puissent utiliser les services Amazon, et non pas les restreindre dans les machines AWS en tant que tel.

Mais après avoir lancé l'instance à l'étape d'après (*Cf* Chapitre 3.4.2), il se trouve qu'il suffisait de créer un utilisateur sur la machine une fois accédée via SSH (*Cf* Chapitre 3.4.3).

#### 3.4.2 — Instanciation des deux machines côté AWS

1. On se rend sur la page d'accueil d'EC2
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 de l'application)
    - Il faut aussi autoriser les connexions HTTP entrantes, pour que l'on puisse accéder au service (qui expose du HTTP).
  - 15 Go de SSD
  - une keypair temporaire pour se connecter une fois à l'instance
3. On télécharge le fichier `.pem` (la clef privée) à mettre dans le répertoire `~/ssh` de notre machine personnelle.

- Puis un changement des droits d'accès dessus (mesure de sécurité standard pour que le fichier ne soit visible que par notre utilisateur sur notre machine) : `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`)
- Nous avons utilisé la commande telle quelle, mais utiliser l'IPv4 publique de la machine en tant qu'hôte est aussi faisable, et sera fait par la suite

```
>>> ssh -i "~/ssh/AWS_DevSecOps2_default.pem" ubuntu@ec2-51-20-87-132.eu-north-1.compute.amazonaws.com
The authenticity of host 'ec2-51-20-87-132.eu-north-1.compute.amazonaws.com (51.20.87.132)' can't be estal
ED25519 key fingerprint is SHA256:xDQHsJjy+aUf7vfVHZSTfDTNabZyQiduuc8YZjzmZY.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-51-20-87-132.eu-north-1.compute.amazonaws.com' (ED25519) to the list of kn
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.2.0-1017-aws x86_64)
```

Fig. 20. – Connexion réussie sur la machine AWS en SSH

Nous avons maintenant accès à la machine. La machine de production a été configuré de la même manière, mais simple avec une keypair différente générée pour l'authentification.

### 3.4.3 – Création d'un utilisateur dédié aux mises à jour de l'app' sur chaque machine

Nous avons en tête de dédier un utilisateur aux droits restreints, mais nous nous sommes vite aperçu que pour des tâches telles que modifier le contenu du répertoire `/var/www/html` ou redémarrer le service php, l'agent aurait besoin des droits d'administrateur. Cependant, nous savions qu'il était déconseillé de se connecter directement en tant qu'utilisateur root avec SSH. Ci dessus sont listées les étapes que nous avons utilisées et mises en place.

1. Une fois dans la machine, on crée un utilisateur (grâce à [cet article de DigitalOcean](#)) : `sudo adduser updater_agent`
  - Suivre les étapes en ne définissant que le mot de passe (le vrai nom, département, etc. ne sont pas pertinents pour nous)
  - Au fur et à mesure de nos tests avec le pipeline sur CircleCI, nous nous avons compris qu'il fallait 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`
  - (Nous avons pu utiliser [cet article-là sur Linuxize pour la suppression des groupes](#))

```
updater_agent@ip-172-31-20-205:~$ sudo systemctl status apache2
[sudo] password for updater_agent:
updater_agent is not in the sudoers file. This incident will be reported.
```

Fig. 21. – L'utilisateur n'est pas ajouté au groupe root et ne peut pas utiliser sudo

2. Changement d'utilisateur pour impersonner `updater_agent` : `su updater_agent`
3. 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`)
4. Nous avons rencontré une erreur qui nous a donné du fil à retordre: `updater_agent@51.20.92.240: Permission denied (publickey)`. Mais finalement, nous avons compris qu'il fallait autoriser la connexion SSH via la clef créée explicitement pour éviter l'erreur
  - `cat ~/.ssh/circleci.key.pub > ~/.ssh/authorized_keys`
5. Enfin, étant donné que certaines commandes telles que `service` allaient être utilisées en mode administrateur, nous avons pu trouver comment autoriser l'utilisateur à exécuter certaines commandes sans demande de mot de passe (lors du script d'update en ssh automatisé).



```

sudo visudo

# Règles établies sur le serveur staging
updater_agent ALL=(ALL) NOPASSWD: /usr/bin/rm
updater_agent ALL=(ALL) NOPASSWD: /usr/bin/cp
updater_agent ALL=(ALL) NOPASSWD: /usr/sbin/service

# Règle établie sur le serveur de production
updater_agent ALL=(ALL) NOPASSWD: /usr/bin/docker

```

Code 8. – Configuration de commandes qui font exception à la demande de mot de passe lors de leur exécution en tant qu'utilisateur root

- En ce qui concerne le pourquoi de telle ou telle commande autorisée, il faut observer les injections de commandes lors de la connexion SSH des agents, qui sont décrites dans [.circleci/config.yml](#), et qui sont expliquées dans le Chapitre 3.4.5

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

À partir de là, la suite de cette configuration se fera sur CircleCI, dans le Chapitre 3.4.5

### 3.4.4 – Configuration des deux machines staging et production

À la fin de cette partie, nous avons obtenu ceci

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4
<input type="checkbox"/>	Staging_server	i-0d2e0520829967de0	Running	t3.micro	2/2 checks passed	View alarms +	eu-north-1a	ec2-51-20-124-61.eu-n...	51.20.124.61
<input type="checkbox"/>	Production_se...	i-0271f21fc97950be5	Running	t3.micro	2/2 checks passed	View alarms +	eu-north-1b	ec2-16-170-243-186.eu...	16.170.243.186

Fig. 22. – Les deux serveurs staging et production lancés

#### 3.4.4.1 – Serveur staging : mises à jour, configuration de PHP, des dépendances, et lancement de l'application

```

sudo apt update
sudo apt upgrade -y
[ -f /var/run/reboot-required ] && sudo reboot -f # Reboot si nécessaire

# Après le reboot

# Ajout de PHP & cie
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

# Installation et mise à jour du projet
cd $HOME
git clone https://github.com/StanleyDINNE/php-devops-tp && cd php-devops-tp
composer install --optimize-autoloader --no-interaction --prefer-dist

# Exposition du service
sudo mkdir -p /var/www/html && sudo rm -rf /var/www/html/* && sudo cp -r ./ /var/www/html # Pour être sûr de
supprimer les fichiers existants
sudo sed -i 's!/var/www/html!/var/www/html/public!g' /etc/apache2/sites-available/000-default.conf
sudo systemctl restart apache2
(flock -w 10 9 || exit 1; sudo -S service php8.2-fpm restart ) 9>/tmp/fpm.lock

```

Code 9. – Installation des services nécessaires, notamment ce qui est fait dans [docker/Dockerfile](#) et dans le job `deploy-ssh-staging` de [.circleci/config.yml](#)

Cette configuration manuelle est sujette à être bancal si des packages manquent, etc. L'un des buts de la containerisation avec Docker est d'avoir des images déjà toutes configurées, et que l'exécution soit reproductible. Avant même de commencer cette configuration, nous nous sommes dits que cela aurait été mieux de simplement instancier un container Docker, dans lequel toute cette configuration était déjà



faite. Ainsi, pour le serveur de production, nous avons entrepris de déployer l'image Docker construite dans le pipeline sur CircleCI et hébergée sur GHRC.io.

#### 3.4.4.2 — Serveur de production : installation de docker et instanciation du container

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 la configuration de l'utilisateur `updater_agent` comme pour la machine Staging (avec mot de passe différent, configuration de la key-pair en mettant la clef privée dans les contextes de CircleCI, etc. : Cf Chapitre 3.4.2 & Chapitre 3.4.3), il suffisait de récupérer l'image depuis notre dépôt GitHub avec ces commandes :

```
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, ce que nous avons fait

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.

#### 3.4.5 — Côté CircleCI

Grâce à la documentation sur [l'ajout de clefs SSH](#), nous avons pu faire cette partie.

1. On se rend sur les paramètres du projet
2. Section « SSH Keys »
3. Ajout d'une clef SSH
  - On met comme hostname l'IPv4 publique de la machine
  - On colle la clef privée copiée de la section précédente Chapitre 3.4.3

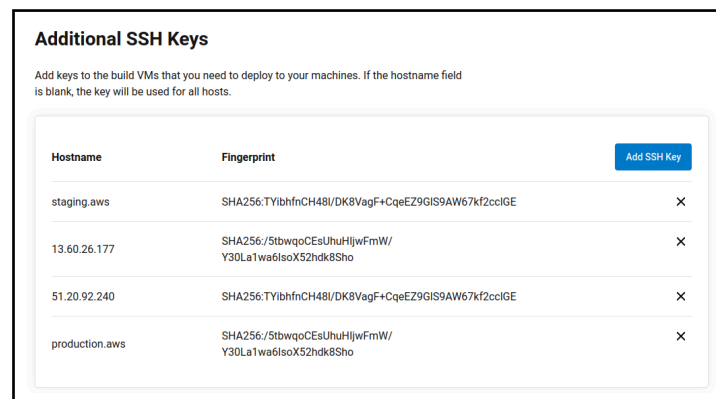


Fig. 23. – Ajout de clefs SSH pour les comptes `updater_agent`

4. On obtient la signature (SHA256: ... )
  - On peut utiliser cette signature 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`

Les étapes sont les mêmes pour configurer CircleCI pour le serveur de production, avec les variables `$PRODUCTION_SSH_FINGERPRINT`; `$PRODUCTION_SSH_USER`; `$PRODUCTION_SSH_HOST`

En ce qui concerne le job `deploy-ssh-staging`, l'idée des commandes git utilisées (`git checkout --track ...`; `git reset --hard ...`) est d'avoir un environnement propre, peu importe l'état de la branche actuelle, potentiellement sur un commit qui n'existe plus (dans le cas d'amend a posteriori sur le dépôt), ou alors avec des fichiers modifiés localement (comme `composer.lock`)

Pour `deploy-ssh-production`, il s'agit simplement d'arrêter le container en cours d'exécution, de le supprimer, de mettre à jour l'image vers sa dernière version avec le tag `latest`, et d'instancier le container, notamment avec le token d'Infisical injecté, pour que l'app' puisse utiliser la CLI d'Infisical pour utiliser ses secrets. **TO[ Infisical]DO**

### 3.5 – Modification des flows pour définir des politiques

La politique de déploiement visible de par la configuration des workflow sur `.circleci/config.yml` est telle que :

- Les jobs de metrics, tests, et security checks seront toujours exécutés sur n'importe quelle branche, dans le workflow principal `main_workflow`
- l'image docker ne sera construite que lorsque des changements sont effectués sur les branches `main` et celles préfixées par `release/`, via le workflow `container_workflow`
- le déploiement de l'application sur le serveur staging ne se fera que si des modifications sont perçues sur des branches `release/*`, via le workflow `main_workflow`
- le déploiement de l'application via son image docker ne sur le serveur de production ne se fera qu'à l'issue de la construction de celle-ci, depuis le workflow `container_workflow`, et que lors de changement sur la branche `main`

Aussi, l'image est tag avec son numéro de version issue d'un tag git, ou issu du nom de la branche et de la date du jour.

### 3.6 – Quelques figures du déploiement

<pre> updater_agent@ip-172-31-42-172:~\$ sudo docker ps CONTAINER ID   IMAGE bbb0201951a8   ghcr.io/stanleydinne/php-devops-tp:latest updater_agent@ip-172-31-42-172:~\$ </pre>	COMMAND	CREATED	STATUS	PORTS	NAMES
"docker-php-entrypoint..."	15 minutes ago	Up 15 minutes	0.0.0.0:80->80/tcp, :::80->80/tcp	php-devops-tp:latestcontainer	

Fig. 24. – Container lancé sur le serveur de production

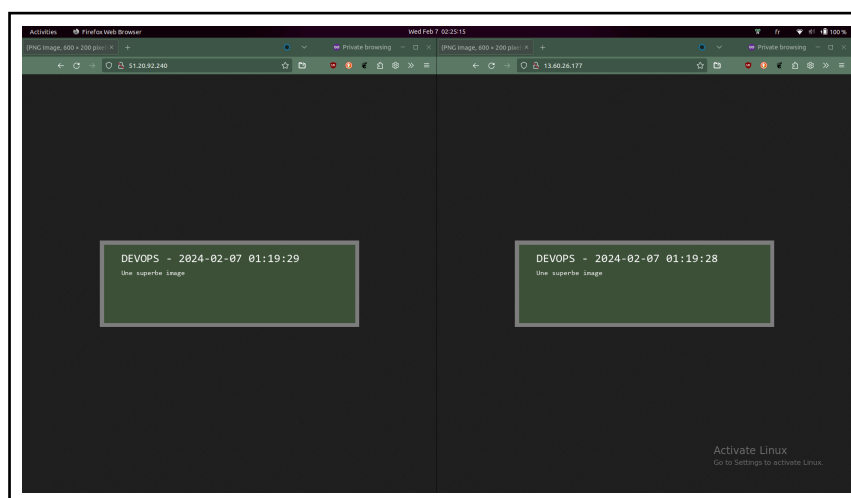


Fig. 25. – Instances de l'application PHP sur le serveur staging et production

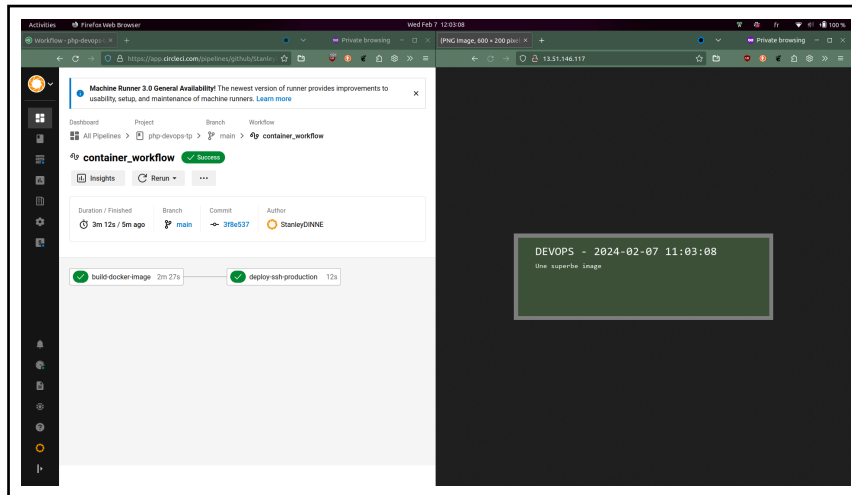


Fig. 26. – Workflow exécuté avec succès

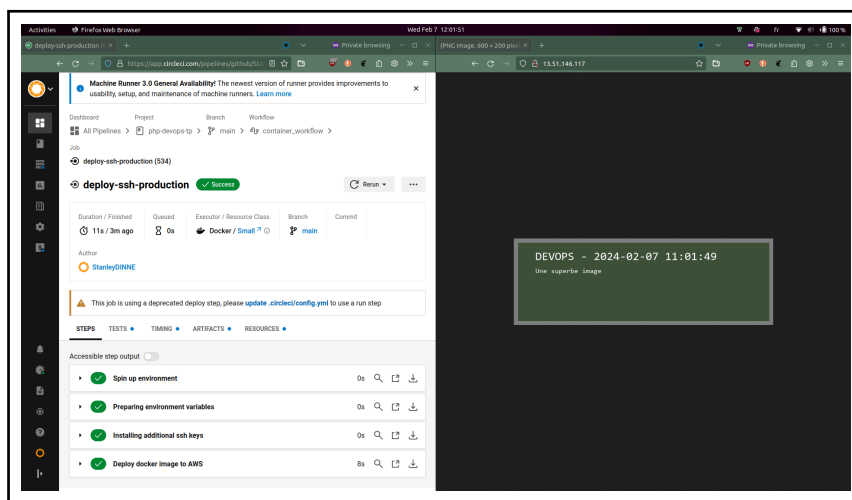


Fig. 27. – Déploiement automatisé via SSH sur le serveur de production fonctionnel

## 4 — Sécurité à tous les étages : idées pour plus de sécurité tout au long du processus de déploiement

### 1. 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 dépôt, crée une branche master, met du code contaminé, et merge ça sur le dépôt principal, la politique de création et déploiement de l'image docker fait que celle-ci sera créée avec le job.

### 2. Suppression de l'utilisateur de l'option `-o StrictHostKeyChecking=no` avec ssh, au vu des commentaires sur [cet article sur HowToUseLinux](#) Mais nous avons compris après certains tests, que cela permet de se débarrasser de

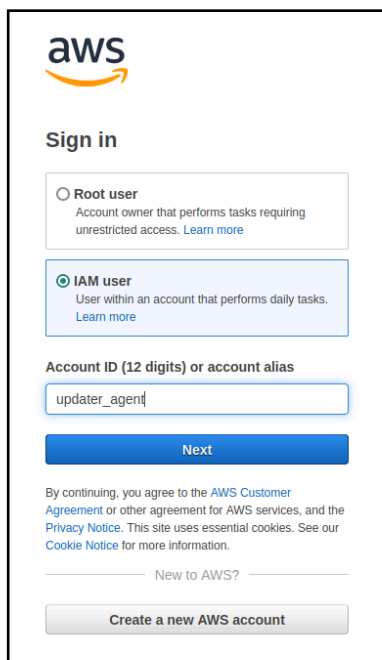
```
The authenticity of host '***** (*****)' can't be established.  
ECDSA key fingerprint is ....  
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

1. Ceci est plus une remarque, mais sur la configuration 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.
2. À un moment, on voulait passer une variable d'environnement récupérée depuis un contexte CircleCI, puis l'injecter dans une commande sudo pour mettre à jour les machines via ssh. Mais c'est une mauvaise idée, car le mot de passe sera ainsi écrit dans `~/.bash_history`. À la place, certaines commandes ont été configurées pour pouvoir s'exécuter dans besoin de mot de passe, et ce en configurant `sudo visudo`
3. Le dépôt est compte GitHub
4. En ce qui concerne la configuration de l'instance EC2, il existe
  - *AWS inspector*, qui permet de faire un scan de vulnérabilités potentielles sur notre machine AWS
    - Nous ne l'avons pas utilisé par manque de temps avant ce rendu, mais il aurait fallu
  - *AWS Identity and Access Management (IAM)*, dont il a été question dans Chapitre 5.1, qui était plutôt Not Applicable dans notre cas
  - des processus et outils tels que CSPM, ou CWPP
    - « *Cloud Security Posture Management (CSPM) is the process of monitoring cloud-based systems and infrastructures for risks and misconfigurations.* » - Microsoft security documentation.
    - « *Cloud Workload Protection Platform (CWPP) is a cloud security solution that helps protect cloud workloads in multicloud and hybrid environments.* » - Microsoft security documentation.
  - Le *AWS Web Application Firewall (WAF)* pour filtrer le trafic entrant vers les applications web, définir des règles pour bloquer ou autoriser certaines requêtes en fonction de critères tels que les adresses IP source, les chaînes de caractères ou les en-têtes HTTP.
    - Nous avons autorisé par défaut les connexion entrantes de toutes les adresses IPv4 car nous avons en tête que notre service allait être accessible depuis n'importe qui, mais cela est sujet à une réflexion qui nous aurait demandé plus de temps, compte tenu de la configuration que nous avons fait jusque-là

## 5 — Annexes

### 5.1 — Configuration d'un « Utilisateur IAM » sur AWS

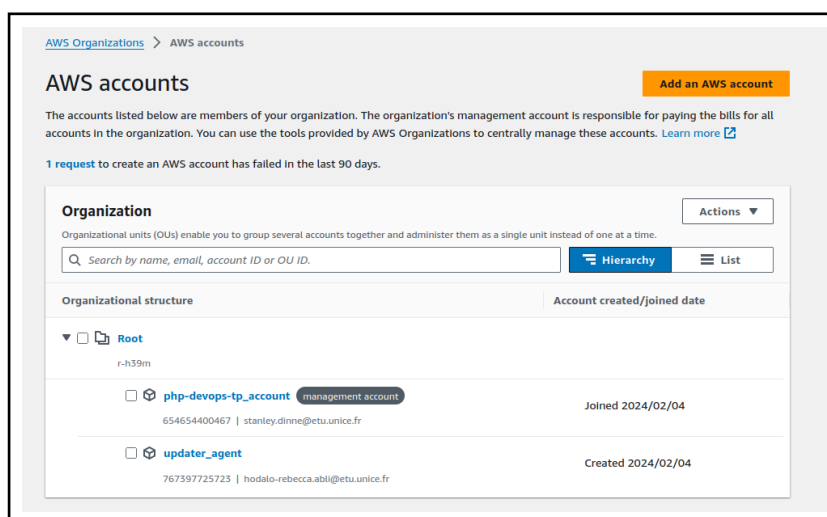
Ces étapes retracent notre volonté à créer un utilisateur avec des droits restreints qui puissent faire des mises à jour de l'application sur les machines AWS. Seulement,



The image shows the AWS 'Sign in' page. At the top is the AWS logo. Below it, the 'Sign in' heading is followed by two radio button options: 'Root user' (unselected) and 'IAM user' (selected). The 'IAM user' option is highlighted with a blue border. Below these options is a text input field containing 'updater\_agent'. A blue 'Next' button is positioned below the input field. At the bottom, there is a link for 'New to AWS?' and a button labeled 'Create a new AWS account'.

Fig. 28. – Interface de connexion à AWS via un utilisateur non administrateur

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



The image shows the AWS Organizations console. At the top, there's a breadcrumb 'AWS Organizations > AWS accounts' and an 'Add an AWS account' button. Below this, a message states: 'The accounts listed below are members of your organization. The organization's management account is responsible for paying the bills for all accounts in the organization. You can use the tools provided by AWS Organizations to centrally manage these accounts. Learn more'. A status message indicates '1 request to create an AWS account has failed in the last 90 days.' The main section is titled 'Organization' and includes a search bar and 'Hierarchy'/'List' tabs. Below this, the 'Organizational structure' is shown as a tree. The 'Root' node is expanded, showing two child accounts: 'php-devops-tp\_account' (management account, joined 2024/02/04) and 'updater\_agent' (created 2024/02/04).

Fig. 29. – Création du compte IAM 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>)

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

Code 10. – Définition d'une politique  
Simple machine connection Policy

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

Code 11. – Définition d'une politique  
DenyEverything

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

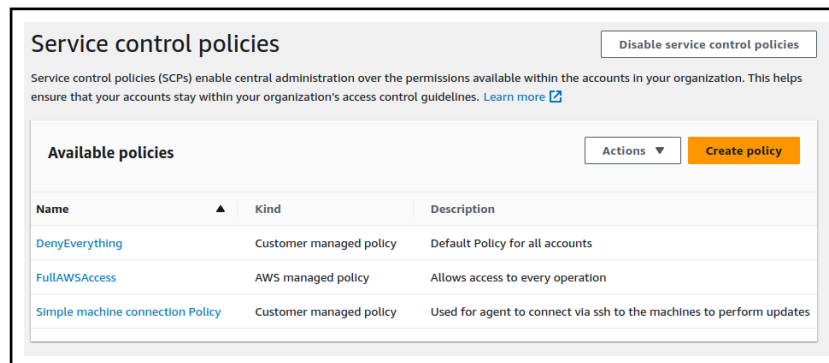


Fig. 30. – Création des Service Control Policies

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