

# TP03 DevSecOps

---

RAPPORT FINAL

Mohamed BOUCHENGUOUR  
Hamadi DAGHAR  
Bilal MESAOURI DEBOUN  
Lamyra BAIDOURI  
POLYTECH UNICE



## Table des matières

Projet Initial : .....	3
Configuration CircleCi .....	4
Lien du projet GitHub à CircleCI .....	4
Configuration du fichier config.yml pour les tests .....	6
Utilisation des workspaces .....	6
Optimisation des Exécuteurs.....	6
Description des jobs essentiels pour la qualité, la sécurité et les tests .....	7
Création et déploiement de l'image Docker de l'application .....	7
Déploiement SSH sécurisé sur l'environnement de staging.....	8
Workflows principaux : main_workflow et container_workflow .....	8
Gestion des variables d'environnement et authentification GHCR.....	9
Organisation des branches et workflows Git .....	10
Extension du pipeline .....	11
Configuration de Infisical .....	13
Déploiement automatisé sur AWS EC2 en Staging .....	15
Étapes de création et configuration de l'instance : .....	15
Connexion à l'instance et configuration initiale : .....	17
Génération et ajout de la clé SSH pour GitHub : .....	18
Clonage du dépôt .....	19
Gestion des secrets avec Infisical .....	20
Déploiement depuis CircleCI .....	23
Ajout des variables d'environnement à CircleCI.....	24
Modification du job existant.....	24
Ajustements et Tests .....	25
Déploiement automatisé sur AWS EC2 en Production .....	27
Étapes de création et configuration de l'instance .....	27
Connexion à l'instance et configuration initiale .....	27
Génération et ajout de la clé SSH pour GitHub .....	28
Clonage du dépôt : .....	28
Gestion des secrets avec Infisical .....	28
Déploiement depuis CircleCI .....	28

Ajout des variables d'environnement à CircleCi.....	29
Ajout du job deploy-ssh-production .....	29
Ajustements et Tests .....	30
Schéma récapitulatif .....	32
Sécurité à tous les étages .....	32
Êtes-vous sûrs d'avoir bien configuré votre repository et compte GitHub ? .....	32
Pensez-vous que votre image Docker est fiable ? Comment le montrer ? .....	34
Êtes-vous sûr d'avoir bien configuré votre instance EC2 ? Comment le prouver ?.....	35
Gestion des secrets et des variables d'environnement sensibles.....	37
Surveillance et audits de sécurité des connexions.....	37
Gestion des mises à jour de sécurité : .....	38

## Projet Initial :

Après avoir récupéré le projet, celui-ci se présente sous forme de conteneur Docker, avec le fichier de configuration principale Dockerfile situé dans le répertoire Docker/. Ce fichier configure un environnement PHP 8.2 avec Apache pour l'exécution de l'application.

Le Dockerfile commence par l'installation des dépendances système et des extensions PHP requises (gd, pdo\_mysql, zip, etc.), ainsi que la configuration de SQLite et Xdebug. Il installe également l'outil CLI Infisical pour la gestion sécurisée des secrets.

L'activation du module mod\_rewrite d'Apache et la définition de la racine du document sur /var/www/html/public garantissent la compatibilité avec l'application. Le code source est copié dans /var/www/html, et des permissions adaptées sont définies pour assurer le bon fonctionnement.

Ce Dockerfile facilite le déploiement et la gestion de l'application dans un environnement reproductible et cohérent.

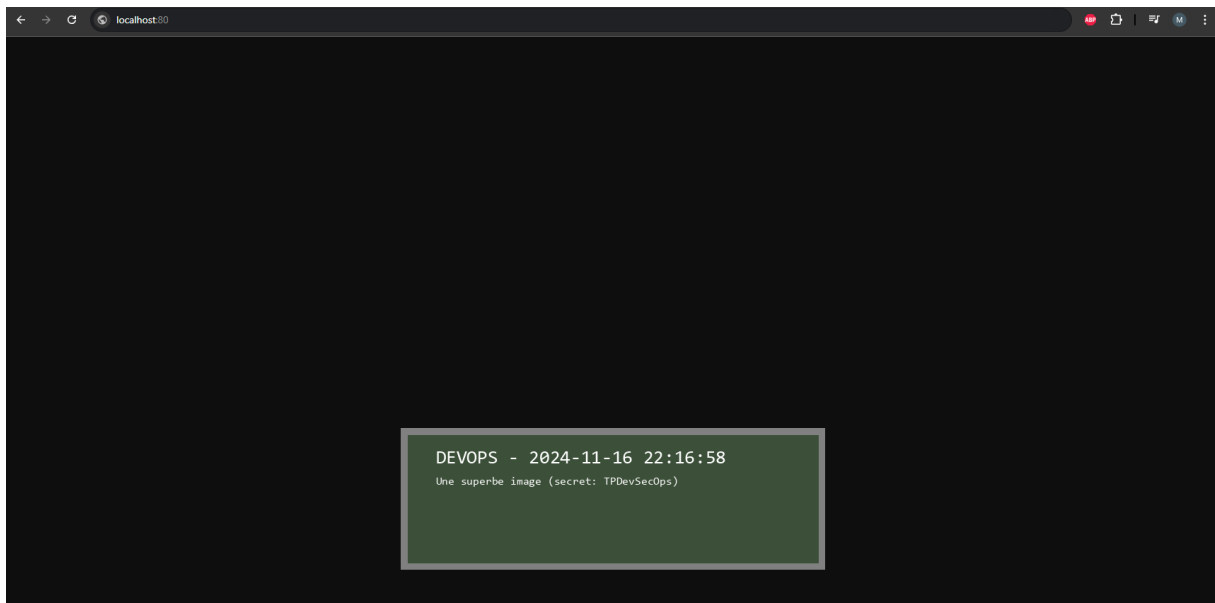
Dans le Dockerfile, le port 80 est exposé afin de permettre l'accès au site via le conteneur. Pour exécuter l'application localement, un fichier docker-compose.yml a été créé comme suit :

```
compose.yml
1  services:
2    web:
3      build:
4        context: .
5        dockerfile: Docker/Dockerfile
6      ports:
7        - "80:80"
8      volumes:
9        - ./var/www/html
10     command: /bin/sh -c "composer install --no-interaction --no-ansi --prefer-dist && apache2-foreground"
```

Ce fichier définit un service web qui utilise l'image construite à partir du Dockerfile et expose le port 80. Le volume monté garantit que les modifications apportées localement sont immédiatement visibles dans le conteneur. La commande d'exécution assure que les dépendances PHP sont installées via composer avant de démarrer le serveur Apache, préparant ainsi l'application à être servie.

Dans le fichier .env.example, il y a APP\_SECRET. Ce dernier est un mot caché qui devra être transmis entre les différentes instances (développeurs et aws). Il simulera dans un vrai contexte un mot de passe par exemple. On le modifie en .env puis on lui assigne une valeur (TPDevSecOps par exemple) et ce dernier sera affiché dans la page index.php (récupéré par src/ImageCreator.php). On lance le docker-compose et on se connecte au

site depuis localhost :80. Une image s'affiche incluant le secret APP\_SECRET. Le fichier .env ne devra jamais être mis dans le dépôt git, il est donc inclut dans .gitignore.



Une fois le site fonctionnel, on crée un projet GitHub. On lui assigne un nom (DevSecOpsTP) et on garde le projet en privée. Cela permet d'éviter que n'importe quelle personne accède au code source. On ajoute ensuite tous les membres du projet et on push le projet dans le GitHub. Chaque membre qui récupère le projet devra créer un .env qui contient la variable d'environnement APP\_SECRET.

## Configuration CircleCi

### Lien du projet GitHub à CircleCI

Pour configurer CircleCI, après la création de l'organisation et du projet, et après avoir ajouté tous les membres, il est nécessaire de lier le projet GitHub à CircleCI. Cela permet de déclencher des actions automatisées à chaque push. Dans les paramètres du projet sur CircleCI, une **pipeline** est ajoutée, définissant les étapes et les workflows à exécuter.

The screenshot shows the CircleCI configuration interface. At the top, the 'Integration' dropdown is set to 'GitHub App'. Below it, a note states: 'This cannot be changed after creating the pipeline.' The 'Connection' section shows 'Connected to mbouchenguour' with a green checkmark. The 'Name' field is 'build-and-test'. The 'Config' section has 'Config Source' set to 'DevSecOpsTP' and 'Config File Path' set to '.circleci/config.yml'. A note explains: 'When a pipeline is triggered, the config file will be pulled from this repo. This cannot be changed after creating the pipeline.' The 'Build Assets' section has 'Checkout Source' set to 'DevSecOpsTP'. A note states: 'When a checkout step is run, this repo will be checked out. The checkout source cannot be edited because this pipeline is connected with a trigger. Delete the associated triggers from the "Triggers" section if you wish to edit this field.' At the bottom, there is a 'Definition ID' field and a 'Copy' button.

Ici, le fichier config.yml est spécifié pour s'exécuter lors d'un push. Ce fichier, situé dans .circleci/, définit le pipeline à suivre pour les tests, la construction et les déploiements de l'application. Le projet GitHub est ainsi configuré pour être automatiquement testé et validé.

Une fois la pipeline définie, un trigger est mis en place : il s'agit d'un déclencheur qui lance automatiquement les processus définis à chaque push sur le dépôt GitHub.

The screenshot shows the 'Edit Trigger' configuration page in CircleCI. The breadcrumb 'Triggers > Edit Trigger' is at the top. The main heading is 'Edit GitHub App Trigger'. There are 'Cancel' and 'Save' buttons. The 'Trigger Name' field contains 'Github App Trigger'. The 'Pipeline to run' dropdown is set to 'build-and-test'. The 'Trigger source' dropdown is set to 'mbouchenguour/DevSecOpsTP'.

L'image montre la configuration du déclencheur GitHub App Trigger qui relie le projet GitHub à la pipeline build-and-test de CircleCI. Le déclencheur, une fois configuré, garantit que toute modification (push) effectuée sur le projet

mbouchenguour/DevSecOpsTP active automatiquement la pipeline définie, permettant ainsi l'exécution des tests, des builds et d'autres étapes prévues dans le workflow CircleCI.

## Configuration du fichier config.yml pour les tests

Maintenant le projet github et circle lié, il faut configurer config.yml pour les tests.

## Utilisation des workspaces

Le config possède des workspaces :

```
# Default configuration for persist_to_workspace and attach_workspace commands
persist_to_workspace: &persist_to_workspace
  persist_to_workspace:
    root: .
    paths:
      - .

attach_workspace: &attach_workspace
  attach_workspace:
    # Must be absolute path or relative path from working_directory
    at: ~/project
```

Les workspaces permettent de sauvegarder et de partager des fichiers ou des résultats entre différents jobs d'un workflow. En utilisant persist\_to\_workspace, les fichiers sont conservés dans un espace de travail, et attach\_workspace permet d'accéder à ces fichiers dans un autre job.

## Optimisation des Exécuteurs

```
executors:
  php-executor:
    resource_class: small
    shell: /bin/bash
    docker:
      - name: localhost
        image: cimg/php:8.2
  builder-executor:
    resource_class: small
    shell: /bin/bash
    docker:
      - image: cimg/php:8.2-node
        name: localhost
  simple-executor:
    resource_class: small
    shell: /bin/bash
    docker:
      - image: cimg/base:stable
        name: localhost
```

Les exécuteurs définissent les environnements dans lesquels les jobs s'exécutent. Par exemple, php-executor utilise une image PHP, tandis que builder-executor combine PHP et NodeJS pour les tâches nécessitant ces deux langages. Cela permet d'optimiser chaque job selon ses besoins spécifiques en termes de ressources et d'environnement.

## Description des jobs essentiels pour la qualité, la sécurité et les tests

Ensuite, nous avons une série de jobs qui automatisent des tâches essentielles pour garantir la qualité, la sécurité et le bon fonctionnement du projet :

1. **debug-info** : Ce job fournit des informations de débogage sur l'environnement CircleCI, telles que l'utilisateur courant, le répertoire, la date, et les variables d'environnement.
2. **build-setup** : Ce job installe les dépendances PHP avec Composer, utilise un cache pour accélérer les builds futurs et partage l'état du projet avec d'autres jobs via un workspace.
3. **lint-phpcs** : Vérifie la conformité du code avec PHP\_CodeSniffer selon des règles spécifiques, tout en générant un rapport stocké comme artefact.
4. **security-check-dependencies** : Ce job analyse les vulnérabilités des dépendances PHP à l'aide de local-php-security-checker et produit un rapport de sécurité.
5. **test-phpunit** : Vérifie la présence de tests unitaires avec PHPUnit, installe l'outil et exécute les tests afin de valider le bon fonctionnement de l'application.

## Création et déploiement de l'image Docker de l'application

Le job **build-docker-image** est responsable de la création et du déploiement de l'image Docker de l'application. Ses principales étapes sont les suivantes :

- **checkout** : Récupère le code source du projet pour la construction de l'image.
- **setup\_remote\_docker** : Configure un environnement Docker distant, en utilisant la version spécifiée (ici 2024) et le cache des couches Docker pour optimiser les builds.
- **Build and Push Docker Image** : Ce job commence par vérifier si la variable d'environnement **SKIP\_BUILD** est définie ; si c'est le cas, la construction est ignorée. Sinon, les variables d'environnement **GITHUB\_PROJECT\_USERNAME** (défini manuellement en variable environnement dans CircleCi), **GITHUB\_PROJECT\_REPONAME** (défini manuellement en variable environnement dans CircleCi), et **CIRCLE\_BRANCH** sont utilisées pour formater correctement le nom du dépôt et le tag de l'image. Le processus d'authentification auprès de GitHub Container Registry (GHCR) utilise la variable **GHCR\_PAT** (GitHub Container Personal Access Token) et **GHCR\_USERNAME** pour permettre la connexion sécurisée. L'image Docker est ensuite construite en utilisant plusieurs arguments de construction comme **BUILD\_DATE**, **TAG**, **GIT\_COMMIT**, **GIT\_URL**, **SQLITE\_VERSION**, **SQLITE\_YEAR**, et **PROJECT\_USERNAME** pour inclure des



métadonnées importantes et contextualiser l'image. Enfin, l'image construite est poussée sur le registre GHCR en utilisant le tag formaté.

## Déploiement SSH sécurisé sur l'environnement de staging

Le job *deploy-ssh-staging* est chargé de déployer l'application sur un environnement de staging via une connexion SSH sécurisée. Ce job utilise l'exécuteur *simple-executor* et commence par ajouter les clés SSH nécessaires pour établir une connexion sécurisée avec le serveur de staging, en utilisant les empreintes digitales spécifiées (*STAGING\_SSH\_FINGERPRINT*).

Ensuite, le processus de déploiement démarre avec une connexion SSH au serveur cible définie par les variables d'environnement *STAGING\_SSH\_USER* et *STAGING\_SSH\_HOST*. Une fois connecté, le job détermine la version de PHP-FPM en cours d'utilisation, se déplace dans le répertoire de déploiement défini par *STAGING\_DEPLOY\_DIRECTORY*, et synchronise le code source en utilisant *git pull* pour récupérer la dernière version du code à partir de la branche spécifiée (*CIRCLE\_BRANCH*).

Après la mise à jour du code, le job installe les dépendances PHP à l'aide de *composer install* avec des options pour optimiser l'autoloader et minimiser les interactions, garantissant une mise à jour efficace des dépendances. Enfin, il utilise une commande de verrouillage (*flock*) pour redémarrer de manière sécurisée le service PHP-FPM, évitant les conflits d'accès simultané au service et assurant une continuité fluide du déploiement.

## Workflows principaux : *main\_workflow* et *container\_workflow*

Ensuite, nous avons deux workflows principaux : *main\_workflow* et *container\_workflow*. Un workflow permet d'orchestrer l'exécution de différents jobs en définissant leur ordre et leurs dépendances, garantissant que chaque tâche s'exécute selon une séquence logique ou en parallèle, en fonction des besoins du projet.

Le premier workflow, *main\_workflow*, se concentre sur l'intégration continue, la validation de la qualité du code et les déploiements. Il commence par exécuter des jobs essentiels tels que *debug-info* et *build-setup*, puis passe par des vérifications de style et de sécurité du code avec *lint-phpcs* et *security-check-dependencies*. Une fois ces étapes validées, il exécute les tests unitaires avec *test-phpunit*. Ces jobs sont exécutés sur toutes les branches, garantissant une validation continue et systématique du code à chaque mise à jour ou modification. Le workflow inclut également un job d'approbation (*hold*) pour sécuriser les déploiements en production et en staging, garantissant que seuls les changements validés sont appliqués. Les déploiements en environnement de production et de staging, réalisés respectivement par les jobs *deploy-ssh-production* et *deploy-ssh-*

staging, sont déclenchés après la validation du hold et ne s'exécutent que sur les branches main, master (pour la production) ou sur les branches correspondant à un schéma de release précis (pour le staging).

Le second workflow, `container_workflow`, est dédié à la construction et à la gestion des images Docker du projet. Ce workflow exécute le job `build-docker-image`, qui construit une image Docker à partir du code source récupéré, en utilisant les variables d'environnement et les métadonnées définies dans le contexte de la CI/CD. Ce workflow est exécuté sur les branches principales telles que main, develop, ainsi que sur les branches de type feature, release, hotfix et bugfix, afin de garantir que chaque version ou modification majeure est intégrée dans une image conteneurisée actualisée. L'image générée est ensuite authentifiée et poussée sur le registre GitHub Container Registry (GHCR), maintenant ainsi une continuité dans les déploiements conteneurisés.

## Gestion des variables d'environnement et authentification GHCR

Pour que ces jobs s'exécutent correctement, il est indispensable de définir les variables d'environnement nécessaires. Par exemple, le job `build-docker-image` requiert les variables `GHCR_USERNAME` et `GHCR_PAT`. La variable `GHCR_USERNAME` correspond au nom d'utilisateur associé au dépôt GitHub. Quant à `GHCR_PAT`, il s'agit d'un token personnel GitHub permettant de s'authentifier auprès de GHCR.IO (GitHub Container Registry) et d'effectuer des actions telles que des *push* et *pull* sur les images Docker.

Pour obtenir ce token, il faut se connecter à GitHub, accéder aux **Paramètres** du compte, puis naviguer vers **Paramètres du développeur**. De là, sélectionner **Tokens personnels (classic)** et cliquer sur **Générer un nouveau token**. Lors de la génération du token, il est crucial de lui accorder les permissions nécessaires pour l'accès au registre de conteneurs, telles que *write* et *read*.

Note

CircleCI GHCR Token

What's this token for?

Expiration

This token expires on **Mon, Dec 16 2024**. To set a new expiration date, you must [regenerate the token](#).

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo:deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows
<input checked="" type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input checked="" type="checkbox"/> read:packages	Download packages from GitHub Package Registry

Une fois le token créé, les variables `GHCR_USERNAME`, `GHCR_PAT`, `GITHUB_PROJECT_USERNAME` et `GITHUB_PROJECT_REPONAME` sont ajoutées dans la configuration du projet CircleCI en tant que variables d'environnement.

GHCR_PAT	xxxx0pnm	Nov 19, 2024, 4:09:08 AM	×
GHCR_USERNAME	xxxxuour	Nov 19, 2024, 4:09:34 AM	×
GITHUB_PROJECT_REPO NAME	xxxxpsTP	Nov 19, 2024, 4:22:27 AM	×
GITHUB_PROJECT_USER NAME	xxxxuour	Nov 19, 2024, 4:22:03 AM	×

Cela garantit l'authentification et le bon déroulement des tâches d'intégration continue et de déploiement lors de la création et la gestion des images Docker.

La variable CIRCLE\_BRANCH est déjà définie par l'environnement CircleCI. Elle correspond à la branche du dépôt sur laquelle l'exécution est déclenchée.

## Organisation des branches et workflows Git

Ensuite, les branches Git sont créées et organisées en fonction des exigences du workflow container\_workflow. Celui-ci est configuré pour s'exécuter sur les branches principales telles que master, main et develop, mais également sur des branches spécifiques comme celles dédiées aux nouvelles fonctionnalités (feature), aux versions (release), aux correctifs urgents (hotfix) et aux corrections de bogues (bugfix).

Default branch	
✓ main	17 hours ago
Recent branches	
🔗 release/v2.1	20 hours ago
🔗 release/v2.0	22 hours ago
🔗 release/v1.0	22 hours ago
🔗 develop	3 days ago
🔗 master	3 days ago
Other branches	
🔗 bugfix/fix-issue123	3 days ago
🔗 feature/new-feature	3 days ago
🔗 realease/v2.0	yesterday

Le bon fonctionnement des workflows est vérifié en effectuant une modification du code, ce qui déclenche l'exécution des workflows container\_workflow et main\_workflow. On peut observer leur succès ou leur nécessité d'approbation selon les étapes définies dans le pipeline CircleCI, garantissant ainsi que les modifications apportées respectent les conditions de validation et de déploiement.

Dev 26	▶ <span>✓ Success</span>	container_workflow	MO main e370f54 Update config.yml	3d ago	2m 59s ↓25%	↺ ↻ ⊗ ...
	▶ <span>   Needs Approval</span>	main_workflow	MO main e370f54 Update config.yml	3d ago	2d 21h	↺ ↻ ⊗ ...

## Extension du pipeline

Pour l'extension du pipeline, plusieurs nouveaux jobs ont été ajoutés afin d'améliorer l'analyse du code et d'assurer une meilleure qualité :

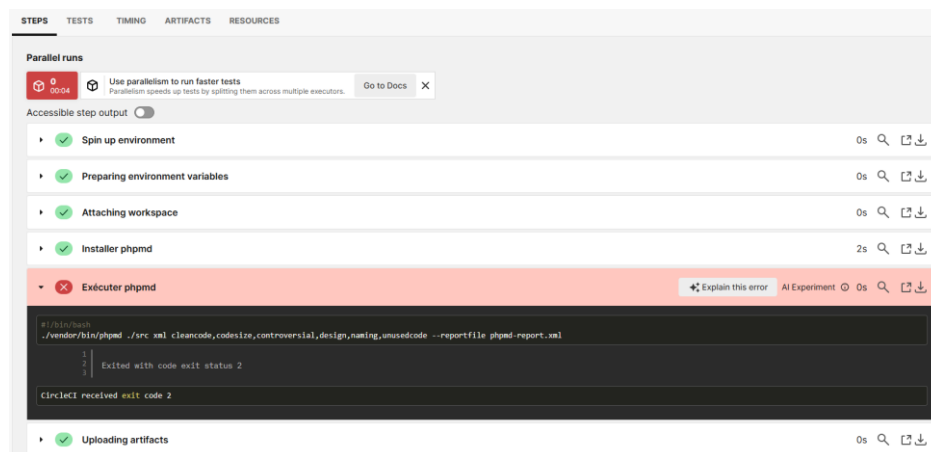
- **metrics-phpmetrics** : Ce job installe et exécute phpmetrics pour analyser les métriques du code PHP et génère un rapport HTML accessible via le répertoire phpmetrics-report.
- **metrics-phploc** : Ce job télécharge et exécute phploc pour collecter des statistiques de niveau de complexité et de structure du code. Le rapport généré est stocké dans phploc-report.
- **lint-phpmd** : Pour renforcer la qualité du code, ce job utilise phpmd (PHP Mess Detector) pour détecter divers problèmes de conception, de taille de code, et de nommage. Un rapport XML est généré et sauvegardé dans phpmd-report.
- **lint-php-doc-check** : Ce job vérifie la présence et la qualité de la documentation PHP avec l'outil php-doc-check et produit un rapport textuel php-doc-check-report.

Ces jobs s'exécutent après l'étape build-setup dans le workflow main\_workflow, garantissant l'analyse, la documentation et l'optimisation du code. Ils sont exécutés sur toutes les branches, ce qui assure une évaluation continue de la qualité du code à chaque mise à jour, et permet de produire des artefacts qui fournissent des informations utiles pour améliorer l'application de manière constante.

Un problème rencontré lors de l'ajout du job **metrics-phploc** était l'impossibilité de l'installer directement via Composer, entraînant des erreurs lors de son exécution. Pour contourner ce problème, la solution retenue a été de télécharger le binaire phploc.phar directement depuis l'URL officielle à l'aide de la commande wget, suivi de l'ajout des permissions d'exécution avec chmod. Cela a permis d'exécuter phploc correctement sur le répertoire ./src sans erreurs et de générer le rapport attendu phploc-report.txt. Cette

approche a résolu les problèmes initiaux d'installation, garantissant que l'analyse de la structure et des statistiques du code puisse s'effectuer sans incident.


Lors de l'ajout des jobs d'analyse de qualité de code, une erreur est apparue dans le job phpmmd, indiquée par un message dans l'interface CircleCI.



Ces ajustements ont permis de corriger les erreurs signalées et de garantir le bon déroulement des analyses de qualité du code, en conformité avec les standards définis dans le pipeline.

## Configuration de Infisical

Après la création du compte et du projet sur Infisical, un secret nommé APP\_SECRET a été créé. Ce secret est configuré pour être accessible dans les environnements de développement, staging et production.

<input type="checkbox"/> NAME ↓	Development	Staging	Production
▼ APP_SECRET	✓	✓	✓
Key	APP_SECRET		
Environment	Value		 Hide Values
Development	TPDevSecOps		
Staging	TPDevSecOps		
Production	TPDevSecOps		

Désormais, depuis l'application en development, en staging ou en production, il faudra récupérer APP\_SECRET depuis infisical directement et ne plus le stocker dans le fichier .env ce qui permettra qu'il soit toujours synchronisé entre les environnements mais également entre les développeurs.

Pour configurer le projet Git et permettre aux développeurs d'y accéder localement, chaque développeur doit disposer d'une machine dédiée. L'utilisation d'une machine dédiée par développeur renforce la sécurité en permettant de filtrer l'accès par adresse IP.

**Edit Universal Auth**

Auth Method  
Universal Auth

Access Token TTL (seconds)  
2592000

Access Token Max TTL (seconds)  
2592000

Access Token Max Number of Uses  
0

Client Secret Trusted IPs  
0.0.0.0/0  
::/0  
+ Add IP Address

Access Token Trusted IPs  
0.0.0.0/0  
::/0  
+ Add IP Address

Edit Cancel Delete Auth Method

Ainsi, même si une personne obtient les mots de passe ou secrets, elle ne pourra pas y accéder si elle n'utilise pas la machine approuvée. Cela garantit que les accès sont strictement limités et contrôlés.

Une fois les machines créées, elles sont affectées au projet, garantissant ainsi un accès personnalisé et sécurisé à chaque développeur.

**Local**

**Identity Details**

Identity ID  
30c90e68-3d5f-4bdf-8028-2338839f1db4

Name  
Local

Organization Role  
member

Metadata  
-

**Authentication**

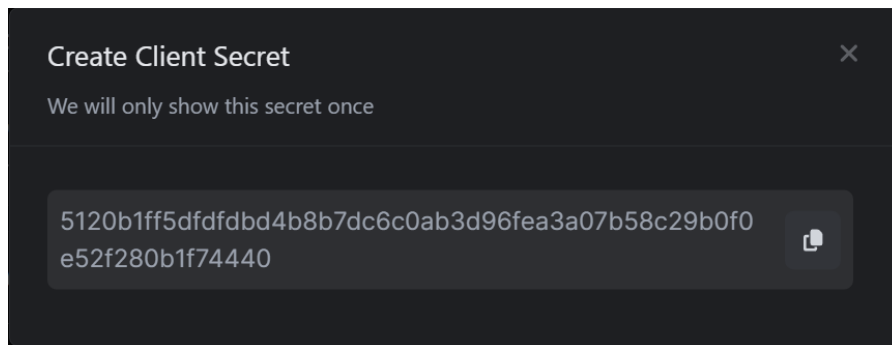
Auth Method  
Universal Auth

Client ID  
8e811abc-f91e-4cc6-8f7f-7bed86b1246a

**Projects**

NAME	ROLE	ADDED ON
DevSecOps	Developer	2024-11-15

Ensuite, une méthode d'authentification "Universal Auth" est configurée pour chaque machine. Cette méthode génère automatiquement un « machine secret », et utilise le project-id associé au projet "DevSecOps", qui contient le secret APP\_SECRET. Ces informations, nécessaires pour l'intégration et l'accès sécurisé, sont récupérées lors de la création d'Universal Auth.



Cela assure une connexion sécurisée entre les machines des développeurs et le projet, tout en permettant un accès contrôlé aux secrets du projet.

## Déploiement automatisé sur AWS EC2 en Staging

Pour le **déploiement automatisé sur AWS EC2**, un compte AWS a été créé afin de mettre en place l'infrastructure nécessaire pour le déploiement de l'application.

Sur la page EC2, une instance a été lancée pour l'environnement de staging.

### Étapes de création et configuration de l'instance :

1. **Lancement de l'instance** : Une instance Amazon EC2 a été créée avec Amazon Linux 2. Nous avons opté pour Amazon Linux 2 comme système d'exploitation pour l'instance EC2, car il est plus optimisé pour cet environnement. Amazon Linux 2 offre des performances améliorées et une sécurité renforcée.





2. **Configuration des paires de clés :** Une paire de clés **"key staging sever AL"** a été générée pour permettre une connexion SSH sécurisée à l'instance.

**Créer une paire de clés** [X]

Nom de la paire de clés  
Les paires de clés vous permettent de vous connecter à votre instance en toute sécurité.  
Key Staging Server AL  
La longueur maximale du nom est de 255 caractères ASCII. Il ne peut pas inclure d'espaces avant ou après.

Type de paire de clés

☒ RSA  
Paire de clés privée et publique chiffrée RSA

☐ ED25519  
Paire de clés privée et publique chiffrée ED25519

Format de fichier de clé privée

☒ .pem  
À utiliser avec OpenSSH

☐ .ppk  
À utiliser avec PuTTY

⚠ Lorsque vous y êtes invité, stockez la paire de clés dans un emplacement sécurisé et accessible sur votre ordinateur. Vous en aurez besoin ultérieurement pour vous connecter à votre instance. [En savoir plus](#)

Annulez [Créer une paire de clés]

3. **Paramétrage du réseau :** Pour assurer l'accessibilité de l'application, un **groupe de sécurité** nommé **launch-wizard-8** a été configuré :
- **Autorisation du trafic SSH** depuis 0.0.0.0/0 (à restreindre idéalement pour plus de sécurité).
  - **Autorisation du trafic HTTP et HTTPS** pour garantir l'accès web sécurisé et non sécurisé à l'application.

▼ Paramètres réseau Informations [Modifier]

Réseau Informations  
vpc-0fff5199483edbd8f

Sous-réseau Informations  
Aucune préférence (sous-réseau par défaut dans n'importe quelle zone de disponibilité)

Attribuer automatiquement l'adresse IP publique Informations  
Activer

Des frais supplémentaires s'appliquent en cas de dépassement de la limite de l'offre gratuite

Pare-feu (groupes de sécurité) Informations  
Un groupe de sécurité est un ensemble de règles de pare-feu qui contrôlent le trafic de votre instance. Ajoutez des règles pour autoriser un trafic spécifique à atteindre votre instance.

☒ Créer un groupe de sécurité ☐ Sélectionner un groupe de sécurité existant

Nous allons créer un nouveau groupe de sécurité appelé « **launch-wizard-8** » avec les règles suivantes :

☒ Autoriser le trafic SSH depuis  
Vous permet de vous connecter à votre instance. N'importe où 0.0.0.0/0

☒ Autoriser le trafic HTTPS depuis l'Internet  
Pour configurer un point de terminaison, par exemple lors de la création d'un serveur web

☒ Autoriser le trafic HTTP depuis l'Internet  
Pour configurer un point de terminaison, par exemple lors de la création d'un serveur web

⚠ Les règles avec la source 0.0.0.0/0 autorisent toutes les adresses IP à accéder à votre instance. Nous vous recommandons de définir des règles de groupe de sécurité pour autoriser l'accès à partir d'adresses IP connues uniquement. [X]

## Connexion à l'instance et configuration initiale :

Après la connexion à l'instance, les commandes suivantes ont été exécutées pour préparer l'environnement requis par le job de déploiement ssh staging utilisant PHP-FPM :

1. **Mettre à jour le système** : `sudo yum update -y`
2. **Installer Apache2** : `sudo yum install -y httpd`
3. **Installer Git** : `sudo yum install -y git`
4. **Installer PHP 8.2 et PHP-FPM** :  
`sudo amazon-linux-extras enable php8.2`  
`sudo yum clean metadata`  
`sudo yum install -y php php-fpm`
5. **Installer les extensions PHP nécessaires** : `sudo yum install -y php-gd php-mysqlnd php-pdo php-zip php-sqlite3 php-xml`
6. **Configurer PHP-FPM** : `sudo nano /etc/php-fpm.d/www.conf`

### Contenu du fichier de configuration :

```
listen = /run/php-fpm/www.sock
listen.owner = apache
listen.group = apache
listen.mode = 0660
```

7. **Créer le répertoire de l'application avec les permissions appropriées** :

```
sudo mkdir -p /var/www/myapp
sudo chown -R ec2-user:apache /var/www/myapp
sudo chmod -R 750 /var/www/myapp
```

8. **Créer le fichier de configuration Apache** : `sudo nano /etc/httpd/conf.d/myapp.conf`

```
<VirtualHost *:80>
    ServerAdmin webmaster@myapp.com
    DocumentRoot /var/www/myapp/public
    ServerName ec2-51-20-70-218.eu-north-1.compute.amazonaws.com
    ErrorLog /var/log/httpd/myapp-error.log
    CustomLog /var/log/httpd/myapp-access.log combined
    <Directory /var/www/myapp/public>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
    <FilesMatch \.php$>
        SetHandler "proxy:unix:/run/php-fpm/www.sock|fcgi://localhost/"
    </FilesMatch>
</VirtualHost>
```

9. **Activer les modules Apache nécessaires** :

```
sudo sed -i 's/#LoadModule rewrite_module/LoadModule rewrite_module/'
/etc/httpd/conf.modules.d/00-base.conf
sudo sed -i 's/#LoadModule proxy_module/LoadModule proxy_module/'
/etc/httpd/conf.modules.d/00-proxy.conf
sudo sed -i 's/#LoadModule proxy_fcgi_module/LoadModule proxy_fcgi_module/'
/etc/httpd/conf.modules.d/00-proxy.conf
```

**10. Installer Composer avec sudo :** `sudo curl -sS https://getcomposer.org/installer | sudo php -- --install-dir=/usr/local/bin --filename=composer`

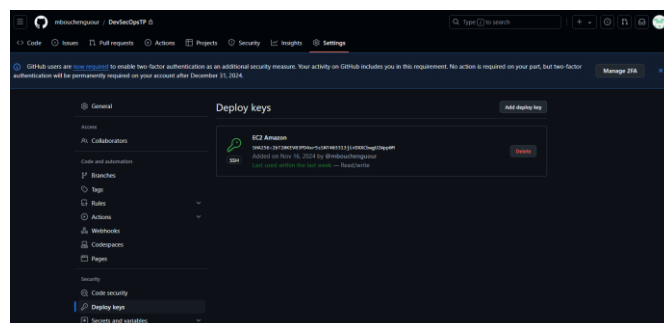
### 11. Démarrer et activer les services :

```
sudo systemctl start httpd
sudo systemctl enable httpd
sudo systemctl start php-fpm
sudo systemctl enable php-fpm
```

## Génération et ajout de la clé SSH pour GitHub :

Pour sécuriser les interactions entre l'instance EC2 et le dépôt GitHub, une clé SSH a été générée et ajoutée comme clé de déploiement.

- 1. Génération de la clé SSH :** La commande suivante a été utilisée pour créer une clé SSH RSA de 4096 bits, associée à l'email mbouchenguour@gmail.com :  
`ssh-keygen -t rsa -b 4096 -C "mbouchenguour@gmail.com"`
- 2. Affichage de la clé publique :** La clé publique nouvellement générée a été affichée avec la commande suivante :  
`cat ~/.ssh/id_rsa.pub`
- 3. Ajout de la clé de déploiement dans GitHub :** La clé publique a ensuite été ajoutée dans la section Deploy keys du dépôt GitHub, avec les autorisations de lecture seule pour garantir la sécurité.



[Deploy keys](#) / Add new

Title

AL EC2 STAGING


Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQADR0OuZ/ewtLtTgvmADRSUecB/rzr52MnNxVcQH548obDBO6B+x4lrW4ns46cW
hW26WbMCsnvg6J5sckm7Iewi2cNm8YRgrbpw/BIPh10YkCM99vtzXaszvDellUFaDida58VaxgHP0G0rsEAFW5SMQ2p7UR
udKc2m16TRZgWKmmjOY651mXpo9Y7X1HjlanYCW5bt7xgTvTbcM+EH0Foj7I2Zoz6lk6wQH4yvBHW0r3Ob99pd7PYGK
UEZeCaP8sY8gS9gFHoF8N456/8jnRn9X47KT12Nvmt3UI49XCBLzcXONYYNcAncQ+LcbbtV4vQWc4YsgeyRxPuzZ5gRVCH
10gawkmOBZyG1Ekws/OK8EdqMk516uqS0C+nR7qHENegS3TR4VvTnH+hSRyE8Xs6ppmdE4tsndpB4Gis1cKtavJKSCB1
HFpa4Y3Kjcelcaz0AiptndP1LK9JdhfmWSL9M2Ktu4ju0IW5VC8KUModNaOtX+/74ArfVKRa1ORxYTO1Bkc9XCaTMWgJ53
FVPhP09qCOyIfMuUL/rsV8ZC6n4Tnfg1A+oDD3jO2BcVgWA7BhmKh6HqJVvgfFpQ4QsDVTUhnVB+8zxG9g0mTIPit72B
KriaaSQ/LrMg0gf1sdt7qsyevbtp1Ud0BJ3KI8INdww43HLYBDUwleQ== mbouchenguour@gmail.com
```

begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'.

☐ Allow write access  
Can this key be used to push to this repository? Deploy keys always have pull access.

Add key


**AL EC2 STAGING**

SHA256:k1j1JgwYm/fEv515PW94fe7kzkyiL0df2MAPR6ssQR4

Added on Nov 19, 2024 by @mbouchenguour

Last used within the last week — Read-only

Delete

## Clonage du dépôt

Pour configurer l'environnement de déploiement sur l'instance EC2, les étapes suivantes ont été réalisées :

1. **Accès au répertoire** : `cd /var/www/myapp`
2. **Clonage du dépôt GitHub** : `git clone git@github.com:mbouchenguour/DevSecOpsTP.git` .

Le dépôt contenant l'application est cloné directement dans le répertoire.

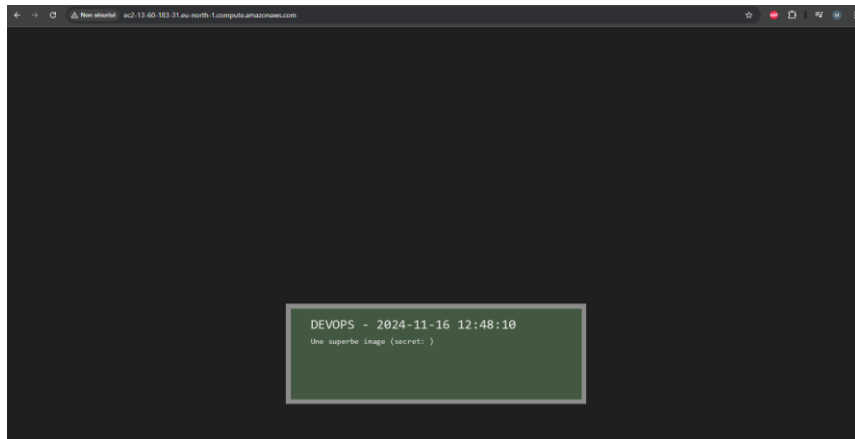
3. **Création et configuration du fichier .env** :
  - **Création et modification des permissions** :  
`sudo touch .env`  
`sudo chmod 640 .env`  
`sudo chown ec2-user:apache .env`

Le fichier .env est créé avec des permissions restrictives pour garantir la sécurité et attribué à l'utilisateur et groupe corrects.

- **Ajout de la variable APP\_SECRET** : Le fichier .env est édité pour inclure la variable :  
`APP_SECRET=TPDev`

Cette configuration est utilisée pour les tests, permettant de vérifier le bon fonctionnement de l'application avec une valeur de secret spécifique.

**Problème rencontré :** Lors de l'ajout de la variable APP\_SECRET=TPDev pour le test, rien ne s'affichait sur la page, laissant la variable vide.



**Solution apportée :** Pour corriger ce problème, une boucle parcourant les variables d'environnement a été ajoutée dans le fichier index.php, qui utilise putenv() pour réinjecter chaque variable d'environnement dans l'environnement global PHP.

```
$dotenv = Dotenv\Dotenv::createImmutable(dirname(__DIR__));
$dotenv->safeLoad();

foreach ($_ENV as $key => $value) {
    putenv("$key=$value");
}
```

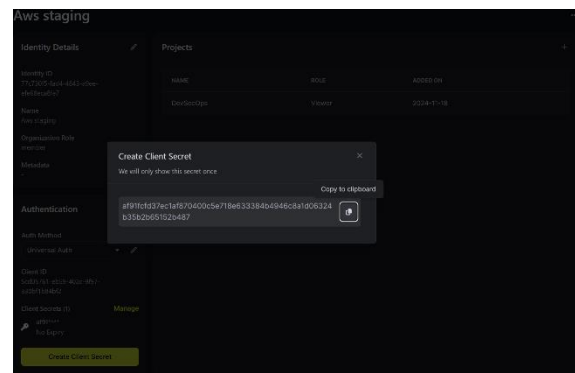
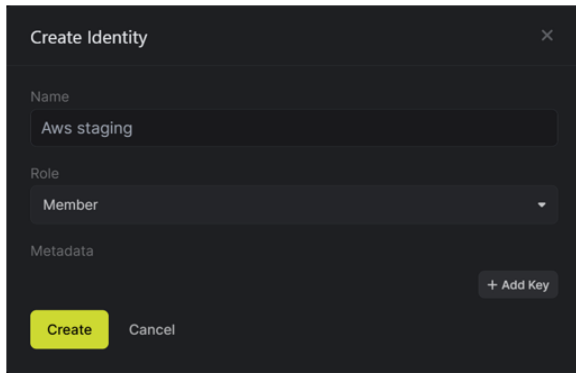
Cela permet à l'application d'accéder correctement à la variable APP\_SECRET.

**Résultat obtenu :** Après l'ajout de la boucle, la variable APP\_SECRET s'affiche correctement dans l'application. Aucune autre solution n'a été trouvée à ce moment-là, mais il est probable que le problème soit lié à la configuration ou au comportement de php-fpm avec les variables d'environnement.

## Gestion des secrets avec Infisical

### Création d'une identité pour l'accès aux secrets :

Une identité Aws staging a été créée avec le rôle Member. Un client secret a été généré pour cette identité afin qu'elle accède aux secrets.



### Ajout de l'identité au projet avec le rôle Viewer :

L'identité *Aws staging* a été ajoutée au projet *DevSecOps* avec le rôle *Viewer* pour garantir que la machine peut accéder aux secrets nécessaires sans permissions supplémentaires. Ce choix limite les actions possibles, renforçant la sécurité en accordant uniquement le niveau d'accès minimal requis tout en liant la machine au projet.

### Connexion et export des secrets :

Le script Bash **fetchSecrets.sh** est créé et mis dans le dépôt Git pour se connecter à Infisical, obtenir un token d'authentification et exporter les secrets nécessaires dans un fichier **.env**. Voici une explication détaillée de chaque section du script :

#### Vérification des Variables d'Environnement

Le script commence par vérifier que les variables d'environnement nécessaires sont définies :

```
# Ensure that the required environment variables are set
if [ -z "$INFISICAL_MACHINE_CLIENT_ID" ] || [ -z "$INFISICAL_MACHINE_CLIENT_SECRET" ] || [ -z "$PROJECT_ID" ] || [ -z "$INFISICAL_SECRET_ENV" ]; then
    echo "Error: One or more required environment variables are not set."
    echo "Please set INFISICAL_MACHINE_CLIENT_ID, INFISICAL_MACHINE_CLIENT_SECRET, PROJECT_ID, and INFISICAL_SECRET_ENV."
    exit 1
fi
```

Cette section vérifie que les variables **INFISICAL\_MACHINE\_CLIENT\_ID**, **INFISICAL\_MACHINE\_CLIENT\_SECRET**, **PROJECT\_ID**, et **INFISICAL\_SECRET\_ENV** sont définies. Si l'une de ces variables n'est pas définie, le script affiche un message d'erreur et arrête l'exécution.

#### Installation de l'Infisical CLI

Le script installe ensuite l'Infisical CLI si celui-ci n'est pas déjà installé :

```
# Install the Infisical CLI if not already installed
if ! command -v infisical &> /dev/null; then
    echo "Infisical CLI not found. Installing..."

    # Add Infisical repository and install the CLI
    curl -sLf 'https://dl.cloudsmith.io/public/infisical/infisical-cli/setup.rpm.sh' | sudo -E bash
    sudo yum install -y infisical

    # Verify installation
    if command -v infisical &> /dev/null; then
        echo "Infisical CLI installed successfully."
    else
        echo "Failed to install Infisical CLI. Please check the logs."
        exit 1
    fi
fi
```

Cette section vérifie si l'Infisical CLI est installé. Si ce n'est pas le cas, le script télécharge et installe l'Infisical CLI, puis vérifie l'installation.

### Obtention du Token Infisical

Le script obtient ensuite un token d'authentification à l'aide des identifiants fournis (client ID et client secret) :

```
# Obtain the Infisical token using the provided client ID and secret
INFISICAL_TOKEN=$(infisical login --method=universal-auth \
--client-id="$INFISICAL_MACHINE_CLIENT_ID" \
--client-secret="$INFISICAL_MACHINE_CLIENT_SECRET" \
--silent --plain \
--domain=https://eu.infisical.com)
```

Cette commande utilise les informations d'identité créées sur Infisical pour s'authentifier et obtenir un token.

### Vérification de l'Obtention du Token

Le script vérifie si le token a été obtenu avec succès. Si ce n'est pas le cas, il affiche un message d'erreur et arrête l'exécution :

```
# Check if the token was obtained successfully
if [ -z "$INFISICAL_TOKEN" ]; then
    echo "Error: Failed to obtain the Infisical token."
    exit 1
fi
```

### Exportation des Secrets

Une fois le token obtenu, le script utilise ce token pour exporter les secrets dans un fichier **.env** :

```
# Export all secrets from the specified environment to the .env file
infisical export \
--env="$INFISICAL_SECRET_ENV" \
--projectId="$PROJECT_ID" \
--token="$INFISICAL_TOKEN" \
--domain=https://eu.infisical.com > .env
```

Cette commande permet de récupérer les secrets nécessaires à l'environnement spécifié et de les écrire dans le fichier **.env** de l'application.

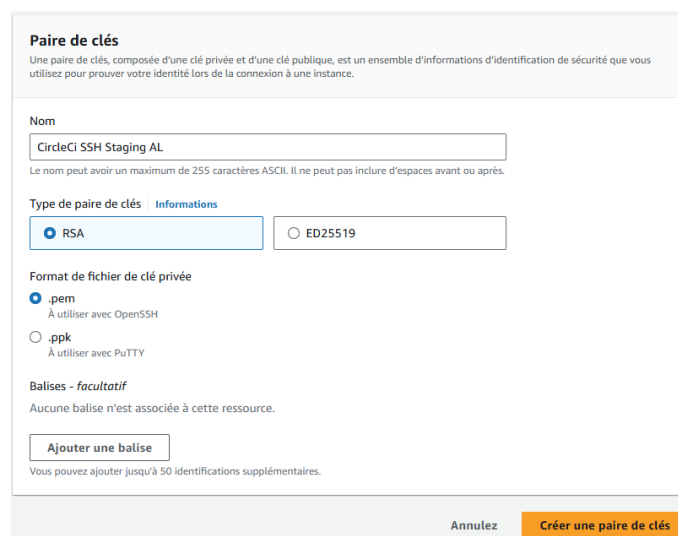
Désormais, les développeurs devront également utiliser ce script, ou l'adapter en fonction du système d'exploitation de leurs machines.

## Déploiement depuis CircleCI

Pour permettre à CircleCI de se connecter à notre serveur, une paire de clés SSH a été générée spécifiquement pour cet usage. Voici le processus détaillé :

### 1. Génération de la paire de clés :

- Une nouvelle paire de clés a été créée via l'interface AWS avec le nom CircleCi SSH Staging AL. Le type de clé choisi est RSA et elle est sauvegardée au format .pem.



- La clé publique a ensuite été générée à partir de la clé privée avec la commande suivante :

```
ssh-keygen -y -f './CircleCi SSH Staging AL.pem' > './CircleCi SSH Staging AL.pub'
```

- La clé publique a été ajoutée au fichier `~/ssh/authorized_keys` de l'instance staging pour autoriser les connexions SSH.

### 2. Ajout de la clé privée à CircleCI :

- La clé privée a été ajoutée à CircleCI dans la section des clés SSH pour permettre à CircleCI d'établir une connexion sécurisée à l'instance EC2.
- Lors de l'ajout, le hostname spécifique (adresse IP) du serveur a été indiqué pour restreindre l'accès de la clé à cette machine.



**Add an SSH Key**

Hostname  
51.20.70.218

Private Key\*

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEA6B7F4pDj7UDXKjw6ix0h0bwfJ0
blGQfN3/Db48/yZrsi371
JsBm6wt7crx20vqhxtzgNV6tRdzH3RAzBZfoMSXCVE
s7lm2vqHuM6wbxw1Y1D67p
Sm0vFIEct9n/H8RnSI R900N13 IVE7Kfa6 IMtutuzi7?R0E1
-----
```

Cancel Add SSH Key

### 3. Empreinte de la clé :

- L'empreinte de la clé (SHA256) a été enregistrée puis ajoutée en tant que variable d'environnement `STAGING_SSH_FINGERPRINT` dans CircleCI pour permettre une vérification de l'authenticité des connexions.

## Ajout des variables d'environnement à CircleCi

Ensuite, les variables `STAGING_SSH_USER` (ec2-user), `STAGING_SSH_HOST` (adresse de l'hôte : 51.20.70.218) et `STAGING_DEPLOY_DIRECTORY` (/var/www/myapp) ont été ajoutées dans CircleCI pour permettre les connexions et les déploiements sur l'environnement de staging lors du job `deploy-ssh-staging`.

Les variables `STAGING_INFISICAL_MACHINE_CLIENT_ID`, `STAGING_INFISICAL_MACHINE_CLIENT_SECRET` et `PROJECT_ID` ont été ajoutées dans CircleCI pour permettre l'exécution du script `./fetchSecrets`

## Modification du job existant

Le job a été modifié pour exécuter le script `./fetchSecrets` avec les variables nécessaires (`STAGING_INFISICAL_MACHINE_CLIENT_ID`, `STAGING_INFISICAL_MACHINE_CLIENT_SECRET` et `PROJECT_ID`), permettant d'injecter les secrets directement dans le fichier `.env`. De plus, avec Amazon Linux 2, il n'est plus nécessaire de spécifier la version de PHP pour lancer `php-fpm`, ce qui permet de redémarrer avec la commande `service php-fpm restart`.

### Problème rencontré lors de l'exécution du job :

Lors de l'exécution, un problème est survenu car les variables n'étaient pas connues par la machine AWS. Elles sont connus uniquement sur CircleCi.

### Solution apportée :

Pour résoudre ce problème, les variables ont été transmises directement dans la commande SSH.

```
ssh -o StrictHostKeyChecking=no $STAGING_SSH_USER@$STAGING_SSH_HOST \  
"CIRCLE_BRANCH='$CIRCLE_BRANCH' \  
STAGING_DEPLOY_DIRECTORY='$STAGING_DEPLOY_DIRECTORY' \  
STAGING_INFISICAL_MACHINE_CLIENT_ID='$STAGING_INFISICAL_MACHINE_CLIENT_ID' \  
STAGING_INFISICAL_MACHINE_CLIENT_SECRET='$STAGING_INFISICAL_MACHINE_CLIENT_SECRET' \  
PROJECT_ID='$PROJECT_ID' \  
bash -s" \<< 'EOF'
```

### Résultat obtenu :

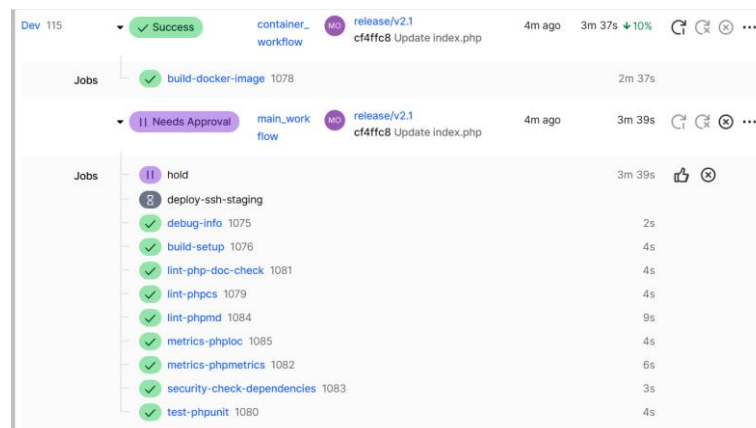
La machine distante reconnaît désormais les variables et le déploiement peut fonctionner correctement.

## Ajustements et Tests

Pour vérifier que le déploiement s'effectue correctement lors du push, les tests ont été réalisés en mettant à jour le contenu d'index.php pour afficher le texte Une superbe image en Staging et en modifiant le secret dans Infisical à TPDevSecOpsStaging.

### Résultat :

1. Tous les tests du workflow CircleCI ont été exécutés avec succès, et après validation de ces tests, le job de déploiement deploy-ssh-staging a été approuvé pour poursuivre le déploiement en staging.



2. Le job Deploy to AWS s'est exécuté avec succès, incluant la transmission des variables d'environnement et la gestion des secrets via Infisical.

```

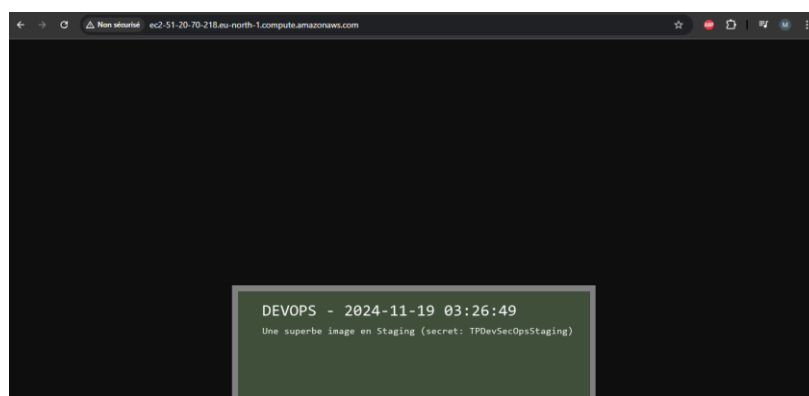
12 git pull origin $CIRCLE_BRANCH
13 chmod +x ./fetchSecrets.sh
14
15 INFISICAL_MACHINE_CLIENT_ID=$STAGING_INFISICAL_MACHINE_CLIENT_ID \
16 INFISICAL_MACHINE_CLIENT_SECRET=$STAGING_INFISICAL_MACHINE_CLIENT_SECRET \
17 PROJECT_ID=$PROJECT_ID \
18 INFISICAL_SECRET_ENV=staging \
19 ./fetchSecrets.sh
20
21 composer install --optimize-autoloader --no-interaction --prefer-dist
22 (flock -w 10 9 || exit 1; sudo -S service php-fpm restart ) 9>/tmp/fpm.lock
23 EOF
24
25 + ssh -o StrictHostKeyChecking=no *****@***** 'CIRCLE_BRANCH='release/v2.1'' STAGING_DEPLOY
26 Warning: Permanently added '*****' (ECDSA) to the list of known hosts.
27 From github.com:*****/DevSecOpsTP
28 * branch      release/v2.1 -> FETCH_HEAD
29 cf4ffc8..f622d62 release/v2.1 -> origin/release/v2.1
30 Updating 476417e..f622d62
31 Fast-forward
32  .circleci/config.yml | 9 ++++++
33  1 file changed, 8 insertions(+), 1 deletion(-)
34 Successfully updated the .env file with secrets from Infisical.
35 Installing dependencies from lock file (including require-dev)
36 Verifying lock file contents can be installed on current platform.
37 Nothing to install, update or remove
38 Generating optimized autoload files
39 35 packages you are using are looking for funding.
40 Use the 'composer fund' command to find out more!
41 Redirecting to /bin/systemctl restart php-fpm.service
42 CircleCI received exit code 0

```

- Le workflow complet a été validé, indiquant que toutes les étapes de construction et de déploiement ont réussi.

Dev 115	<div> <div>✓ Success</div> <div>container_workflow</div> <div>MO</div> <div>release/v2.1</div> <div>cf4ffc8 Update index.php</div> <div>47m ago</div> <div>3m 38s ↓9%</div> <div> </div> </div>
	<div> <div>✓ Success</div> <div>main_workflow</div> <div>MO</div> <div>release/v2.1</div> <div>cf4ffc8 Update index.php</div> <div>47m ago</div> <div>5m 0s ↓39%</div> <div> </div> </div>

- Le déploiement a été finalisé avec succès, affichant la page de l'application mise à jour avec le texte Une superbe image en Staging et le secret TPDevSecOpsStaging récupéré via Infisical, confirmant que toutes les étapes ont fonctionné correctement.



# Déploiement automatisé sur AWS EC2 en Production

Pour le déploiement automatisé en production sur AWS EC2, nous avons utilisé un serveur Amazon Linux 2, comme pour l'environnement de staging. Cela est essentiel pour garantir une homogénéité entre les deux environnements, ce qui simplifie la gestion, le déploiement et la résolution des problèmes tout en minimisant les différences entre les configurations de staging et de production.

## Étapes de création et configuration de l'instance

Pour le déploiement en production, nous avons repris les mêmes étapes que pour l'environnement de staging.

### Lancement de l'Instance

Une instance Amazon EC2 a été créée avec Amazon Linux 2, identique à celle utilisée pour staging, afin de garantir une cohérence entre les environnements.

### Configuration des Paires de Clés

Une paire de clés nommée key production a été générée pour permettre une connexion SSH sécurisée à l'instance.

### Paramétrage du Réseau

Un groupe de sécurité launch-wizard-production a été configuré pour autoriser le trafic SSH depuis des adresses IP spécifiques et permettre l'accès HTTP et HTTPS pour garantir la sécurité et l'accessibilité de l'application.

## Connexion à l'instance et configuration initiale

Pour la configuration de l'instance de production, nous avons exécuté les mêmes commandes que pour l'environnement de staging afin de préparer l'environnement requis, incluant la mise à jour du système, l'installation d'Apache, Git, PHP 8.2, PHP-FPM, ainsi que les extensions nécessaires. Le répertoire de l'application a été créé avec les permissions appropriées, et les modules Apache nécessaires ont été activés.

La seule différence notable est le changement du ServerName dans le fichier de configuration VirtualHost pour refléter le nom de l'instance de production (ec2-16-16-211-108.eu-north-1.compute.amazonaws.com).

## Génération et ajout de la clé SSH pour GitHub

Pour la production, nous avons suivi la même procédure :

### Génération et ajout de la clé SSH pour GitHub :

Pour sécuriser les interactions entre l'instance de production et le dépôt GitHub, une clé SSH a été générée et ajoutée comme clé de déploiement.

1. **Génération de la clé SSH** : Une clé SSH RSA de 4096 bits a été créée avec la commande : `ssh-keygen -t rsa -b 4096 -C "mbouchenguour@gmail.com"`
2. **Affichage de la clé publique** : La clé publique a été affichée avec la commande : `cat ~/.ssh/id_rsa.pub`
3. **Ajout de la clé de déploiement dans GitHub** : La clé publique a été ajoutée dans la section **Deploy keys** du dépôt GitHub, avec des autorisations de lecture seule pour garantir la sécurité.

## Clonage du dépôt :

Pour la production, nous avons effectué la même procédure que pour l'environnement de staging, incluant le clonage du dépôt, la création et la configuration du fichier `.env` avec les permissions appropriées et l'ajout de la variable `APP_SECRET`.

## Gestion des secrets avec Infisical

Pour la gestion des secrets en production avec Infisical, nous avons suivi le même processus que pour l'environnement de staging. Une identité pour la machine de production a été créée avec les mêmes permissions et rôles nécessaires pour accéder aux secrets du projet. Le rôle assigné à cette identité permet uniquement un accès en lecture, garantissant la sécurité en limitant les actions possibles.

Le script `fetchSecrets.sh`, qui a déjà été défini, sera directement depuis le dépôt lors du clonage. Ce script se charge de se connecter à Infisical, d'obtenir un token d'authentification, puis d'exporter les secrets requis dans un fichier `.env`. Cela assure que l'environnement de production dispose des secrets nécessaires sans nécessiter de configurations supplémentaires manuelles.

## Déploiement depuis CircleCI

Pour le déploiement depuis CircleCI, nous avons suivi la même procédure que pour l'environnement de staging afin de permettre une connexion sécurisée à notre serveur de production.

## Génération de la paire de clés SSH

Une paire de clés SSH a été créée spécifiquement pour la connexion à l'instance, avec le même type de clé et format que pour staging. La clé publique a été ajoutée au fichier `~/.ssh/authorized_keys` de l'instance de production pour autoriser les connexions SSH.

## Ajout de la clé privée à CircleCI

La clé privée a été ajoutée à CircleCI pour permettre l'établissement de connexions sécurisées depuis CircleCI vers l'instance EC2. Le hostname du serveur a également été spécifié pour restreindre l'accès de la clé à cette machine.

## Empreinte de la clé

L'empreinte de la clé (SHA256) a été enregistrée et configurée en tant que variable d'environnement dans CircleCI pour vérifier l'authenticité des connexions, tout comme dans l'environnement de staging.

## Ajout des variables d'environnement à CircleCi

Pour l'environnement de production, les mêmes variables que pour l'environnement de staging ont été ajoutées dans CircleCI, mais adaptées pour la production. Les variables `PRODUCTION_SSH_USER`, `PRODUCTION_SSH_HOST` et `PRODUCTION_DEPLOY_DIRECTORY` ont été configurées pour permettre les connexions et les déploiements lors du job `deploy-ssh-production`.

De plus, les variables nécessaires à l'exécution du script `./fetchSecrets` telles que `PRODUCTION_INFISICAL_MACHINE_CLIENT_ID`, `PRODUCTION_INFISICAL_MACHINE_CLIENT_SECRET` et `PROJECT_ID` ont également été ajoutées pour garantir que les secrets sont correctement récupérés lors du déploiement.

## Ajout du job `deploy-ssh-production`

Pour le job de déploiement en production, nous utilisons le même job que celui configuré pour l'environnement de staging, avec les variables adaptées pour la production. Cela permet de maintenir une cohérence dans le processus tout en garantissant que les connexions et les opérations sont spécifiques à l'environnement de production grâce aux variables `PRODUCTION_SSH_USER`, `PRODUCTION_SSH_HOST`, `PRODUCTION_DEPLOY_DIRECTORY` et autres nécessaires.

Pour ajouter le job `deploy-ssh-production` dans le `main_workflow` de manière similaire au job `deploy-ssh-staging`, il suffit de le configurer de la même manière en respectant les mêmes dépendances et filtres, mais en adaptant les branches pour le `main/master`.

```
- deploy-ssh-production:
  requires:
    - hold
  filters:
    branches:
      only:
        - main
        - master
- deploy-ssh-staging:
  requires:
    - hold
  filters:
    branches:
      only:
        - /^release\/.*/
```

## Ajustements et Tests

Pour vérifier le déploiement en production, plusieurs ajustements et tests ont été effectués afin de valider que tout fonctionne correctement.

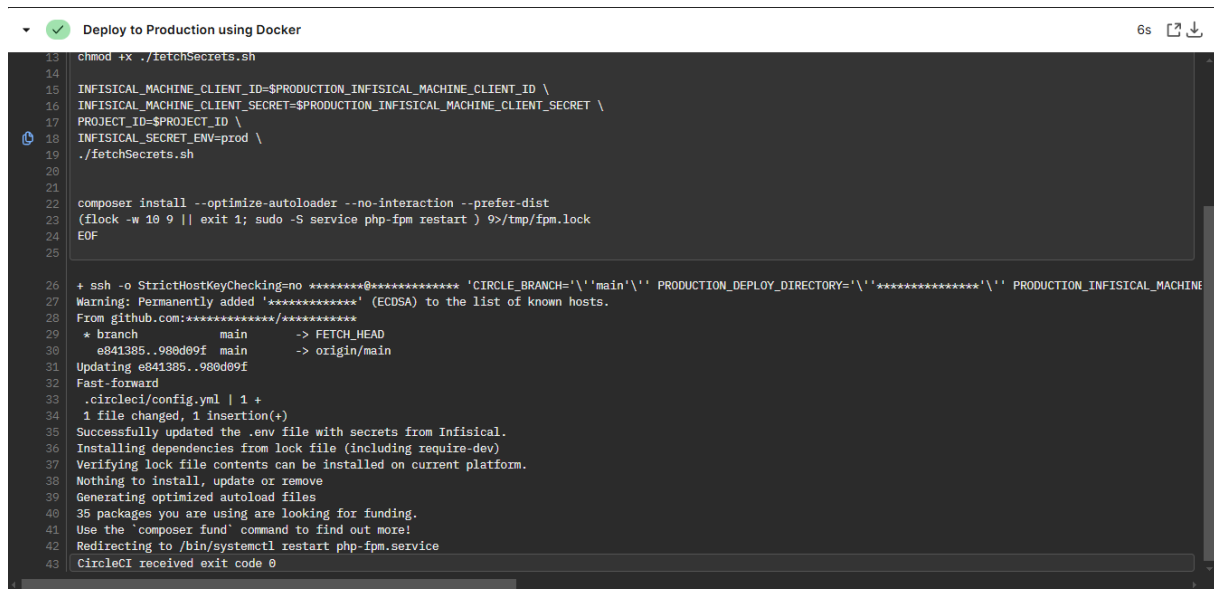
### 1. Modifications nécessaires avant le déploiement :

- **Modification dans `index.php`** : Le texte a été mis à jour de "Une superbe image" à "Une superbe image en Production".
- **Mise à jour des secrets dans `Infisical`** : Le secret `TPDevSecOps` a été remplacé par `TPDevSecOpsProduction` pour les besoins de l'environnement de production.

### 2. Tests des workflows `CircleCI` : Les workflows configurés ont été validés avec succès, confirmant que tout est en ordre. Suite à cette validation, le job `deploy-ssh-production` a été approuvé pour lancer le déploiement, puis réussi.

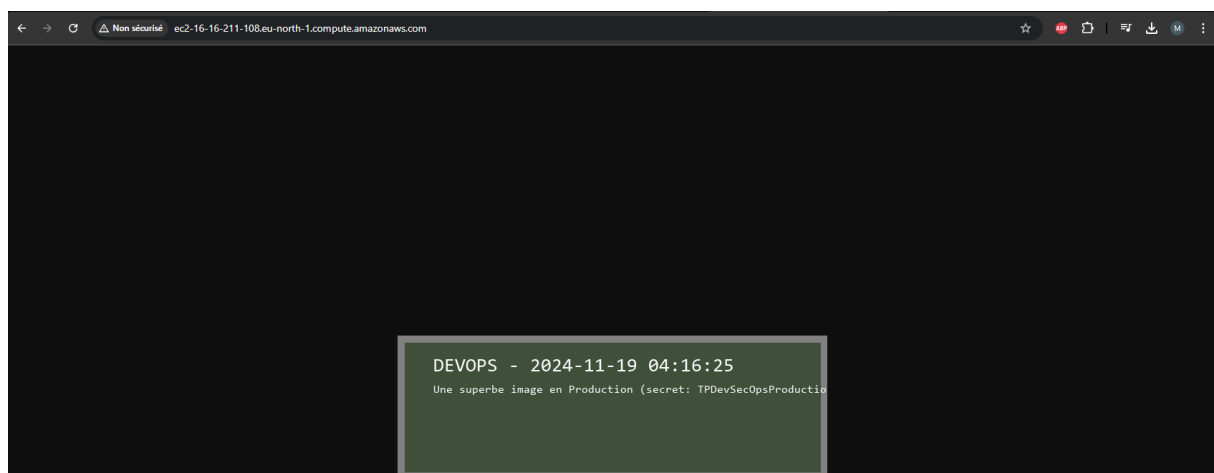
The screenshot displays the CircleCI interface for a repository named 'DevSecOps 11'. It shows two successful workflow runs. The first workflow, 'container\_workflow', has a job 'build-docker-image' (ID 122) that completed successfully 2m 28s ago. The second workflow, 'main\_workflow', also completed successfully 4m ago. This workflow includes a series of jobs: 'hold' (121), 'deploy-ssh-production' (123), 'debug-info' (119), 'build-setup' (120), 'lint-php-doc-check' (127), 'lint-phpcs' (125), 'lint-phpmd' (130), 'metrics-phploc' (128), 'metrics-phpmetrics' (126), 'security-check-dependencies' (124), and 'test-phpunit' (129). All jobs in the 'main\_workflow' are marked as successful with green checkmarks.

3. **Exécution du déploiement avec Docker** : Le job deploy-ssh-production a téléchargé et déployé l'image la plus récente de l'application depuis le registre GHCR. Le conteneur existant a été arrêté, puis le nouveau conteneur a démarré avec les configurations nécessaires.



```
13 chmod +x ./fetchSecrets.sh
14
15 INFISICAL_MACHINE_CLIENT_ID=$PRODUCTION_INFISICAL_MACHINE_CLIENT_ID \
16 INFISICAL_MACHINE_CLIENT_SECRET=$PRODUCTION_INFISICAL_MACHINE_CLIENT_SECRET \
17 PROJECT_ID=$PROJECT_ID \
18 INFISICAL_SECRET_ENV=prod \
19 ./fetchSecrets.sh
20
21
22 composer install --optimize-autoloader --no-interaction --prefer-dist
23 (flock -w 10 9 || exit 1; sudo -S service php-fpm restart ) 9>/tmp/fpm.lock
24 EOF
25
26
27 + ssh -o StrictHostKeyChecking=no *****@***** 'CIRCLE_BRANCH='main'' PRODUCTION_DEPLOY_DIRECTORY='*****' PRODUCTION_INFISICAL_MACHINE
Warning: Permanently added '*****' (ECDSA) to the list of known hosts.
28 From github.com:*****/*
29 * branch          main      -> FETCH_HEAD
30 e841385..980d09f  main      -> origin/main
31 Updating e841385..980d09f
32 Fast-forward
33  .circleci/config.yml | 1 +
34  1 file changed, 1 insertion(+)
35 Successfully updated the .env file with secrets from Infisical.
36 Installing dependencies from lock file (including require-dev)
37 Verifying lock file contents can be installed on current platform.
38 Nothing to install, update or remove
39 Generating optimized autoload files
40 35 packages you are using are looking for funding.
41 Use the 'composer fund' command to find out more!
42 Redirecting to /bin/systemctl restart php-fpm.service
43 CircleCI received exit code 0
```

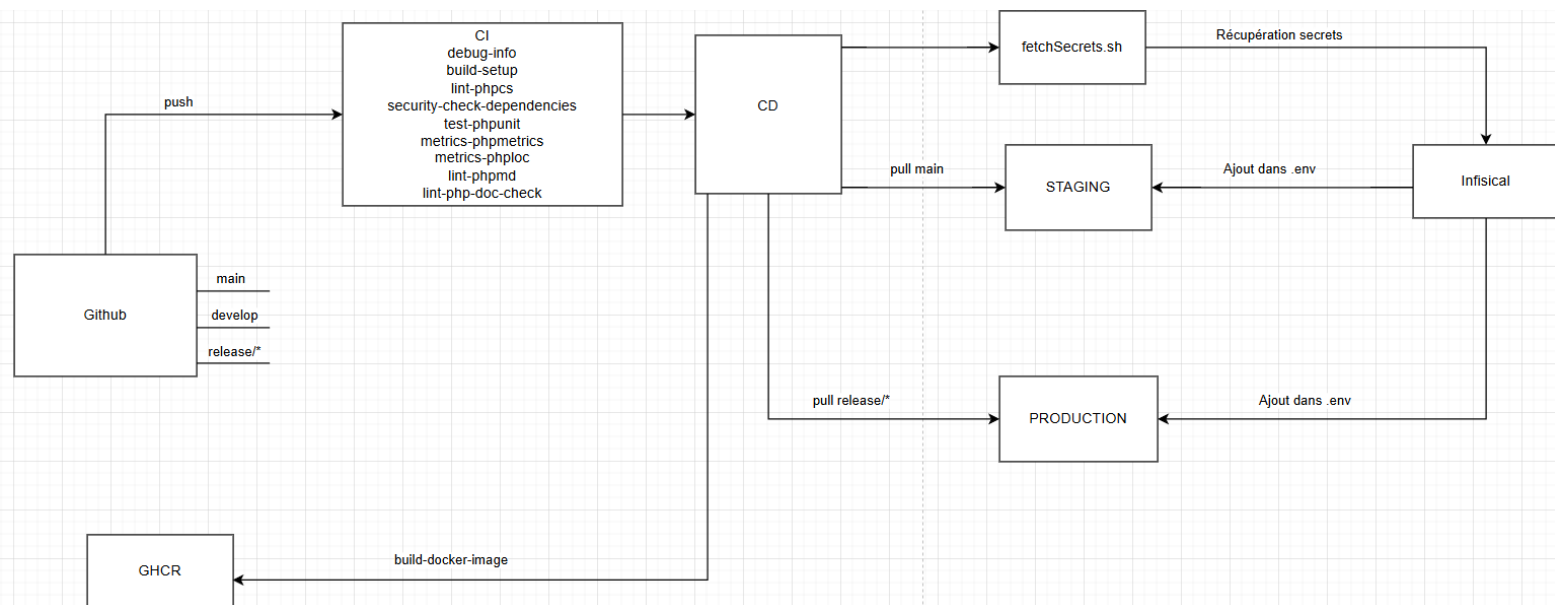
4. **Affichage du site** : Une capture d'écran du site de production confirme que le déploiement a été effectué avec succès, affichant le texte "Une superbe image en Production" dans index.php et confirmant l'utilisation du secret TPDevSecOpsProduction à partir d'Infisical.



Ces ajustements et tests ont permis de valider le bon déploiement de l'application en production.



## Schéma récapitulatif



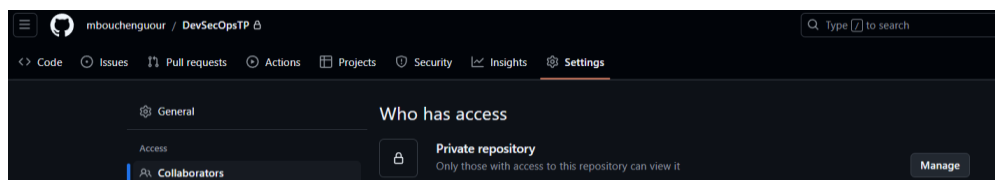
## Sécurité à tous les étages

### Êtes-vous sûrs d'avoir bien configuré votre repository et compte GitHub ?

Pour vérifier la sécurité du dépôt GitHub et du compte associé, plusieurs mesures ont été mises en place et sont présentées ci-dessous :

#### 1. Le dépôt GitHub est privé

Le dépôt est configuré comme **privé**, ce qui signifie que l'accès au code source est limité aux membres autorisés du projet. Cela empêche les utilisateurs non autorisés d'accéder aux fichiers et protège les données sensibles.

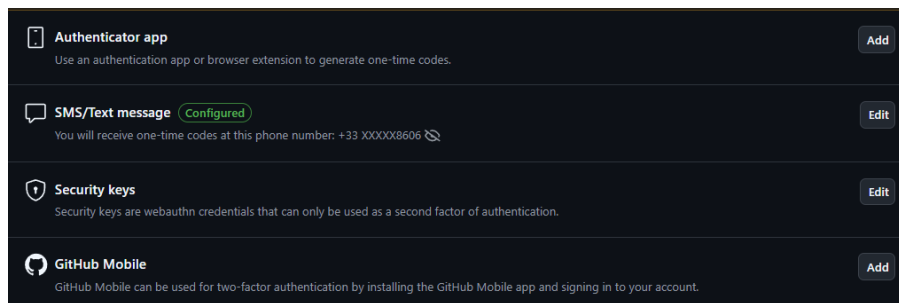


#### 2. Activation de l'authentification à deux facteurs (2FA)

L'authentification à deux facteurs a été **activée** pour chaque membre de l'équipe afin de sécuriser les accès aux comptes GitHub. Cette mesure exige un code de vérification lors

de chaque connexion, en plus du mot de passe. Certains membres utilisent la méthode **SMS/Text message** pour recevoir le code de vérification.

**Limite** : Bien que la 2FA par SMS renforce la sécurité, elle est plus vulnérable aux attaques par interception par rapport aux autres méthodes. Les membres utilisant cette méthode doivent envisager de la renforcer en passant à une application d'authentification (Authenticator app) ou à des clés de sécurité (Security keys), offrant un niveau de sécurité supérieur.



### 3. Gestion des permissions et règles de protection des branches

#### Protéger les branches principales

L'application de règles de protection avancées pour les branches principales (par exemple, main ou master) ne peut pas être mise en place actuellement, car elle nécessite de migrer le projet vers une **organisation GitHub de type Team**, qui est un service payant. Cela empêche l'activation de certaines fonctionnalités avancées de contrôle des modifications et des permissions. Cependant, il serait fortement recommandé de le faire afin d'améliorer la sécurité et la qualité du projet. Pour garantir l'intégrité et la qualité du code, il serait recommandé de mettre en place ces règles après la migration :

- **Require pull request reviews before merging** : Obliger chaque modification à passer par une Pull Request nécessitant l'approbation d'un membre de l'équipe permettrait de garantir la qualité des modifications et de bénéficier de revues par des pairs.
- **Require status checks to pass before merging** : L'exigence de tests automatiques validés avant toute fusion empêcherait l'intégration de code avec des erreurs, renforçant la stabilité de la branche principale.

- **Require signed commits** : Cela garantirait que seuls les commits signés par des utilisateurs autorisés soient acceptés, renforçant l'authenticité et la traçabilité des changements.
- **Restrict who can push to matching branches** : Interdire les push directs sur les branches critiques obligerait les contributions à passer par un processus de validation et de revue.

**Importance des règles de protection** : Bien qu'il ne soit pas possible de les activer actuellement, leur mise en place renforcerait la sécurité, garantirait que seules les modifications validées soient fusionnées, et favoriserait une collaboration rigoureuse entre les membres de l'équipe.

### **Gestion des permissions des collaborateurs**

- La gestion granulaire des permissions pour chaque collaborateur ne peut pas être réalisée dans le cadre actuel sans GitHub Team. Cela permettrait, par exemple, de définir des rôles spécifiques pour chaque membre, limitant les actions critiques et renforçant la sécurité du dépôt. Activer cette gestion garantirait que seules les actions nécessaires soient autorisées aux membres, tout en préservant une collaboration efficace et sécurisée.

## **4. Gestion des secrets et des variables d'environnement sensibles**

### **Protection des secrets avec Infisical**

Les secrets sont protégés grâce à l'exclusion du fichier .env via .gitignore et à l'utilisation d'**Infisical** pour gérer les secrets de manière centralisée et sécurisée. Chaque machine dispose d'une identité dédiée pour accéder aux secrets via Infisical.

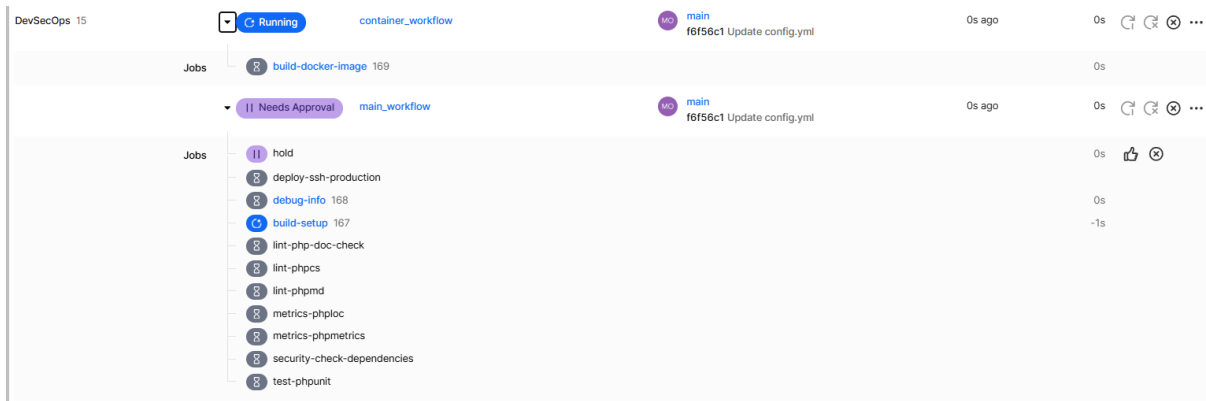
## **Pensez-vous que votre image Docker est fiable ? Comment le montrer ?**

La fiabilité de l'image Docker repose sur un ensemble de tests et validations qui sont réalisés par le main\_workflow pour la vérification du code et par le container\_workflow pour la construction de l'image. Actuellement, ces deux workflows s'exécutent en parallèle, ce qui soulève des préoccupations quant à la synchronisation et la validation complète avant la création de l'image.

### **1. Vérifications de qualité et de sécurité du code :**

Le main\_workflow inclut des jobs comme lint-phpcs, lint-phpmd, security-check-dependencies, et test-phpunit qui assurent la qualité du code, vérifient les

dépendances et exécutent des tests unitaires pour valider l'application. Cependant, l'exécution parallèle des workflows signifie que l'image Docker peut être construite avant que toutes les vérifications ne soient terminées ou validées.



## 2. Amélioration recommandée :

Il serait pertinent d'introduire une étape hold pour retarder la construction de l'image Docker jusqu'à ce que les tests du main\_workflow soient terminés et validés. Cela garantirait que l'image Docker n'est construite que si toutes les vérifications critiques ont réussi.

## 3. Preuves de sécurité :

Les scans de sécurité effectués via security-check-dependencies vérifie si les dépendances de l'application PHP contiennent des vulnérabilités connues. Cela garantit que l'image Docker ne contient pas de paquets vulnérables connus.

La gestion des secrets avec Infisical assure que les variables sensibles ne sont jamais intégrées directement dans l'image Docker, limitant ainsi les risques d'exposition de données sensibles.

En conclusion, bien que les vérifications actuelles fournissent une base solide, la mise en place d'un hold avant la construction de l'image Docker renforcerait sa fiabilité en s'assurant que l'image est générée uniquement à partir de code validé et testé.

## Êtes-vous sûr d'avoir bien configuré votre instance EC2 ? Comment le prouver ?

Pour garantir que l'instance EC2 est bien configurée, plusieurs vérifications et mesures ont été mises en place :

### Groupes de sécurité :

- **SSH** : Actuellement ouvert à 0.0.0.0, ce qui expose l'instance à des connexions de toute source. Pour renforcer la sécurité, il est essentiel de restreindre l'accès aux adresses IP spécifiques des membres de l'équipe et aux IP de CircleCI, disponibles ici : <https://circleci.com/docs/ip-ranges/>.
- **HTTP/HTTPS** : L'accès public aux applications web est autorisé. Cependant, le site est actuellement uniquement accessible en HTTP, ce qui représente une faille de sécurité. Il est recommandé de passer le site en HTTPS avec un certificat SSL valide et de supprimer l'autorisation d'accès HTTP pour garantir la sécurité des connexions.

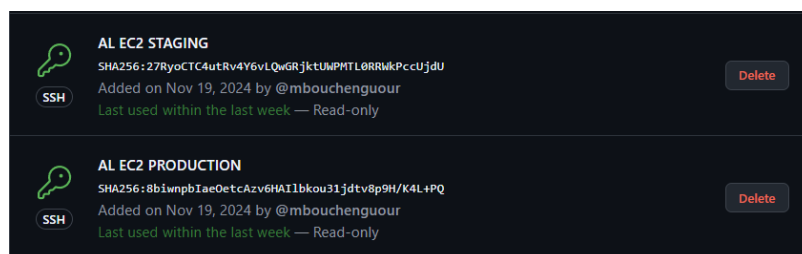
### Authentification SSH sécurisée :

- Une paire de clés SSH spécifique a été générée pour accéder à l'instance, évitant ainsi l'utilisation de mots de passe. Cela inclut également des clés dédiées permettant à CircleCI de se connecter de manière sécurisée à l'instance EC2 lors des déploiements automatisés.

```
ec2-user@ip-172-31-26-43 ~]$ cat .ssh/authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCKg1DCHXuwaXexj7AYGqqtRoLTw8HxTPlMNXUQB3xhisi7NVUawuQj8HOWzJoyzAHf7Tfe5avzXI159wvcM3Setlea4ju8/wCu0y67nagxaTvpPOgn7x9j2aAyV00/FLIchAwkPnSAWk/V
p1HxoE5yRk82ncwbDqHgsnud5a6aM2bc+hJO/604bWOKT8CTQzWE5XkoYC6h1kU1PKS7o8Ri60I3McAVXxVD2XoU9MeOfjX+W+2zHmFYCk21okFrB0dGvWak49oTppCp08Cbbt92tt8tRVEolnr9n0vFeKry8+n2g6j5WKhISBj6iXBPjvnz5J
we06ZfgaW8R8eBv CircleCI SSH Prod AL
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCKg1DCHXuwaXexj7AYGqqtRoLTw8HxTPlMNXUQB3xhisi7NVUawuQj8HOWzJoyzAHf7Tfe5avzXI159wvcM3Setlea4ju8/wCu0y67nagxaTvpPOgn7x9j2aAyV00/FLIchAwkPnSAWk/V
p1HxoE5yRk82ncwbDqHgsnud5a6aM2bc+hJO/604bWOKT8CTQzWE5XkoYC6h1kU1PKS7o8Ri60I3McAVXxVD2XoU9MeOfjX+W+2zHmFYCk21okFrB0dGvWak49oTppCp08Cbbt92tt8tRVEolnr9n0vFeKry8+n2g6j5WKhISBj6iXBPjvnz5J
we06ZfgaW8R8eBv
```

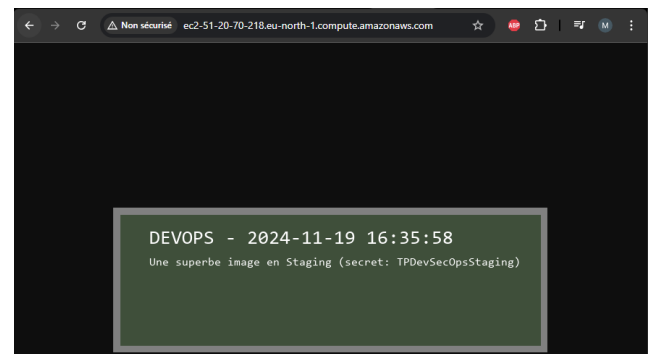
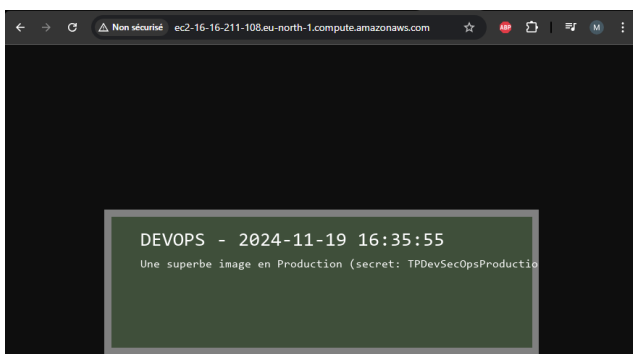
### Utilisation de la clé publique dans GitHub pour récupérer le projet :

Une clé publique SSH a été ajoutée dans les "Deploy keys" du dépôt GitHub, permettant à l'instance EC2 de cloner ou de récupérer les mises à jour du projet de manière sécurisée. Cette clé est configurée en mode lecture seule, garantissant que l'accès est restreint et sécurisé.



### Affichage correct du site et optimisation des performances avec Amazon Linux 2

Le site web déployé sur l'instance EC2 s'affiche correctement, démontrant ainsi la bonne configuration du serveur web et la compatibilité de l'application. Cela garantit que l'application répond aux requêtes comme prévu et que les fonctionnalités essentielles sont opérationnelles. De plus, l'instance EC2 utilise Amazon Linux 2, ce qui offre des performances optimisées et une sécurité accrue, grâce à des mises à jour régulières et des optimisations spécifiques pour l'environnement AWS. Cette combinaison assure un fonctionnement fluide, sécurisé et performant de l'application sur le long terme.



## Gestion des secrets et des variables d'environnement sensibles

### Points d'amélioration :

- **Accès restreint aux secrets** : La possibilité de limiter l'accès à Infisical par IP spécifique pour les machines autorisées n'a pas été mise en place, car cette fonctionnalité est payante. Cela aurait permis de restreindre l'accès aux secrets aux seuls systèmes approuvés, renforçant ainsi la sécurité.
- **Audits réguliers** : Réaliser des audits réguliers des accès aux secrets pourrait permettre de détecter toute activité inhabituelle ou tentative d'accès non autorisée.
- **Rotation des secrets** : Introduire une rotation régulière des secrets afin de limiter les risques liés à une éventuelle compromission.

## Surveillance et audits de sécurité des connexions

- **Logs de connexion et audit des accès** : Mettre en place une surveillance centralisée des logs d'accès SSH pour détecter toute activité suspecte ou tentative de connexion non autorisée. Cela peut inclure l'utilisation d'AWS CloudWatch Logs pour stocker et analyser les événements.

- **Systèmes d'alerte en temps réel :** Configurer des alertes pour les connexions SSH non autorisées ou pour tout comportement anormal détecté sur l'instance EC2, en utilisant des services comme AWS GuardDuty ou des outils de surveillance tiers.

### Gestion des mises à jour de sécurité :

- **Automatisation des mises à jour de sécurité :** Activer les mises à jour automatiques de sécurité pour Amazon Linux 2 afin de garantir que tous les paquets critiques et vulnérabilités potentielles soient corrigés sans intervention manuelle.
- **Analyse des configurations :** Utiliser AWS Inspector ou des outils similaires pour scanner régulièrement l'instance à la recherche de configurations non sécurisées, de vulnérabilités connues ou de paramètres exposant l'instance à des risques.