TP03 DevSecOps

DOCUMENTATION

Mohamed BOUCHENGUOUR
Hamadi DAGHAR
Bilal MESAOURI DEBOUN
Lamya BAIDOURI
POLYTECH UNICE

POLYTECH
NICE SOPHIA

Table des matières

Introduction

Ce document vise à fournir une explication détaillée de chaque étape de la configuration du pipeline CI/CD définie dans le fichier config.yml de CircleCI. Le pipeline est conçu pour automatiser les processus de construction, de test, d'analyse de code, de génération de métriques, de construction d'images Docker et de déploiement sur les environnements de staging et de production.

Variables d'Environnement Clés

Outre les variables intégrées, plusieurs variables d'environnement personnalisées sont utilisées pour configurer et sécuriser le pipeline CI/CD. Ces variables sont définies dans les paramètres du projet CircleCI et ne doivent jamais être exposées dans le code source. À noter que les variables intégrées par CircleCI, telles que celles commençant par CIRCLE_, sont définies et sécurisées par défaut dans l'environnement du pipeline.

Variables Personnalisées Principales

GHCR_USERNAME
Description : Nom d'utilisateur associé au dépôt GitHub Container Registry (GHCR).
Obtention : Disponible dans le compte GitHub sous les paramètres du registre de conteneurs.
Utilisation : Utilisée pour authentifier la connexion au GHCR lors de la construction et du push des images Docker.

GHCR_PAT
Description : Token d'accès personnel GitHub (Personal Access Token) pour GHCR.
Obtention :

- Se connecter à GitHub.
- Accéder à Paramètres > Paramètres du développeur > Tokens personnels (classic).
- Cliquer sur Générer un nouveau token et sélectionner les permissions *read* et *write* pour le registre de conteneurs.

Utilisation : Utilisé pour authentifier les opérations de push images Docker vers GHCR.

GITHUB_PROJECT_USERNAME
Description : Nom de l'utilisateur ou de l'organisation GitHub associé au projet.
Utilisation : Utilisée pour formater les noms de dépôt et les tags Docker.

GITHUB_PROJECT_REPONAME
Description : Nom du dépôt GitHub.
Utilisation : Utilisée dans la construction des noms d'images Docker.

STAGING_SSH_FINGERPRINT
Description : Empreinte digitale de la clé SSH pour l'environnement de staging.
Obtention : Générée lors de l'ajout de clé privée dans CircleCi pour staging.
Utilisation : Utilisée pour ajouter les clés SSH nécessaires lors du déploiement en staging.

STAGING_SSH_USER
Description : Nom d'utilisateur SSH pour accéder à l'instance de staging.
Valeur Typique : ubuntu

STAGING_SSH_HOST
Description : Adresse IP ou nom d'hôte de l'instance de staging.

STAGING_DEPLOY_DIRECTORY
Description : Répertoire de déploiement sur le serveur de staging.
Valeur Typique : /var/www/myapp

PRODUCTION_SSH_FINGERPRINT
Description : Empreinte digitale de la clé SSH pour l'environnement de production.
Obtention : Générée lors de la création de la paire de clés SSH pour production.
Utilisation : Utilisée pour ajouter les clés SSH nécessaires lors du déploiement en production.

PRODUCTION_SSH_USER
Description : Nom d'utilisateur SSH pour accéder à l'instance de production.
Valeur Typique : ec2-user (pour Amazon Linux)

PRODUCTION_SSH_HOST
Description : Adresse IP ou nom d'hôte de l'instance de production.

PRODUCTION_DEPLOY_DIRECTORY
Description : Répertoire de déploiement sur le serveur de production.
Valeur Typique : /var/www/myapp

PROJECT_ID
Description : Le project ID contenant les secrets dans infisical.
Utilisation : Utilisés dans le script fetchSecrets pour accéder aux secrets dans infisical.

STAGING_INFISICAL_MACHINE_CLIENT_ID,
STAGING_INFISICAL_MACHINE_CLIENT_SECRET
Description : Identifiants pour l'intégration avec Infisical, utilisés pour la gestion des secrets en staging.
Utilisation : Utilisés dans les scripts pour authentifier et récupérer les secrets nécessaires à l'application.

PRODUCTION_INFISICAL_MACHINE_CLIENT_ID,
PRODUCTION_INFISICAL_MACHINE_CLIENT_SECRET
Description : Identifiants pour l'intégration avec Infisical, utilisés pour la gestion des secrets en production.

Utilisation : Utilisés dans les scripts pour authentifier et récupérer les secrets nécessaires à l'application.

Variables Intégrées de CircleCI

Les variables suivantes sont intégrées par CircleCI et sécurisées par défaut :

CIRCLE_BRANCH
Description : Branche Git actuelle sur laquelle le pipeline est exécuté.
Utilisation : Utilisée pour déterminer le tag de l'image Docker et pour conditionner les déploiements.

Configuration par défaut pour persist_to_workspace et attach_workspace

Dans les workflows, l'utilisation d'ancres YAML permet de définir des configurations réutilisables pour les commandes persist_to_workspace et attach_workspace, réduisant ainsi la redondance et assurant la cohérence de leur application.

Configuration de persist_to_workspace

Cette configuration permet de persister le répertoire de travail actuel dans l'espace de travail global. Cela facilite le partage de fichiers entre différents jobs du même workflow, en veillant à ce que les résultats produits par un job soient disponibles pour d'autres jobs ultérieurs.

Configuration de attach_workspace

Cette configuration permet d'attacher l'espace de travail précédemment sauvegardé à un chemin spécifique, ici défini comme ~/project, dans le job actuel. Cela permet d'accéder aux fichiers partagés d'un job à un autre et de garantir la continuité des données nécessaires au bon déroulement des étapes suivantes du workflow.

Définition des exécuteurs

Les exécuteurs définissent l'environnement d'exécution des jobs, notamment l'image Docker à utiliser, les ressources allouées, le shell par défaut, etc.

Exécuteur php-executor

resource_class : Indique la taille des ressources allouées (CPU, RAM).

shell : Définit le shell par défaut utilisé dans les étapes run.

docker : Spécifie l'image Docker à utiliser, ici une image PHP 8.2 officielle de CircleCI.

Exécuteur builder-executor

Cet exécuteur est similaire à php-executor mais inclut également Node.js, utile pour les jobs nécessitant Node.js en plus de PHP.

Exécuteur simple-executor

Un exécuteur de base utilisant une image Docker générique, adapté aux tâches ne nécessitant pas d'environnement spécifique.

Définition des jobs

Les jobs sont les unités de travail exécutées dans le pipeline. Chaque job est composé de plusieurs étapes (steps).

Job debug-info

Description

Affiche des informations de débogage sur l'environnement d'exécution

Étape run : Exécute une série de commandes pour afficher les variables d'environnement et des informations système.

Job build-setup

Description

Prépare l'environnement en installant les dépendances via Composer et en mettant en cache le dossier vendor.

Détails des étapes

- **Étape checkout** : Clone le dépôt Git dans l'environnement.
- **Étape restore_cache** : Restaure le cache des dépendances si disponible.

- **Étape run** : Installe les dépendances Composer.
- **Étape save_cache** : Sauvegarde le dossier vendor dans le cache pour les builds futurs.
- **Étape persist_to_workspace** : Persiste le répertoire actuel dans l'espace de travail.

Job lint-phpcs

Description

Exécute PHP_CodeSniffer pour vérifier le respect des standards de codage PHP.

Détails des étapes

- **Étape attach_workspace** : Attache l'espace de travail précédemment persisté.
- **Étape run** : Installe PHP_CodeSniffer et l'extension PHPCompatibility.
- **Étape run** : Exécute PHP_CodeSniffer avec les options spécifiées, gère les codes de sortie pour éviter l'échec du job en cas de warnings.
- **Étape store_artifacts** : Sauvegarde le rapport généré comme artefact.

Job security-check-dependencies

Description

Vérifie les dépendances PHP pour détecter les vulnérabilités connues.

Détails des étapes

- **Étape run** : Télécharge et installe l'outil local-php-security-checker.
- **Étape run** : Exécute l'analyse de sécurité sur les dépendances.
- **Étape store_artifacts** : Sauvegarde le rapport de sécurité.

Job test-phpunit

Description

Exécute les tests unitaires avec PHPUnit si le fichier de configuration phpunit.xml est présent.

Détails des étapes

- **Étape run** : Vérifie la présence du fichier phpunit.xml et saute le job si absent.
- **Étape run** : Installe PHPUnit via Composer.
- **Étape run** : Exécute les tests unitaires.

Job metrics-phpmetrics

Description

Génère des métriques de code avec phpmetrics et produit un rapport HTML.

Détails des étapes

- **Étape run** : Installe phpmetrics.
- **Étape run** : Exécute phpmetrics sur le répertoire ./src.
- **Étape store_artifacts** : Sauvegarde le rapport HTML généré.

Job metrics-phploc

Description

Compte le nombre de lignes de code PHP et génère un rapport texte.

Détails des étapes

- **Étape run** : Télécharge le fichier exécutable phploc.phar.
- **Étape run** : Exécute phploc et génère un rapport texte.
- **Étape store_artifacts** : Sauvegarde le rapport.

Job lint-phpmd

Description

Utilise PHP Mess Detector pour détecter les problèmes potentiels dans le code.

Détails des étapes

- **Étape run** : Installe PHP Mess Detector.
- **Étape run** : Exécute phpmd avec les règles spécifiées et génère un rapport XML.
- **Étape store_artifacts** : Sauvegarde le rapport.

Job lint-php-doc-check

Description

Vérifie la documentation des méthodes et des classes PHP en utilisant php-doc-check.

Détails des étapes

- **Étape run** : Installe l'outil php-doc-check.
- **Étape run** : Exécute la vérification de la documentation et génère un rapport texte.
- **Étape store_artifacts** : Sauvegarde le rapport.

Job build-docker-image

Description

Construit une image Docker de l'application et la pousse vers GitHub Container Registry (GHCR).

Détails des étapes

- **Étape checkout** : Clone le dépôt Git.

- **Étape setup_remote_docker** : Initialise un environnement Docker distant avec la version spécifiée et active le caching des couches Docker.
- **Étape run** : Construit et pousse l'image Docker :
    - Vérifie si le build doit être sauté en vérifiant SKIP_BUILD.
    - Prépare le nom du dépôt et le tag en nettoyant les noms.
    - Se connecte à GHCR en utilisant les identifiants fournis.
    - Construit l'image Docker en utilisant le Dockerfile spécifié et en passant des arguments de build.
    - Pousse l'image vers GHCR avec le tag approprié.

Job deploy-ssh-staging

Description

Déploie l'application sur l'environnement de staging en utilisant SSH.

Détails des étapes

- **Étape add_ssh_keys** : Ajoute les clés SSH nécessaires pour la connexion.
- **Étape run** : Exécute le déploiement en se connectant au serveur de staging :
    - Active le mode verbeux pour le débogage.
    - Se connecte via SSH en désactivant la vérification stricte des hôtes.
    - Exécute le script fetchSecrets avec les identifiants de la machine en staging.
    - Exécute des commandes sur le serveur distant pour mettre à jour le code (git pull) et redémarrer PHP-FPM.

Job deploy-ssh-production

Description

Déploie l'application sur l'environnement de production en utilisant SSH.

Détails des étapes

- **Étape add_ssh_keys** : Ajoute les clés SSH nécessaires pour la connexion.
- **Étape run** : Exécute le déploiement en se connectant au serveur de production :
    - Active le mode verbeux pour le débogage.
    - Se connecte via SSH en désactivant la vérification stricte des hôtes.
    - Exécute le script fetchSecrets avec les identifiants de la machine en production.
    - Exécute des commandes sur le serveur distant pour mettre à jour le code (git pull) et redémarrer PHP-FPM.

Définition des workflows

Les workflows orchestrent l'exécution des jobs en définissant l'ordre et les dépendances.

Workflow main_workflow

Détails

- Les jobs lint-phpcs, security-check-dependencies, test-phpunit, metrics-phpmetrics, metrics-phploc, lint-phpmd et lint-php-doc-check dépendent tous de build-setup.
- Le job hold est une étape d'approbation manuelle, nécessaire avant les déploiements.
- Le déploiement en production (deploy-ssh-production) n'est exécuté que sur les branches main et master.
- Le déploiement en staging (deploy-ssh-staging) n'est exécuté que sur les branches correspondant au motif release/*.

Workflow container_workflow

Détails

- Le job build-docker-image est exécuté sur un ensemble spécifique de branches, y compris les branches de développement et de fonctionnalité.
- Cela permet de construire et de pousser des images Docker pour les environnements de test et de développement.