Documentation pipeline CI/CD

Membres du groupe:

- REMY Clément
- DUMANOIS Arnaud
- ZOU Weishen
- BEUREL Simon

Introduction:

Le but de ce document est de présenter une documentation complète de chaque “job” et de chaque “workflow” présents dans le fichier de configuration .circleci/config.yml. Ce document sera constitué de 5 parties:

1. Présentation du système de pipeline CircleCI
2. Présentation des executors
3. Présentation des jobs
4. Présentation des workflows
5. Conclusion

L’objectif de ce rapport est simplement de documenter, il est complémentaire du document “Rapport_detaille_pipeline_CICD” présentant en détail les choix décidés durant ce projet, leur justification et leurs intérêts.

Partie I: Système CircleCI

Le système CircleCI permet de mettre en place rapidement un pipeline CI/CD grâce à un simple fichier de configuration présent dans un dossier nommé “.circleci/”. A l’intérieur de ce fichier, nous pouvons trouver plusieurs éléments:

- **Workspace** : Il permet de partager des fichiers ou des artefacts entre différents jobs d’un même workflow, facilitant ainsi la continuité des tâches (par exemple, transférer les artefacts de compilation vers un job de test ou de déploiement).
- **Executors** : Ils définissent l'environnement dans lequel les tâches s'exécuteront, comme une image Docker, une machine virtuelle ou un environnement distant. Les executors spécifient également les ressources nécessaires, comme les images Docker ou les configurations matérielles.
- **Jobs** : Ce sont les unités principales de travail dans CircleCI. Un job contient une série d'étapes ou de commandes à exécuter, comme construire un projet, exécuter des tests, ou déployer une application. Chaque job s'exécute dans un executor.
- **Workflows** : Ils orchestrent l'ordre d'exécution des jobs et permettent d'établir des dépendances entre eux. Grâce aux workflows, vous pouvez paralléliser des tâches ou les

exécuter conditionnellement selon le résultat d'autres jobs.

Par exemple, si nous prenons en situation suivante “Un restaurant italien” nous pourrions avoir:

- Workspace: La cuisine du restaurant
- Executors: Un cuisinier
- Jobs: “Préparer la pâte”, “Disposer les fromages sur la pizza”, “Commander de la sauce tomate”, etc.
- Workflow: “Préparer une pizza 4 fromages”

Partie II: Présentation des executors

Au sein de notre pipeline, nous possédons 3 executors, voici leurs caractéristiques:

Nom	Description	Image docker utilisée	Version PHP/Node	Shell utilisé	Classe de ressources
php-executor	Principal executor pour tous les jobs relatifs à PHP, sauf indication contraire.	cimg/php:8.2	PHP 8.2	`/bin/bash`	small
builder-executor	Utilisé pour le job relatif à la construction de l’image docker et à son publiement	cimg/php:8.1-node	PHP 8.1 / Node.js	`/bin/bash`	small
simple-executor	Executor basique pour des jobs nécessitant uniquement une image légère, comme des scripts génériques ou tests rapides.	cimg/base:stable	N/A	`/bin/bash`	small

Partie III: Présentation des jobs

A l’intérieur de notre pipeline, nous possédons un total de 14 jobs différents qui vont chacun avoir leur propre tâche spécifique à réaliser. Voici leurs caractéristiques:

Job Name	*Executor*	*Description*	*Principales étapes*
debug-info	php-executor	Collecte des informations système pour le debug.	Exécute des commandes shell pour afficher l'utilisateur, le chemin, l'OS, et les variables d'environnement.

quality_check	php-executor	Vérifie la qualité du code avec PHPMD et PHP Doc Check, et génère des rapports.	Installe PHPMD et PHP Doc Check, lance les analyses, et sauvegarde les rapports.
build-setup	php-executor	Prépare l'environnement et installe les dépendances avec Composer.	Restaure le cache, installe les dépendances, et enregistre le cache mis à jour.
lint-phpcs	php-executor	Analyse le code avec PHPCS en suivant un ensemble de règles personnalisées.	Installe PHP_CodeSniffer, lance une analyse avec un ruleset spécifique, et sauvegarde les rapports.
security-check-dependencies	php-executor	Analyse les dépendances PHP pour détecter des vulnérabilités avec `local-php-security-checker`.	Télécharge et exécute `local-php-security-checker`, puis sauvegarde le rapport au format JSON.
test-phpunit	php-executor	Exécute les tests PHPUnit de l'application (tests unitaires).	Vérifie la présence de tests PHPUnit, installe PHPUnit, et exécute les tests unitaires.
test-phpunit-feature	php-executor	Exécute les tests PHPUnit de l'application (tests de fonctionnalités).	Vérifie la présence de tests PHPUnit, installe PHPUnit, et exécute les tests de fonctionnalités.
test-end-to-end	php-executor	Exécute les scripts de tests de bout en bout (E2E) s'ils existent dans le dossier spécifié.	Vérifie la présence du dossier `tests/E2E`, exécute les scripts de tests E2E trouvés.
build-docker-image	builder-executor	Construit et pousse une image Docker vers le GitHub Container Registry (GHCR).	Configure Docker, construit l'image avec des métadonnées, et pousse l'image vers GHCR.
deploy-ssh-staging	simple-executor	Déploie l'application sur un serveur de staging via SSH.	Accède au serveur staging, télécharge la dernière image, arrête l'ancienne instance, et relance le container.
deploy-ssh-production	simple-executor	Déploie l'application sur un serveur de production via SSH.	Accède au serveur production, télécharge la dernière image, arrête l'ancienne instance, et relance le container.
phpmetrics	php-executor	Analyse le code avec PHP Metrics et génère des rapports HTML, XML, et JSON.	Installe PHP Metrics, exécute l'analyse, et sauvegarde les rapports dans différents formats.
phploc	php-executor	Analyse le code avec PHPLOC pour collecter des métriques sur le projet.	Télécharge et exécute PHPLOC, génère un rapport, et sauvegarde les résultats.
security-docker-image	simple-executor	Analyse la sécurité de l'image Docker avec Trivy et génère un rapport.	Installe Trivy, construit une image Docker de test, exécute l'analyse avec Trivy, et sauvegarde le rapport.

configure-infisical	simple-executor	Se connecte et récupère les secret de Infisical et les stock dans un dossier environnement	Se connecte pour obtenir un token, export les secrets dans un format .dotenv et les stock dans un fichier environnement pour chaque environnement

Partie IV: Présentation des workflows

A l'intérieur de notre pipeline, nous possédons un total de 4 workflows différents qui vont chacun avoir leur propre tâche spécifique à réaliser. Voici leurs caractéristiques:

Workflow Name	Description	Jobs
metrics_workflow	Analyse les métriques du code source pour évaluer la qualité du projet et sauvegarde des sorties dans différents rapports.	- `debug-info` - `build-setup` - `quality_check` - `phpmetrics` - `lint-phpcs` - `phploc`
security_workflow	Évalue la sécurité des dépendances et de l'image Docker crée pour détecter des vulnérabilités.	- `debug-info` - `build-setup` - `security-check-dependencies` - `security-docker-image`
tests_workflow	Exécute différents types de tests (unitaires, intégration, end2end) pour garantir la stabilité et la fonctionnalité du code.	- `debug-info` - `build-setup` - `test-phpunit` - `test-phpunit-feature` - `test-end-to-end`
deployment_workflow	Automatisation du processus de déploiement de l'image Docker vers les environnements de staging et production.	- `build-setup` - `build-docker-image` - `hold-for-deployment` - `deploy-ssh-production` - `deploy-ssh-staging`

Partie V: Conclusion

Ce document sert de documentation basique sur les différents éléments présents au sein de notre pipeline CI/CD CircleCI. Nous vous invitons également à lire le deuxième rapport, détaillant l'ensemble des processus, les différentes stratégies choisies et les différentes extensions apportées au pipeline original.