**Documentation du Déploiement avec GitLab CI/CD**

**1. Introduction**

Cette documentation décrit l'utilisation de GitLab CI/CD pour l'automatisation du déploiement d'une application web multi-régionale sur AWS. Elle explique les différentes étapes du pipeline définies dans le fichier gitlab.yaml, leur rôle, ainsi que les bonnes pratiques pour un déploiement sécurisé et efficace.

**2. Structure du Pipeline GitLab CI/CD**

Le pipeline est organisé en plusieurs stages :

1. **Validate** : Vérification du code et de la configuration.
2. **Test** : Exécution des tests unitaires et de sécurité.
3. **Build** : Préparation des dépendances et des fichiers nécessaires au déploiement.
4. **Deploy** : Déploiement de l'application sur différentes instances AWS.

**3. Explication des Étapes**

**3.1 Validation du Code**

**3.1.1 Vérification de la Syntaxe PHP**

lint:

stage: validate

image: php:8.2-cli

script:

- echo "Checking PHP syntax..."

- find . -name "\*.php" -not -path "./vendor/\*" -print0 | xargs -0 -n1 php -l

* Vérifie la syntaxe de tous les fichiers PHP avant l'exécution.

**3.1.2 Validation de la Configuration Apache**

apache\_config:

stage: validate

image: httpd:2.4

script:

- echo "Checking Apache configuration..."

- apache2ctl -t

* Vérifie la validité des configurations Apache pour éviter des erreurs au déploiement.

**3.2 Tests**

**3.2.1 Tests Unitaires avec PHPUnit**

phpunit:

stage: test

image: php:8.2-cli

script:

- echo "Running PHPUnit tests..."

- vendor/bin/phpunit --testdox

* Lance les tests unitaires pour s'assurer du bon fonctionnement du code.

**3.2.2 Vérification de Sécurité**

security\_check:

stage: test

image: php:8.2-cli

script:

- echo "Running security checks..."

- symfony security:check

* Vérifie les vulnérabilités connues des dépendances PHP.

**3.3 Compilation et Préparation**

**3.3.1 Installation des Dépendances**

build:

stage: build

image: composer:latest

script:

- echo "Installing dependencies..."

- composer install --no-dev --optimize-autoloader

artifacts:

paths:

- vendor/

- composer.lock

expire\_in: 1 day

* Télécharge et installe les dépendances nécessaires à l'application.

**3.4 Déploiement**

Un template de déploiement est utilisé pour faciliter le déploiement sur plusieurs régions AWS.

**3.4.1 Déploiement en Europe**

deploy\_eu:

stage: deploy

variables:

INSTANCE\_IP: ${EU\_INSTANCE\_IP}

script:

- echo "Deploying to Europe..."

- rsync -avz ./ ubuntu@${INSTANCE\_IP}:/var/www/html/

- ssh ubuntu@${INSTANCE\_IP} "sudo systemctl restart apache2"

* Synchronise les fichiers sur le serveur en Europe et redémarre Apache.

**3.4.2 Déploiement aux USA et en Asie**

* Similaire au déploiement en Europe, mais utilise les variables des instances US et AP.

**3.5 Nettoyage Post-Déploiement**

cleanup:

stage: .post

script:

- echo "Cleaning up temporary files..."

- rm -rf vendor/

- rm -rf .composer-cache/

when: always

* Nettoie les fichiers temporaires après le déploiement pour libérer de l'espace.

**4. Bonnes Pratiques**

* **Utilisation de variables sécurisées** : Stocker les credentials dans les variables de GitLab CI/CD plutôt que dans le code.
* **Tests avant déploiement** : Toujours valider la syntaxe et exécuter des tests avant de déployer.
* **Déploiement manuel** : Pour éviter un déploiement non maîtrisé, certaines étapes sont configurées en manual.

**5. Conclusion**

Ce pipeline GitLab CI/CD permet d'automatiser le déploiement et d'assurer un contrôle qualité avant la mise en production. En suivant ces étapes, on garantit une infrastructure robuste et sécurisée.