Rapport détaillé pipeline CI/CD:

Membres du groupe:

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

Introduction:

Le but de ce document est de détailler les différents processus intégrés au pipeline CI/CD, notamment en expliquant la mise en place du déploiement automatique sur une instance EC2 d'AWS, une explication détaillée des workflows, d'Infisical et une explication de la stratégie adoptée pour le branching du projet.

Réponses aux questions:

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

Pour pouvoir sécuriser au maximum notre dépôt Github nous avons décidé de faire des choix spécifiques pour assurer une sécurité sur le projet.

1. Premièrement, le dépôt est en visibilité "Privé". Cette visibilité permet de seulement autoriser certaines personnes à avoir accès au dépôt pour lire les fichiers que contient ce dernier et effectuer différents commits. Les différents contributeurs autorisés sont les 4 membres du projet.
2. Deuxièmement, les différents tokens générés par Github et utilisés dans ce projet comme pour le push d'image sur GHCR par exemple sont des tokens qui possèdent le strict minimum d'autorisations. Ainsi, si malheureusement une personne vole le token, ce ne sera pas grave car elle sera très limitée en terme d'actions.
3. Troisièmement, le dépôt GitHub est lié à l'utilisateur "simonbeurel". Cet utilisateur possède un mot de passe fort composé de plusieurs lettres, chiffres et caractères spéciaux ce qui évite de subir une attaque par brute

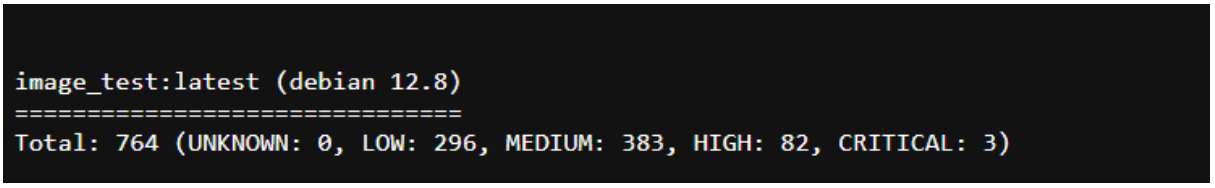
force sur le mot de passe et ainsi renforce la sécurité du compte et indirectement du dépôt.

Pensez-vous que votre image docker est fiable ? Comment le montrer ?

Pour pouvoir tester notre image docker, nous avons trouvé un outil très pratique qui s'appelle Trivy:

> https://github.com/aquasecurity/trivy

Grâce à cet outil, nous pouvons réaliser différents tests de sécurité sur l'image Docker générée pour pouvoir suivre si de potentielles menaces sont détectées. Nous avons également intégré cet outil dans un job spécifique de la pipeline CI/CD pour effectuer un test de sécurité et en garder la trace grâce à un artefact pour chaque push sur les branches develop/staging/main. Malheureusement, nous avons pu observer que notre image docker possède de nombreuses failles de sécurité, il serait judicieux dans un futur de comprendre pourquoi cette image possède autant de failles:



De plus, nous avons remarqué que l'un des problèmes lié à l'outil Trivy est que nous pouvons rapidement devenir temporairement bannis de l'outil car nous effectuons trop de requêtes ce qui est contraignant, c'est notamment pour cela que l'analyse de sécurité de l'image Docker n'est réalisée qu'à partir de la branche Develop (voir partie **"Stratégie de branching"**)

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

Pour la configuration de l'instance EC2, nous avons plusieurs points à vérifier:

1. Regarder si la gestion des alertes de paiement est bien mise en place (pour éviter de se retrouver avec une facture de XXXX€ à la fin du mois)

2. Regarder si la gestion du pare-feu est bien mise en place pour n'ouvrir que les ports nécessaires à notre application
3. S'assurer que la clé SSH pour se connecter sur le serveur ne soit pas être présente dans la nature

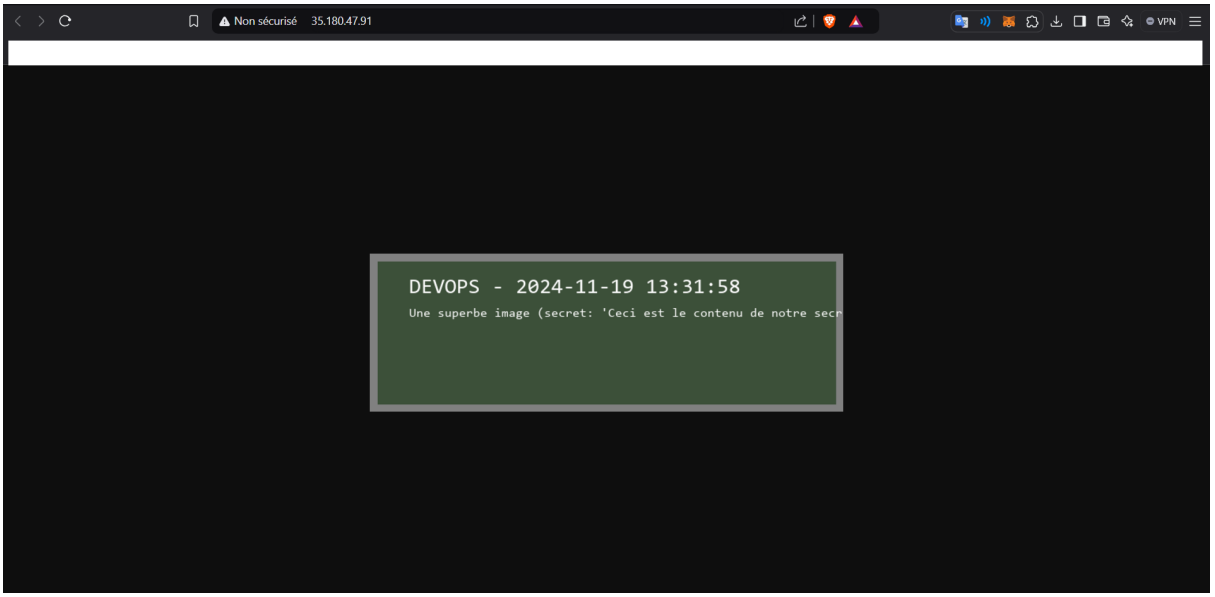
Pour les deux premiers points, nous avons utilisé une documentation fournie et utilisée lors du projet PS8 en 4ème année à Polytech Nice Sophia. En effet, lors de ce projet de fin d'année, nous devions réaliser un site web et le déployer en utilisant une instance EC2 d'AWS. Lors de ce projet, notre encadrant Mr. Benjamin Vella nous avait fourni une documentation complète sur les bonnes pratiques à utiliser pour configurer une instance EC2. Nous le remercions pour cette documentation qui aura été très utile pour notre projet PS8 de l'année dernière, et pour ce projet également.

Concernant le point numéro 3, la clé SSH permettant de se connecter au serveur est présente seulement sur l'ordinateur d'un des membres du groupe, ainsi cela réduit la possibilité de perte/vol de la clé par une personne étrangère. De plus, nous avons ajouté une variable d'environnement dans CircleCi qui contient l'adresse et le nom d'utilisateur pour se connecter au serveur.

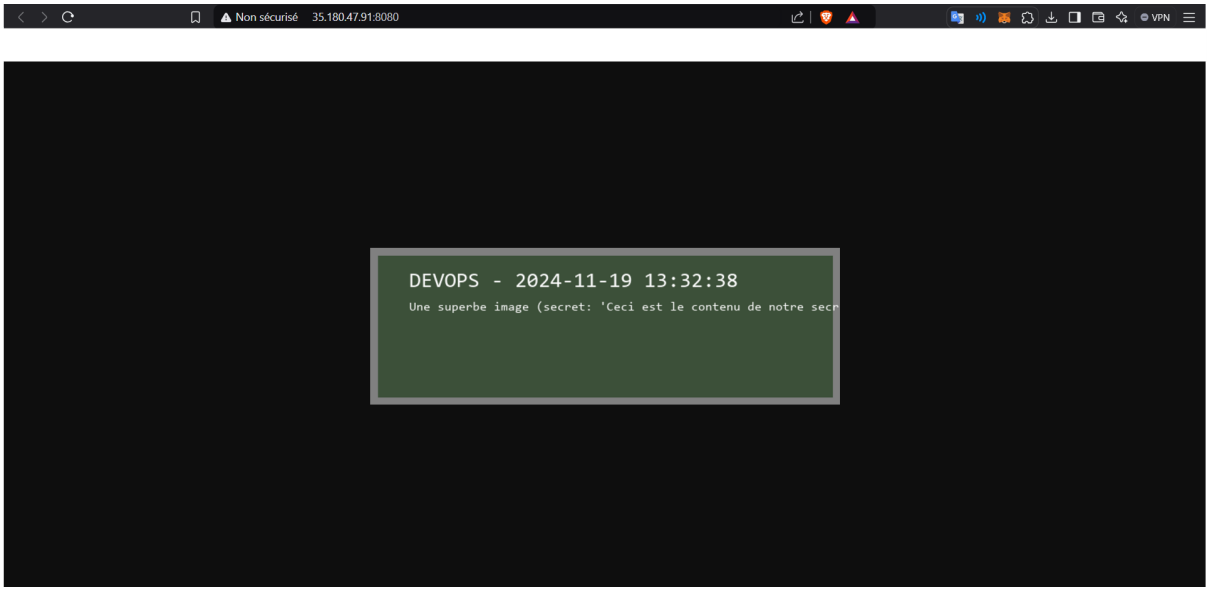
Mise en place du serveur AWS:

Pour réaliser un déploiement continu de notre application, nous avons utilisé le service Amazon Web Service et sa fonctionnalité EC2 pour déployer un serveur privé. Lors de la création de ce serveur, nous avons spécifié que nous voulions un serveur basé sur Ubuntu. Ce serveur possède également des règles de pare-feu assez strictes pour permettre seulement des connexions sur les ports TCP 80 (image de production) et 8080 (image de staging). L'IP du serveur n'est reliée à aucun nom de domaine, mais si vous voulez vérifier que les images Docker du projet sont bien en exécution vous pouvez vous rendre sur ces liens:

- http://35.180.47.91/ (lien de l'image de production)



- http://35.180.47.91:8080/ (lien de l'image de staging)



Tout ceci est géré grâce à Docker comme nous pouvons le voir quand nous listons les différents images en cours d'exécution sur notre serveur:

```
root@ip-172-31-40-13:/home/ubuntu# docker ps
CONTAINER ID   IMAGE                                          COMMAND                  CREATED       STATUS       PORTS                                       NAMES
fdd30ae0c46b   ghcr.io/simonbeurel/tddevsecoppspns:staging   "docker-php-entrypoi…"   2 hours ago   Up 2 hours   0.0.0.0:8080->80/tcp, [::]:8080->80/tcp     tddevsecoppspns
-staging
2186474d1925   ghcr.io/simonbeurel/tddevsecoppspns:main      "docker-php-entrypoi…"   2 hours ago   Up 2 hours   0.0.0.0:80->80/tcp, :::80->80/tcp           tddevsecoppspns
-prod
root@ip-172-31-40-13:/home/ubuntu#
```

Explication des workflows:

Pour que cette pipeline fonctionne correctement et de manière utile, nous avons décidé d'utiliser 4 workflows qui possèdent chacun un comportement et

un objectif bien précis.

- metrics_workflow: Le but principal de ce workflow est de réaliser les différents jobs liés à la production de différents artefacts pour faire l'état de l'art du code de notre projet. Ce workflow va notamment appeler les jobs quality_check, phpmetrics, lint-phpcs et phploc. Ce workflow va permettre aux différents membres de l'équipe d'avoir une trace écrite des différentes analyses produites.
- security_workflow: Le but de ce workflow est de réaliser les différents tests de sécurité liés à l'application PHP ou à l'image Docker produite avant chaque déploiement. Pour ce faire, nous faisons appel au job security-check-dependencies (qui va permettre d'analyser les différentes dépendances liées au projet) et security-docker-image qui va permettre de pouvoir réaliser une analyse de l'image Docker grâce à Trivy (mentionné précédemment dans le rapport)
- tests_workflow: Le but de ce workflow est d'exécuter les différents tests permettant de garantir que notre application est fonctionnelle et sans bug. Nous avons décidé de créer 3 types de tests différents
    - Test unitaires: Permettent de vérifier le comportement d'une fonction ou d'une classe spécifique seulement. Ils sont présents dans le dossier tests/Unit
    - Tests de feature: Permettent de vérifier une fonctionnalité complète du système, pouvant appeler plusieurs fonctions/classes. Ils sont présents dans le dossier tests/Feature
    - Test end2end: Ce sont des tests très larges qui permettent, d'un point de vue utilisateur, de garantir la fiabilité du système. Ils sont présents dans le dossier tests/E2E
- deployment_workflow: Le but de ce workflow e st de créer l'image Docker correspondant à la branche dans laquelle ce dernier est exécuté, et de déployer cette image Docker en production ou en staging. Grâce à ce workflow nous pouvons assurer un déploiement automatique sur le serveur EC2 AWS

Explication d'Infisical:

Nous utilisons Infisical pour gérer les secrets de notre application, son avantage principal est la gestion des rotations de secrets, c'est à dire que si une clé est compromise ou alors que nous souhaitons changer la valeur d'un secret, Infisical le permet facilement. Dans ce dernier nous y renseignerons des valeurs sensible telles que des clés d'API, identifiants de base de données (adresse serveur etc.) ou autres valeurs nécessaires pour le fonctionnement de notre application. Pour les ajouter à notre application nous exportons les secrets de Infisical dans un format .dotenv afin de générer des environnements utilisables dans l'application.

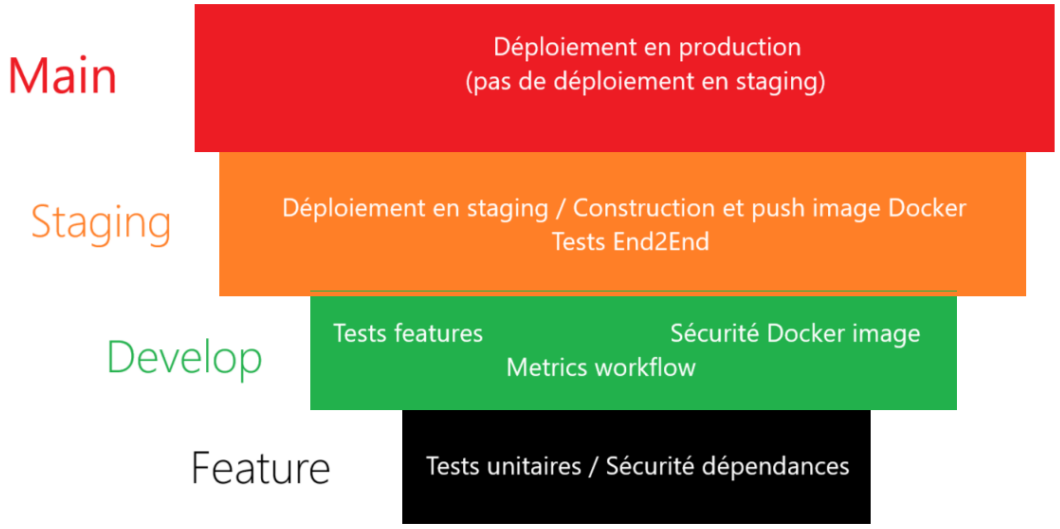
Nous utilisons en complément de ceci les variables d'environnement de CircleCI. A la différence que dans circleCI les variables d'environnement que nous stockons seront plutôt des clés pour des outils CI\CD.

Malheureusement Infisical possède un léger problème, la version européenne de l'outil provoque parfois quelques problèmes de connectivité, parfois ne chargeant pas les secrets ou ne permettant pas de se connecter tout simplement au service. Néanmoins une solution direct pourrait être de faire en sorte de self host Infisical sur un serveur et ainsi ne pas dépendre des serveurs de l'outil.

Stratégie de branching:

Au cours de la construction des différents workflows nous avons dû réfléchir sur la stratégie de branching que nous voulions implémenter pour notre projet. En effet, nous devons nous mettre dans la peau d'une entreprise qui possède des ressources limitées sur la pipeline CI/CD car dans notre cas nous sommes seulement 4 membres dans ce projet, mais dans de grandes entreprises les équipes peuvent être 100x plus grandes et le nombre de commits peut rapidement dépasser la centaine par jour. Ainsi il faut optimiser la pipeline CI/CD pour que cette dernière ne consomme pas de temps de calcul/de ressources inutiles.

Nous avons décidé de partir sur une stratégie par couches, c'est-à-dire qu'au fur et à mesure que la branche devient de plus en plus importante, celle-ci accumulera de plus en plus de jobs liés aux workflows à exécuter. Voici un schéma présentant notre stratégie:



Tout d'abord, nous avons tout en bas de cette pyramide inversée la branche Feature. Cette branche représente de manière générale toutes les branches crées par des développeurs pour coder de nouvelles features sur l'application. Elle est peu importante et subira beaucoup de pushs, ainsi il faut réaliser le minimum de jobs sur cette dernière, c'est pour cela que nous avons décidé de réaliser seulement:

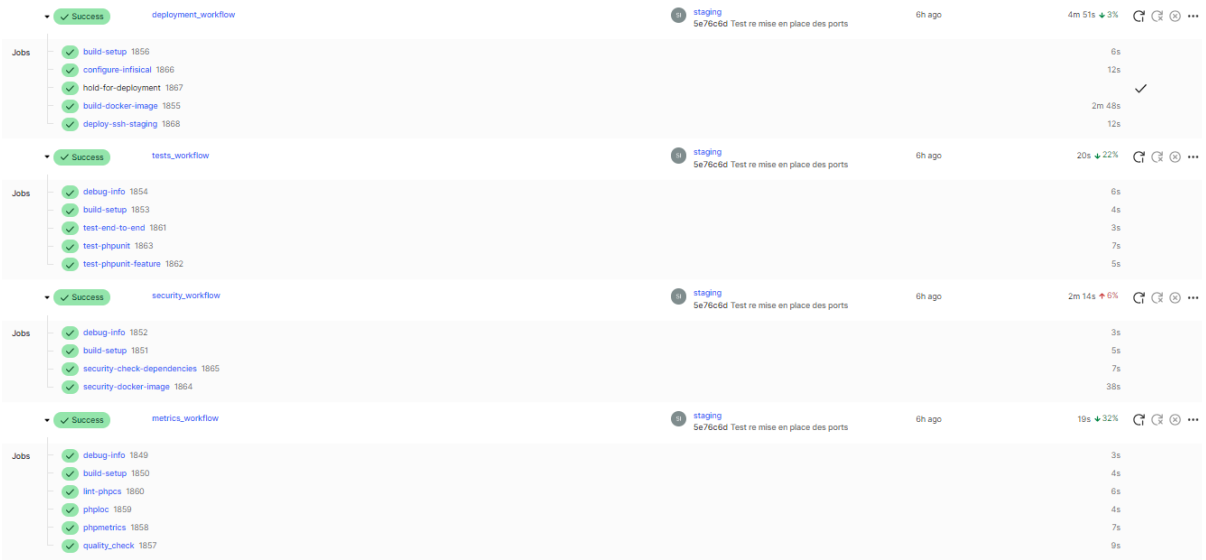
- Les tests unitaires
- L'analyse de sécurité des dépendances.

Deuxièmement nous avons la branche Develop. Cette branche est plus importante que la précédente car elle va permettre de mettre en commun les différentes features implémentées par les développeurs. Ainsi, les jobs réalisés sur cette dernière seront:

- Les tests unitaires
- L'analyse de sécurité des dépendances
- Les tests features
- L'analyse de sécurité de l'image Docker
- Le workflow lié aux métriques

Troisièmement nous pouvons voir la branche Staging. Cette branche est extrêmement importante car c'est la dernière étape avant la fameuse "mise en production", il est donc important de s'assurer que cette dernière ne possède aucun problèmes, notamment liés au déploiement. Ainsi, sur cette branche nous allons réaliser les différents jobs suivants:

- Les tests unitaires
- L’analyse de sécurité des dépendances
- Les tests features
- L’analyse de sécurité de l’image Docker
- Le workflow lié aux métriques
- Construction et push de l’image Docker sur GHCR
- Déploiement en staging (nécessite une validation humaine sur CircleCI)
- Les tests E2E



(Exemple de sortie que l’on peut voir sur l’interface CircleCI suite à un push réalisé sur la branche Staging)

Quatrièmement, nous avons la fameuse branche Main. Cette branche est la principale du projet, il est donc très important de s’assurer que tout fonctionne correctement et qu’aucun bug n’est présent car le code présent sur cette branche est le codé utilisé sur le serveur de production, c’est-à-dire le serveur où est déployé l’application finale utilisée par le client. Pour tous les commits présents sur cette branche, nous allons réaliser les jobs suivants:

- Les tests unitaires
- L’analyse de sécurité des dépendances
- Les tests features
- L’analyse de sécurité de l’image Docker

- Le workflow lié aux métriques
- Construction et push de l'image Docker sur GHCR
- Déploiement en production (nécessite une validation humaine sur CircleCI)
- Les tests E2E

Il est important de noter que lors d'un commit réalisé sur la branche Main, nous ne faisons pas de déploiement en staging car cela serait inutile.

Conclusion:

En conclusion, nous pouvons dire que notre pipeline CI/CD est assez complète et permet de prendre en compte plusieurs problématiques importantes en entreprise comme:

- Séparation du métier à travers différents jobs
- Optimisation de l'exécution des jobs lors des commits
- Utilisation d'un service Cloud reconnu (AWS)

Cependant, ce projet n'est pas parfait. En effet, certains points pourraient être améliorés par la suite comme notamment:

- Utilisation de serveurs différents pour différencier la production et le staging
- Ajout de nouveaux types de tests (Tests d'intégration, Tests de performance)
- Ajout de nouvelles règles liées au branching (ex: Si un rapport de sécurité est mauvais, le commit est annulé)