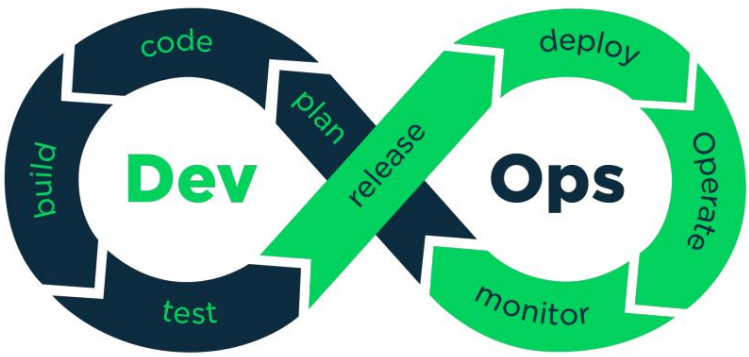
B2 - Introduction au DevOps

B-DOP-200

Chocolatine

Améliorez l'intégration et les tests avec GitHub Actions



Chocolatine

• La totalité de vos fichiers sources, à l'exception de tous les fichiers inutiles (binary, temp files, obj fichiers,...), doivent être inclus dans votre livraison.

Quelles que soient vos opinions, c'est une chocolatine.

Mais à quoi ça sert ? Cela vous permet de mieux traverser la journée et de mieux travailler. En d'autres termes, il facilite votre travail et vous permet de vous concentrer sur les choses qui comptent (discuter de son nom n'appartient pas à cette catégorie).

Vos référentiels sont hébergés sur GitHub, et GitHub dispose d'un moyen intuitif d'automatiser le lancement d'actions prédéfinies, déclenchées par des événements de votre choix. C'est, au même titre que les chocolatines, un excellent moyen de faciliter et d'améliorer votre développement et votre flux de travail.

Dans ce projet, vous allez configurer un flux de travail GitHub Actions pour appliquer les bonnes pratiques et diverses règles dans un référentiel.

Les actions GitHub sont gratuites et illimitées pour les dépôts publics, et gratuites jusqu'à 2 000 minutes par mois pour les dépôts privés personnels, ce qui est assez généreux !

DÉTAILS TECHNIQUES

Vous devrez rendre un seul fichier de flux de travail YAML nommé chocolatine.yml, placé soit :

- à la racine de votre référentiel ; • ou dans le dossier .github/workflows .

Votre workflow doit être utilisable avec vos différents projets Epitech et leurs technologies respectives et outils associés.

Pour ce projet, les seules actions externes autorisées sont les actions/paiement et pixta-dev/repository-mirroring-action.
Toutes les autres actions externes (que vous pouvez trouver sur la place de marché GitHub par exemple) sont strictement interdites.

Si des paramètres ou des éléments particuliers ne sont pas spécifiés ou abordés dans le sujet, vous êtes libre d'en faire ce que vous voulez.

ÉVALUATION

Votre flux de travail sera testé en copiant votre fichier chocolatine.yml , et uniquement ce fichier, dans le répertoire .github/workflows d'un référentiel de test.
Assurez-vous ensuite que le flux de travail est autonome et n'a pas besoin de fichiers externes.

SECRETS

Vous devrez peut-être utiliser des données ou des valeurs sécurisées pour que votre flux de travail s'exécute correctement.
Dans ce cas, vous devez utiliser des secrets.

Si une valeur sensible codée en dur est trouvée dans votre fichier de flux de travail, l'ensemble de votre projet échouera. Tu étais prévenu.

CARACTÉRISTIQUES

À l'aide d'un workflow GitHub Actions, vous devrez implémenter un ensemble de fonctionnalités.

Le workflow doit être exécuté à chaque push et à chaque création de pull request, sauf si le nom de la branche commence par ga-ignore-, auquel cas le workflow ne doit pas être exécuté du tout.

De plus, chaque emploi doit :

- commencer par extraire le référentiel auprès de la branche concernée ; •
être exécuté uniquement si le travail précédent a réussi.

L'ordre dans lequel les travaux sont censés être exécutés est l'ordre dans lequel ils sont définis ci-dessous dans le sujet.

VARIABLES D'ENVIRONNEMENT

Vous devez définir plusieurs variables d'environnement, disponibles au niveau du workflow.

- MIRROR_URL : l'URL du référentiel Epitech qui fera office de miroir ; • EXECUTABLES :
une liste séparée par des virgules des chemins des exécutables à produire (il peut y avoir un seul exécutable, comme « pushswap » ou « lib/my/libmy.a », ou plusieurs, comme « antman/antman, homme géant/homme géant").

Toutes les valeurs des variables d'environnement seront des chaînes.

"disponible au niveau du workflow" signifie que les variables d'environnement ne doivent être définies qu'une seule fois et être disponibles pour tous les travaux et étapes du workflow.

VÉRIFICATION DU STYLE DE CODAGE

Créez une tâche avec un ID check_coding_style qui s'exécute dans un conteneur Docker ghcr.io/epitech/coding-style-checker:latest , et cela dans l'ordre :

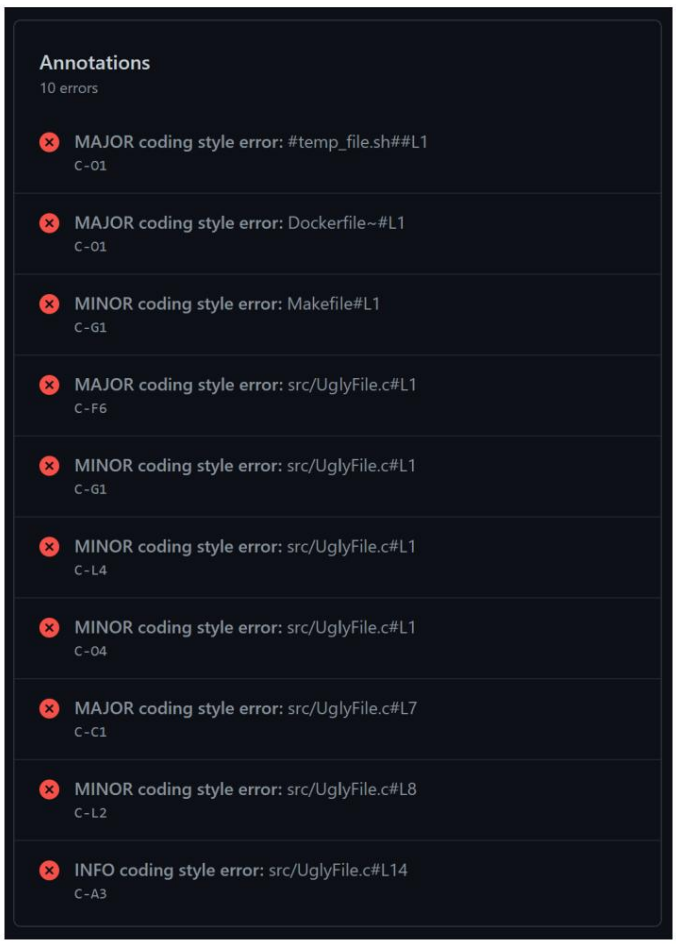
• exécute le script de vérification du style de codage avec la commande suivante :

Terminal - + X

```
check.sh $(mot de passe) $(mot de passe)
```

• le cas échéant, affiche chaque erreur de style de codage sous la forme d'une annotation d'erreur ; • fait échouer la tâche s'il y a des erreurs de style de codage.

La sortie de la tâche doit ressembler à ceci :



En utilisant les paramètres d'annotation d'erreur appropriés, vous pourrez cliquer dessus et être dirigé vers la ligne précise du fichier incriminé.

VÉRIFICATION QUE LE PROGRAMME SE COMPILE CORRECTEMENT

Créez une tâche avec un ID check_program_compilation qui, dans l'ordre :

- lance make à la racine du dépôt (cette étape doit avoir un timeout de 2 minutes) ; • lance (dans une étape distincte) make clean à la racine du référentiel ; • vérifie que chaque fichier spécifié dans la variable d'environnement EXECUTABLES existe et est exécutable (le le travail doit échouer si l'un des exécutables n'est pas là ou n'est pas exécutable).

Ce travail doit être exécuté dans un epitechcontent/epitest-docker Conteneur Docker.

EXÉCUTER DES TESTS (PARCE QUE C'EST CE QUE FONT LES HÉROS)

Créez une tâche avec un ID run_tests qui lance make tests_run à la racine du référentiel (cette étape doit avoir un délai d'attente de 2 minutes).

Ce travail doit être exécuté dans un epitechcontent/epitest-docker Conteneur Docker.

POUSSER VERS LE RÉFÉRENTIEL MIROIR

Créez une tâche avec un ID push_to_mirror qui exécute une mise en miroir vers le référentiel spécifié dans la variable d'environnement MIRROR_URL .

Il doit utiliser un secret nommé GIT_SSH_PRIVATE_KEY pour spécifier la clé privée SSH à utiliser.

Ce travail ne doit être exécuté que lorsqu'un push est effectué vers le référentiel.