Cahier de TP « Gitlab / Gitlab CI»

Pré-requis :

- Bonne connexion Internet
- Système d'exploitation recommandé : Linux ou Windows 10
- Utilisation de Docker
- Bon éditeur : .yml
- Pour le dernier TP, un cluster Kubernetes disponible, possibilité d'utiliser minikube

TP1 : Commandes de base Git

1.1 Installation et configuration de Git

Après l'installation de Git, le configurer afin de pouvoir committer

```
$ git config --global user.name "John Doe"

$ git config --global user.email johndoe@example.com
```

1.2 Création de dépôt et premier commit

Récupérer les sources fournies, initialiser un dépôt, vérifier le fichier *.gitignore* et ajouter toutes les sources du projet

1.3 Création de branche et fusion

Basculer vers une nouvelle branche ***dev***
Éditer un fichier, par exemple le fichier *pom.xml* à la racine

Revenir à la branche ***master***
Éditer le même fichier et les mêmes lignes

Intégrer les modifications de ***dev*** dans ***master*** avec la commande ***merge***, résoudre les conflits

Visualisez les changements avec *gitk*

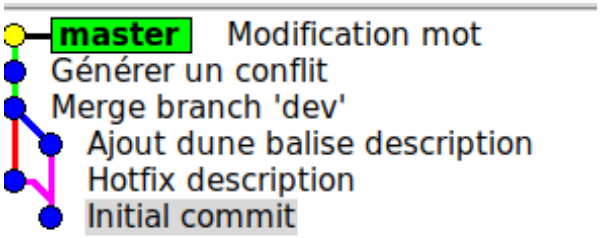
Supprimer la branche ***dev***

1.4 Création de branche et rebase

Effectuer les mêmes opérations que la section précédente.

Rebaser ensuite ***dev*** sur ***master***
Faire une fusion rapide de master, (Sur master : *git merge dev*)
Visualisez les changements avec *gitk*
Supprimer la branche ***dev***

A la fin du TP, l’historique du projet doit ressembler à :



TP2 : Configuration équipe Gitlab

Pré-requis: Installation gitlab

Via docker :

```
sudo docker run --detach \
  --hostname gitlab.formation.org \
  --publish 443:443 --publish 80:80 --publish 22:22 \
  --name gitlab \
  --volume /srv/gitlab/config:/etc/gitlab \
  --volume /srv/gitlab/logs:/var/log/gitlab \
  --volume /srv/gitlab/data:/var/opt/gitlab \
  gitlab/gitlab-ce:latest
```

Configurer également votre */etc/hosts* pour faire pointer *gitlab.formation.org* vers *localhost*

Se connecter sur http://gitlab.formation.com avec un navigateur

Avec un compte admin, créer

- Un utilisateur avec votre identité
- Un login ***developer***
- Un login ***reporter***

Se connecter avec votre compte

Créer 1 groupe de projet

- ***myteam***: Visibilité privé

Répartir les rôles sur les groupes de projet avec les utilisateurs précédents

Définir 2 projets appartenant à ce groupe :

- 1 nommé ***dummy***. Ce projet sera initialisé avec un fichier README.md
- 1 nommé ***multi-module.*** Ce projet sera initialisé avec le projet multi-module du TP précédent

Pour votre utilisateur, créer une paire de clés *ssh* et la configurer sur le serveur

Effectuer en suite :

- un *git clone* sur le projet dummy
- un *git push* avec le dépôt créer dans le premier atelier sur le projet ***multi-module***

TP3 : Gestion des issues

Créer 3 labels :

- BUG
- RFC
- Refactoring

Créer plusieurs issues, accéder au tableau de bord, utiliser les listes par défaut (to do, doing) déplacer les issues

Démarrer une merge request à partir d'une issue.

TP4 : Merge Request

Sur le projet *multi-module*

Avec votre compte, vérifier la configuration des MRs, en particulier si toutes les discussions doivent être résolues.

Se logger avec le compte ***developer***

- Créer une branche feature1
- Créer une Merge Request ***WP : Feature1*** et l'associer à la branche précédente

- Commencer à collaborer et ouvrir une discussion avant de pousser des modifications.
- Faire un commit directement par le site web
- Faire un push à partir de votre repository local
- Changer le statut de la MR

Se logger avec votre compte (***maintainer***)

- Faire une revue de code
- Répondre à la discussion ouverte et résoudre la discussion
- Effectuer le Merge et vérifier la suppression de la branche

TP5 : Installation et Mise en place de runner

En fonction de votre OS, installer un runner approprié et démarrer le service.

Dans un environnement Linux :

- Installer le gitlab-runner en service via le dépôt: https://docs.gitlab.com/runner/install/linux-repository.html

Editer la configuration de gitlab-runner ***/etc/gitlab-runner/config.toml,*** vérifier que vous pouvez exécuter 2 exécuteurs en //

Si vous modifiez la config, redémarrer le service

Enregistrer 2 runners/shell sur le projet multi-module

TP6 : Exécution de pipeline

Dans le projet fourni, créer une branche et un merge request nommé ***pipeline***.

6.1 Simple pipeline

Mettre en place un fichier ***.gitlab-ci.yml*** qui effectue une phase de compilation et de tests unitaires.

Faire en sorte que les dépendances Maven soient cachés.

6.2 Pipeline à 3 phases

Ensuite, mette en place une pipeline à 3 phases :

- 1 phase de packaging stockant l'artefact généré :
  Commande Maven :
  `./mvnw clean package`
- Une phase de test incluant 2 jobs :
  - Un job exécutant des tests d'intégration
    Commande maven :
    `./mvnw integration-test`
  - Un job exécutant une analyse qualité Sonar. Il nécessite une installation préalable de Sonar (Voir DockerHub)
    Commande maven :
    `./mvnw verify sonar:sonar`
- Une phase de déploiement récupérant l'artefact généré et le copiant dans une arborescence

TP7 : GITLAB_STRATEGY, when, Utilisation de gabarit

7.1 Amélioration de la pipeline précédente

Pour le job *deploy-integration*, s'assurer que le dépôt n'est pas cloné en positionnant la variable ***GIT_STRATEGY***

Variabiliser l'emplacement de copie du jar dans le job *deploy-integration*

S'assurer que tous les jobs ne s'exécutent que sur une branche excepté master

7.2 Utilisation de gabarits

Créer un nouveau projet nommé ***gabarit***

Ajouter un fichier ***maven.yml*** qui reprend toutes les phases définies dans multi-module. Commiter dans la branche *master*

Dans le projet *multi-module*, créer une nouvelle branche ***template*** modifier le fichier ***.gitlab-ci.yml*** afin qu'il utilise le gabarit précédent

Visualiser les gabarits proposés par Gitlab

TP8 : Intégration de docker

8.1 Utiliser une image docker pour builder

Option 1 (simple) :

Utiliser le service *gitlab.com* qui permet de facilement mettre en place des runner exécutant des images docker

Créer un projet vide sur *gitlab.com*, y pousser votre dépôt local

Option 2 (+compliqué) :

Configurer un runner docker en local.
Dans la configuration *config.toml* Ajouter dans la section [runners.docker] la ligne :

```
network_mode = "host"
```

Créer ensuite une branche ***docker*** dans votre projet et modifier le fichier *.gitlab-ci.yml* afin que ce soit l'image ***openjdk:8-jdk-alpine*** qui exécute la pipeline.

8.2 Construire un artefact docker et le publier dans un registre

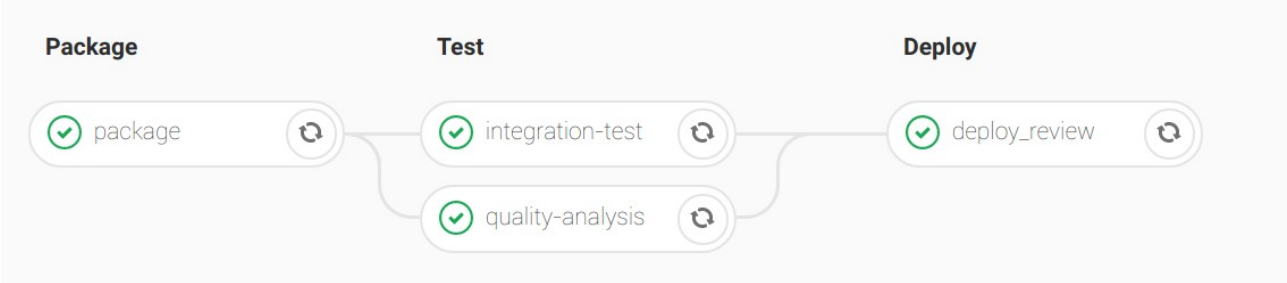
Construisez une image applicative en utilisant le fichier Dockerfile présent dans *application/src/main/docker* et pousser l'image construite sur DockerHub (nécessite un compte docker-hub)

TP9 : Mise en place d'environnements

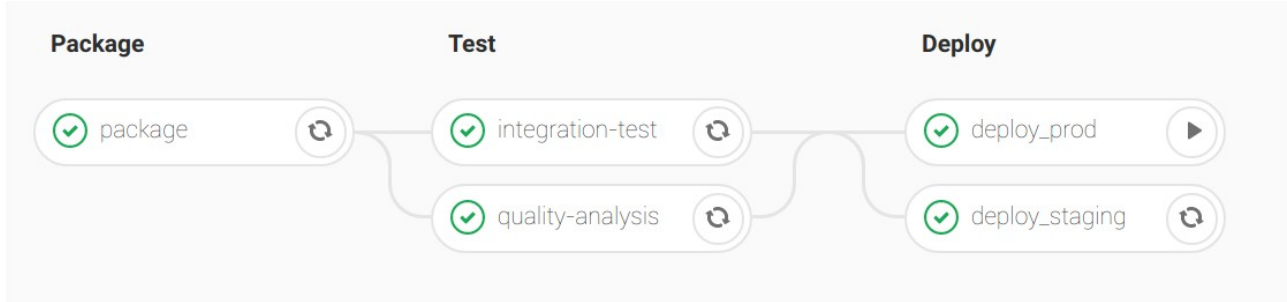
Reprendre la pipeline précédente (version docker ou version classique)

- Définir des jobs de déploiements dans les environnements suivants :
  - Environnement dynamique reprenant le nom de branche
  - Environnement de QA, réservé à la branche master, le déploiement est automatique
  - Environnement de production, réservé à la branche master, le déploiement est manuel

Pipeline feature-branch :



Pipeline master :



Visualiser les différents déploiements exécutés

TP10 : Intégration Kubernetes

Nécessite une pré-installation d'un cluster Kubernetes

Suivre :
https://docs.gitlab.com/ee/user/project/clusters/add_remove_clusters.html

Cas de minikube, il faut autoriser des appels sur le réseau interne :

Admin Area -> Settings -> Outbound requests : Autoriser et ajouter l'API Url fournit par minikube

Modifier les tâches de déploiement afin de déployer sur le cluster Kubernetes . Utiliser les variables d'environnement fourni par Gitlab et en particulier le namespace projet

TP10 : AutoDevOps

Démonstration de l'auto devops avec Google Kubernetes Engine