Ateliers

Gitlab CI/CD

Pré-requis :

- Bonne connexion Internet
- Environnement Linux de préférence
- Minimum 16Go de RAM
- ***Docker***
- ***git*** et ***gitk***
- JDK21
- Un serveur gitlab avec des runners docker et optionnellement un accès à un cluster Kubernetes

Table des matières

Atelier 1 : Démarrage

Objectifs : Premier accès, parcours de l'interface utilisateur, mise en place clé ssh

1.1 Accès administrateur et création de son compte

Accèder à : ***gitlab.formation.org*** avec ***:***

- username : **root**
- password : **/Welcome1**

Créer vous un compte et définissez vous un mot de passe.

1.2 Mise en place clé ssh

Commencer par générer une clé ssh sur votre poste :

```
ssh-keygen -t rsa -b 2048 -C "<comment>"
```

Appuyer sur Entrée à chaque question de l'assistant sans préciser de Pass Phrase. A la fin 2 fichiers ont été créés dans le répertoire ***~/.ssh*** :

- ***id_rsa*** : La clé privé. Ne doit pas jamais transiter par le réseau
- ***id_rsa.pub*** : La clé publique. Elle peut être fournie à des tiers

Se connecter avec votre compte et accéder à *Preferences → SSH Keys* et ajouter une nouvelle clé en copiant/collant le contenu de votre clé public

Tester en commande en ligne avec :

```
ssh -T git@gitlab.formation.org
```

Atelier 2: Workflow de collaboration

Objectifs :

- Comprendre les différents acteurs accédant aux projet
- Visualiser le support pour la planification et le suivi de tâches
- Comprendre le workflow de résolution d'issues

2.1 Groupes / Projets / Membres

Avec votre compte,

- Création d'un groupe de projet à votre nom et affecter des membres parmi les stagiaires avec des rôles *developer*
- Création d'un projet privé nommé ***delivery-service***, en initialisant un dépôt. (Présence d'un fichier *README*)

Parcourir les menus du projet ***delivery-service,*** en particulier ***settings***

2.2 Branche protégée

Vérifier que la branche main est votre branche par défaut. Si ce n'est pas le cas faites les opérations nécessaires.

En tant que mainteneur de votre projet configurer une branche protégée ***production*** avec les permissions suivantes :

- *Allow to Merge* : Mainteneur
- *Allow tu push and merge* : No one
- *Force push* : Non

2.3 MergeRequest

1. Création de merge request à partir d'une issue

Commencer par créer une Issue '*CRUD pour delivery-service*',
A partir de cette, créer une *Merge Request et* l'affecter à un développeur
=> La merge request est préfixée par ***Draft*** et a pour effet de créer une branche portant le nom de l'issue

2. Mise en place environnement de développement + développement

En tant que développeur sur votre poste de travail :

- Récupérer la branche de la merge request :

`git clone <url-ssh-depot>`
`git checkout <nom-de-branche>`

- **Reprendre le code source fourni dans le répertoire TPs/2.3_MergeRequest/FirstCommit**

Construire l'application :
`./mvnw clean package`

Exécuter l'application :
`java -jar application/target/delivery-service-0.0.1-SNAPSHOT.jar`

Accéder à l'application :
http://localhost:8080/swagger-ui.html

3. Pousser les modifications
Le développeur pousse les modifications
`git add .`

`git commit -m 'Implémentation CRUD'`

`git push`

En tant que *developer* sur gitlab, supprimer le préfixe *Draft* de la MR

4. Revue de code
En tant que *owner/mainteneur*, faire une revue de code et refuser la fusion.

5. Compléments de développement et maj dépôt
Reprise du code fourni dans le répertoire TPs/2.3_MergeRequest/SecondCommit

Exécuter les tests et s'assurer qu'ils passent :
`./mvnw test`

Push les modifications vers gitlab

6. Accepter la MR
En tant que *Owner/Mainteneur* faire une revue de code
Accepter le Merge Request, supprimer la branche et éventuellement un *« squash commit »*

7. Nettoyage local
En local, en tant que développeur supprimer la branche locale et exécuter
`git  remote prune origin`

Ateliers 3 : Syntaxe .gitlab-ci.yml

3.1 Stages et Job

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

La pipeline peut utiliser l'image : *openjdk:21-jdk-oracle*

Visualise le tag du runner disponible et déclarer ce tag dans la pipeline

La commande pour compiler est :

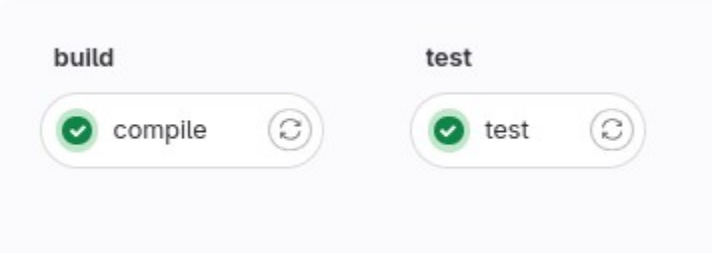
```
./mvnw $MAVEN_OPTS compile
```

Pour tester :

```
./mvnw $MAVEN_OPTS test
```

Les options maven suivantes peuvent être positionnées en une variable MAVEN_OPTS:

```
-Dhttps.protocols=TLSv1.2 -Dmaven.repo.local=$CI_PROJECT_DIR/.m2/repository
-Dorg.slf4j.simpleLogger.log.org.apache.maven.cli.transfer.Slf4jMavenTransferListener=WARN
-Dorg.slf4j.simpleLogger.showDateTime=true
-Djava.awt.headless=true
```



Observer l'exécution de la pipeline, les dépendances sont elles cachées ?

3.2 Artefact et dependencies

Modifier la pipeline afin qu'elle intègre 3 phases :

- 1 phase de **packaging** stockant l'artefact généré :
  Commande Maven :

```
./mvnw clean package
L’artefact se trouve dans le répertoire target
```

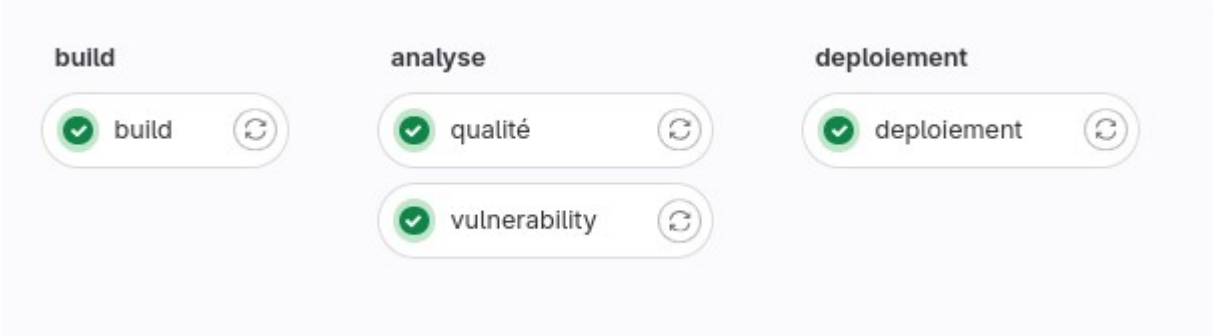
- Une phase **de test** incluant 2 jobs :
  - Une analyse de vulnérabilités des dépendances, elle n’a pas besoin du jar pécédent
    Commande maven :

```
./mvnw dependency-check:aggregate
```

  - Un job démarrant l’application construite précédemment et executant les tests JMeter
    Commande maven :

```
./mvnw -Dsonar.host.url=http://gitlab-sonar:9000 -Dsonar.token=${SONAR_TOKEN} verify sonar:sonar
```

- Une phase de **déploiement** récupérant l’artefact généré et le copiant dans une arborescence variabilisée. Utiliser l’exécuteur shell pour ce job



Mise en place de sonarqube

- Se connecter à localhost:9000 avec admin/admin
- Changer le mot de passe
- Dans le Menu en haut à droit ***My Account → Security***
- Générer un jeton « Global Analysis »,
- Déclarer une variable groupe masquée via l’interface Gitlab SONAR_TOKEN qui contient le jeton

4.3 Conditions et GITLAB_STRATEGY

Pour le job *deploy-integration*, s’assurer que le dépôt n’est pas cloné en positionnant la variable *GIT_STRATEGY*
S’assurer qu’il ne s’exécute que si l’on est sur une branche protégée

Interdire l’exécution simultanée de 2 pipelines sur des commits différents

4.4 Gabarits et inclusions

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

Ajouter un fichier ***maven.yml*** qui reprend toutes les phases définies dans la pipeline précédente

Commiter dans la branche main.

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

4.5 Construction d'une image docker et publication dans un registre

Dans un branche ***docker***, récupérer le fichier Dockerfile fourni et le placer à la racine du projet

Committer et pousser.

Créer vous un compte docker hub et définir des variables DOCKER_LOGIN et DOCKER_PWD dans les settings du groupe

Modifier le fichier *.gitlab-ci.yml* en ajoutant un job qui :

- Récupère les artefacts jar et Dockerfile
- Se connecte sur dockerHub
- Construit une image docker
- La pousse sur votre registre

4.6 Mise en place d'environnements

Reprendre la pipeline précédente

- Définir des jobs de déploiements dans les environnements suivants :

- Environnement dynamique reprenant le nom de branche
- Environnement de QA, réservé à la branche main, le déploiement est automatique
- Environnement de production, réservé à la branche main, le déploiement est manuel

Le déploiement pourra être effectué avec une commande *docker run*

Ateliers 5 : Support gitlab pour jobs standard

5.1 Publication des tests unitaires

Dans la phase de build, publier le résultat des tests unitaires présents dans
/target/surefire-reports/*.xml

5.2 Couverture des tests

Le rapport de couverture test est fourni par jacoco dans le répertoire ***target/site/jacoco/index.html***
L'expression régulière permettant d'extraire le résultat est : ***/Total.*?([0-9]{1,3})%/***

5.3 Analyses statique

Inclure les gabarits permettant :

- L'analyse qualité
- La détection de secret
- SAST

L'analyse qualité nécessite un runner partagé avec ***DockerInDocker***

```
sudo gitlab-runner register -n \
--url http://gitlab/ \
--registration-token REGISTRATION_TOKEN \
--executor docker \
--description "My Docker Runner" \
--docker-image "docker:20.10.16" \
--docker-privileged \
--docker-volumes "/certs/client"
```

+ éventuellement si démarrage par docker-compose :

```
links = ["gitlab.formation.org"]
network_mode = "<nom-du-network-docker-compose>"
```

5.4 Gitlab Registry (Maven)

Ajouter un fichier ***settings.xml*** comme suit :

```
<settings>
  <servers>
```

```
    <server>
      <id>gitlab-maven</id>
      <configuration>
        <httpHeaders>
          <property>
            <name>Job-Token</name>
            <value>${env.CI_JOB_TOKEN}</value>
          </property>
        </httpHeaders>
      </configuration>
    </server>
  </servers>
</settings>
```

Ajouter dans le fichier ***pom.xml*** la configuration de déploiement

```
<repositories>
  <repository>
    <id>gitlab-maven</id>
    <url>${CI_API_V4_URL}/projects/${CI_PROJECT_ID}/packages/maven</url>
  </repository>
</repositories>
<distributionManagement>
  <repository>
    <id>gitlab-maven</id>
    <url>${CI_API_V4_URL}/projects/${CI_PROJECT_ID}/packages/maven</url>
  </repository>
  <snapshotRepository>
    <id>gitlab-maven</id>
    <url>${CI_API_V4_URL}/projects/${CI_PROJECT_ID}/packages/maven</url>
  </snapshotRepository>
</distributionManagement>
```

Ajouter un script dans le premier job de build :

```
./mvnw -s settings.xml deploy
```

5.5 Release

En reprenant l'exemple des slides, ajouter un job de release ne s'exécutant que sur la branche par défaut

Faire en sorte que le n° de version corresponde à celui indiqué dans le fichier pom.xml

Pour obtenir le n° de version, vous pouvez utiliser la commande suivante pour obtenir le n° de version :

```
./mvnw org.apache.maven.plugins:maven-help-plugin:3.1.0:evaluate -Dexpression=project.version -q -DforceStdout
```

5.6 Kubernetes

Installer ***kind***

Modifier la pipeline afin de déployer sur le cluster kind, essayer d'obtenir les bons liens sur les environnements de développement.

Implémenter le ***on_stop*** permettant de nettoyer les ressources lors de la suppression d'une branche

Etape 1 : Vérifier l'exécution de kubectl

Etape 2 : Fournir manifeste kubernetes

5.7 AutoDevOps

Rennomer le fichier *.gitlab-ci.yml* et pousser le projet sur gitlab.com