

Ateliers

Gitlab CI/CD

Pré-requis pour jouer les démos :

- Bonne connexion Internet
- Environnement Linux de préférence
- **Docker**
- **git** et **gitk**
- Optionnellement JDK21
- Optionnellement service **gitlab-runner**
- Optionnellement cluster Kubernetes : **kind**

Manipulations pour visualiser les solutions :

Dans un répertoire de travail, exécuter

```
mkdir multi-module
```

```
git clone https://github.com/dthibau/gitlab-solutions.git
```

Pour chaque atelier où il y a un tag, il suffit d'appeler le script **goto.sh** du répertoire *gitlab-solutions* pour mettre à jour le répertoire *multi-module* :

```
cd gitlab-solutions
```

```
./goto.sh <tag>
```

=> Le projet **multi-module** est alors dans l'état du tag correspondant à la démo.

Table des matières

Atelier 1 : Démarrage de la plateforme.....	3
Atelier 2: Workflow de collaboration.....	4
2.1 Groupes / Projets / Membres.....	4
2.2 Milestones, Issues, Labels, Tableaux de bord.....	4
2.3 MergeRequest.....	5
Ateliers 3 : Bases pipelines Gitlab.....	7
3.1 Enregistrement de Runners.....	7
3.2 Première pipeline et gabarits.....	8
Ateliers 4 : Syntaxe .gitlab-ci.yml.....	9
4.1 Stages et Job.....	9
4.2 Artefact et dependencies.....	9
4.3 Conditions et GITLAB_STRATEGY.....	10
4.4 Gabarits et inclusions.....	11
4.5 Construction d'une image docker et publication dans un registre.....	11
4.6 Mise en place d'environnements.....	11
Ateliers 5 : Support gitlab pour jobs standard.....	12
5.1 Publication des tests unitaires.....	12
5.2 Couverture des tests.....	12
5.3 Analyses statique.....	12
5.4 Gitlab Registry (Maven).....	12
5.5 Release.....	13
5.6 AutoDevOps.....	13

Atelier 1 : Démarrage de la plateforme

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

Option1 : Installation locale via Docker

Démarrage installation gitlab via Docker :

Visualiser le fichier docker-compose fourni et l'adapter à votre environnement

```
docker compose up -d
```

Modifier `/etc/hosts` ou `C:\Windows\System32\Drivers\etc\hosts` afin que **gitlab.formation.org** pointe sur `localhost`

```
127.0.0.1 gitlab.formation.org
```

Récupérer le mot de passe **root** avec :

```
sudo docker exec -it gitlab grep 'Password:'  
/etc/gitlab/initial_root_password
```

Se logger avec **root** et changer le mot de passe

Visualisation interface administrateur

Création de comptes

Avec le compte administrateur, mettre en place 2 comptes Gitlab

Un propriétaire de projet : **leader** (Mot de passe : `/Welcome1`)

Un développeur vous représentant

Se connecter avec le compte **developer** et mettre en place de clé ssh

Option2 : Utilisation plateforme en ligne

Se créer un compte sur `gitlab.com`, installer sa clé ssh

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 le compte *leader*,

- Création d'un groupe de projet *formation* et affecter les membres *productowner* et *developer* dans leur différents rôles
- Création d'un projet privé nommé *multi-module*, en initialisant un dépôt. (Présence d'un fichier *README*)

Parcourir les menus du projet *multi-module*, en particulier *settings*

2.2 Milestones, Issues, Labels, Tableaux de bord

Mise en place des labels, Milestone, et tableaux de bord

En tant que mainteneur de projet, créer 2 milestones :

- **Sprint1**
- **Sprint2**

Au niveau groupe, définir les labels suivants :

- **In progress**
- **Review**

Définir ensuite un tableau de bord ajoutant des colonnes pour les 2 labels précédents

Au niveau projet, utiliser les labels par défaut de Gitlab +

- **API**
- **DevOps**

Création d'issues

Avec le compte *reporter*

- Saisir plusieurs issues dont une s'appelant : « *CRUD pour delivery-service* »
- Tagger avec *API*

Discussion sur une issue entre Reporter/Mainteneur projet :

- Saisir quelques commentaires
- Saisir quelques issues techniques :
 - « Mise en place pipeline CI »,
 - « Configuration repository », ...

Les tagger avec DevOps

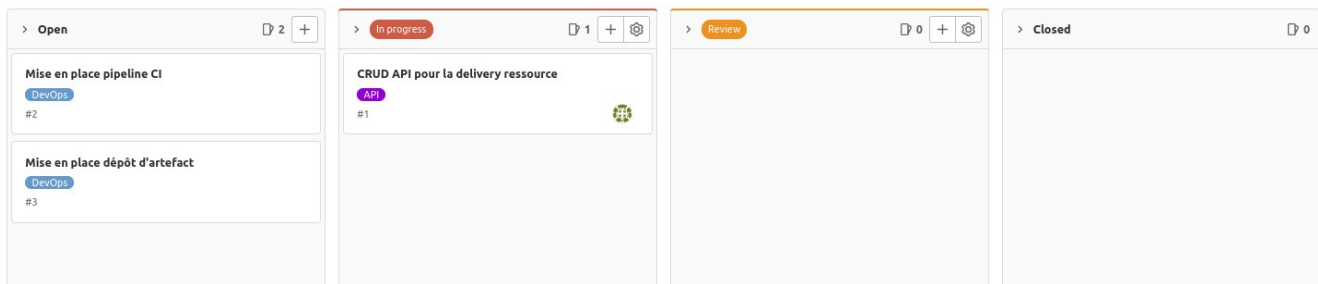
Avec le compte *owner/mainteneur* :

- mise au planning et affectation
- Tagger les issues

Avec le compte *developer*, accès au tableau de bord et déplacement du post-it «*CRUD pour delivery-service*»

To Do -> *In progress*

A la fin de ces opérations, le tableau de bord pourra ressembler à ce qui suit :



2.3 MergeRequest

1. Création de merge request sur gitlab

En tant que développeur sur gitlab, à partir de l'issue, '*CRUD pour delivery-service*', créer une *Merge Request*

=> 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 :

- Installer une clé ssh
- Récupérer la branche de la merge request :

```
git clone <url-ssh-depot>
```

```
git checkout <nom-de-branche>
```

- **Reprendre le tag 2.3 des solutions** (Dans le répertoire solutions : *./goto.sh 2.3*)

Construire l'application :

```
./mvnw clean package
```

Exécuter l'application :

```
java -jar application/target/multi-module-application-0.0.1-SNAPSHOT.jar
```

Accéder à l'application :

<http://localhost:8080>

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*

4. Revue de code

En tant que *owner/mainteneur*, faire une revue de code et ajouter comme commentaire :

« *Et les tests ?* »

5. Compléments de développement et maj dépôt

Reprise du tag 2.4

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 : Bases pipelines Gitlab

3.1 Enregistrement de Runners

Nous enregistrons 2 runners :

- Un runner partagé avec un exécuteur docker
- Un runner dédié au projet avec un exécuteur shell

Runner partagé

Avec le login root

Admin Area → *CI/CD* → *runners* → *New Instance Runner*

Indiquer le tag **docker**

Cocher également la case **Run untagged jobs**

Copier coller la commande d'enregistrement

Se logger sur le container *gitlab-runner* :

```
docker exec -it <nom-container-gitlab-runner> /bin/bash
```

Coller la commande d'enregistrement

Choisir les choix par défaut

Et pour la question « exécuteur » choisir **docker**

Indiquer également une image par défaut **ruby:2.7** par exemple

Vérifier que le runner s'est bien enregistré dans l'interface administrateur

Vérifier que le runner partagé est disponible pour le projet

Vérifier si une pipeline AutoDevOps a démarré sur notre projet

Pour que la pipeline s'exécute correctement, ajouter cette ligne dans la configuration du runner partagé (fichier **config/config.toml** dans le répertoire partagé du runner) ayant l'exécuteur Docker:

```
privileged = true
links = ["gitlab.formation.org"]
network_mode = "<nom-du-network-docker-compose>"
volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache"]
```

Pour voir le *<nom-du-network-docker-compose>*, vous pouvez exécuter :

```
docker network ls
```

Modifier également au niveau global

```
concurrent = 4
```

Runner dédié au projet

Avec le login **leader**

Project → Settings → CI/CD → Runner → New Instance Runner

Indiquer le tag **shell**

Se logger sur le container *gitlab-runner* :

`docker exec -it <nom-container-gitlab-runner> /bin/bash`

Coller la commande d'enregistrement

Choisir les choix par défaut

Et pour la question « exécuteur » choisir **shell**

Vérifier le bon enregistrement dans la page runners du projet

3.2 Première pipeline et gabarits

Dans une **branche de feature** avec le compte **developer**, créer un fichier **.gitlab-ci.yml** à partir des templates proposés par Gitlab.

Vous pouvez utiliser le projet Java précédent ou le projet dans la technologie de votre choix.

Repository → <feature-branch> → + → .gitlab-ci.yml → Sélectionner le template

Committer

Vérifier l'exécution de la pipeline, quel runner a été utilisé ?

Ateliers 4 : Syntaxe .gitlab-ci.yml

4.1 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*

La commande pour compiler est :

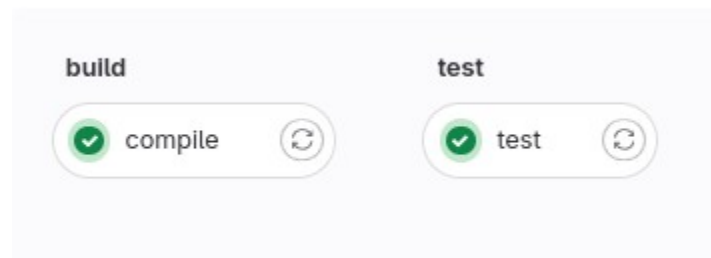
```
./mvnw $MAVEN_OPTS compile
```

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 ?

4.2 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`

- Une phase **d'analyse** incluant 2 jobs :
 - Une analyse de vulnérabilités des dépendances Un job exécutant des tests d'intégration

Commande maven :

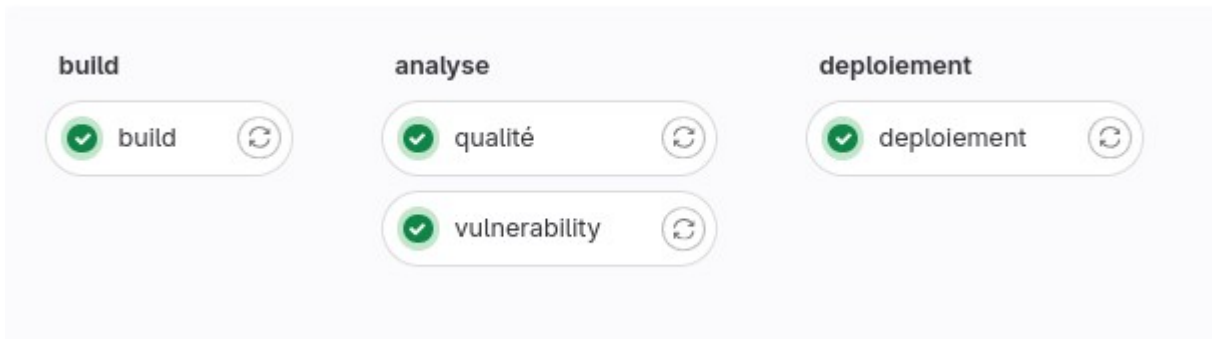
```
./mvnw dependency-check:aggregate
```

- Une analyse qualité Sonar. Il nécessite une installation préalable de Sonar (Voir + loin)

Commande maven :

```
./mvnw -Dsonar.host=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 droite **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

Committer 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

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

5.6 AutoDevOps

Renommer le fichier `.gitlab-ci.yml` et pousser le projet sur `gitlab.com`