

Démonstrations

Gitlab une plateforme DevOps complète

Pré-requis pour jouer les démos :

- Bonne connexion Internet
- Environnement Linux
- **Docker**
- **git** et **gitk**
- Optionnellement JDK11
- Optionnellement service **gitlab-runner**
- Optionnellement cluster Kubernetes : **kind**

Manipulations pour visualiser les solutions :

- `git clone https://github.com/dthibau/gitlabplatform-solutions.git solutions-delivery-service`

Pour chaque démo où il y a un tag, il suffit d'appeler le script **goto.sh** du répertoire *solutions-delivery-service* pour mettre à jour le projet *delivery-service* :

```
cd solutions-delivery-service
./goto.sh <tag>
```

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

Table des matières

Démonstration 1 : Démarrage de la plateforme.....	3
Démonstration 2: Pilotage de projet.....	4
2.1 Création de groupe de projets avec rôles pré-définis, Initialisation du projet.....	4
2.2 Milestones, Issues, Labels, Tableaux de bord.....	4
Démonstration 3 : Gestion des sources et collaboration.....	6
3.1 MergeRequest et GitlabFlow.....	6
Ateliers 4 : Concepts pipelines CI/CD.....	8
4.1 Enregistrement de Runners.....	8
4.2 Première pipeline.....	9
4.3 Dépendances et conditions.....	9
4.4 Gabarits et inclusions.....	9
4.5 Construction d'une image docker et publication dans un registre.....	10
4.6 Mise en place d'environnements.....	10
Démonstrations 5 : Phases d'une pipeline.....	11
5.1 Publication des tests unitaires.....	11
5.2 Couverture des tests.....	11
5.3 Analyse qualité.....	11
5.4 Analyse sécurité.....	11
5.5 Gitlab Registry.....	12
5.6 Push vers un registre Docker.....	12
5.7 Environnement et ReviewApp.....	12
5.8 Release.....	12
5.9 Tests de post déploiements.....	12
5.10 Intégration Terraform.....	13
5.11 Intégration Kubernetes.....	13

Démonstration 1 : Démarrage de la plateforme

Objectifs : Premier accès, parcours de l'interface utilisateur

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` afin que ***gitlab.formation.org*** pointe sur localhost

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 3 comptes Gitlab

Un propriétaire de projet : *leader/welcome1* :

Un utilisateur *métier* : *productowner/welcome1*

Un développeur : *developer/welcome1*

Se connecter avec leader, visualisation du menu du Account :

Mise en place de clé ssh pour le compte developer

Option2 : Utilisation plateforme en ligne

Se créer un compte sur gitlab.com

Démonstration 2: Pilotage de projet

Objectifs : Comprendre les différents acteurs accédant aux outils de pilotage et le support pour la gestion des Issues

2.1 Création de groupe de projets avec rôles pré-définis, Initialisation du projet

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 *delivery-service*, en initialisant un dépôt. (Présence d'un fichier *README*)

Parcourir les menus du projet *delivery-service*, 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

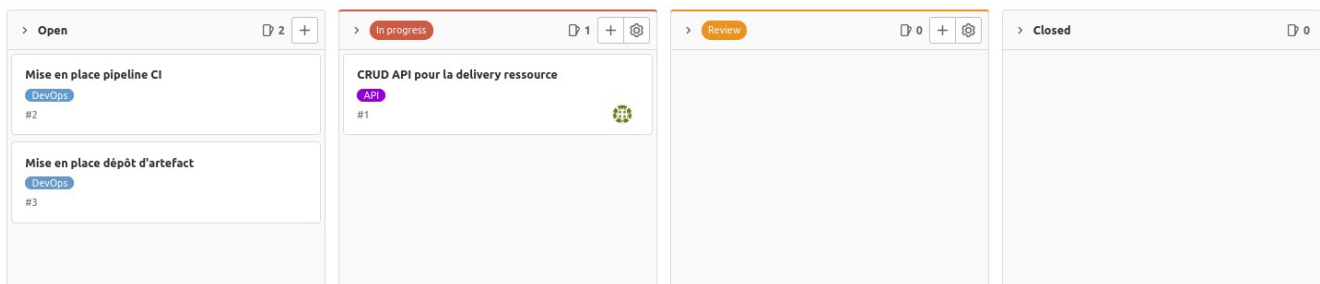
Avec le compte *owner/mainteneur* :

- 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 :



Démonstration 3 : Gestion des sources et collaboration

3.1 MergeRequest et GitlabFlow

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 3.2 des solutions** (Dans le répertoire solutions : `./goto.sh 3.2`)

Construire l'application :

```
./mvnw clean package
```

Exécuter l'application :

```
java -jar target/delivery-service-0.0.1-SNAPSHOT.jar \
--spring.profiles.active=swagger
```

Accéder à l'application :

<http://localhost:8080/swagger-ui.html>

<http://localhost:8080/actuator>

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 3.5 (Dans le répertoire solutions : `./goto.sh 3.5`)

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 4 : Concepts pipelines CI/CD

4.1 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**

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

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:

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

Modifier également

```
concurrent = 4
```

```
privileged = true
```

```
volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache"]
```

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

```
docker network ls
```

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

4.2 Première pipeline

Reprise du tag 4.2 (Dans le répertoire solutions : `./goto.sh 4.2`)

Visualiser le fichier `.gitlab-ci.yml`

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

4.3 Dépendances et conditions

Modifier le fichier `.gitlab-ci.yml` afin que le pipeline déclare 3 variables :

- **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"
- **INTEGRATION_URL**: /home/gitlab-runner/Integration
- **PRODUCTION_URL**: /home/gitlab-runner/Production

Afin que la pipeline déclare 3 stages :

- **build** associé au job **package** précédent et utilisant le tag **docker**, le job package crée un artifact à partir de **target/*.jar**
- **deploy-integration** associé au job **deploy-integration** et utilisant le tag **shell**, le job s'exécute pour toutes les branches à l'exception de la main
Le job récupère l'artefact créé par le job package et le copie dans le répertoire INTEGRATION_URL
- **deploy-production** : Identique à **deploy-integration** mais ne s'exécute que pour la branche **main** et copie l'artefact dans PRODUCTION_URL

Vérifier la bonne exécution de la pipeline et les changements

Comparer avec le tag 4.3 des solution (Dans le répertoire solutions : `./goto.sh 4.3`)

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_PASSWORD dans les settings du projet

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 master, le déploiement est automatique
- Environnement de production, réservé à la branche master, le déploiement est manuel

Le déploiement pourra être effectué avec une commande docker run

Démonstrations 5 : Phases d'une pipeline

5.1 Publication des tests unitaires

Reprise du tag 5.1 (Dans le répertoire solutions : *./goto.sh 5.1*)

Visualiser les changements dans *.gitlab-ci.yml*

Vérifier la bonne exécution de la pipeline et les changements

5.2 Couverture des tests

Reprise du tag 5.2 (Dans le répertoire solutions : *./goto.sh 5.2*)

Visualiser les changements dans *.gitlab-ci.yml* et *pom.xml*

Vérifier la bonne exécution de la pipeline et les changements

5.3 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"
```

Reprise du tag 5.3 (Dans le répertoire solutions : *./goto.sh 5.3*)

Visualiser les changements dans *.gitlab-ci.yml* et *pom.xml*

Vérifier la bonne exécution de la pipeline et les changements

5.4 Analyse sécurité

Reprise du tag 5.4 (Dans le répertoire solutions : *./goto.sh 5.4*)

Visualiser les changements dans *.gitlab-ci.yml* et *pom.xml*

Vérifier la bonne exécution de la pipeline et les changements

5.5 Gitlab Registry

Créer un nouveau projet «Registry Formation » en mode public sans l'initialiser.

Reprise du tag 5.5

Visualiser les changements dans *.gitlab-ci.yml*

Adapter le *pom.xml* à votre environnement

Vérifier la bonne publication du package Maven

5.6 Push vers un registre Docker

Définir 2 variables Projet contenant les crédits 'un compte DockerHub

Reprise du tag 5.6

Visualiser les changements dans *.gitlab-ci.yml* et l'adapter à votre environnement.

Visualiser la mise à jour du registre DockerHub

5.7 Environnement et ReviewApp

Nécessite un cluster Kubernetes configuré, mode démonstration

Reprise du tag 5.7

Visualiser les changements dans *.gitlab-ci.yml*

5.8 Release

Reprise du tag 5.8

Fusionner la branche de features dans la branche main

Observer la création de release.

Editer la release et y ajouter des packages

5.9 Tests de post déploiements

Reprise du tag 5.9

Visualisation des changements

Exécution de la pipeline et récupération des résultats de performance

5.10 Intégration Terraform

Créer une nouvelle branche *terraform* à partir de *main*

Reprise du tag 5.10

Visualiser le fichier *.gitlab-ci.yml* ainsi que le fichier *sample.tf* dans le répertoire *terraform*

Pousser les modifications dans la branche *terraform* et visualiser l'exécution de la pipeline

Fusionner dans la branche *main* et visualiser l'exécution de la pipeline, effectuer la dernière étape manuelle

Visualiser ensuite le stockage de l'état Terraform dans Gitlab

Visualiser également les variables du projet *Settings* → *CI/CD* → *Variables*

Créer une nouvelle branche de *tf_update*

Reprise du tag 5.10.2 visualiser les différences dans *sample.tf*

Pousser les modifications et attendre que le pipeline s'exécute correctement

Faire une fusion

Visualiser les variables du projet

5.11 Intégration Kubernetes

Étapes d'installation d'un agent :

<http://localhost/help/user/clusters/agent/install/index>

Nécessite d'autoriser une installation TLS

Reprise du tag 5.11