Ateliers

Formation Infrastructure DevOps : Les outils

Pré-requis :
Poste développeur avec accès réseau Internet libre
Linux (Recommandé) ou Windows 10
Pré-installation de :

- JDK11
- Git
- Oracle VirtualBox
- Docker et distribution minikube de Kubernetes

Installation durant les Tps :

- jenkins
- vagrant
- ansible (nécessite OS Linux)
- kubectl
- kind / minikube
- terraform

Manipulations pour visualiser les solutions :

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

Pour chaque atelier 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 l'atelier.

Atelier1: Outil de pilotage

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

Démarrage installation gitlab via Docker :

Positionner la variable d'environnement GITLAB_HOME. Par exemple, pour linux

```
export GITLAB_HOME=/formations/srv/gitlab
```

```
docker run --detach \
  --hostname gitlab.formation.org \
  --publish 443:443 --publish 80:80 --publish 22:22 \
  --name gitlab \
```

```
  --volume $GITLAB_HOME/config:/etc/gitlab \
  --volume $GITLAB_HOME/logs:/var/log/gitlab \
  --volume $GITLAB_HOME/data:/var/opt/gitlab \
   gitlab/gitlab-ce:latest
```

Modifier ***/etc/hosts*** afin que ***gitlab.formation.org*** pointe sur localhost

Création de comptes
Avec le compte administrateur
Mise en place de 4 comptes Gitlab
Adminisrateur : root
Owner / Maintener : Vous
Reporter : Un fonctionnel
Developer : Un développeur

Création de groupe de projets avec rôles pré-définis, Initialisation du projet
Avec le compte Owner,

- Création d'un groupe de projet ***formation*** et affecter les membres *Reporter* et *Developer* dans leur différents rôles
- Création d'un projet privé nommé ***delivery-service***, en initialisant un dépôt. (Présence d'un fichier *README*)
- Créer plusieurs ***milestone*** sur le projet

Saisie des issues
Avec le compte *reporter*

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

Discussion sur une issue entre Reporter/Mainteneur projet :

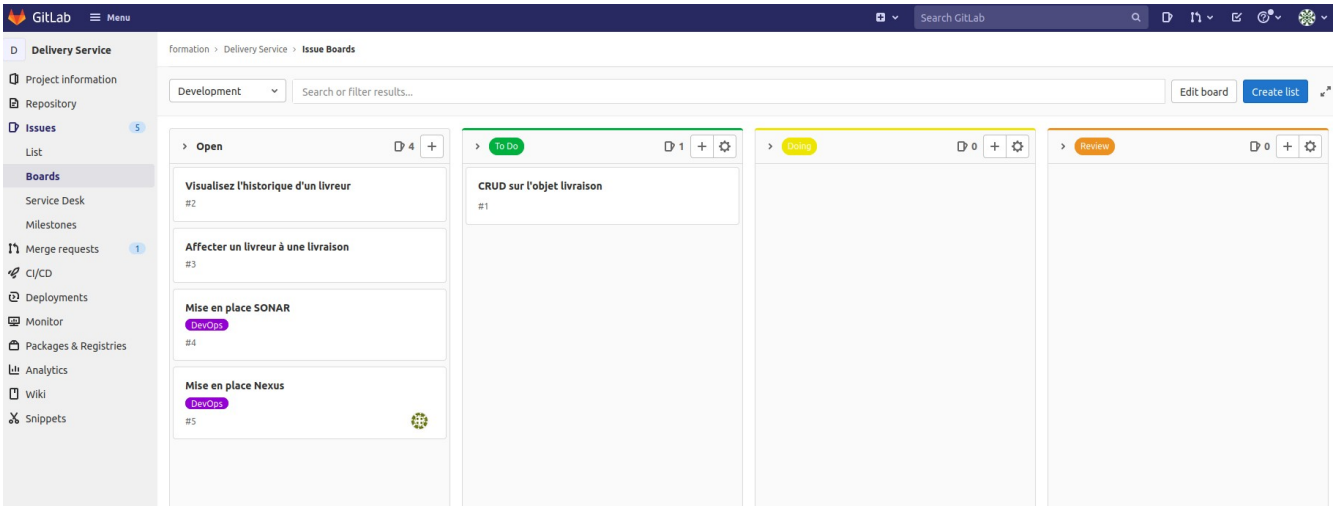
- Saisir quelques commentaires
- Saisir quelques issues techniques : « Mise en place Sonar », « Mise en place Nexus », ...

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
To Do -> Doing

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


GitLab
Menu
Search GitLab
Delivery Service
Project information
Repository
Issues
5
List
Boards
Service Desk
Milestones
Merge requests
1
CI/CD
Deployments
Monitor
Packages & Registries
Analytics
Wiki
Snippets
formation › Delivery Service › Issue Boards
Development
Search or filter results...
Edit board
Create list
Open
4
Visualisez l'historique d'un livreur
#2
Affecter un livreur à une livraison
#3
Mise en place SONAR
DevOps
#4
Mise en place Nexus
DevOps
#5
To Do
1
CRUD sur l'objet livraison
#1
Doing
0
Review
0

Atelier 2 : SCM et Workflow de collaboration *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 2.1 des solutions** (Dans le répertoire solutions : ***./goto.sh 2.1***)

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 2.2

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

Atelier 3 : L’outils de build Maven

3.1 Graphe de tâches et build reproductible

Dans le répertoire de Maven *$HOME/.m2/*
Créer un fichier *settings.xml* avec les lignes suivantes :

```
<settings>
  <pluginGroups>
        <pluginGroup>fr.jcgay.maven.plugins</pluginGroup>
  </pluginGroups>
</settings>
```

Afficher le graphe de tâches avec

```
./mvnw buildplan:list
```

3.2 Build reproductible :

Un nouveau développeur intègre l’équipe, il est capable de mettre en place son environnement de développement :

```
sudo adduser demodev
sudo su demodev
cd
git clone
./mvnw install
```

3.3 Analyse statique du code, Intégration de Sonar

- Démarrage sonar :

```
docker run -d --name sonarqube -p 9000:9000 sonarqube
```

Accéder à l’interface en tant qu’administrateur (***admin/admin***) et désactiver la sécurité
Administration → Sécurité → Force authentication (décocher)
Ou mieux configurer la sécurité de Sonar dans le fichier serrings.xml de Maven

Dans Gitlab, Création d’une MergeRequest ‘*Mettre en place Sonar*’
Du côté dév. Reprendre la branche créée

```
-git fetch origin, git checkout
```

Reprise du tag 3
Visualisez les différences via le *pom.xml* et via

```
./mvnw buildplan:list
```

Démarrer une analyse avec

```
./mvnw clean verify
```

Accéder aux résultats

Atelier 4 – Nexus : Dépôts d’artefacts

Démarrer un serveur Nexus via Docker

```
docker volume create --name nexus-data
docker run -d -p 8081:8081 --name nexus -v nexus-data:/nexus-data
sonatype/nexus3
```

Visualiser le mot de passe administrateur :

```
docker exec -it nexus /bin/bash
cat /nexus-data/admin.password
```

Se connecter en tant qu’administrateur (***admin***) :

- définir comme nouveau mot de passe ***admin123***
- *Autoriser les connexions anonymes*

Présentation des dépôts gérés par Nexus, en particulier ***maven-central, maven-public, maven-releases, maven-snapshot***

Reprise du tag 4

Effectuer un déploiement vers le dépôt des snapshots puis vers le dépôts de release

Atelier 5 – Jenkins : Plateforme CI/CD

5.1 Installation Jenkins

Télécharger une distribution .war de Jenkins
Démarrer le serveur dans un terminal via

```
java -jar jenkins.war --httpPort=8082
```

Connecter vous à *localhost:8082* et finaliser l'installation en installant les plugins suggérés

Visualiser le lien Administration Jenkins et les différents menus proposés :

- Configuration système
- Crédentiels
- Configuration des outils
- Gestion des plugins
- Nœuds Esclave et exécuteurs

Vous pouvez également installer les plugins suivants : ***Blue Ocean (Agregator), Performance, Docker, Docker pipeline***

5.2 Mise en place d'une pipeline CI

- Créer une merge request sur gitlab *« Mise en place CI »*
- Création dans Jenkins d'un job ***« Multi-branch Pipeline »*** et configurer les sources du job vers le dépôt Gitlab. Configuration du scan du dépôt chaque minute
- Dans l'environnement projet :
  - **Reprendre le tag 5.2**
  - Visualiser le fichier *JenkinsFile* et le compléter
  - Dans la branche de features, committer puis push
- Attendre que le pipeline s'exécute

Atelier 6 – Vagrant, Ansible : Gestion de configuration

6.1 Mise à disposition des serveur d'intégration via *vagrant*

Reprendre le tag 6.1

Visualiser le fichier ***Vagrantfile***, générer une paire clé-publique/clé-privé et l'ajouter dans le dossier

Démarrage des machines virtuelles Vagrant

```
vagrant up
vagrant ssh host1
ssh vagrant@192.168.99.2
ssh vagrant@192.168.99.3
```

6.2 Mise en place d'un playbook Ansible

Installer ansible

Reprendre le tag 6.2

Vérifier la connexion d'ansible aux 2 machines virtuelles

Visualiser les changements dans *pom.xml* en particulier le profil prod

Visualiser le playbook ***ansible/delivery.yml*** et essayer de compléter ce qu'il manque

Exécuter le playbook

```
cd ansible
ansible-playbook delivery.yml -i hosts
```

Vérifier le bon déploiement de l'application

Accès à l'appli via :

http://192.168.99.2:8080/swagger-ui.html

Reprendre le tag 6.3 et comparer avec votre solution

6.3 Pipeline CI : Déploiement vers serveurs d'intégration

Intégrer l'appel du playbook à la pipeline Jenkins, si la branche est différente de *master*

Exécution de la pipeline et observer le déploiement automatique du service

6.4 Tests post-déploiement sur environnement d'intégration

Reprendre le tag 6.4 et visualiser la pipeline Jenkins (ajout de 2 phases : Tests fonctionnels et tests de performance)

Exécuter la pipeline

Récupérer et visualiser le rapport de performance

Atelier 7 : Pipeline CD (Livraison continue)

Reprendre le tag 7

Observer les changements sur la pipeline.

Exécuter la pipeline sur la branche ***main***

Effectuer une release

Atelier 8 : Docker

8.1 Familiarisation docker, docker-compose
Quelques commandes docker

Reprendre le tag 8.1

Visualiser le fichier *src/main/docker/postgres-docker-compose.yml*

Démarrer la stack et créer une base de donnée via *pgAdmin*

8.2 Mis en place d'un *docker-compose* pour l'application en profil *prod*

Reprendre le tag 8.2
Visualiser les changements dans :

- *src/main/docker/docker-compose.yml*
- *src/main/resources/application.yml*
- *pom.xml*

Test du profil de *prod* dans l'environnement de Développement

Atelier 9 : Pipeline CI/CD avec stack docker-compose

9.1 Mise en place des dépôts Docker dans Nexus

Se logger sur Nexus en tant qu'admin et déclarer les dépôts Docker suivant :
- ***hosted*** : avec un connecteur http, 18082 et *« Allow anonymous docker pull »*
- ***proxy*** : avec adresse proxy https://registry-1.docker.io et Docker Index = Use DockerHub
- ***group*** : Regroupant les 2 précédents et avec un connecteur http 10000

Vous pouvez vous inspirer de : https://blog.sonatype.com/using-nexus-3-as-your-repository-part-3-docker-images

Autoriser le service docker à utiliser des dépôts en http :
Editer */etc/docker/daemon.json* avec :

```
{
    "insecure-registries" : [
     "localhost:18082",
     "localhost:10000"]
}
```

Relancer le conteneur nexus pour autoriser l'accès aux 2 nouveaux ports :

```
docker stop nexus
docker rm nexus
docker run -d -p 8081:8081 -p 10000:10000 -p 18082:18082 --name nexus
-v nexus-data:/nexus-data sonatype/nexus3
```

Vous pouvez tester votre configuration en poussant une image vers le dépôt.

```
docker pull hello-world
docker login -u admin -p admin123 localhost:18082
docker tag hello-world:latest localhost:18082/hello-world
docker push localhost:18082/hello-world
```

9.2 Intégration dans pipeline Jenkins

Intégrer le déploiement d'images Docker vers le registre nexus dans la pipeline.

Supprimer le déploiement via Ansible et effectuer les tests JMeter en démarrant une stack via *docker-compose*

Configurer Nexus pour autoriser les pull d'images anonymes :
Security > Realms, autoriser *"Docker Bearer Token Realm"*

Reprendre le tag 9.2

Comparer votre solution

Atelier 10 - Kubernetes

Démarrage *kubectl* et *minikube*

Accéder au dashboard

10.1 : Déploiements à partir d'une image

Créer un déploiement à partir d'une image docker
```
kubectl create deployment delivery-service --image=dthibau/delivery-service:0.0.1-SNAPSHOT
```
Exposer le déploiement via un service
```
kubectl expose deployment delivery-service --type LoadBalancer \
  --port 80 --target-port 8080
# Vérifier exécution des pods
kubectl get pods
# Accès aux logs
kubectl logs <pod_id>

kubectl get service delivery-service
#Forwarding de port
kubectl port-forward service/delivery-service 8080:80
```

Accès à l'application via l*ocalhost:8080*

Mise à jour du déploiement
```
kubectl set image deployment/delivery-service delivery-service=dthibau/delivery-service:0.0.3-SNAPSHOT
```

Statut du roll-out
```
kubectl rollout status deployment/delivery-service
```

Accès à l'application : *http:<IP>/actuator/info*

#Visualiser les déploiements
```
kubectl rollout history deployment/delivery-service
```

#Effectuer un roll-back
```
kubectl rollout undo deployment/delivery-service
```

```
#Scaling
kubectl scale deployment/delivery-service --replicas=5
```

10.2 : Déploiement d'une stack SB/Postgres

Reprendre le tag 10.2

Visualiser les fichiers du répertoire s***rc/main/k8***
Déployer la stack et accéder à l'application.

Pour accéder au service et voir le LoadBalancing

```
kubectl expose deployment delivery-service --type NodePort  --port 80
--target-port 8080
kubectl describe services delivery-service
kubectl cluster_info
```

http://<cluster-ip>:<NodePort>/actuator/env

Atelier 11 – Pipeline avec Kubernetes

Le but de cet atelier est d'avoir un environnement de recette par feature branch/MergeRequest

11.1 Création d'un compte sur DockerHub
Pour faciliter la récupération d'image par Kubernetes, nous utilisons le dépôt DockerHub.
Création d'un compte sur *DockerHub*
Configuration d'un crédentiel dans Jenkins
Manage Jenkins → Manage Credentials

11.2 ReviewApp pour une issue

Créer une issue « *Déploiement Kubernetes* », puis une merge request

Récupérer la branche en local et **reprendre le tag 11.1**

Visualiser les changements dans le JenkinsFile et dans les fichiers manifeste Kubernetes

Pousser les modifications et visualiser l'exécution de la pipeline

Accèder à l'environnement de review de branche

11.2. Mise à jour de la production

Effectuer le merge dans la branche main et déclencher une mise à jour de la production

Atelier 12 – Terraform

12.1 Prise en main terraform
Installation de Terraform et de kind

Reprendre le tag 12 et visualiser les nouveaux fichiers dans ***src/main/terraform-kind***

Le fichier terraform provisionne :

- Un cluster kind 3 nœuds
- 3 ressources kubernetes :
  - Le namespace formation
  - Le déploiemennt delivery-service
  - Le service delivery-service en précisant le NodePort

Se placer dans le répertoire puis

```
terraform init
terraform plan
terraform apply --auto-approve
```

Visualiser les pods démarrés

12.2 Utilisation de terraform dans la pipeline
Modifier le jenkinsfile pour déployer la branche main via terraform