

Ateliers

Usine Logicielle et Livraison Continue : Mise en œuvre

Pré-requis :

Poste développeur avec accès réseau Internet libre

Linux (Recommandé) ou Windows 10

Pré-installation de :

- JDK8+
- Git
- Docker
- Si possible : Oracle VirtualBox
- Si possible : Distribution minikube de Kubernetes

Manipulations pour visualiser les solutions :

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

```
./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 User Story ou Issues

Option1 : Utilisation Gitlab local

Démarrage *gitlab* via Docker :

```
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
```

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

Mise en place de 4 comptes Gitlab

Adminisrateur :

Owner / Maintener : Vous

Reporter : Un fonctionnel

Developpeur : Un développeur

Avec le compte Owner,

- Création d'un groupe de projet et avec affectation des membres 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

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

Avec le compte *owner/mainteneur*, mise au planning et affectation

Avec le compte *developer*, accès au tableau de bord et déplacement du post-it

To Do -> Doing

Option2 : Utilisation de gitlab.com

Créer vous un compte sur **gitlab.com**, un groupe de projet , puis un projet

Éventuellement, inviter d'autres stagiaires comme développeur sur votre projet

Définir des milestone

Saisir des issues

Mettre en place des tableaux de bord

Définir des labels (ex : DevOps, Bug, RFC, ...)

Configuration développeur

Configurer l'accès **ssh** sous le compte développeur en suivant les indications fournies par gitlab

Atelier 2.1 (Optionel) Commandes de base Git, merge et rebase

2.1.1 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
```

2.1.2 Création de dépôt et premier commit

Initialiser un dépôt
`mkdir delivery-service`
`git init`

Récupérer les sources fournis et les ajouter dans le dépôt :
`git add .`
`git commit -m 'Initial commit'`

2.1.3 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*

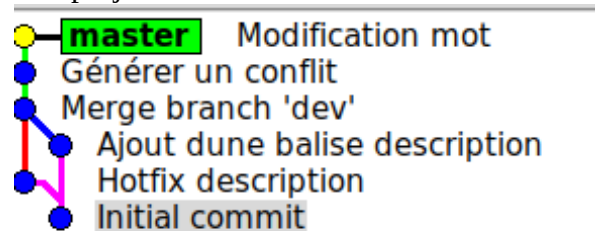
Supprimer la branche **dev**

2.1.4 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 :



Vous pouvez supprimer le dépôt à la fin de cet atelier

Atelier 2.2 : Workflow de collaboration *GitlabFlow*

1. 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 WIP et a pour effet de créer une branche portant le nom de l'issue

2. En tant que développeur sur son poste de travail, Récupérer la branche de la merge request, (*git clone + git checkout*)

Récupérer les sources fournies :

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. Effectuer un

git push

4. En tant que *developer* sur gitlab, supprimer le préfixe *WIP*

5. En tant que *owner/mainteneur*, faire une revue de code et ajouter comme commentaire : « Et les tests ? »

Reprise du tag 2.2

Exécuter les tests et s'assurer qu'ils passent :

`./mvnw test`

6. Push les modifications vers gitlab

7. En tant que *Owner/Mainteneur* faire une revue de code

8. Accepter le Merge Request et supprimer la branche

9. En local, en tant que développeur supprimer la branche locale et exécuter

git remote prune origin

Atelier 3 : 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`

Optionnel : Build reproductible :
`sudo adduser demodev`
`sudo su demodev`
`cd`
`git clone`
`./mvnw install`

3.2 Adaptation du build (Intégration de Sonar dans la phase *verify* de Maven)

Démarrage d'un serveur sonar :
`docker run -d --name sonarqube -p 9000:9000 sonarqube`

Connecter vous au serveur Sonar :
`localhost:9000 admin/admin`

Menu : *MyAccount* → *Security*

Générer un token, conserver le token dans un fichier

Workflow Gitlab

- Création d'une MergeRequest 'Mettre en place Sonar'
- Récupération de la branche en local
- `git fetch origin, git checkout`

Modification pom.xml

Ajouter 2 plugins :

- *jacoco* : Outil permettant de calculer la couverture des tests
- *sonar* : Permettant de démarrer une analyse Sonar

Le cycle Maven doit être modifié comme suit :

- Phase ***initialize***, démarrage de ***jacoco :prepare-agent***
- Phase ***test*** démarrage de ***jacoco:report***
- Phase ***prepare-package*** démarrage de ***sonar:sonar***

Faites vos tests et vérifiez avec
`./mvnw buildplan:list`

Démarrer une analyse avec
`./mvnw -Dsonar.login=<token-sonar> clean verify`

Comparer votre solution avec le tag 3

Reprise du tag 3

3.3 Mise en place de profil

Définit un profil de production qui a un build différent.

- Il utilise le plugin **git-commit-id-plugin** qui démarre l'objectif revision dans la phase **initialize**
- Il configure le plugin **spring-boot-maven-plugin** afin que l'objectif **build-info** soit exécuté en début de cycle et que le jar généré soit un exécutable
(Voir <https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/html/#goals-repackage>)

Comparer avec le tag 3.1

Atelier 4 – Nexus : Dépôts d'artefacts

Reprise du tag 4

Visualiser la configuration dans *settings.xml* et *pom.xml*

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
```

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

Modifier le mot de passe admin de Nexus afin qu'il corresponde à la config Maven

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

Atelier 5 – Tests

5.1 Tests unitaires, d'intégration

Visualisez les tests effectués. Le rapport de test. Comment les qualifierez vous ?

5.2 Tests fonctionnels et de performance

Récupérer le tag 5.2

Démarrer JMeter avec `./apache-jmeter-5.2.1/bin/jmeter &`

Visualisez les 2 scripts jmx fourni :

- **Fonctionnel.jmx** et en particulier les éléments d'assertions
- **Load.jmx**, le nombre d'utilisateur simulés

Démarrer l'application et exécuter les scripts vers *localhost:8080*

5.3 Tests d'acceptation

Récupérer le tag 5.3

Visualisez les nouvelles sources, en particulier :

- ***delivery-service.feature*** : Le test en langage pseudo-naturel, écrit par le métier
- ***StepDefsIntegrationTest.java*** : Le code glue fourni par le développeur
- ***SpringIntegrationTest*** : La couche d'intégration avec Spring

Exécuter le tests avec :

./mvnw test

Atelier 6 – Jenkins : Plateforme CI/CD

6.1 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

Création et exécution d'une pipeline via l'assistant, utiliser l'exemple déclaratif / Maven

6.2 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 6.2**
 - Visualiser le *JenkinsFile* et le compléter
 - Dans la branche de features, committer puis push
- Attendre que le pipeline s'exécute
- Éventuellement fixer vos erreurs

Reprendre le tag 6.3 et comparer avec votre solution

Optionnel : Faire en sorte que la pipeline Jenkins échoue si la porte qualité Sonar n'est pas franchie
Vous pouvez utiliser la fonction Groovy fournie qui nécessite également le plugin Jenkins ***Pipeline Utility Steps***

Reprendre le tag 6.4 et comparer avec votre solution

Atelier 7 – Vagrant, Ansible : Gestion de configuration

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

Reprendre le tag 7.1

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
```

```
ssh vagrant@192.168.99.2
```

```
ssh vagrant@192.168.99.3
```

7.2 Mise en place d'un playbook Ansible

Installer *Ansible*

Reprendre le tag 7.2

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 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 7.3 et comparer avec votre solution

7.3 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 bon déploiement de l'application

7.4 Tests post-déploiement

Intégrer les tests fonctionnels et de performance dans la pipeline

Exécuter la pipeline

Récupérer et visualiser le rapport de performance

Reprendre le tag 7.4 et comparer avec votre solution

Atelier 8 : Pipeline CD, Déployer une Release

Reprendre le tag 8

Observer les changements sur la pipeline.

Exécuter la pipeline sur la branche master

Effectuer une release

Voir également : <https://plugins.jenkins.io/scmskip/>

Atelier 9 : Docker

9.1 Familiarisation docker, *docker-compose*

Quelques commandes docker

Reprendre le tag 9.1

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

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

9.2 Mis en place d'un *docker-compose* pour l'application en profile prod

Travailler dans le répertoire ***src/main/docker***

Mettre au point un fichier *docker-compose.yml* permettant de démarrer la stack applicative dans le profil production

Reprendre le tag 9.2 , visualisez les changements et comparer avec votre solution

Atelier 10 : Pipeline CD avec image Docker

10.1 Mise en place des dépôts Docker

Voir : <https://blog.sonatype.com/using-nexus-3-as-your-repository-part-3-docker-images>

Déploiement latest et release

10.2 Intégration dans pipeline Jenkins

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

Reprendre le tag 10.2

10.3 Démarrage stack via *docker-compose*

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

Reprendre le tag 10.3

Comparer votre solution

10.4 Release image Docker

Modifier la pipeline pour publier une release de l'image Docker

Atelier 11 - Kubernetes

Démarrage *minikube* ou *microk8s*

Accès au dépôt privé Nexus : voir <https://kubernetes.io/fr/docs/tasks/configure-pod-container/pull-image-private-registry/>

```
kubectrl create secret generic regcred
--from-file=.dockerconfigjson=/home/dthibau/.docker/config.json
--type=kubernetes.io/dockerconfigjson
```

10.1 : Déploiements à partir d'une image

```
# Créer un déploiement à partir d'une image docker
kubectrl create deployment delivery-service
--image=nexus:18082/delivery-service:0.0.1-SNAPSHOT
# Exposer le déploiement via un service
kubectrl expose deployment delivery-service --type LoadBalancer \
--port 80 --target-port 8080
# Vérifier exécution des pods
kubectrl get pods
# Accès aux logs
kubectrl logs <pod_id>
```

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

Accès à l'application via localhost:8080

```
# Mise à jour du déploiement
kubectl set image deployment/delivery-service delivery-
service=nexus:18082/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
```

11.2 : Déploiement d'une stack SB/Postgres

Essayer d'écrire les fichiers ressources Kubernetes pour déployer la stack

Reprendre le tag 11.2 visualiser les fichiers du répertoire *src/main/k8* et comparer avec votre travail

Déployer la stack et accéder à l'application, l'utiliser

Atelier 12 – Pipeline avec Kubernetes

12.1 Un environnement par Merge Request

But avoir un environnement de recette par feature branch/MergeRequest

12.2. Roll-out de la production

Dans la branche master, mettre à jour un déploiement de minikube

Reprendre le tag 12 et comparer

Atelier 13 : Blue-green deployment

Reprendre le tag 13

Démarrer le projet et visualiser le endpoint */actuator/health*

Visualiser les manifestes Kubernetes fournis dans src/main/k8 ainsi que le script de déploiement .sh, exécuter un premier déploiement de la stack et vérifier son déploiement correct en accédant aux endpoints intéressant.

Effectuer un roll-out et vérifier le blue/green deployment

Atelier 14 : Canary deployment avec Istio

14.1 Installation

Attention, il est recommandé d'installer **kind** : distribution kubernetes plus performante que *minikube*, pour les ateliers suivants qui nécessitent beaucoup de containers

<https://kind.sigs.k8s.io/docs/user/quick-start/#installation>

Installation Istio :

Voir <https://istio.io/docs/setup/getting-started/#download>

14.2 Installation

Inspiré de <https://www.digitalocean.com/community/tutorials/how-to-do-canary-deployments-with-istio-and-kubernetes>

Créer un nouveau descripteur de déploiement pour *delivery-service*

- qui pointe vers un nouvelle image, par exemple ***dthibau/delivery-service:2.0.0***
- qui affecte un nouveau label ***version=v2***

Modifier le descripteur de déploiement existant de *delivery-service* en lui affectant un label ***version=1***

Définit une Gateway Istio comme suit :

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: store-gateway
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"

```

Définir une Virtual Service effectuant du routing et s'appuyant sur 2 ***subsets*** :

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:

```

```
name: store
spec:
  hosts:
  - "*"
  gateways:
  - store-gateway
  http:
  - match:
    - uri:
      prefix: /delivery-service/
    rewrite:
      uri: /
    route:
    - destination:
      host: delivery-service
      subset: v1
      weight: 80
    - destination:
      host: delivery-service
      subset: v2
      weight: 20
```

Ajouter également une ressource ***DestinationRule*** définissant les subsets :

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: delivery-service
spec:
  host: delivery-service
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

Déployer ces ressources

Accéder à des micro-services via la gateway :
[minikube](#)

```
export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')
```

```
export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].nodePort}')
```

```
export INGRESS_HOST=$(minikube ip)
```

Dans un autre terminal :

```
minikube tunnel
```

kind

```
export INGRESS_HOST=$(kubectl get po -l istio=ingressgateway -n istio-system -o jsonpath='{.items[0].status.hostIP}')
```

```
export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http2")].nodePort}')
```

```
export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].nodePort}')
```

Accéder à l'URL

[http://\\$INGRESS_HOST:\\$INGRESS_PORT/delivery-service/actuator/info](http://$INGRESS_HOST:$INGRESS_PORT/delivery-service/actuator/info)

Effectuer des reload, on doit voir accéder à différentes versions sur la même URL