École Mohammadia d'Ingénieurs

Génie Logiciel / DevOps

TP n°2 : Pipeline CI/CD multi-projets — Partie Tomcat (Diapos 1 à 6)

Déploiement Spring Boot sur Tomcat avec GitLab CI/CD (Version 1.0)

Élèves :
Sami FAOUZI

Encadré par :
Asmaa Retbi

2ème année Génie Informatique

1er décembre 2025

Table des matières

1 Contexte et objectif (jusqu'à la diapo 6)

L'objectif de cette première version est de mettre en place la chaîne minimale permettant :

- d'installer et démarrer Apache Tomcat localement ;
- d'adapter une application Spring Boot pour produire un fichier `.war` (déployable sur Tomcat) ;
- d'ajouter un **stage Deploy dans GitLab CI** et de lancer le pipeline ;
- de valider le déploiement par un test navigateur sur `localhost`.

2 Prérequis : installation et démarrage de Tomcat (Diapos 1–2)

2.1 Installation et structure du dossier Tomcat (capture attendue)

Tomcat a été installé localement. La capture suivante montre la **composition du dossier décompressé / installé**.

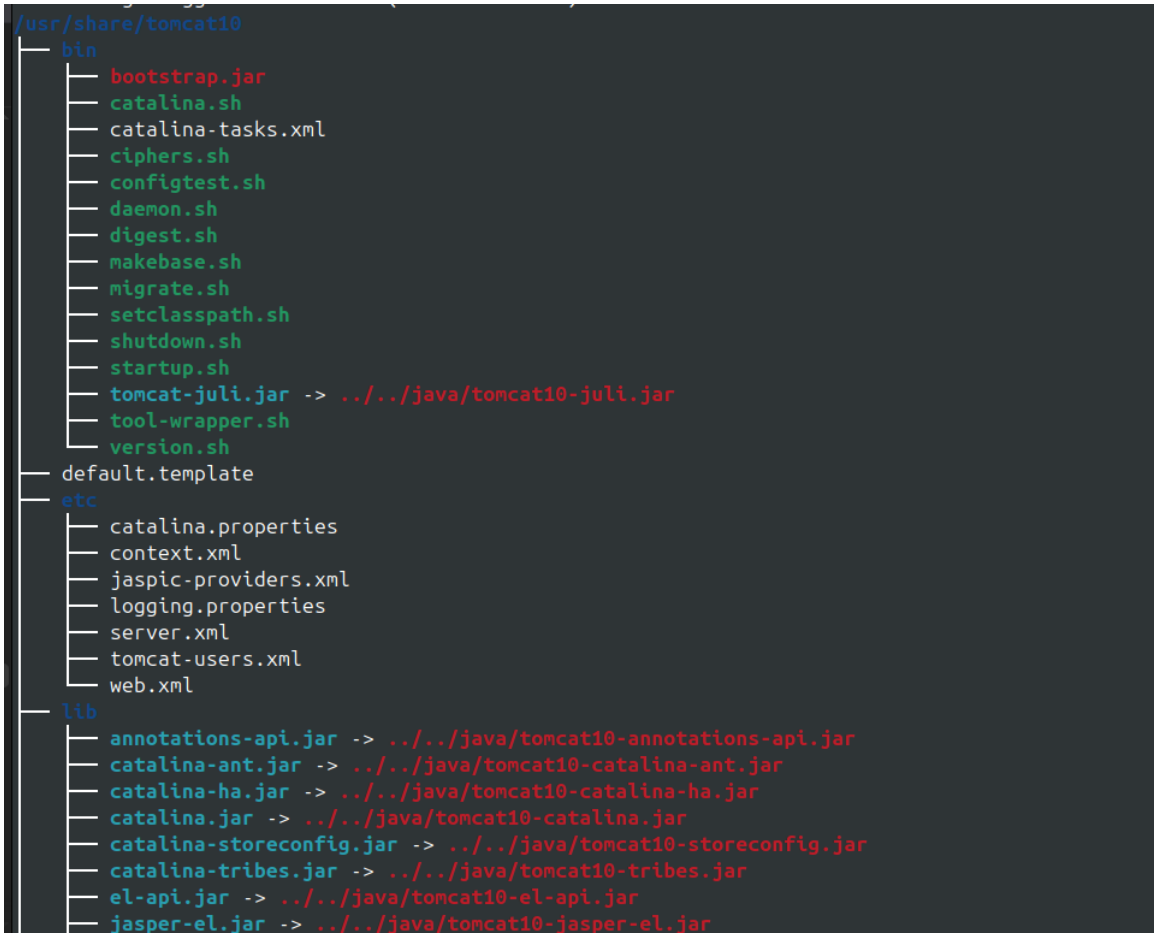

FIGURE 1 – Composition du dossier Tomcat (vue 1).

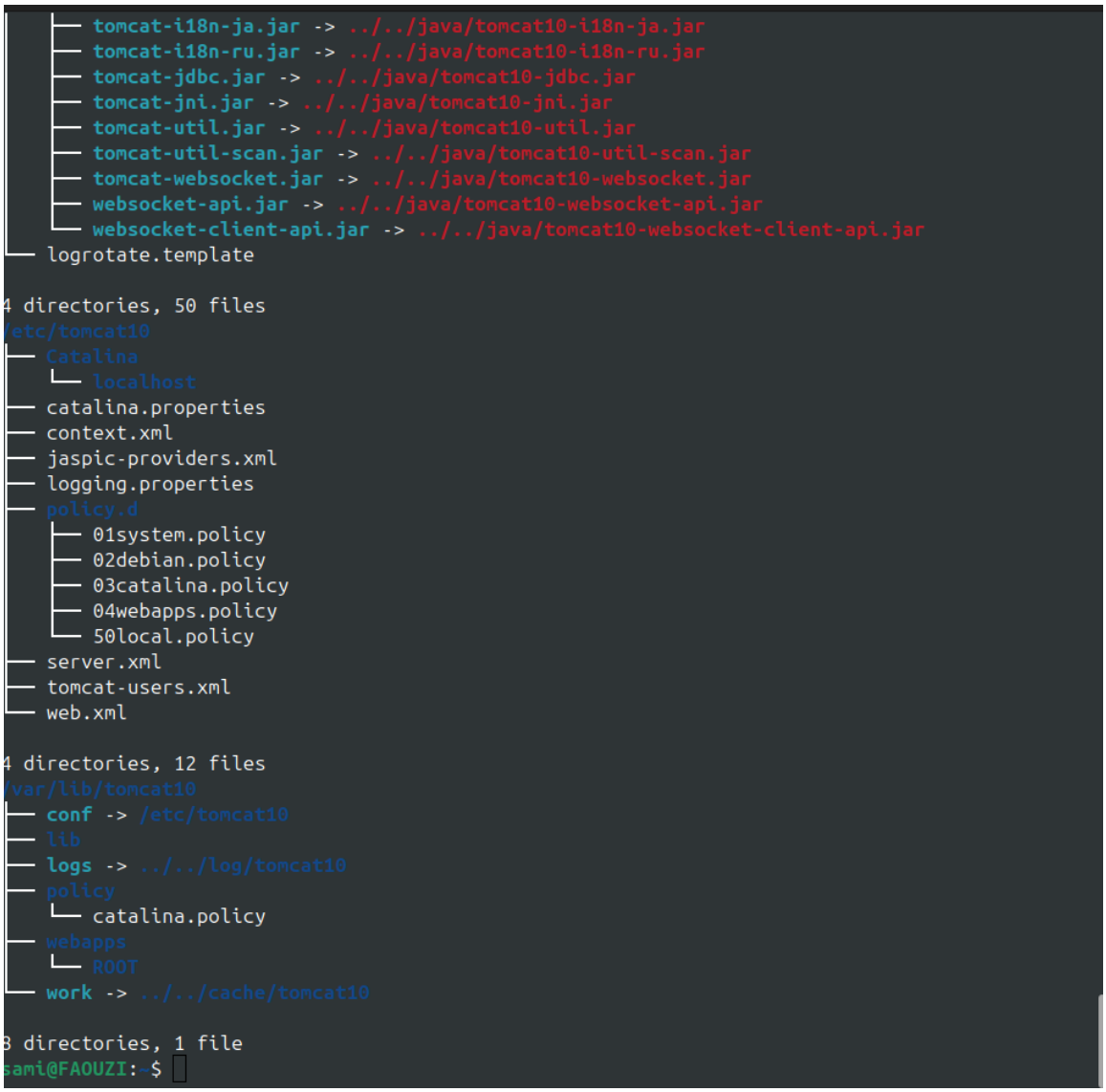

FIGURE 2 – Composition du dossier Tomcat (vue 2).

2.2 Démarrage et vérification (captures attendues)

Après installation, Tomcat est démarré en tant que service et son état est vérifié.

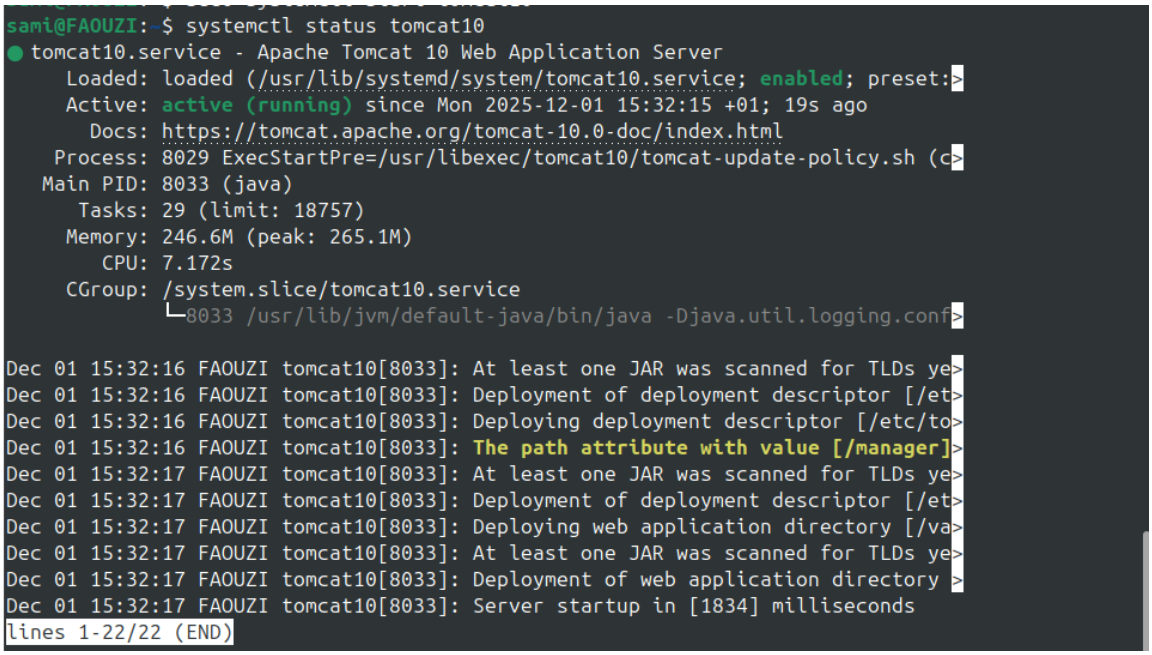

FIGURE 3 – Vérification côté terminal : Tomcat en cours d'exécution (*service started / active*).

Ensuite, un accès navigateur à `http://localhost:8081` confirme le bon fonctionnement.

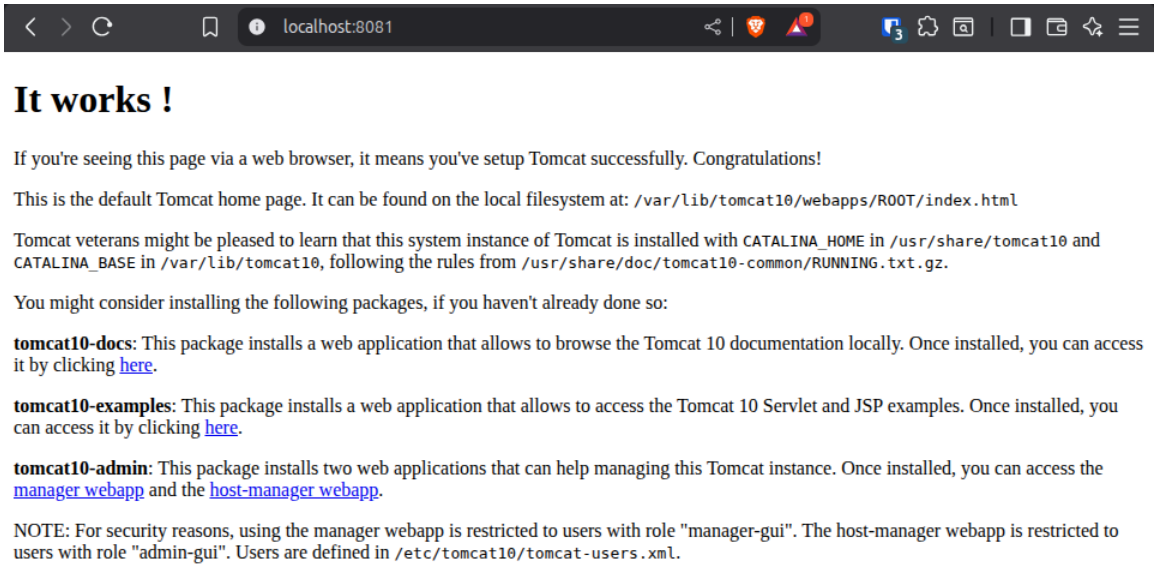

FIGURE 4 – Page d'accueil par défaut Tomcat (`localhost:8081`).

3 Configuration Spring Boot en WAR (Diapos 3.1–3.3 et 4)

3.1 Modification du `pom.xml` : packaging `war` (Diapo 3.1)

Le type de packaging a été modifié de `jar` vers `war` afin de produire un artefact déployable sur Tomcat.

```
GNU nano 7.2                                pom.xml *
        <packaging>war</packaging>

        <version>0.0.1-SNAPSHOT</version>
        <name>testapp</name>
        <description>Demo project for Spring Boot</description>
        <url/>
        <licenses>
                <license/>
        </licenses>
        <developers>
                <developer/>
        </developers>
        <scm>
                <connection/>
                <developerConnection/>
                <tag/>
                <url/>
        </scm>
        <properties>
                <java.version>17</java.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-webmvc</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-webmvc-test</artifactId>
                        <scope>test</scope>
                </dependency>
                <dependency>
                         <groupId>org.springframework.boot</groupId>
                         <artifactId>spring-boot-starter-tomcat</artifactId>
                         <scope>provided</scope>
                </dependency>
```

FIGURE 5 – Modification du `pom.xml` : `<packaging>war</packaging>` et dépendance Tomcat en `provided`.

3.2 Ajout de la dépendance Tomcat en `provided` (Diapo 3.2)

Pour un déploiement sur un Tomcat externe, la dépendance `spring-boot-starter-tomcat` est déclarée avec le scope `provided`, car elle sera fournie par le serveur Tomcat au moment de l'exécution.

3.3 Création de `ServletInitializer` (Diapo 3.3)

Une classe `ServletInitializer` est ajoutée pour permettre à Spring Boot de s'initialiser correctement dans un conteneur Servlet (Tomcat).

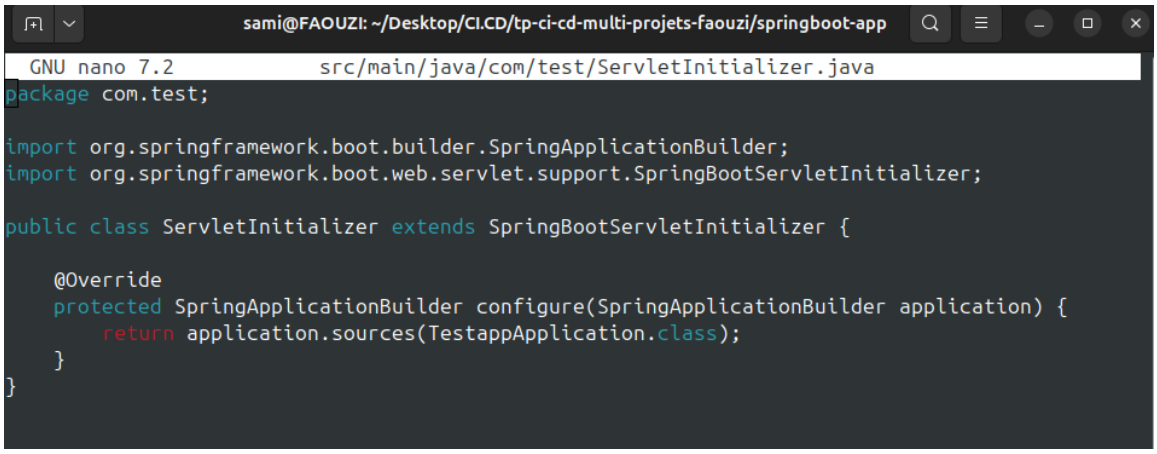

FIGURE 6 – Création de la classe `ServletInitializer` étendant `SpringBootServletInitializer`.

3.4 Génération du WAR et validation (Diapo 4)

La compilation Maven est exécutée avec `-DskipTests`. Deux captures sont attendues : **BUILD SUCCESS** et **présence du fichier** `.war`.

FIGURE 7 – Console Maven montrant **BUILD SUCCESS**.

```
sami@FAOUZI:~/Desktop/CI.CD/tp-ci-cd-multi-projets-faouzi/springboot-app$ ls -lh target/*.war
-rw-rw-r-- 1 sami sami 19M Dec  1 16:54 target/testapp-0.0.1-SNAPSHOT.war
sami@FAOUZI:~/Desktop/CI.CD/tp-ci-cd-multi-projets-faouzi/springboot-app$ ls -lh target
total 32M
drwxrwxr-x 3 sami sami 4.0K Dec  1 16:54 classes
drwxrwxr-x 3 sami sami 4.0K Dec  1 16:54 generated-sources
drwxrwxr-x 3 sami sami 4.0K Dec  1 16:54 generated-test-sources
drwxrwxr-x 2 sami sami 4.0K Dec  1 16:54 maven-archiver
drwxrwxr-x 3 sami sami 4.0K Dec  1 16:54 maven-status
drwxrwxr-x 4 sami sami 4.0K Dec  1 16:54 testapp-0.0.1-SNAPSHOT
-rw-rw-r-- 1 sami sami  19M Dec  1 16:54 testapp-0.0.1-SNAPSHOT.war
-rw-rw-r-- 1 sami sami  13M Dec  1 16:54 testapp-0.0.1-SNAPSHOT.war.original
drwxrwxr-x 3 sami sami 4.0K Dec  1 16:54 test-classes
sami@FAOUZI:~/Desktop/CI.CD/tp-ci-cd-multi-projets-faouzi/springboot-app$
```

FIGURE 8 – Vérification du fichier `.war` généré dans `target/`.

4 Ajout du stage Deploy sur GitLab (Diapo 5)

4.1 Principe

Le stage `deploy` est exécuté après le stage `build`. Dans notre cas, le déploiement cible **Tomcat local**, donc il nécessite un **runner de type Shell** sur la machine qui possède Tomcat (sinon, impossible d'accéder à `/var/lib/tomcat10` et `systemctl`).

4.2 Extrait du job `deploy-tomcat` (à ajouter en bas de `.gitlab-ci.yml`)

Ci-dessous, l'extrait correspondant à la diapo 5 (adapté Linux/Tomcat10 et runner local) :

```
stages:
  - build
  - test
  - deploy

build-springboot:
  stage: build
  image: maven:3.9.9-eclipse-temurin-21-alpine
  script:
    - cd springboot-app
    - mvn -B clean package -DskipTests
  artifacts:
    paths:
      - springboot-app/target/*.war

deploy-tomcat:
  stage: deploy
```

```
  needs:
    - job: build-springboot
      artifacts: true
  script:
    - echo "===== Début Deploy Tomcat ====="
    - ls -lh springboot-app/target/*.war
    - sudo cp springboot-app/target/*.war /var/lib/tomcat10/webapps/springboot-ci.war
    - sudo systemctl restart tomcat10
    - echo "===== Fin Deploy Tomcat ====="
  only:
    - main
  tags:
    - local-shell
```

4.3 Point critique (runner)

Pour que `deploy-tomcat` s'exécute, un **Project Runner** doit être enregistré sur GitLab avec le tag `local-shell` et un executor `shell`.

5 Exécution du pipeline GitLab CI et test localhost (Diapo 6)

5.1 Commit & Push (déclenchement pipeline)

Après modification du `.gitlab-ci.yml`, les commandes suivantes déclenchent le pipeline :

```
git add .
git commit -m "add deploy stage for tomcat"
git push
```

5.2 Capture GitLab attendue : pipeline SUCCESS

La capture attendue est : `CI/CD > Pipelines` montrant le pipeline **vert** (build/test/deploy).

Capture à ajouter : GitLab `CI/CD > Pipelines` avec pipeline **SUCCESS** (jobs verts).
Nom conseillé : $\texttt{pipeline}_{s}success_{g}itlab.png$

FIGURE 9 – Pipeline GitLab CI/CD en succès (build/test/deploy).

5.3 Test navigateur attendu : endpoint déployé sur Tomcat

Après succès du deploy, l'application est testée sur le navigateur via Tomcat :

— URL : `http://localhost:8081/springboot-ci/hello`
— (si l'endpoint diffère, utiliser l'URL réellement implémentée)

Capture à ajouter : navigateur sur `http://localhost:8081/springboot-ci/hello`
Nom conseillé : $\texttt{test}_h ello_l ocalhost.png$

FIGURE 10 – Test navigateur confirmant le déploiement sur Tomcat.

6 Conclusion (version 1.0)

Cette version 1.0 couvre l'avancement jusqu'à la diapo 6 :
— Tomcat installé et opérationnel (`localhost:8081`) ;
— application Spring Boot configurée pour produire un `.war` (packaging et `ServletInitializer`) ;
— ajout du stage `deploy` dans GitLab et préparation de l'exécution du pipeline ;
— captures restantes à compléter : **pipeline GitLab SUCCESS** et **test /hello**.