



## ÉCOLE MOHAMMADIA D'INGÉNIEURS

CI/CD — DEVOPS  
SONARCLOUD + MAVEN/GITLAB CI (AVEC PIPELINE  
MULTI-PROJETS)

---

# Atelier 4 — Analyse Qualité (SonarCloud) & Intégration GitLab CI/CD

---

*Elèves :*

Sami FAOUZI

*Encadré par :*

Pr. ASMAA RETBI

2ème année Génie Informatique  
14 décembre 2025

## Table des matières

|  |           |
|--|-----------|
| <b>1 SonarCloud — Rappel (Diapo 55)</b>  | <b>3</b>  |
| <b>2 Pré-requis (optionnel) — Plugin SonarQube (capture)</b>                   | <b>3</b>  |
| <b>3 Création du compte SonarCloud (depuis GitLab) — Diapo 57</b>              | <b>3</b>  |
| 3.1 Connexion via GitLab . . . . .   | 3         |
| 3.2 Tableau de bord après connexion . . . . .                                  | 4         |
| <b>4 Organisation SonarCloud (depuis un groupe GitLab) — Diapo 59</b>          | <b>5</b>  |
| 4.1 Création du Personal Access Token GitLab (scope api) . . . . .             | 5         |
| 4.2 Création de l'organisation (Group ID + token) . . . . .                    | 6         |
| 4.3 Organisation créée et connectée . . . . .                                  | 6         |
| <b>5 Importer le projet Spring Boot existant — Diapo 60</b>                    | <b>7</b>  |
| 5.1 Liste des projets GitLab visibles dans SonarCloud . . . . .                | 7         |
| 5.2 Projet Spring Boot sélectionné . . . . .                                   | 7         |
| <b>6 Configuration SonarCloud (Maven/GitLab CI) — Diapos 61 &amp; 62</b>       | <b>8</b>  |
| 6.1 Variables CI/CD à ajouter dans GitLab . . . . .                            | 8         |
| 6.2 Bloc YAML proposé par l'assistant SonarCloud . . . . .                     | 10        |
| <b>7 Adapter le fichier <code>.gitlab-ci.yml</code> — Diapo 63</b>             | <b>10</b> |
| 7.1 Fichier <code>.gitlab-ci.yml</code> (version utilisée) . . . . .           | 10        |
| 7.2 Capture du fichier modifié dans GitLab . . . . .                           | 13        |
| <b>8 Modification du code — Projet Spring Boot (Diapo 58)</b>                  | <b>14</b> |
| 8.1 Classe Java volontairement “sale” (code smells) . . . . .                  | 14        |
| <b>9 Lancer l'analyse SonarCloud — Diapo 64</b>                                | <b>15</b> |
| 9.1 Pipeline GitLab incluant le stage Sonar . . . . .                          | 15        |
| <b>10 Jenkins &amp; SonarCloud — Intégration dans Jenkins (Diapos 65 → 69)</b> | <b>15</b> |
| 10.1 Étape 7 — Installation du plugin SonarQube dans Jenkins . . . . .         | 15        |
| 10.2 Étape 8 — Configurer SonarCloud dans Jenkins . . . . .                    | 16        |
| 10.2.1 Création du credential SonarCloud (Secret text) . . . . .               | 16        |
| 10.2.2 Déclaration du serveur SonarCloud (SonarQube servers) . . . . .         | 16        |
| 10.3 Étape 9 — Ajouter un stage SonarCloud dans le Jenkinsfile . . . . .       | 17        |
| 10.4 Étape 10 — Lancer le pipeline Jenkins . . . . .                           | 19        |
| 10.4.1 Log du stage SonarCloud (analyse réussie) . . . . .                     | 20        |
| 10.5 Étape 11 — Analyse des résultats (SonarCloud) . . . . .                   | 21        |
| 10.6 Comparaison synthétique : GitLab CI vs Jenkins (même analyse SonarCloud)  | 21        |

**11 Annexe — Rappel variables GitLab attendues** **22**

## 1 SonarCloud — Rappel (Diapo 55)

SonarCloud analyse automatiquement la **qualité** et la **sécurité** du code :

- Bugs / Vulnérabilités
- Code Smells / Duplications
- Couverture de tests / Complexité

## 2 Pré-requis (optionnel) — Plugin SonarQube (capture)

Capture demandée dans certains supports : plugin SonarQube sélectionné.



FIGURE 1 – Plugin SonarQube (pré-requis selon configuration Jenkins).

## 3 Création du compte SonarCloud (depuis GitLab) — Diapo 57

### 3.1 Connexion via GitLab

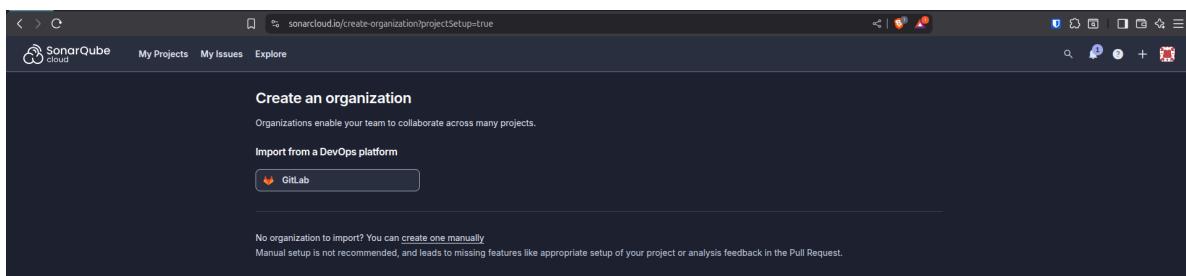


FIGURE 2 – Page de connexion SonarCloud avec l'option GitLab.

### 3.2 Tableau de bord après connexion

The screenshot shows the SonarCloud interface after logging in. The top navigation bar includes links for SonarQube, My Projects, My Issues (which is the active tab), and Explore. On the left, there's a sidebar with various filtering options:

- Filters**:
  - Software quality: Security (0), Reliability (0), Maintainability (0)
  - Severity: Blocker (0), High (0), Medium (0), Low (0), Info (0)
  - Code attribute
  - Type
  - Type Severity
  - Status
  - Security Category
  - Creation Date

The main content area has a header "Select issues" with navigation arrows and a status bar showing "0 issues 0 effort". A central message states: "There are no issues assigned to you."

FIGURE 3 – Tableau de bord SonarCloud (après connexion).

## 4 Organisation SonarCloud (depuis un groupe GitLab) — Diapo 59

### 4.1 Crédit du Personal Access Token GitLab (scope api)

**Personal access tokens**

Add new token

You can generate a personal access token for each application you use that needs access to the GitLab API. You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Token name  
sonarcloud-token

Description (optional)

Expiration date  
2026-01-07

An administrator has set the maximum expiration date to 2026-12-08. Learn more.

Select scopes

Scopes set the permission levels granted to the token. Learn more.

**read\_user**  
Grants read-only access to your profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

**read\_repository**  
Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.

**read\_virtual\_registry**  
Grants read-only access to container images through the dependency proxy in private projects and virtual registries.

**read\_registry**  
Grants read-only access to container registry images on private projects.

**read\_api**  
Grants read access to the API, including all groups and projects, the container registry, and the package registry.

**self\_rotate**  
Grants permission for token to rotate itself.

**write\_repository**  
Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).

**write\_virtual\_registry**  
Grants read, write, and delete access to container images through the dependency proxy in private projects.

**write\_registry**  
Grants write access to container registry images on private projects. You need both read and write access to push images.

**api**  
Grants complete read/write access to the API, including all groups and projects, the container registry, the dependency proxy, and the package registry.

FIGURE 4 – Crédit du token GitLab (scope api) utilisé pour l’organisation SonarCloud.

## 4.2 Création de l'organisation (Group ID + token)

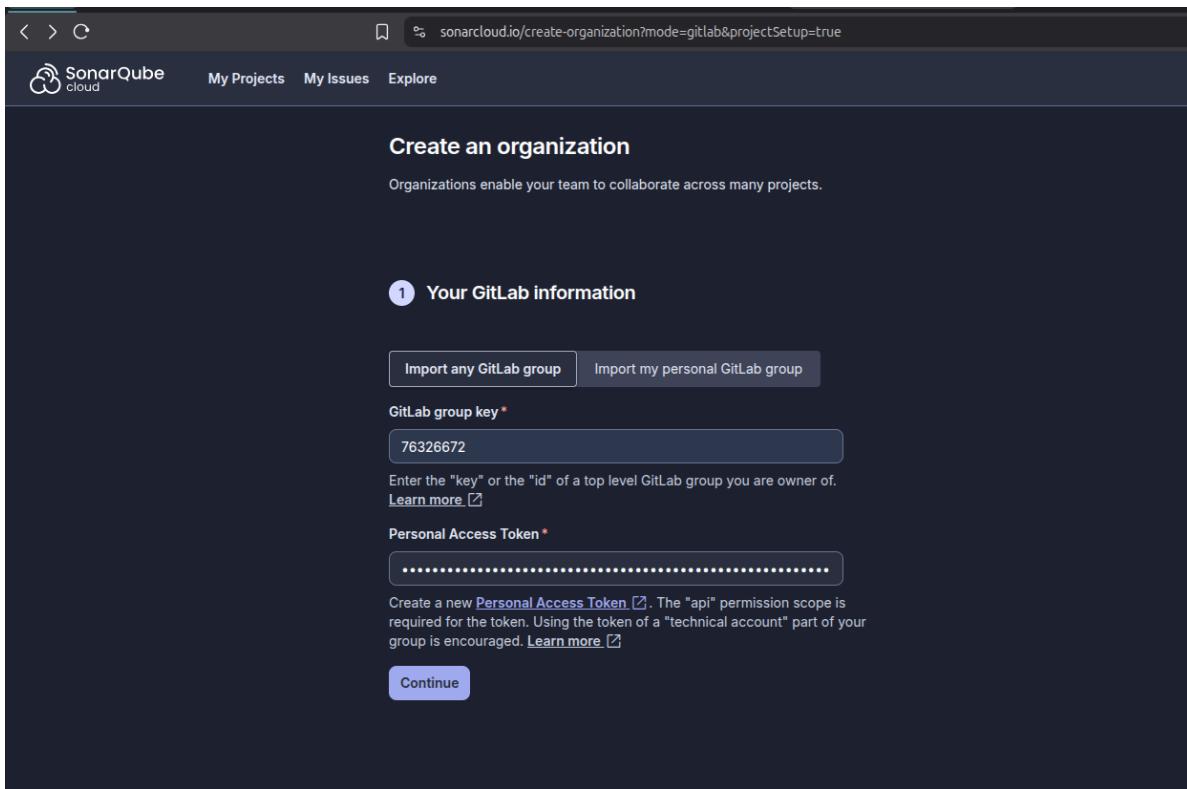


FIGURE 5 – Création de l'organisation SonarCloud liée au groupe GitLab.

## 4.3 Organisation créée et connectée

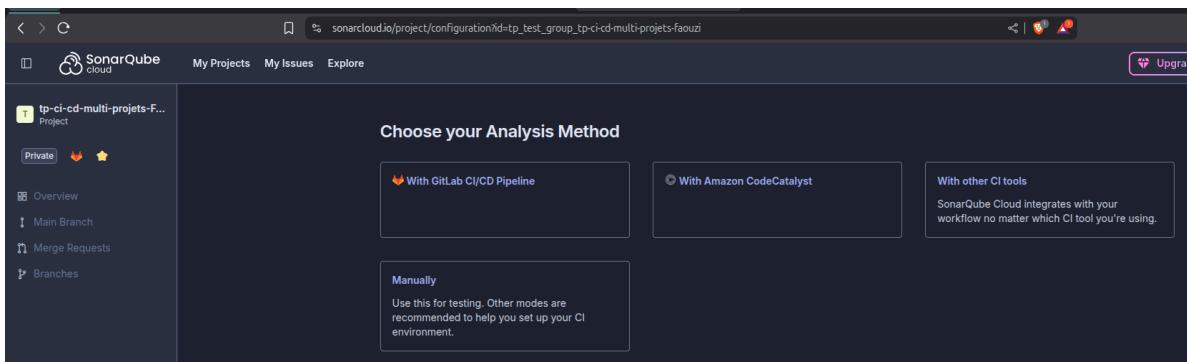


FIGURE 6 – Organisation SonarCloud créée et reliée à GitLab.

## 5 Importer le projet Spring Boot existant — Diapo 60

### 5.1 Liste des projets GitLab visibles dans SonarCloud

The screenshot shows the SonarCloud interface for selecting projects to analyze. It has a dark theme with a header bar at the top. Below the header, there's a section titled 'Analyze projects' with the sub-instruction 'Select repositories from one of your organization'. A dropdown menu is open under 'Organization' set to 'Tp\_test\_group'. There are two sections of project lists:

- Import another organization:** Contains a checkbox 'Select all on this page' and a list with one item: 'tp-ci-cd-FAOUZI [Private]'. This item has a red heart icon.
- Already imported:** Contains a checked checkbox 'tp-ci-cd-multi-projets-FAOUZI [Private]' with a red heart icon and the status 'Already imported'.

On the right side of the page, there are promotional banners for 'Already have a coupon?' and 'Just testing? You can create a project manually.' with a link to 'Setup a monorepo.'

FIGURE 7 – Liste des projets GitLab détectés dans SonarCloud.

### 5.2 Projet Spring Boot sélectionné

This screenshot shows the detailed view of the selected project 'tp-ci-cd-multi-projets-FAOUZI' on SonarCloud. At the top, it displays the project name, a 'Free' badge, and a 'Code' button. Below the header, there's a commit history table:

| Name               | Last commit                            | Last update |
|--------------------|--|-------------|
| dotnet-app         | add deploy stage                       | 6 days ago  |
| nodejs-app         | add deploy stage                       | 6 days ago  |
| springboot-app     | add deploy stage                       | 6 days ago  |
| springboot-app_old | add deploy stage                       | 6 days ago  |
| .gitignore         | add deploy stage                       | 6 days ago  |
| .gitlab-ci.yml     | Ajout de deploy to .gitlab-ci.yml file | 1 week ago  |
| README.md          | add deploy stage                       | 6 days ago  |

The commit 'Ajout de deploy to .gitlab-ci.yml file' by 'Sami Faouzi' is highlighted with a green box. The entire screenshot is framed by a green border.

FIGURE 8 – Sélection du projet Spring Boot à analyser dans SonarCloud.

## 6 Configuration SonarCloud (Maven/GitLab CI) — Diapos 61 & 62

### 6.1 Variables CI/CD à ajouter dans GitLab

The screenshot shows a dark-themed interface for SonarQube Cloud. At the top, there's a navigation bar with 'SonarQube cloud', 'My Projects', 'My Issues', 'Explore', and a user icon. Below the navigation, the URL is shown as 'Tp\_test\_group > tp-ci-cd-multi-projets-FAOUZI > Administration / Analysis Method > Analyze a project with GitLab CI/CD Pipeline'. The main content area has a title 'Analyze a project with GitLab CI/CD Pipeline'. It starts with a section '1 Add environment variables' which is further divided into two parts: 'a. Define the SonarQube Cloud Token environment variable' and 'b. Define the SonarQube Cloud URL environment variable'. Each part contains four numbered steps with corresponding icons: 1. In the Key field, enter SONAR\_TOKEN (with a copy icon). 2. In the Value field, enter 58867259337c9832546facd0d349cfce6cd85bb5 (with a copy icon and a pencil icon). 3. Make sure that the Protect variable checkbox is unticked. 4. Make sure that the Mask variable checkbox is ticked. Part 'b' also includes a note: 'Still in Settings > CI/CD > Variables add a new variable and make sure it is available for your project:'. The steps for 'b' are identical to part 'a'.

FIGURE 9 – Variables recommandées par SonarCloud : [SONAR\\_TOKEN](#), [SONAR\\_HOST\\_URL](#), [SONAR\\_ORGANIZATION](#).

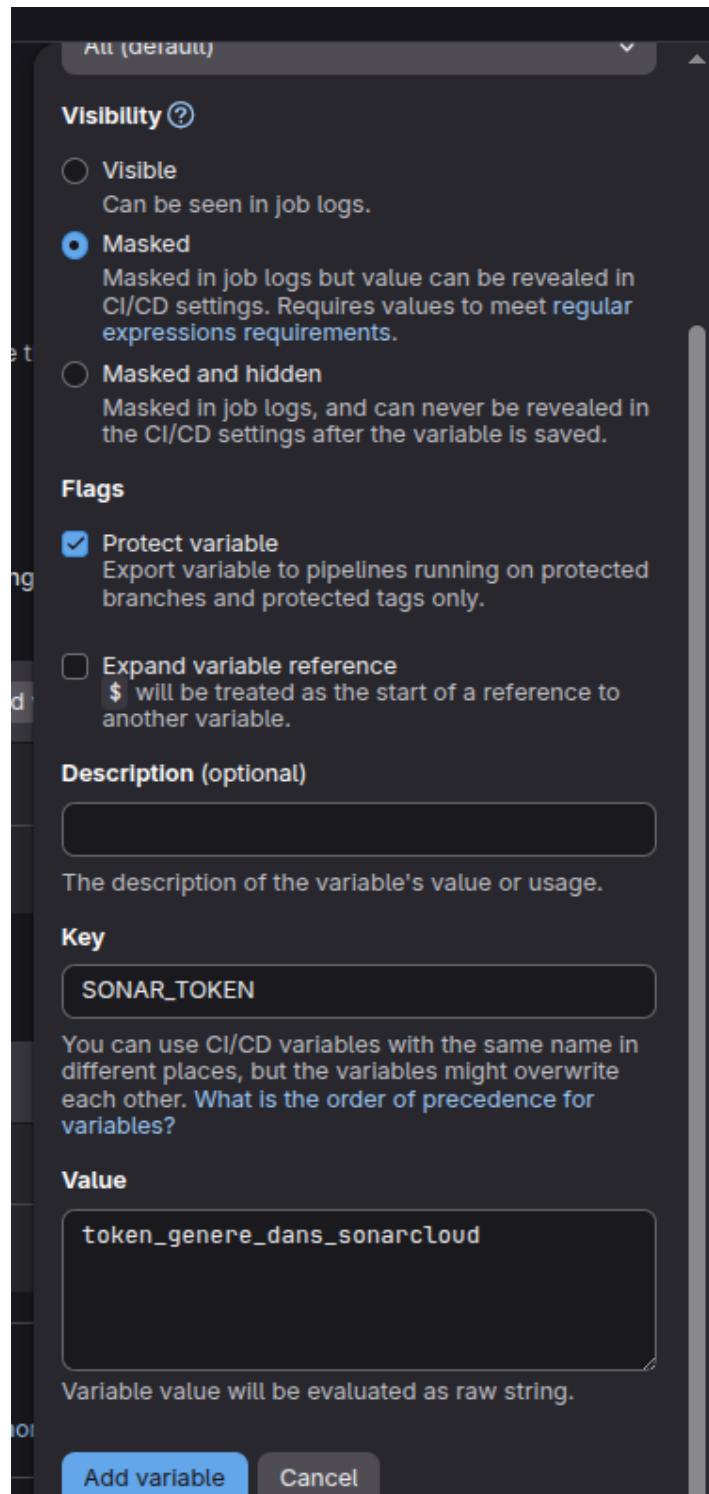


FIGURE 10 – Configuration des variables CI/CD dans GitLab (SONAR\_TOKEN, SONAR\_HOST\_URL, SONAR\_ORGANIZATION).

## 6.2 Bloc YAML proposé par l'assistant SonarCloud

The screenshot shows the SonarCloud interface with the following details:

- Header:** SonarQube cloud, My Projects, My Issues, Explore, Upg
- Breadcrumb:** Tp\_test\_group > tp-ci-cd-multi-projets-FAOUIZI > Administration / Analysis Method > Analyze a project with GitLab CI/CD Pipeline
- Section:** Create or update a `.gitlab-ci.yml` file
- Question:** What option best describes your project? (Maven selected)
- Text:** Update your `pom.xml` file with the following properties:

```
<properties>
  <sonar.organization>tp-test-group</sonar.organization>
</properties>
```

- Text:** Create or update your `.gitlab-ci.yml` yaml file with the following content:

```
variables:
  SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar" # Defines the location of the analysis task cache
  GIT_DEPTH: "0" # Tells git to fetch all the branches of the project, required by the analysis task
sonarcloud-check:
  image: maven:3-openjdk-17
  cache:
    key: "${CI_JOB_NAME}"
    paths:
      - .sonar/cache
  script:
    - mvn verify sonar:sonar -Dsonar.projectKey=tp_test_group_tp-ci-cd-multi-projets-faouzi
only:
  - merge_requests
  - master
  - develop
```

FIGURE 11 – Bloc YAML proposé par SonarCloud (Maven/GitLab CI).

## 7 Adapter le fichier `.gitlab-ci.yml` — Diapo 63

### 7.1 Fichier `.gitlab-ci.yml` (version utilisée)

Le fichier suivant correspond au format demandé (stages build/test/sonar/deploy) et évite l'erreur Maven `No plugin found for prefix '-Dsonar.host.url=...'` en utilisant un bloc YAML plié >.

Listing 1 – Fichier `.gitlab-ci.yml` (pipeline multi-projets + SonarCloud + déploiement Tomcat)

```
stages:
  - build
  - test
  - sonar
  - deploy
```

```

# ----- JOBS DE BUILD -----

build-springboot:
    stage: build
    image: maven:3.9-eclipse-temurin-17
    script:
        - echo "=====Dbut du build Spring Boot====="
        - cd springboot-app
        - mvn -B clean package -DskipTests
        - echo "=====Fin du build Spring Boot====="
    artifacts:
        paths:
            - springboot-app/target/*.war
        expire_in: 1 week

build-nodejs:
    stage: build
    image: node:20-alpine
    script:
        - echo "=====Dbut du build Node.js====="
        - cd nodejs-app
        - npm install
        - npm run build
        - echo "=====Fin du build Node.js====="

build-dotnet:
    stage: build
    image: mcr.microsoft.com/dotnet/sdk:8.0
    script:
        - echo "=====Dbut du build .NET====="
        - cd dotnet-app/demoapp
        - dotnet restore
        - dotnet build --configuration Release
        - echo "=====Fin du build .NET====="

# ----- JOBS DE TEST -----

test-springboot:
    stage: test
    image: maven:3.9.9-eclipse-temurin-21-alpine
    script:
        - echo "----Dbut des tests Spring Boot----"

```

```

- cd springboot-app
- mvn test
- echo "----_Fin_des_tests_Spring_Boot_----"

test-nodejs:
  stage: test
  image: node:20-alpine
  script:
    - echo "----_Dbut_des_tests_Node.js_----"
    - cd nodejs-app
    - npm test
    - echo "----_Fin_des_tests_Node.js_----"

test-dotnet:
  stage: test
  image: mcr.microsoft.com/dotnet/sdk:8.0
  script:
    - echo "----_Dbut_des_tests_.NET_----"
    - cd dotnet-app/demoapp
    - dotnet test
    - echo "----_Fin_des_tests_.NET_----"

# ----- JOB SONARCLOUD (MAVEN) -----

variables:
  SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar"
  GIT_DEPTH: "0"

sonarcloud-check:
  stage: sonar
  image: maven:3.9.9-eclipse-temurin-17
  cache:
    key: "${CI_JOB_NAME}"
    paths:
      - .sonar/cache
  script:
    - echo "====_Lancement_de_l'analyse_SonarCloud_===="
    - mvn -f springboot-app/pom.xml -B
      -DskipTests
      -Dsonar.projectKey=tp_test_group_tp-ci-cd-multi-projets-faouzi
      -Dsonar.organization=$SONAR_ORGANIZATION
      -Dsonar.host.url=$SONAR_HOST_URL

```

```

-Dsonar.token=$SONAR_TOKEN
clean verify sonar:sonar
- echo "=====Fin de l'analyse SonarCloud====="
only:
- main

# ----- DEPLOYMENT TOMCAT -----

deploy-tomcat:
stage: deploy
needs:
- job: build-springboot
  artifacts: true
script:
- echo "=====Dbut 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

```

## 7.2 Capture du fichier modifié dans GitLab

The screenshot shows the GitLab CI/CD Catalog interface. At the top, there are tabs for 'Edit', 'Visualize', 'Validate', 'NEW', and 'Full configuration'. Below the tabs, the title 'CI/CD Catalog' is visible. The main area contains the content of the .gitlab-ci.yml file:

```

stages:
- build
- test
- sonar
- deploy
# ----- JOB SONARCLOUD (NOUVEAU STAGE) -----
variables:
SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar"
GIT_DEPTH: "7" # Sonarcloud a besoin de tout l'historique Git
sonarcloud-check:
stage: sonar
image: maven:3-openjdk-17
cache:
key: "${CI_JOB_NAME}"
path:
.sonar/cache
script:
# IMPORTANT : on cible le pom du projet Spring Boot dans le sous-dossier springboot-app
- mvn -f springboot-app/pom.xml -B verify sonar:sonar -DskipTests -Dsonar.projectKey=tp_tp_ci_cd-multi-projets-faouzi
only:
- main
- merge_requests

```

At the bottom of the code editor, there are two buttons: 'Commit message' and 'Update .gitlab-ci.yml file'.

FIGURE 12 – Capture du fichier `.gitlab-ci.yml` modifié (GitLab).

## 8 Modification du code — Projet Spring Boot (Diapo 58)

### 8.1 Classe Java volontairement “sale” (code smells)

Création d'une classe avec :

- un champ non utilisé ;
- une duplication volontaire ;
- aucun test unitaire associé à cette classe.

Listing 2 – BadQualityService.java (champ non utilisé + duplication)

```
package com.test;

// Classe volontairement "sale" pour SonarCloud
public class BadQualityService {

    // Champ non utilis
    private String unusedField;

    public int computeScoreV1(int a, int b) {
        int sum = a + b;
        if (sum > 10) {
            return sum;
        } else {
            return sum + 10;
        }
    }

    // Mthode quasi-identique => duplication volontaire
    public int computeScoreV2(int a, int b) {
        int sum = a + b;
        if (sum > 10) {
            return sum;
        } else {
            return sum + 10;
        }
    }
}
```

## 9 Lancer l'analyse SonarCloud — Diapo 64

### 9.1 Pipeline GitLab incluant le stage Sonar

| Status                                    | Job  | Pipeline                      | Coverage        |
|---|--|-------------------------------|-----------------|
| <span style="color: green;">Passed</span> | #12379872740: deploy-tomcat<br>Y main -> 809825f1<br>local-shell | #2205550470 created by [User] | [Coverage icon] |
| <span style="color: green;">Passed</span> | #12379872738: sonarcloud-check<br>Y main -> 809825f1             | #2205550470 created by [User] | [Coverage icon] |
| <span style="color: green;">Passed</span> | #12379872735: test-dotnet<br>Y main -> 809825f1                  | #2205550470 created by [User] | [Coverage icon] |
| <span style="color: green;">Passed</span> | #12379872734: test-nodejs<br>Y main -> 809825f1                  | #2205550470 created by [User] | [Coverage icon] |
| <span style="color: green;">Passed</span> | #12379872733: test-springboot<br>Y main -> 809825f1              | #2205550470 created by [User] | [Coverage icon] |
| <span style="color: green;">Passed</span> | #12379872730: build-dotnet<br>Y main -> 809825f1                 | #2205550470 created by [User] | [Coverage icon] |
| <span style="color: green;">Passed</span> | #12379872728: build-nodejs<br>Y main -> 809825f1                 | #2205550470 created by [User] | [Coverage icon] |
| <span style="color: green;">Passed</span> | #12379872726: build-springboot<br>Y main -> 809825f1             | #2205550470 created by [User] | [Coverage icon] |

FIGURE 13 – Connexion SonarCloud réussie via GitLab.

## 10 Jenkins & SonarCloud — Intégration dans Jenkins (Diapos 65 → 69)

### 10.1 Étape 7 — Installation du plugin SonarQube dans Jenkins

Installation du plugin **SonarQube Scanner for Jenkins** via : [Manage Jenkins](#) → [Plugins](#) → recherche puis installation.

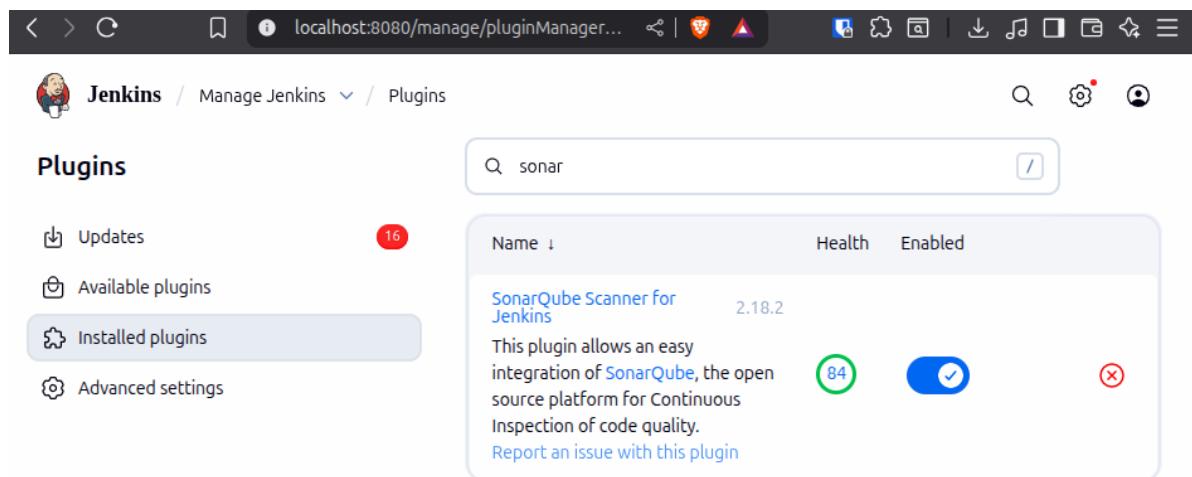


FIGURE 14 – Installation du plugin SonarQube Scanner dans Jenkins (installation réussie).

## 10.2 Étape 8 — Configurer SonarCloud dans Jenkins

### 10.2.1 Crédentiel SonarCloud (Secret text)

Ajout du token SonarCloud en tant que credential **Secret text** dans : [Manage Jenkins](#) → [Credentials](#) → [Global](#) → [Add Credentials](#).

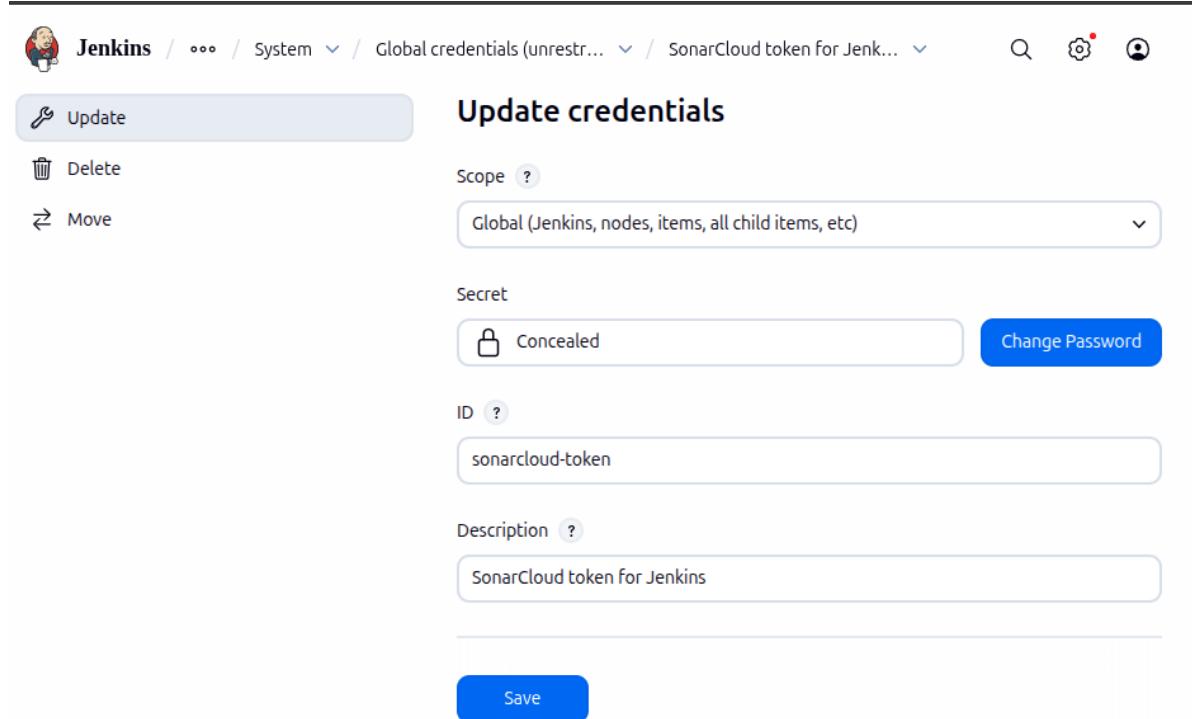


FIGURE 15 – Credential SonarCloud (Secret text) créé dans Jenkins.

### 10.2.2 Déclaration du serveur SonarCloud (SonarQube servers)

Configuration dans : [Manage Jenkins](#) → [Configure System](#) → [SonarQube servers](#).

### SonarQube servers

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

 Environment variables

#### SonarQube installations

[List of SonarQube installations](#)

**Name** ×

**Server URL**  
Default is <http://localhost:9000>

**Server authentication token**  
SonarQube authentication token. Mandatory when anonymous access is disabled.  

▼ + Add

Advanced ▼

FIGURE 16 – Section *SonarQube servers* remplie : Name = [SonarCloudServer](#), URL = <https://sonarcloud.io>, token associé.

### 10.3 Étape 9 — Ajouter un stage SonarCloud dans le Jenkinsfile

Le Jenkinsfile local a été modifié pour ajouter un stage **SonarCloud Analysis**. L'extrait ci-dessous correspond au stage d'analyse (exécuté dans le dossier [springboot-app](#)).

Listing 3 – Extrait du Jenkinsfile : stage SonarCloud Analysis

```
pipeline {
    agent any
    options { timestamps() }

    environment {
        SONAR_INSTANCE = 'SonarCloudServer'
        SONAR_ORG = 'tp-test-group'
        SONAR_PROJECT_KEY = 'tp_test_group_tp-ci-cd-multi-projets-faouzi'
        APP_DIR = 'springboot-app'
    }

    stages {
        stage('Checkout') {
            steps { checkout scm }
        }
    }
}
```

```

stage('Build') {
    steps {
        dir(env.APP_DIR) {
            script {
                if (isUnix()) {
                    sh 'chmod +x mvnw || true'
                    sh './mvnw -B -DskipTests clean verify'
                } else {
                    bat 'mvnw.cmd -B -DskipTests clean verify'
                }
            }
        }
    }
}

stage('SonarCloud Analysis') {
    steps {
        dir(env.APP_DIR) {
            withSonarQubeEnv(env.SONAR_INSTANCE) {
                script {
                    if (isUnix()) {
                        sh '''
                            chmod +x mvnw || true
                            ./mvnw -B -DskipTests sonar:sonar \
                            -Dsonar.projectKey=$SONAR_PROJECT_KEY \
                            -Dsonar.organization=$SONAR_ORG \
                            -Dsonar.host.url=https://sonarcloud.io \
                            -Dsonar.token=$SONAR_AUTH_TOKEN
                        '''
                    } else {
                        bat """
                            mvnw.cmd -B -DskipTests sonar:sonar ^
                            -Dsonar.projectKey=%SONAR_PROJECT_KEY% ^
                            -Dsonar.organization=%SONAR_ORG% ^
                            -Dsonar.host.url=https://sonarcloud.io ^
                            -Dsonar.token=%SONAR_AUTH_TOKEN%
                        """
                    }
                }
            }
        }
    }
}

```

}

**Remarque :** selon la version du plugin Maven, le log affiche souvent **BUILD SUCCESS** au lieu de **ANALYSIS SUCCESSFUL**. Dans les deux cas, cela indique que l'analyse SonarCloud s'est terminée correctement.

## 10.4 Étape 10 — Lancer le pipeline Jenkins

Lancement via : **Job** → **Build Now**. Les stages observés dans notre exécution apparaissent dans la vue *Stages*.

FIGURE 17 – Liste des stages du pipeline Jenkins (Checkout/Build/SonarCloud Analysis).

#### 10.4.1 Log du stage SonarCloud (analyse réussie)

The screenshot shows the Jenkins interface for a pipeline job named "springboot-sonarcloud-pip...". The left sidebar has a "Console Output" tab selected. The main area is titled "Console Output" with a green checkmark icon. It displays the following log output:

```

Started by user sami
Obtained Jenkinsfile from git
https://gitlab.com/tp_test_group/tp-ci-ci-multi-projets-fauzzi.git
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/lib/jenkins/workspace/springboot-sonarcloud-pipeline

```

FIGURE 18 – Extrait du log du stage SonarCloud : exécution réussie (*BUILD SUCCESS*).

The screenshot shows the Jenkins interface for a pipeline job named "springboot-sonarcloud-pip...". The left sidebar has a "Console Output" tab selected. The main area is titled "Console Output" with a green checkmark icon. It displays the following log output, with the "Timestamps" section highlighted:

**Timestamps**

- System clock time
- Use browser timezone
- Elapsed time
- None

[View as plain text](#)

```

23:02:15 [INFO] 23:02:15.060 Sensor cache published successfully
23:02:15 [INFO] 23:02:15.068 Analysis total time: 25.437 s
23:02:15 [INFO] -----
-----
23:02:15 [INFO] BUILD SUCCESS
23:02:15 [INFO] -----
-----
23:02:15 [INFO] Total time: 44.052 s
23:02:15 [INFO] Finished at: 2025-12-14T23:02:15+01:00
23:02:15 [INFO] -----
-----
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // withSonarQubeEnv
[Pipeline] }
[Pipeline] // dir
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // timestamps
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

FIGURE 19 – Deuxième extrait du log confirmant la réussite du pipeline.

## 10.5 Étape 11 — Analyse des résultats (SonarCloud)

Après l'exécution du pipeline Jenkins, une nouvelle analyse apparaît sur SonarCloud. Cela confirme que Jenkins a bien envoyé l'analyse vers SonarCloud.

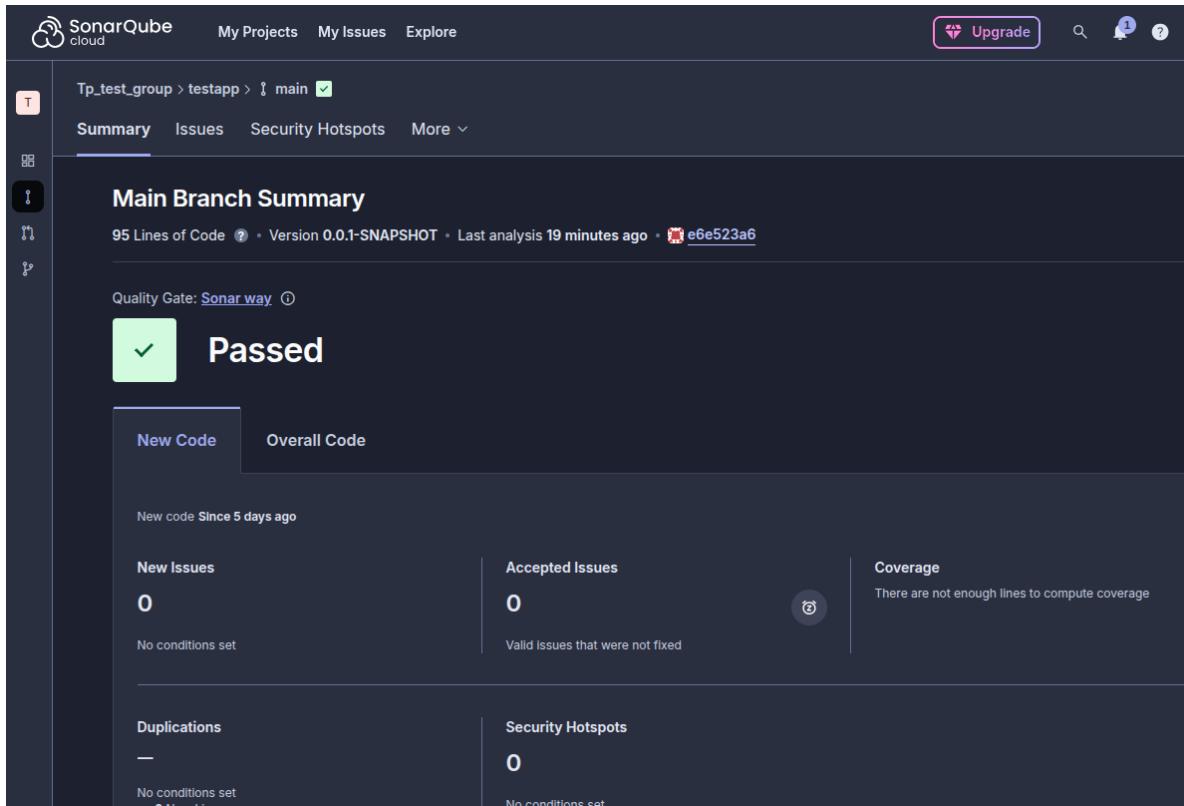


FIGURE 20 – Nouvelle analyse reçue et visible dans SonarCloud après exécution depuis Jenkins.

## 10.6 Comparaison synthétique : GitLab CI vs Jenkins (même analyse SonarCloud)

Une analyse SonarCloud a déjà été réalisée via GitLab CI. Les métriques SonarCloud restent cohérentes, car l'outil d'analyse est le même (SonarCloud) ; seule la plateforme CI change.

| Critère              | GitLab CI + SonarCloud           | Jenkins + SonarCloud                        |
|----------------------|----------------------------------|---|
| Fichier pipeline     | <a href="#">.gitlab-ci.yml</a>   | <a href="#">Jenkinsfile</a>                 |
| Exécution            | GitLab Runner                    | Jenkins (controller/agent)                  |
| Déclenchement        | push/merge (CI)                  | manuel ou automatique (Build Now / webhook) |
| Résultats SonarCloud | Identiques (mêmes règles/projet) | Identiques (mêmes règles/projet)            |
| Visualisation        | Pipeline GitLab                  | Vue Stages + Console détaillée              |

TABLE 1 – Comparaison GitLab CI vs Jenkins pour l'analyse SonarCloud.

## 11 Annexe — Rappel variables GitLab attendues

Dans GitLab → **Settings** → **CI/CD** → **Variables** :

- **SONAR\_TOKEN** : token généré dans SonarCloud
- **SONAR\_HOST\_URL** : <https://sonarcloud.io>
- **SONAR\_ORGANIZATION** : clé de l'organisation SonarCloud