École Mohammadia d'Ingénieurs

CI/CD — DevOps
SonarCloud + Maven/GitLab CI (avec pipeline multi-projets)

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

Élèves :
Sami FAOUZI

Encadré par :
Pr. Asmaa RETBI

2ème année Génie Informatique
14 décembre 2025

Table des matières

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.

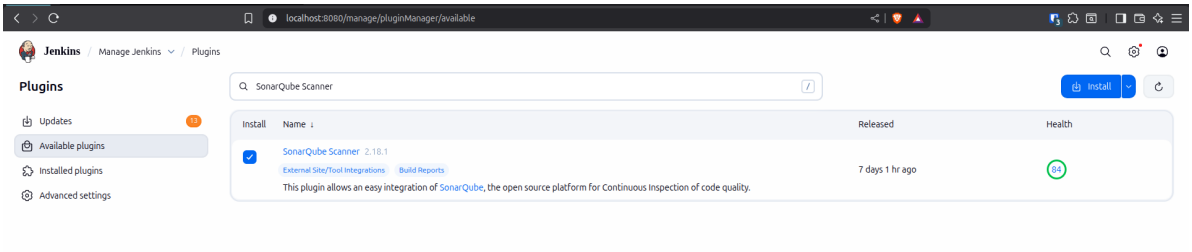

FIGURE 1 – Plugin SonarQube (pré-requis selon configuration Jenkins).

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

3.1 Connexion via GitLab

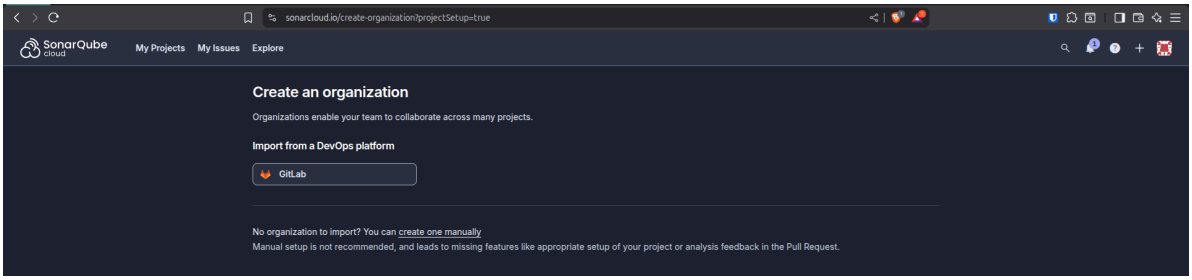

FIGURE 2 – Page de connexion SonarCloud avec l'option GitLab.

3.2 Tableau de bord après connexion

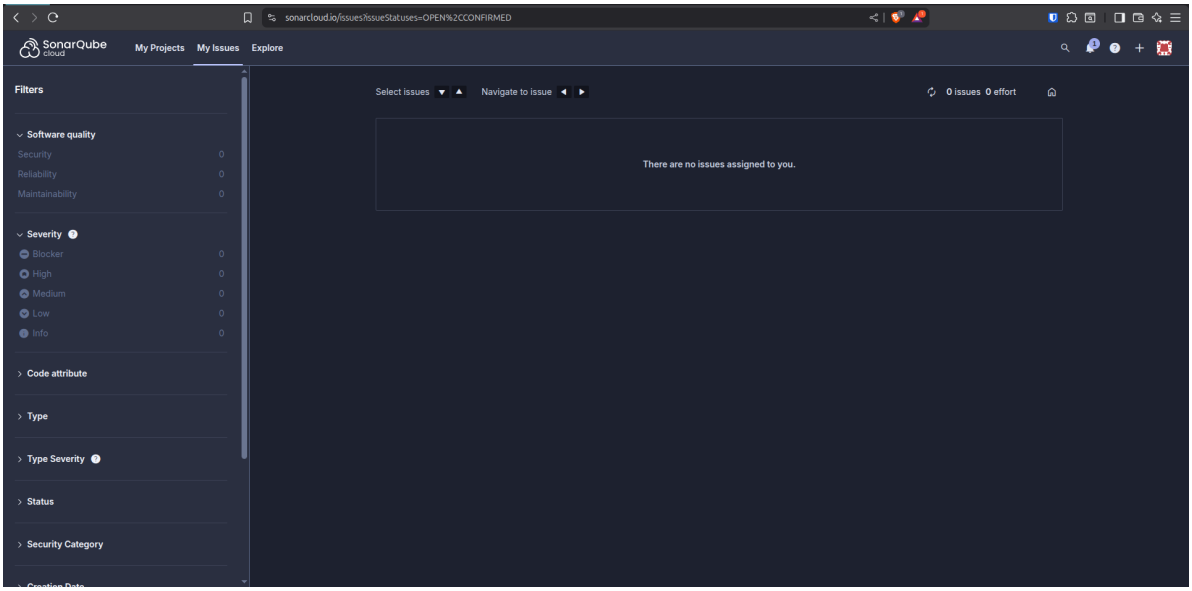

FIGURE 3 – Tableau de bord SonarCloud (après connexion).

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

4.1 Création du Personal Access Token GitLab (scope api)

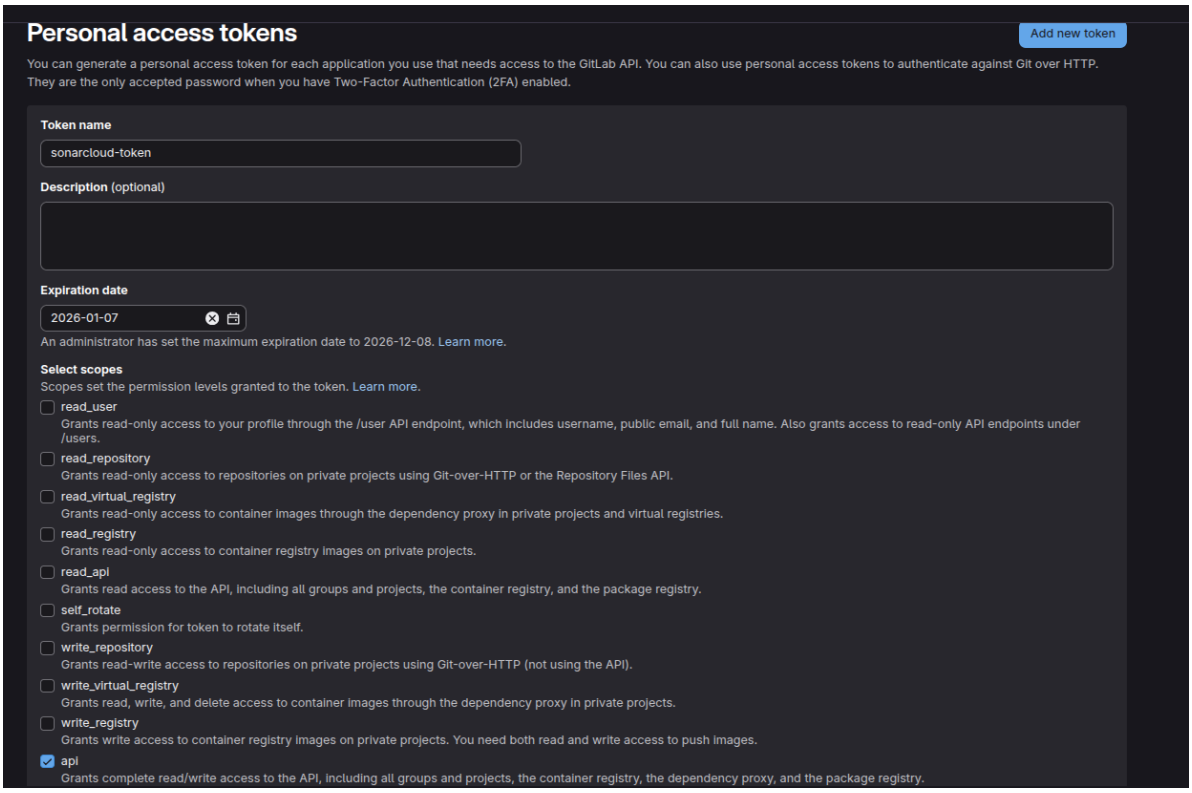

FIGURE 4 – Création du token GitLab (scope api) utilisé pour l'organisation SonarCloud.

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

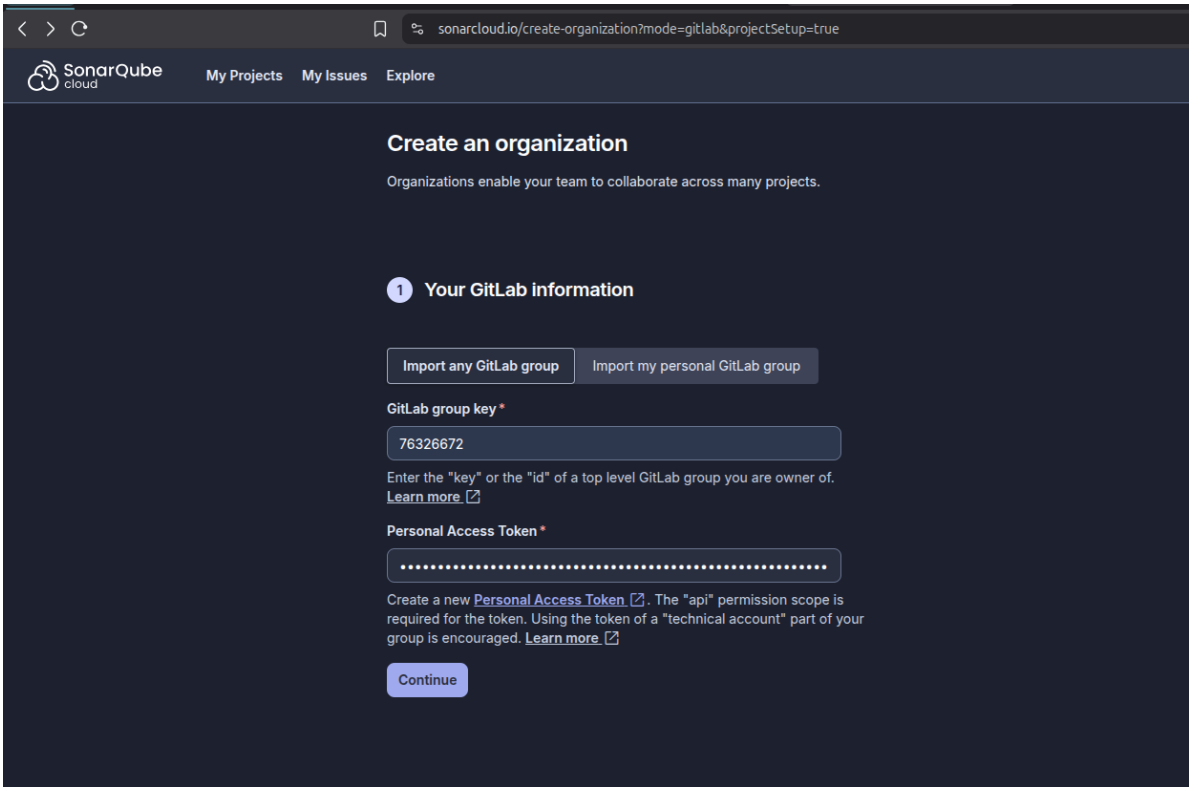

FIGURE 5 – Création de l'organisation SonarCloud liée au groupe GitLab.

4.3 Organisation créée et connectée

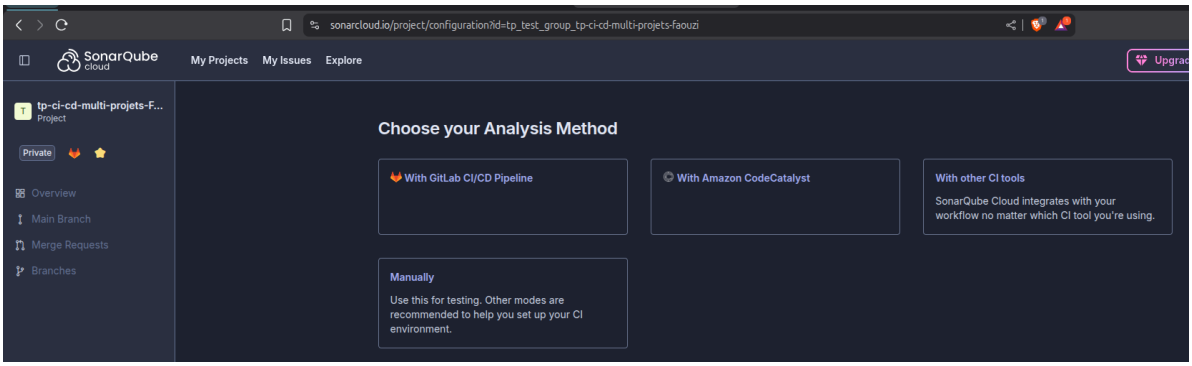

FIGURE 6 – Organisation SonarCloud créée et reliée à GitLab.

5 Importer le projet Spring Boot existant — Diapo 60

5.1 Liste des projets GitLab visibles dans SonarCloud

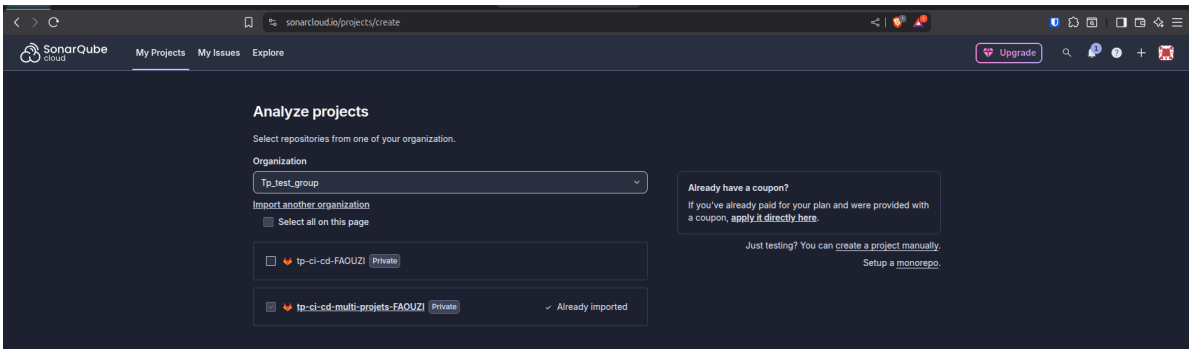

FIGURE 7 – Liste des projets GitLab détectés dans SonarCloud.

5.2 Projet Spring Boot sélectionné

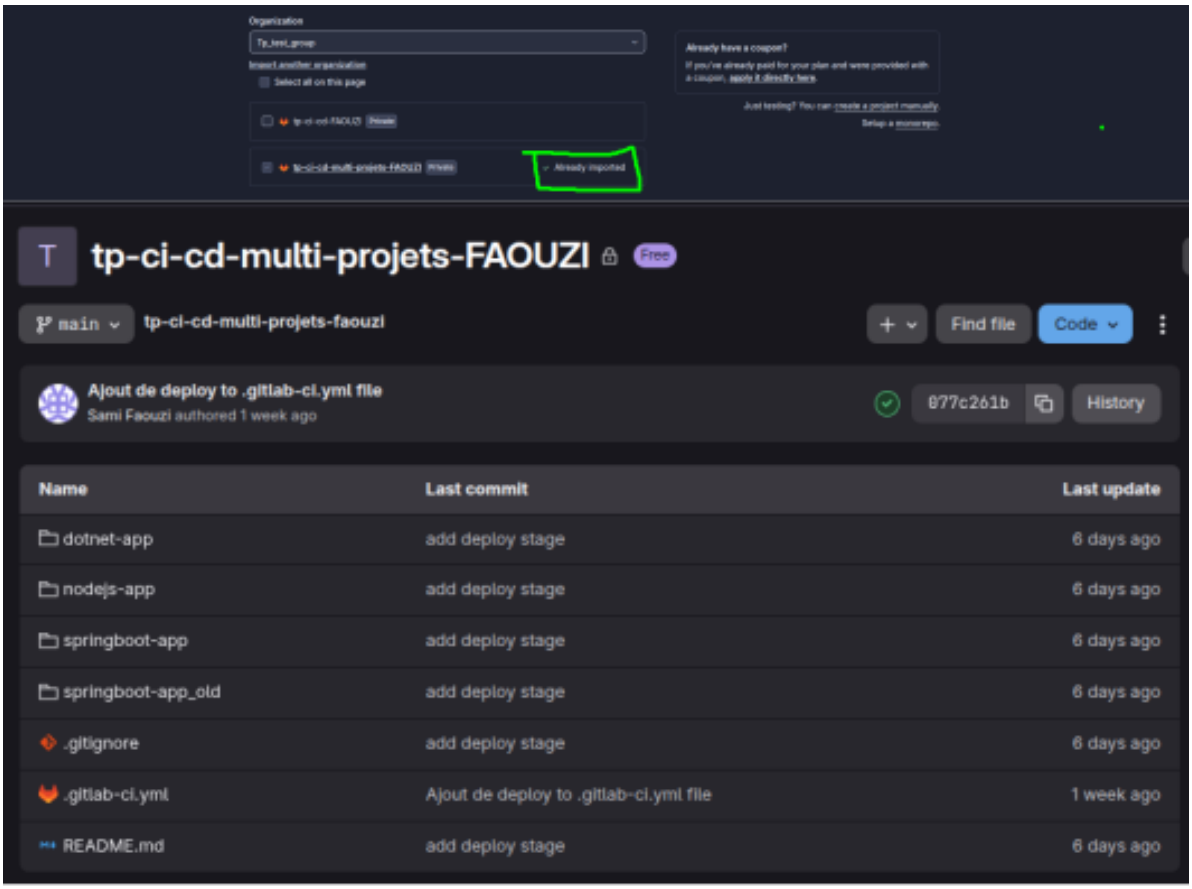

FIGURE 8 – Sélection du projet Spring Boot à analyser dans SonarCloud.

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

6.1 Variables CI/CD à ajouter dans GitLab

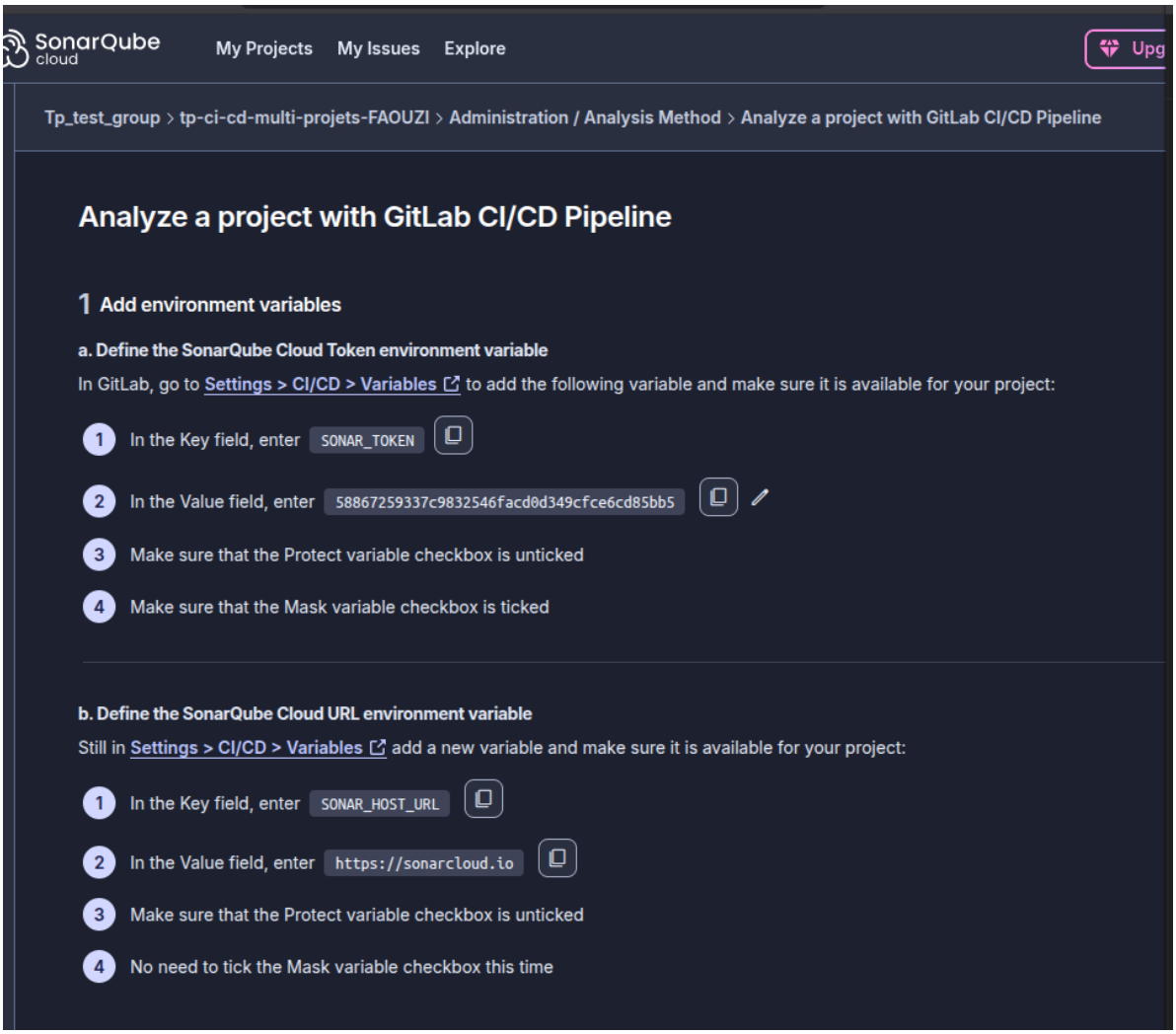

FIGURE 9 – Variables recommandées par SonarCloud : `SONAR_TOKEN`, `SONAR_HOST_URL`, `SONAR_ORGANIZATION`.

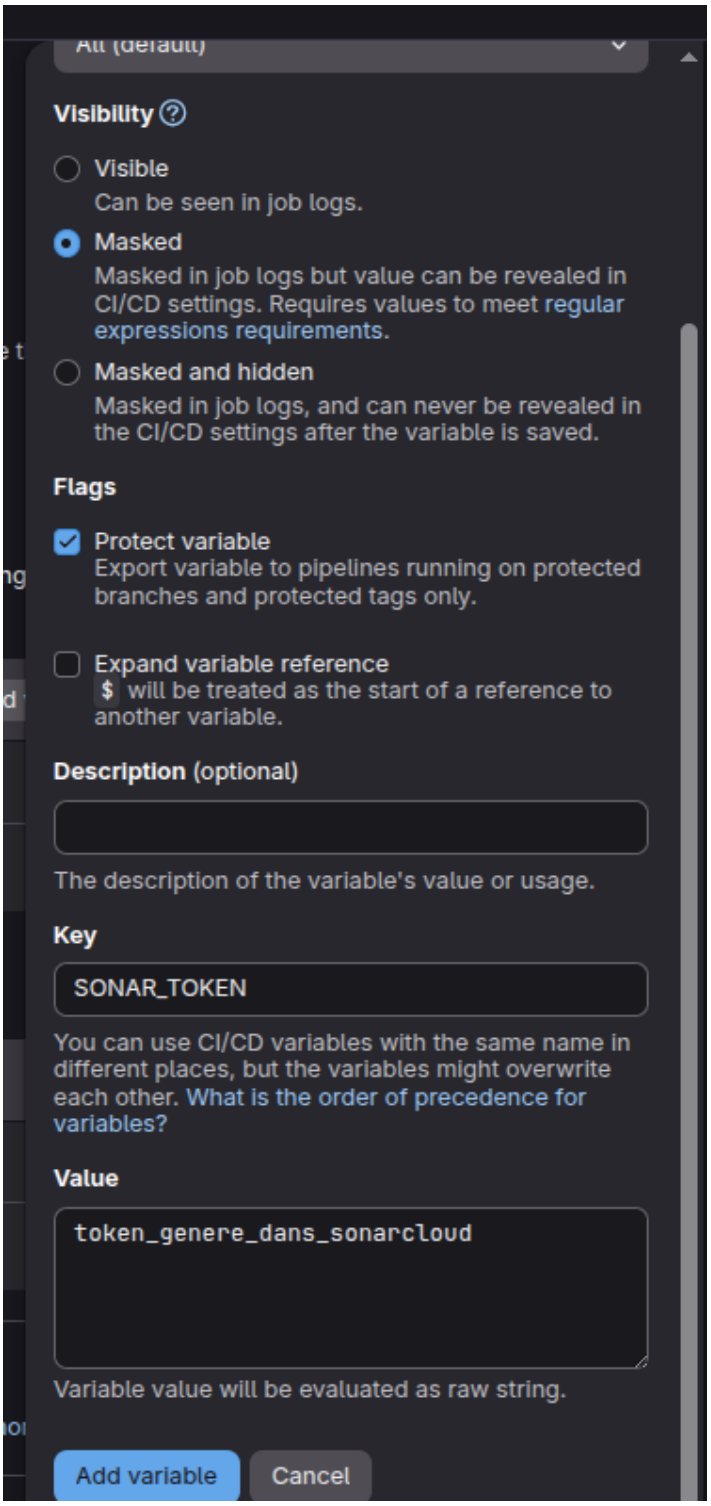

FIGURE 10 – Configuration des variables CI/CD dans GitLab (SONAR_TOKEN, SONAR_HOST_URL, SONAR_ORGANIZATION).

6.2 Bloc YAML proposé par l'assistant SonarCloud

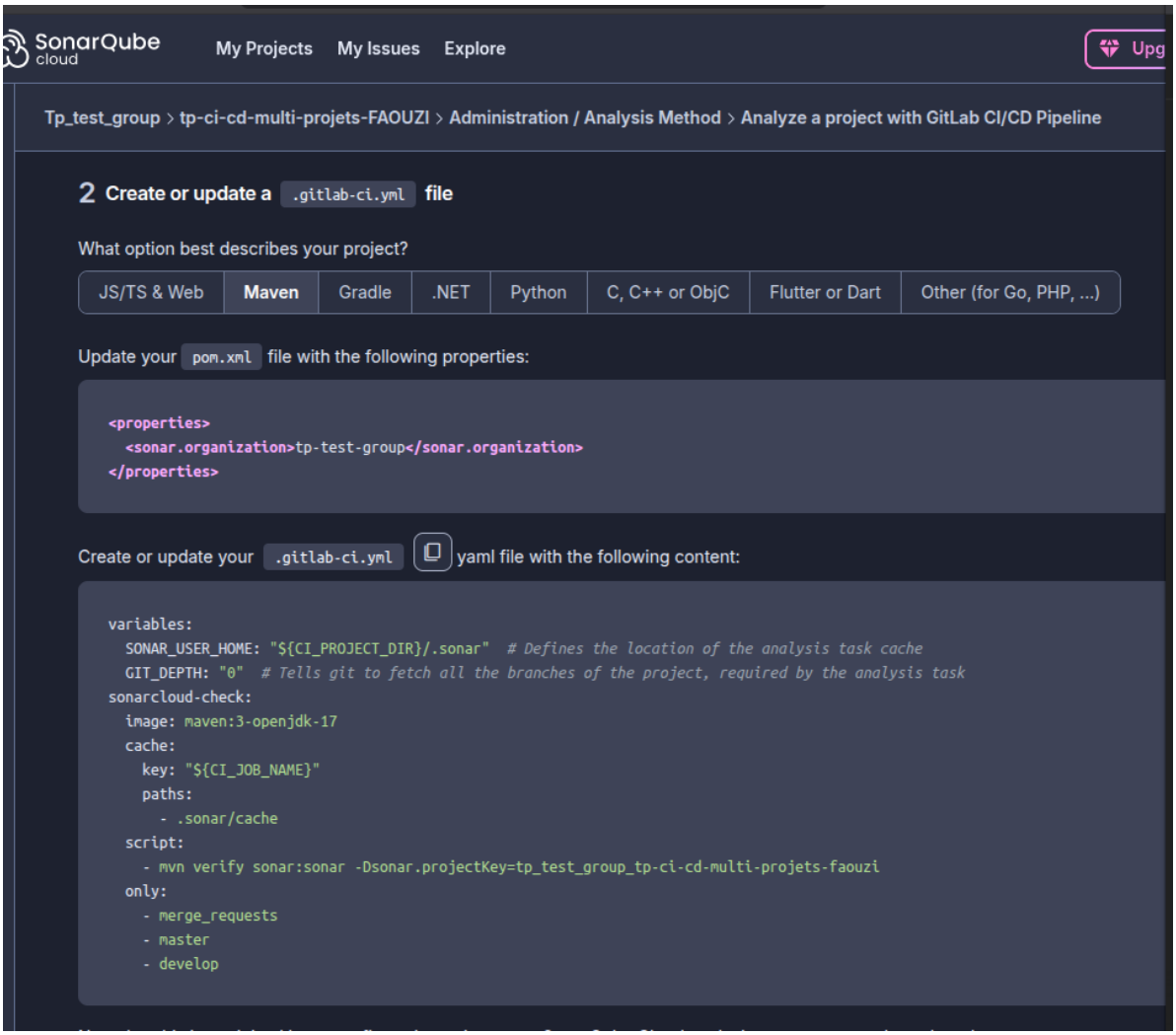

FIGURE 11 – Bloc YAML proposé par SonarCloud (Maven/GitLab CI).

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

# ---------- DPLOIEMENT 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

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

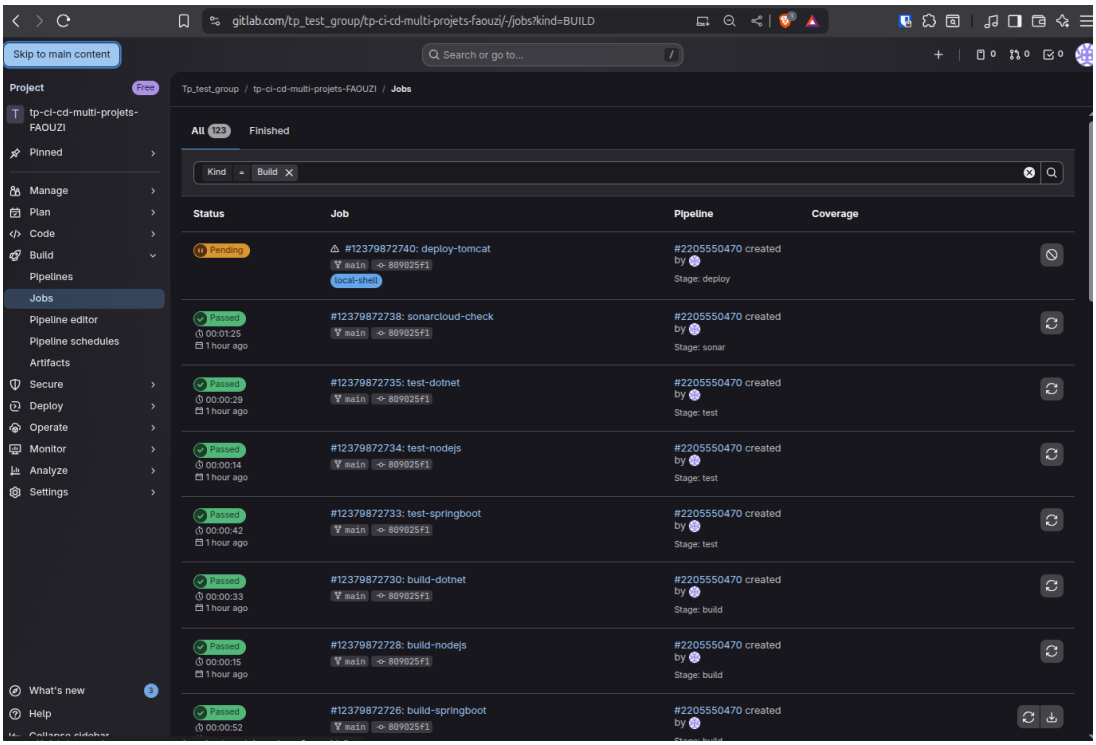

FIGURE 13 – Connexion SonarCloud réussie via GitLab.

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.

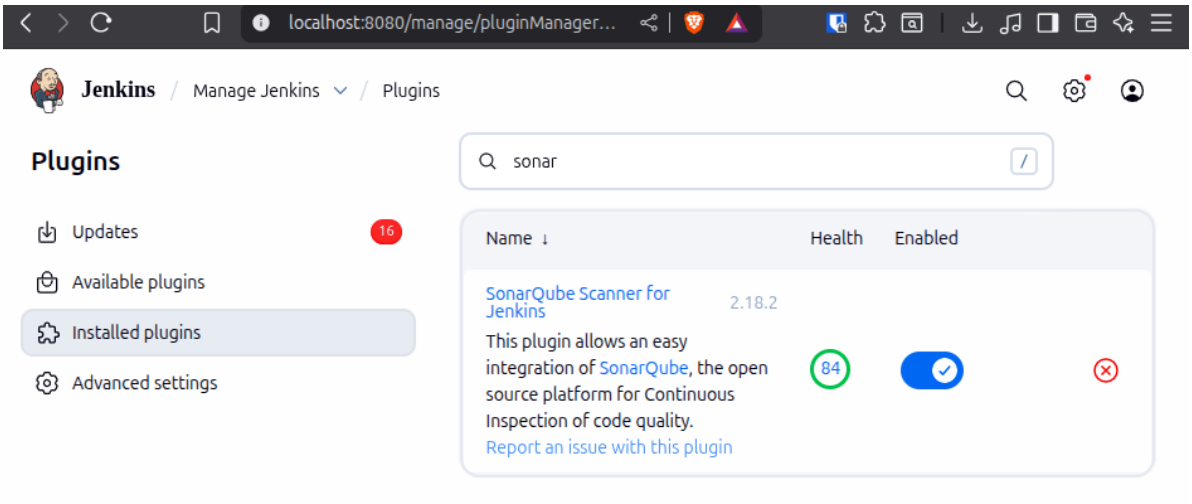

FIGURE 14 – Installation du plugin SonarQube Scanner dans Jenkins (installation réussie).

10.2 Étape 8 — Configurer SonarCloud dans Jenkins

10.2.1 Création du credential SonarCloud (Secret text)

Ajout du token SonarCloud en tant que credential **Secret text** dans : `Manage Jenkins` → `Credentials` → `Global` → `Add Credentials`.

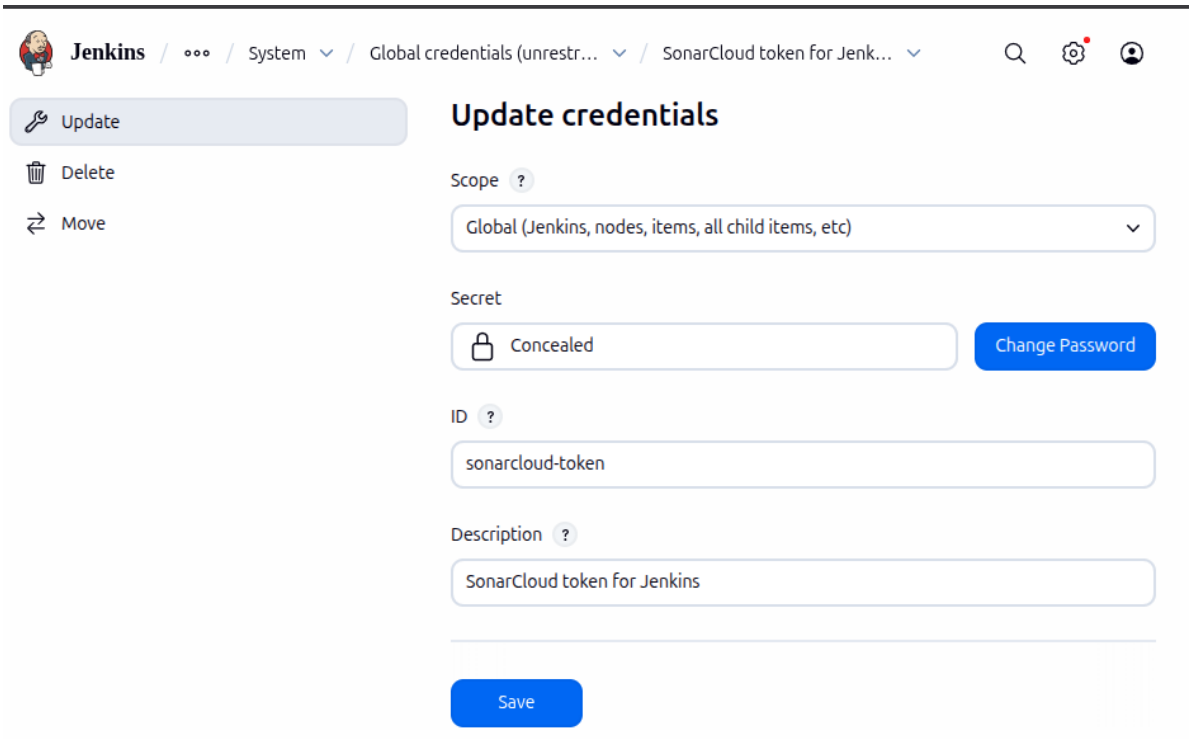

FIGURE 15 – Credential SonarCloud (Secret text) créé dans Jenkins.

10.2.2 Déclaration du serveur SonarCloud (SonarQube servers)

Configuration dans : `Manage Jenkins` → `Configure System` → `SonarQube servers`.

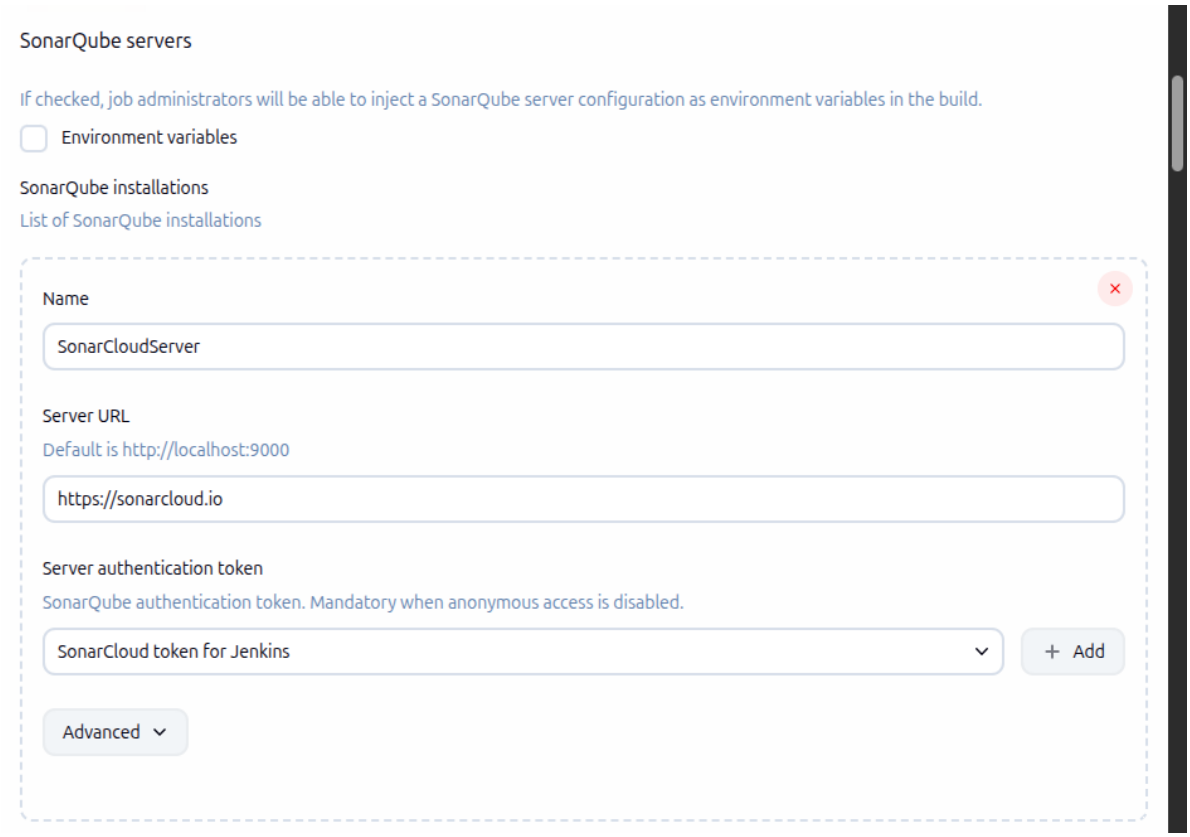

FIGURE 16 – Section *SonarQube servers* remplie : Name = `SonarCloudServer`, URL = `https://sonarcloud.io`, token associé.

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*.

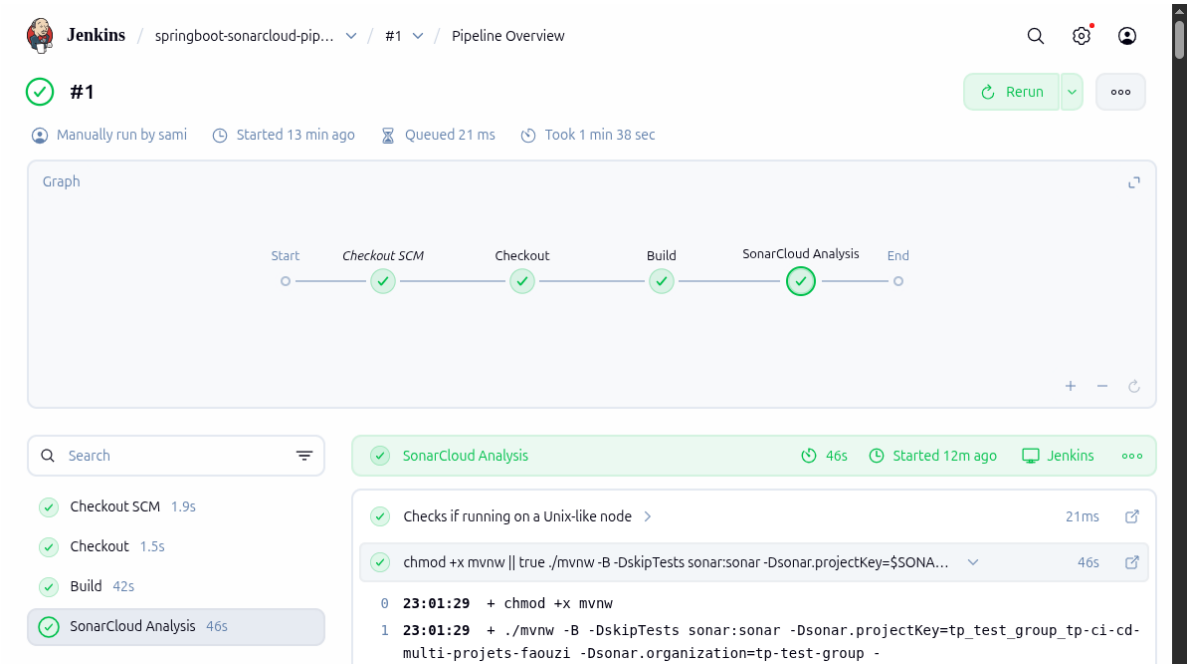

FIGURE 17 – Liste des stages du pipeline Jenkins (Checkout/Build/SonarCloud Analysis).

10.4.1 Log du stage SonarCloud (analyse réussie)

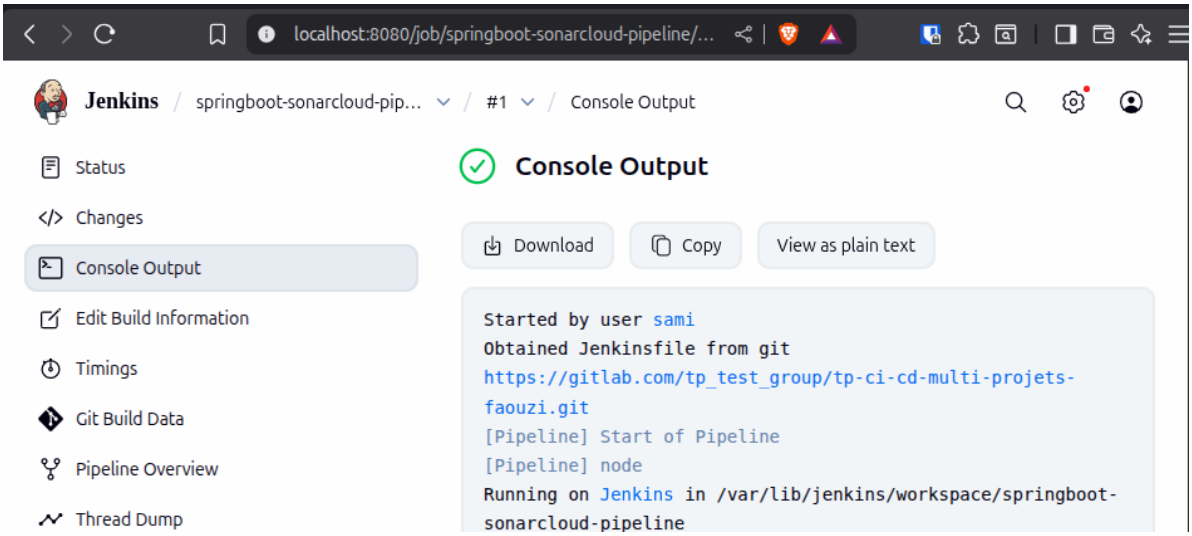

FIGURE 18 – Extrait du log du stage SonarCloud : exécution réussie (*BUILD SUCCESS*).

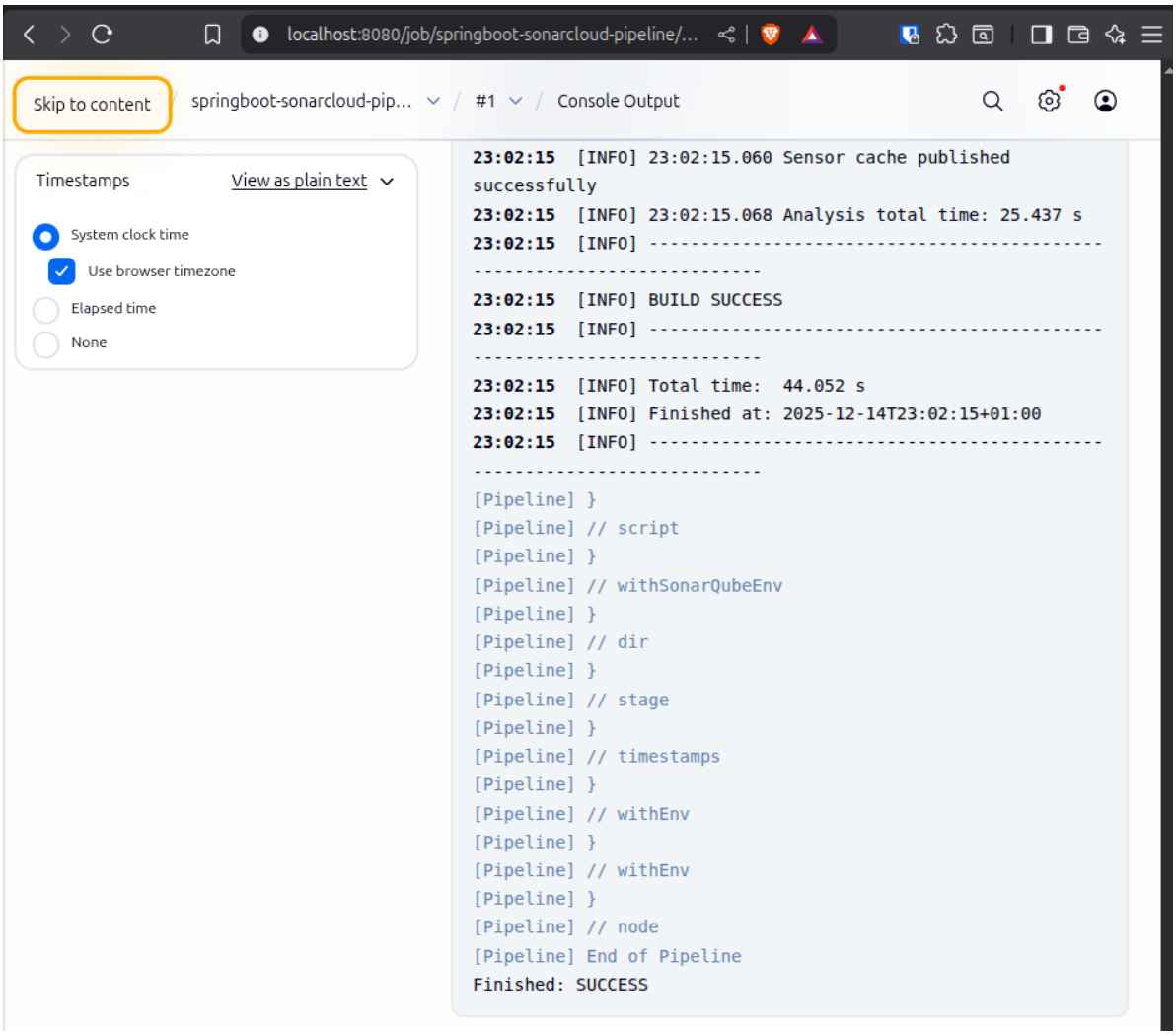

FIGURE 19 – Deuxième extrait du log confirmant la réussite du pipeline.

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.

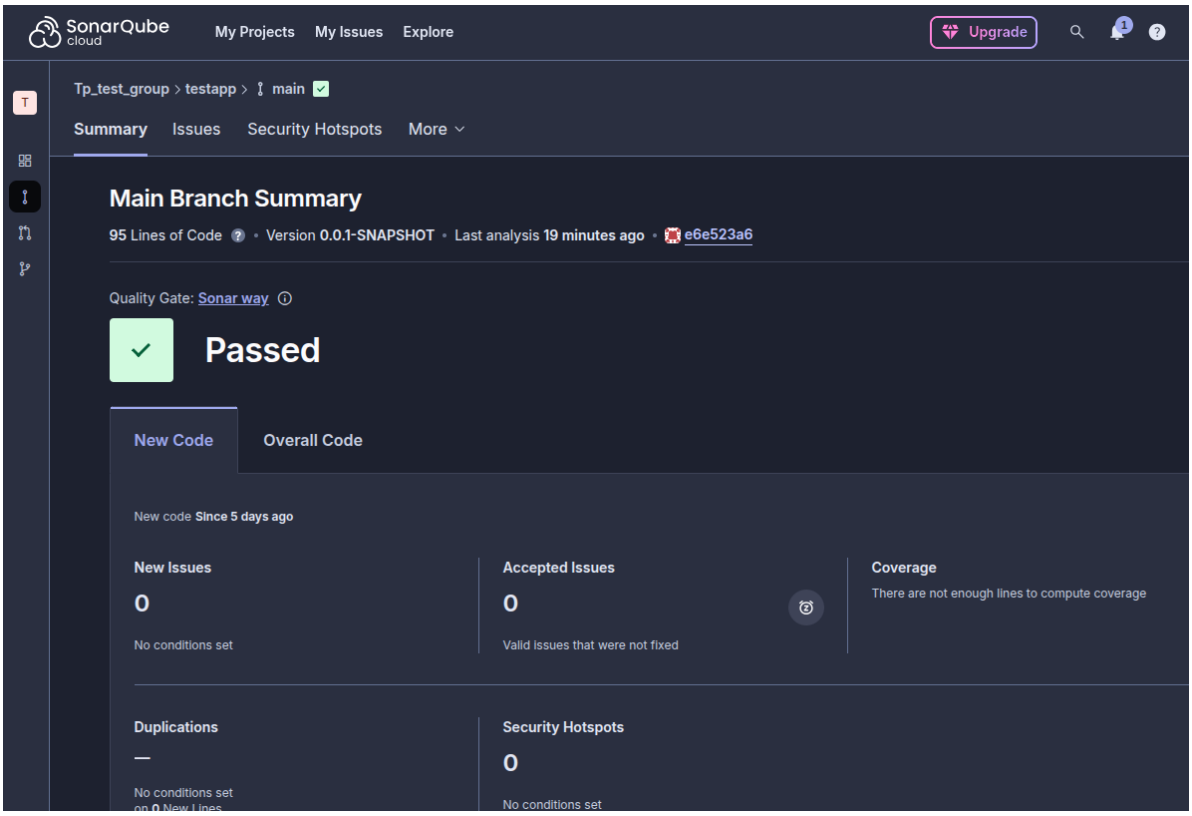

FIGURE 20 – Nouvelle analyse reçue et visible dans SonarCloud après exécution depuis Jenkins.

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	`.gitlab-ci.yml`	`Jenkinsfile`
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