

# Analyseur de Code : Étude, Mise en Place et Exploitation de SonarQube

## Table des matières

Analyseur de Code : Étude, Mise en Place et Exploitation de SonarQube.....	1
1. Introduction Générale.....	2
Qu'est-ce qu'un Analyseur de Code ? .....	2
Objectifs du Projet .....	3
2. Étude de Marché et Choix de l'Outil.....	3
Types d'Analyseurs .....	3
Critères de Sélection .....	3
Pourquoi SonarQube ? .....	4
3. Fonctionnement d'un Analyseur de Code .....	4
Étapes de l'Analyse Statique .....	4
4. Mise en Place de l'Infrastructure .....	5
A. Prérequis serveur Debian .....	5
B. Configuration base de données PostgreSQL .....	5
C. Installation de SonarQube .....	5
D. Configuration.....	6
E. Création utilisateur et lancement du service .....	6
F. Accès à l'interface web .....	6
5. Analyse de Code avec SonarScanner.....	6
A. Installation .....	6
B. Préparation d'un Projet .....	7
C. Lancement de l'analyse .....	7
6. Exemple de Vulnérabilités Détectées.....	8
Cas 1 : Identifiants exposés (mots de passe en clair) .....	8
7. Exploitation et Utilisation en Environnement Réel.....	9
En production : .....	9
8. Conclusion .....	9
Recommandation : .....	10

## 1. Introduction Générale

### Qu'est-ce qu'un Analyseur de Code ?

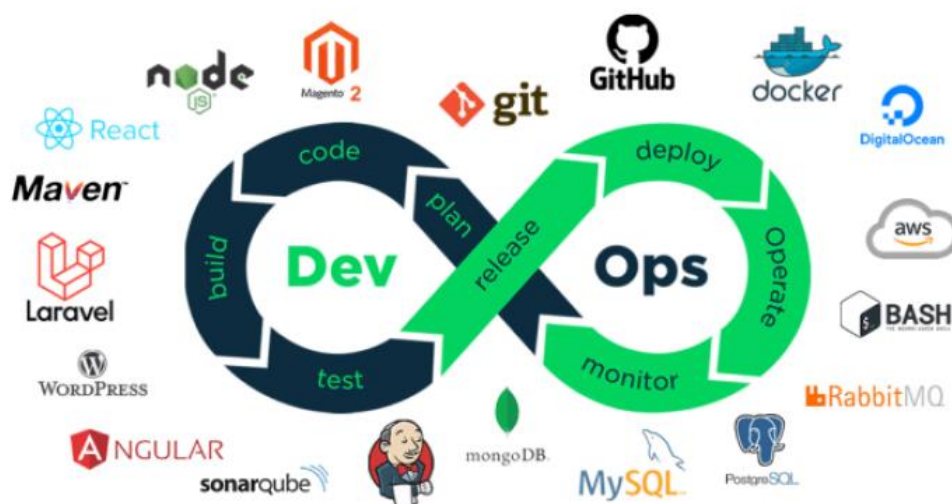
Un analyseur de code est un outil logiciel conçu pour examiner le code source d'une application afin d'identifier :

- Des erreurs potentielles
- Des failles de sécurité
- Des duplications
- Des violations de conventions de codage

Ces outils permettent d'améliorer la **qualité du code**, la **maintenabilité** et la **sécurité** des applications, et peuvent intervenir à différentes étapes du cycle de développement.

Dans un contexte de développement moderne, les équipes DevOps adoptent des workflows CI/CD (Intégration Continue / Déploiement Continu) pour livrer plus rapidement et de manière plus fiable du code en production. Ce processus automatisé permet d'enchaîner les phases de développement (planification, codage, tests), d'intégration, de livraison, de déploiement et de supervision dans un cycle fluide et répétable.

Le schéma ci-dessous illustre le cycle DevOps avec les principales étapes de CI/CD, et les technologies souvent associées à chaque phase (analyseur = test) :



## Objectifs du Projet

Ce projet de fin d'études en cybersécurité a pour but de sélectionner, déployer et exploiter un analyseur de code. L'objectif est d'intégrer des vérifications de sécurité dans un environnement de développement continu, tout en évaluant plusieurs outils du marché avant de porter notre choix sur SonarQube.

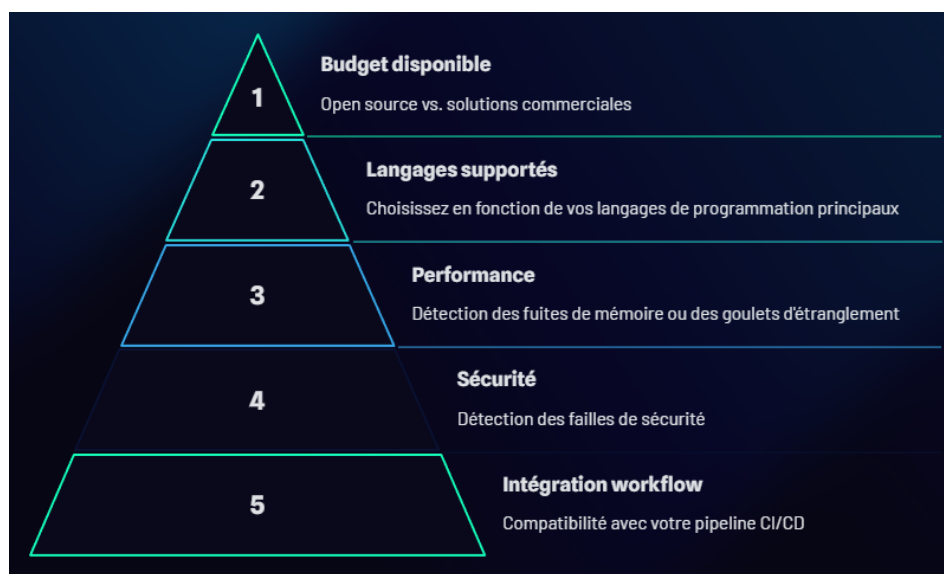
## 2. Étude de Marché et Choix de l'Outil

### Types d'Analyseurs

1. **Analyseurs Statiques** : N'exécutent pas le programme. Rapides pour détecter des erreurs dans le code source. Ex. : SonarQube, ESLint.
2. **Analyseurs Dynamiques** : Nécessitent l'exécution du programme. Détectent les fuites mémoire ou problèmes de performance. Ex. : Valgrind, AppDynamics.
3. **Analyseurs Hybrides** : Combinent les deux. Ex. : Checkmarx (SAST + DAST).

### Critères de Sélection

1. **Budget** : Open source de préférence, mais évolutif vers du payant si besoin.
2. **Langages supportés** : Outils multi-langages privilégiés.
3. **Performance** : Soumis aux ressources du serveur de l'école pour le projet
4. **Sécurité** : Détection des failles OWASP et autres vulnérabilités.
5. **Intégration CI/CD** : Doit s'adapter à GitLab, GitHub Actions, etc.



## Pourquoi SonarQube ?

Critère	SonarQube	ESLint	Checkmarx
Langages supportés	Large	JS/TS	Large
Open Source	Oui	Oui	Non
Sécurité	Moyenne	Faible	Élevée
Performance	Moyenne	N/A	Moyenne
Intégration CI/CD	Oui	Oui	Oui
Coût	Gratuit/Payant	Gratuit	Payant

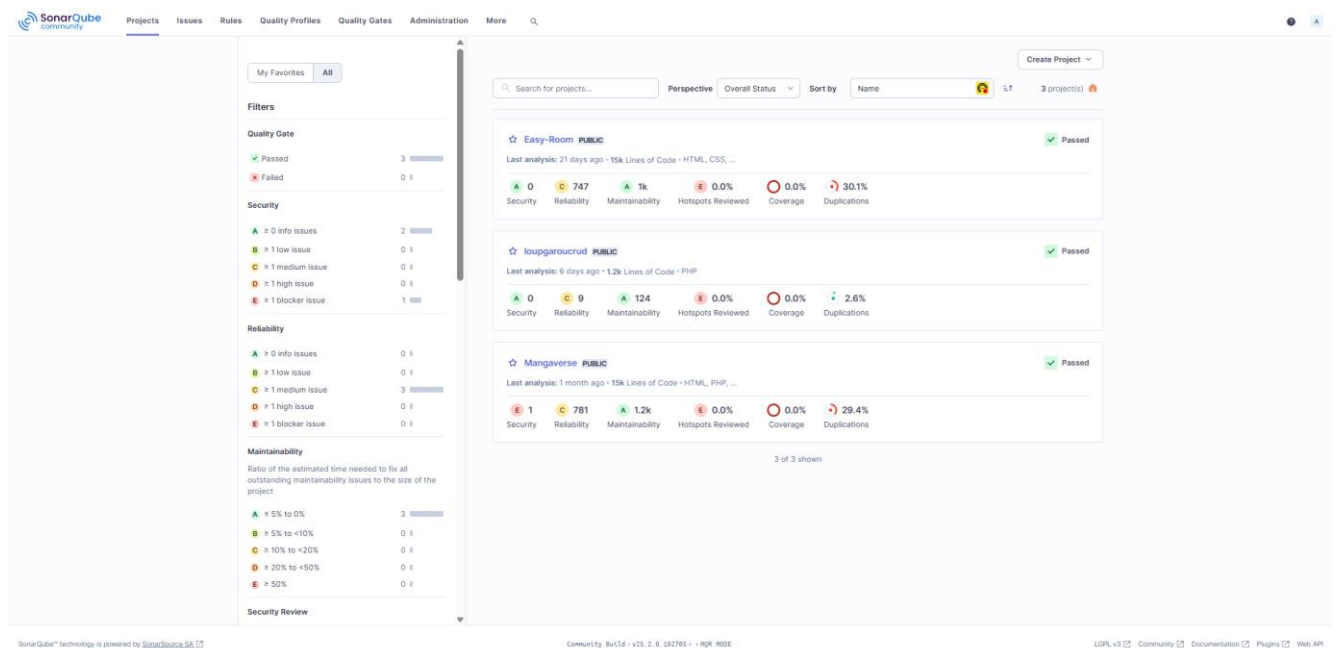
- Support multi-langages
- Version Community gratuite
- Détection efficace des duplications, dettes techniques, failles basiques
- Intégration fluide avec CI/CD

---

## 3. Fonctionnement d'un Analyseur de Code

### Étapes de l'Analyse Statique

1. **Extraction** : Récupération du code localement ou via Git
2. **Analyse syntaxique** : Création d'un arbre syntaxique abstrait (AST)
3. **Application des règles** : Détection de mauvaises pratiques, vulnérabilités, duplications, etc.
4. **Rapport** : Génération de résultats détaillés accessibles via une interface web



## 4. Mise en Place de l'Infrastructure

### A. Prérequis serveur Debian

```
sudo apt install sudo -y
```

```
sudo apt install openjdk-17-jdk unzip wget -y
```

```
sudo apt install postgresql postgresql-contrib -y
```

### B. Configuration base de données PostgreSQL

```
sudo -u postgres createuser sonar
```

```
sudo -u postgres createdb sonarqube
```

```
sudo -u postgres psql
```

```
ALTER USER sonar WITH PASSWORD 'mot_de_passe';
```

```
GRANT ALL PRIVILEGES ON DATABASE sonarqube TO sonar;
```

```
\q
```

### C. Installation de SonarQube

```
cd /opt
```

```
sudo wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-25.2.0.102705.zip
```

```
sudo unzip sonarqube-25.2.0.102705.zip
```

```
sudo mv sonarqube-25.2.0.102705 sonarqube
```

## D. Configuration

- Modifier conf/sonar.properties pour y insérer les identifiants PostgreSQL

## E. Création utilisateur et lancement du service

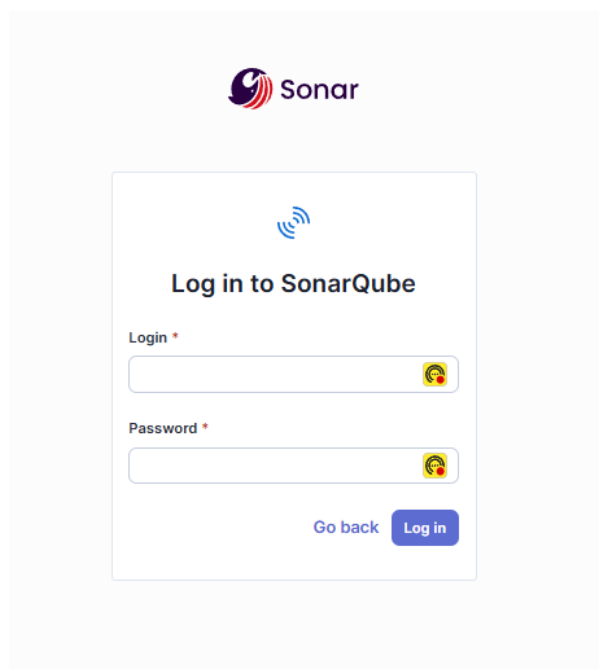
```
sudo useradd -m -d /opt/sonarqube sonaruser
```

```
sudo chown -R sonaruser:sonaruser /opt/sonarqube
```

```
sudo -u sonaruser /opt/sonarqube/bin/linux-x86-64/sonar.sh start
```

## F. Accès à l'interface web

- URL : <http://192.168.27.12:9000>
- Login par défaut : admin / admin, changé en S@nar22S@nar22



---

## 5. Analyse de Code avec SonarScanner

### A. Installation

```
cd /opt
```

```
sudo wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-5.0.1.3006-linux.zip
```

```
sudo unzip sonar-scanner-cli-5.0.1.3006-linux.zip
```

```
sudo mv sonar-scanner-cli-5.0.1.3006-linux sonar-scanner
```

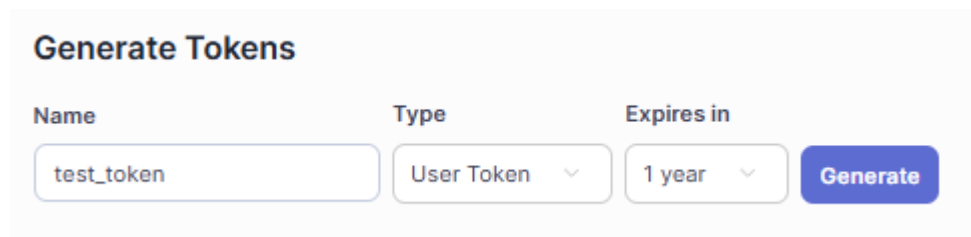
Ajout au PATH :

```
echo 'export PATH="/opt/sonar-scanner/bin:$PATH"' >> ~/.bashrc  
source ~/.bashrc
```

## B. Préparation d'un Projet

```
cd ~/nom_du_projet  
nano sonar-project.properties  
sonar.projectKey=occurrence_du_projet  
sonar.projectName=nom_sur_l_interface_web  
sonar.projectVersion=1.0  
sonar.sources=.  
sonar.host.url=http://192.168.27.12:9000  
sonar.login=JETON_GENERÉ
```

*note : pour générer un jeton depuis l'interface web : Myaccount/security/*



Name	Type	Expires in	
test_token	User Token	1 year	Generate

## C. Lancement de l'analyse

```
sonar-scanner
```

```
aftec@debian:~/sonarqube_test/DVWA$ sonar-scanner
INFO: Scanner configuration file: /opt/sonar-scanner/conf/sonar-scanner.properties
INFO: Project root configuration file: /home/aftec/sonarqube_test/DVWA/sonar-project.properties
INFO: SonarScanner 5.0.1.3000
INFO: Java 17.0.7 Eclipse Adoptium (64-bit)
INFO: Linux 5.10.0-34-amd64 amd64
INFO: User cache: /home/aftec/.sonar/cache
INFO: Analyzing on SonarQube server 25.2.0.102705
INFO: Default locale: "fr_FR", source code encoding: "UTF-8" (analysis is platform dependent)
INFO: Load global settings
INFO: Load global settings (done) | time=130ms
INFO: Server id: 86e1f440-a2052YXc12VPz35klHwR
INFO: Loading required plugins
INFO: Load plugins index
INFO: Load plugins index (done) | time=62ms
INFO: Load/download plugins
INFO: Load/download plugins (done) | time=51ms
INFO: Process project properties
INFO: Process project properties (done) | time=23ms
INFO: Project key: dvwa
INFO: Base dir: /home/aftec/sonarqube_test/DVWA
INFO: Working dir: /home/aftec/sonarqube_test/DVWA/.scannerwork
INFO: Load project settings for component key: 'dvwa'
INFO: Load quality profiles
INFO: Load quality profiles (done) | time=933ms
INFO: Load active rules
INFO: Load active rules (done) | time=1218ms
INFO: Load analysis cache
INFO: Load analysis cache (404) | time=14ms
WARN: The property 'sonar.login' is deprecated and will be removed in the future. Please use the 'sonar.token' property instead when passing a token.
INFO: Preprocessing files...
INFO: 8 languages detected in 225 preprocessed files
INFO: 1 file ignored because of scm ignore settings
INFO: Loading plugins for detected languages
INFO: Load/download plugins
INFO: Load/download plugins (done) | time=456ms
INFO: Load project repositories
INFO: Load project repositories (done) | time=11ms
INFO: Indexing files...
INFO: Project configuration:
INFO: Some of the project files were automatically excluded because they looked like generated code. Enable debug logging to see which files were excluded. You can disable bundle detection by setting sonar.javascript-detectedBundles=false
INFO: 222 files indexed
INFO: Quality profile for css: Sonar way
INFO: Quality profile for docker: Sonar way
INFO: Quality profile for js: Sonar way
INFO: Quality profile for json: Sonar way
INFO: Quality profile for php: Sonar way
INFO: Quality profile for py: Sonar way
INFO: Quality profile for web: Sonar way
INFO: Quality profile for yaml: Sonar way
INFO: ----- Run sensors on module dvwa
INFO: Load metrics repository
INFO: Load metrics repository (done) | time=68ms
INFO: Sensor pythonSensor (python)
WARN: Your code is analyzed with all Python 3 versions by default. You can get a more precise analysis by setting the exact Python version in your configuration via the parameter "sonar.python.version"
```

```
INFO: Analysis report generated in 1008ms, dir size=1.4 MB
INFO: Analysis report compressed in 1124ms, zip size=820.4 kB
INFO: Analysis report uploaded in 5955ms
INFO: ANALYSIS SUCCESSFUL, you can find the results at: http://192.168.27.12:9000/dashboard?id=dvwa
INFO: Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
INFO: More about the report processing at http://192.168.27.12:9000/api/ce/task?id=2cd0848f-8d61-4a17-94f7-22e5de1d69b7
INFO: Analysis total time: 40.918 s
INFO: -----
INFO: EXECUTION SUCCESS
INFO: -----
INFO: Total time: 43.957s
INFO: Final Memory: 30M/107M
INFO: -----
aftec@debian:~/sonarqube_test/DVWA$
```

☆

dvwa

PUBLIC

✓

Passed

Last analysis: 1 minute ago • 8.5k Lines of Code • PHP, CSS, ...

<div>E 3</div>	<div>C 63</div>	<div>A 991</div>	<div>E 0.0%</div>	<div>0.0%</div>	<div>10.1%</div>
Security	Reliability	Maintainability	Hotspots Reviewed	Coverage	Duplications

## 6. Exemple de Vulnérabilités Détectées

### Cas 1 : Identifiants exposés (mots de passe en clair)

compose.yml

☐

Make sure this MySQL database password gets changed and removed from the code.

Responsibility

Security

cwe +

Open

Not assigned

L31 • 30min effort • 1 year ago



- Présents dans un fichier compose.yml ou .env
- Risques : escalade de privilège, accès direct à la BDD
- Correction : utilisation de variables d'environnement et fichier .env ignoré par Git

The screenshot displays the SonarQube web interface. At the top, a message states: "Make sure this MySQL database password gets changed and removed from the code." Below this, it specifies "MySQL database passwords should not be disclosed" with a link to "secrets:S6697". It also shows "Line affected: L31", "Effort: 30min", and "Introduced: 1 year ago". On the right, a sidebar indicates "Software qualities impacted" with a "Security" icon, and "Clean code attribute" with "Responsibility" and "Not trustworthy" tags. Below the message, there are tabs for "Where is the issue?", "Why is this an issue?", "How can I fix it?", "Activity", and "More info". The main area shows a code editor for "dvwa > /compose.yml". The code is a Docker Compose file. A red box highlights the line: "MYSQL\_PASSWORD=p@ssw0rd". Below the code, a red box contains the same message: "Make sure this MySQL database password gets changed and removed from the code." The code in the editor is as follows:

```
26 baauco... image: docker.io/library/mariadb:10
27 environment:
28   - MYSQL_ROOT_PASSWORD=dvwa
29   - MYSQL_DATABASE=dvwa
30   - MYSQL_USER=dvwa
31   - MYSQL_PASSWORD=p@ssw0rd
32 volumes:
33   - dvwa:/var/lib/mysql
34 networks:
35   - dvwa
36 restart: unless-stopped
37
```

## 7. Exploitation et Utilisation en Environnement Réel

En production :

- Analyse automatique via GitLab CI ou GitHub Actions
- Historique de qualité de code accessible sur l'interface
- Surveillance continue des dettes techniques

## 8. Conclusion

Ce projet nous a permis de :

- Comparer plusieurs solutions d'analyse de code
- Comprendre les critères de choix pertinents dans un contexte de cybersécurité

- Déployer et configurer un environnement SonarQube fonctionnel
- Analyser et corriger des vulnérabilités dans des projets PHP

### Recommandation :

- Pour un projet pédagogique : SonarQube Community suffit largement
- Pour un besoin en sécurité avancée : combiner SonarQube avec Snyk ou Checkmarx

Ce travail a renforcé nos compétences en administration système, en sécurité applicative et en automatisation DevSecOps.

---

**Auteur : Ilan Cado**