SonarQube : analyser la qualité et la sécurité de son code

SonarQube est souvent confondu avec un simple outil de SAST (Static Application Security Testing). En réalité, son cœur de métier est la **qualité de code** — la dette technique, les bugs potentiels, la duplication — avec une couche sécurité qui vient en complément, pas en remplacement d'outils dédiés comme Semgrep ou Trivy.

Installation locale avec Docker

```
docker run -d --name sonarqube -p 9000:9000 sonarqube:lts-community
```

Le démarrage prend une à deux minutes la première fois — SonarQube initialise sa base de données interne. Vérifiez les logs pour confirmer que le serveur est prêt :

```
docker logs sonarqube --tail 20
```

Cherchez la ligne :

```
SonarQube is operational
```

Connectez-vous ensuite sur `http://localhost:9000` avec les identifiants par défaut `admin` / `admin` — un changement de mot de passe sera demandé immédiatement.

Créer un projet et générer un token

Dans l'interface, choisissez **Manually** pour créer un projet sans connecter de plateforme DevOps externe. Renseignez un nom et une clé de projet, puis générez un **token** d'analyse.

Le token ne s'affiche qu'une seule fois — copiez-le immédiatement dans un emplacement sûr. Il sert à authentifier le scanner lors de l'envoi des résultats au serveur.

Installer le scanner

Attention à la compatibilité de version. Si votre instance SonarQube tourne avec Java 11 (cas du Community Edition récent en LTS), la dernière version compatible du SonarScanner CLI est la branche **4.8.x** — les versions plus récentes exigent Java 17 ou supérieur.

Téléchargez directement la version Windows :

```
https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-4.8.1.3023-windows.zip
```

Extrayez l'archive, repérez le dossier `bin/` contenant `sonar-scanner.bat`, et ajoutez ce chemin à la variable d'environnement `PATH` — la même manipulation que pour n'importe quel outil en ligne de commande sur Windows.

Vérifiez l'installation dans un nouveau terminal :

```
        sonar-scanner.bat -h
```

Lancer une analyse

Positionnez-vous dans le dossier racine du projet à analyser, puis lancez :

```
        sonar-scanner.bat -Dsonar.projectKey=mon-projet -Dsonar.sources=. -
Dsonar.host.url=http://localhost:9000 -Dsonar.login=<TOKEN>
```

Le scanner indexe les fichiers, charge les profils de qualité actifs selon le langage détecté, exécute les règles, puis envoie le rapport au serveur :

```
INFO: ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?
id=mon-projet
INFO: EXECUTION SUCCESS
```

Le serveur a ensuite besoin de quelques secondes pour traiter le rapport avant que le dashboard ne s'actualise.

Lire le dashboard

Le dashboard présente cinq métriques principales :

Bugs — erreurs de logique susceptibles de provoquer un comportement incorrect à l'exécution.

Vulnerabilities — failles de sécurité identifiées avec certitude dans le code.

Security Hotspots — points de code jugés sensibles mais qui nécessitent une vérification humaine pour déterminer s'ils constituent un risque réel dans le contexte du projet. Le score de "Security Review" reste à E (le pire) tant qu'aucun hotspot n'a été revu manuellement, indépendamment du nombre réel de hotspots.

Code Smells / Debt — mauvaises pratiques et dette technique, exprimée en temps estimé de correction.

Coverage et Duplications — pourcentage de lignes couvertes par des tests automatisés, et taux de duplication de code.

Le **Quality Gate** synthétise tout cela en un statut Passed/Failed selon des seuils configurables — c'est ce statut qui peut bloquer une pipeline CI/CD si on l'intègre en continu.

Security Hotspots : un exemple concret

Sur l'analyse d'un Dockerfile, deux hotspots typiques remontent fréquemment :

```
        COPY . .
```

"Make sure that recursively copying directories is safe here." — Cette instruction copie tout le contenu du répertoire courant dans l'image, y compris d'éventuels fichiers sensibles si le

`.dockerignore` n'est pas suffisamment strict.

```
FROM python:3.13-slim
# pas de USER déclaré
```

"The python image runs with root as the default user. Make sure it is safe here." — Sans instruction `USER` explicite, le conteneur s'exécute en `root` par défaut.

Ces deux alertes recoupent directement ce que des outils comme Trivy config ou Kubesec détectent côté manifests Kubernetes (`runAsNonRoot` , `readOnlyRootFilesystem`) — SonarQube le signale une étape plus tôt, au niveau du Dockerfile lui-même, avant même le déploiement.

Ce que SonarQube ne fait pas

La Community Edition (gratuite) ne couvre pas le **SCA** (Software Composition Analysis) — l'analyse approfondie des CVE dans les dépendances tierces (`requirements.txt` , `package.json`) reste une fonctionnalité payante des éditions supérieures. Pour cette couverture, des outils comme `bun audit` , `pip-audit` ou Dependabot restent nécessaires.

SonarQube ne scanne pas non plus les images Docker construites — un rôle que remplissent Trivy ou Grype.

Où SonarQube s'insère dans une chaîne de sécurité existante

Outil	Rôle
Gitleaks	Détection de secrets dans le code et l'historique Git
Semgrep	SAST — patterns de vulnérabilités dans le code source
SonarQube	Qualité de code, dette technique, Security Hotspots
`bun audit` / `pip-audit`	SCA — vulnérabilités dans les dépendances
Trivy / Dockle	Scan d'image Docker et bonnes pratiques de configuration

SonarQube ne remplace aucun de ces outils — il ajoute une dimension qualité que les scanners de sécurité purs ne couvrent pas, avec une vue Security Hotspots complémentaire à Semgrep plutôt que redondante.

Avant d'intégrer en CI/CD

Trois prérequis avant de basculer d'un usage local à une intégration continue :

1. Héberger SonarQube en continu plutôt qu'en conteneur local éphémère — sur un VPS dédié ou un service managé.
2. Stocker le token d'analyse comme variable CI/CD masquée et protégée, jamais en clair dans un fichier de configuration versionné.

3. Définir un Quality Gate adapté au projet existant — les seuils par défaut sont souvent trop stricts pour un code déjà en production et peuvent bloquer chaque pipeline sans réelle valeur ajoutée au démarrage.