Ateliers Sonarqube

Pré-requis :

- 16 Go RAM, espace disque libre > 10 %
- Java 17
- Docker, Git
- Node js 15+

Table des matières

Atelier 1 : Installation Sonar, Configuration de prod

Dans ce TP, nous installons une instance Sonar

Objectifs

- Mise en place d'une BD de production avec Sonar
- Configuration JVM

Option 1 : Installation à partir d'une archive

1.1.1 Installation BD

Utiliser le fichier docker-compose fourni et démarrer une base Postgres avec un client pgAdmin comme suit :

```
docker-compose -f postgres-docker-compose.yml up -d
```

Connecter vous à pgAdmin :

- http://localhost:81
- admin@admin.com / admin

Déclarer la connexion au serveur Servers → Register → Server

Donner un nom à la connexion et dans l'onglet Connexion indiquer :

- Host name : ***sonar_postgresql***
- User name : ***postgres***
- Password : ***postgres***

Créer un schéma ***sonarqube*** vide et un utilisateur ***sonarqube*** propriétaire de la base avec le mot de passe ***sonarqube***

1.1.2 Installation Serveur Sonar

1. Récupérer la release LTS fournie et dézipper
2. Modifier *conf/sonar.properties* afin de renseigner l'adresse de la base et les options JVM

adéquates :
    - ***sonar.jdbc.username = sonarqube***
    - ***sonar.jdbc.password=sonarqube***
    - ***sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube***
3. Démarrer le serveur via :
    - ***bin/<YOUR OS>/sonar.sh start***
      OU
    - ***bin/windows-x86-XX/StartSonar.bat***
4. Vérifier les logs d'ElasticSearch :
   `tail -f ${SONAR_HOME}/logs/es.log`
5. Puis ceux du serveur web
   `tail -f ${SONAR_HOME}/logs/server*.log`

Combien de processus Java sont démarrés ?

Accéder au serveur sur ***localhost:9000***

Visualiser les tables de la bases de données dans la base *Postgres*

Option 2 : Démarrage via docker-compose

Utiliser le fichier docker-compose.yml fourni

Vérifier le démarrage du serveur avec

docker logs -f sonarqube

Lorsque le serveur est démarré, vous pouvez visualiser les tables utilisées par SonarQube via pgAdmin :

Connecter vous à pgAdmin :

- http://localhost:81
- admin@admin.com / admin

Déclarer la connexion au serveur *Servers → Register → Server*

Donner un nom à la connexion et dans l'onglet Connexion indiquer :

- Host name : ***db***
- User name : ***sonar***
- Password : ***sonar***

Atelier 2 : Première analyse

Dans ce TP, nous effectuons la première analyse d'un projet avec différentes technologies via le scanner sonar

Objectifs

- Installer *sonar-scanner*
- Générer un jeton d'analyse
- Exécuter une première analyse
- Prendre en main l'interface Web Sonar

2.1 Installation de sonar-scanner

Télécharger une distribution de ***sonar-scanner*** :

https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/

Dézipper et mettre le répertoire dans votre PATH
Dans un terminal Linux :

export PATH=$PATH:<sonar-scanner-home>/bin

Tester de votre répertoire HOME

sonar-scannner -v

2.2 Création jeton d'analyse

- S'authentifier avec l'utilisateur ***admin*** sur sonarqube
- Dans les préférences du compte, Naviguer sur l'onglet ***Security***
- Générer un token de type *Global Analysis*
- Conserver ce jeton

2.3 Analyse d'un projet multi langages

Récupérer le projet fourni, visualiser le fichier ***sonar-project.properties***

Dans le répertoire du projet, exécuter :

sonar-scanner -Dsonar.token=<TOKEN>

Observer les traces de l'analyse

Visualisez ensuite les résultats sur le serveur SonarQube :

1. Visualiser les règles activées, le statut de la porte qualité

2. Visualiser les issues
3. Accéder au code source
4. Visualiser tous les métriques de l'analyse
5. Visualiser les « background tasks »

Atelier 3 : Administration

Intégration serveur de mail

Configurer le serveur SonarQube avec un serveur *smtp*.
Configuration → General

Éventuellement, celui-ci :

Compte : **stageojen@plbformation.com**

Password : **stageojen**

Serveur sortant :**smtp.plbformation.com**, port **587**

Création d'utilisateur, permission *sonar-user*

Avec le compte *admin*, ajouter un utilisateur et lui donner votre email

Administration → Security → Users

Se logger avec le nouvel utilisateur visualiser les différences avec l'interface admin

Activer les notifications concernant les *issues*

My Account → Notifications

Issues et HotSpots

Atelier 4 : Workflow des issues

Supprimer le projet précédent dans Sonarqube

Configurer votre client Git avec votre adresse email et votre identité

Initialiser un dépôt git dans le répertoire *sonar-scanning* :

```
git init
```

```
git add .
```

```
git commit -m 'Initial commit'
```

Effectuer une première analyse

Modifier ensuite le code source pour y ajouter 2 issues.

Committer et relancer une analyse.

Vérifier les notifications emails

Accéder à SonarQube en tant qu'admin et :

- Accepter une issue
- Marquer l'autre comme faux-positif

Relancer une analyse et viusualiser le statut des issues

Atelier 5 : Personnalisation projet

Objectifs

- Configuration Maven
- Configuration de la couverture de test
- Création d'un profil qualité
- Exclusion/Inclusion
- Création d'une porte qualité

5.1 Configuration sources et couverture de test

Décompresser les sources du projet fourni. Il s'agit d'un projet Java11/Angular5 utilisant Maven comme outil de build Java et ***ng*** + ***npm*** pour outil de build typescript.

Initialiser un dépôt et faire le premier commit

Visualiser le *pom.xml*

Ajouter une configuration du plugin Sonar org.sonarsource.scanner.maven:sonar-maven-plugin en allant rechercher la dernière version sur Maven central

Exécuter l'analyse via le plugin Maven :

```
./mvnw -Dsonar.token=<token>  clean test sonar:sonar
```

Observez les résultats :

- il n'y a pas de calcul de la couverture de test.
- Les fichiers typescript ne sont pas détectés

Modifier la configuration via le *pom.xml* afin que les fichiers typescript soient pris en compte

Pour configurer la couverture des tests avec *jacoco*, modifier le fichier *pom.xml* en ajoutant des phases de build pour *jacoco* (https://www.jacoco.org/jacoco/trunk/doc/examples/build/pom.xml)

Dans la balise ***build/plugins*** ajouter :

```
<plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>${jacoco-maven-plugin.version}</version>
      <executions>
            <execution>
```

```
                <id>pre-unit-tests</id>
                <goals>
                      <goal>prepare-agent</goal>
                </goals>
          </execution>
      <!-- Ensures that the code coverage report for unit tests is created
           after unit tests have been run -->
          <execution>
                <id>post-unit-test</id>
                <phase>test</phase>
                <goals>
                      <goal>report</goal>
                </goals>
          </execution>
      </executions>
</plugin>
```

Relancer l'analyse et observer les résultats
Si ils sont corrects, committer vos changements dans votre repository git

5.2 Exclusions

Nous voulons exclure le code source de l'analyse :

- Toutes les classes présentes dans les packages ***entities***
- Tous les fichiers ***css***
- Le fichier ***polyfill.ts***

Vérifier vos configuration en lançant l'analyse.

Observer les changement dans le menu ***Activity*** du projet

Si c'est correct, committer

Nous voulons exclure de la couverture de test les packages :

- Toutes les classes présentes dans les packages ***entities et repository***
- La partie Angular

Vérifier vos configuration en lançant l'analyse.

Si c'est correct, committer

5.3 Création d'un profil qualité

1. Créer un profil en copiant le profil SonarWay
2. Désactiver toutes les règles générant des Code Smells
3. Créer un 2ème profil héritant du premier
4. Activer toutes les règles Java sauf les règles dépréciées

Relancer une analyse

Reprendre le profil SonarWay

Retrouver l'identifiant de la règle qui interdit de déclarer 2 variables sur la même ligne.

Désactiver la ainsi que la règle *java:S100* pour les classes du package ***service***

Relancer une analyse, si vous êtes satisfait committer

5.4 Création d'une porte qualité

Créer une nouvelle porte qualité à partir de SonarWay

- Baisser le taux de couverture de test
- Ajouter une contrainte au niveau de la documentation
- Ajouter une contrainte au niveau de la dette technique
- Ajouter une contrainte au niveau de la complexité d'une méthode

Relancer une analyse

Atelier 6 : Règle personnalisée

Objectifs

- Création d'une règle à partir d'un template
- Création d'un règle custom en Java

6.1 A partir d'un gabarit

Créer une nouvelle règle à partir du template « *Track uses of disallowed classes* »

Générer un code smell lors de l'utilisation de la classe *java.util.Date*

Activer la règle dans un nouveau profil qualité et l'associer au projet

6.2 Règle codée

6.2.1 Reprise des exemples

Récupérer les exemples fournis par SonarQube :

```
git clone https://github.com/SonarSource/sonar-java
```

Construire le projet Maven :

mvn install

Copier le jar contenant les nouvelles règles dans le répertoire d'extensions

cp docs/java-custom-rules-example/target/java-custom-rules-example-1.0.0-SNAPSHOT.jar SONAR_HOME/extensions/plugins

Redémarrer le serveur et visualiser le nouvelles règles

Rules → Repository → My Company Custom Repository

6.2.2 Écriture d'une nouvelle règle

Nous voulons ajouter une règle de type *Code Smells* qui vérifie que le nombre d'arguments des méthodes doivent être inférieur à 4

Voir https://docs.sonarqube.org/display/PLUG/Writing+Custom+Java+Rules+101

Atelier 7 : SonarLint

Dans votre IDE, importer le projet Maven des Tps précédents.

Installer SonarLint via le MarketPlace

Configurer SonarLint :

Exemple Eclipse

- Déclarer le serveur
  Window > Preferences > SonarQube > Servers.

- Associer le projet au projet sur SonarQube
  Project Explorer, Click-Droit **Configure > Associate with SonarQube**.

- Visualiser les fenêtres *SonarLint*

- Traiter quelques issues dans l'IDE

Atelier 8 : Intégration Jenkins

Dans ce TP, nous allons implémenter différents jobs axés sur certains types de tests (unitaire, intégration, couverture des tests, …) .

.

Objectifs

- Chaîner les tests d'intégration après les tests unitaires
- Intégrer l'analyseur de qualité : Sonar

8.1 Installation Jenkins

Récupérer une instance de Jenkins *generic*, décompresser et démarrer le serveur.
Utiliser les plugins proposés

8.2 Intégration Sonar

8.2.1 Plugin Jenkins

Installation du plugin SonarQube Scanner
Définir dans la configuration du système et la configuration globae des outils :

- L'adresse du serveur Sonar
- Le scanner à utiliser (installation automatique)

8.2.2 Job Freestyle

1. Créer un nouveau job freestyle
2. Le job doit lancer l'exécution des tests par Maven
3. Ensuite, il utilise le scanner Sonar. Mettre en place en fichier *sonar-project.properties* fixant les propriétés de l'analyse
4. Faire en sorte de faire échouer la pipeline si la porte qualité échoue

8.2.3 Pipeline

Installer le plugin *Pipeline Utility Steps*
Récupérer le fichier Jenkinsfile fourni et visualiser la fonction Groovy inclut

1. Créer un job de type pipeline
2. Faire en sorte que le job effectue des tâches après que la porte qualité soit passée

Atelier 9 : Intégration Plateforme DevOps

Choisir une plateforme DevOps pour effectuer l'intégration