Ateliers Sonarqube

Pré-requis :

- 16 Go RAM, espace disque libre > 10 %
- JRE11
- Docker, Git

TP1 : 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

1. 1 Installation BD

Utiliser le fichier docker-compose fourni et démarrer une base Postgres comme suit :

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

Connecter vous via pgAdmin (localhost:81)

Créer un schéma sonar vide et un utilisateur ***sonarqube***

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
3. Démarrer le serveur
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

TP2 : Première analyse

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

Objectifs

- Installer sonar-scanner
- Exécuter une première analyse
- Prendre en main l'interface Web Sonar

Pré-requis Installation JDK8

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 Analyse d'un projet multi langages

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

Dans le répertoire du projet, exécuter :

sonar-scanner -Dsonar.login=admin -Dsonar.password=<adminpassword>

Visualisez les résultats sur le serveur SonarQube

Vous pouvez également exécuter le scanner en mode DEBUG pour visualiser toutes les détections de technologies :

sonar-scanner -X -Dsonar.login=admin -Dsonar.password=<adminpassword>

2.3 Accéder à l'interface

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

2.3 Workflow

Configurer Git avec votre adresse email et votre identité

Configurer le serveur avec un serveur smtp.

Éventuellement, celui-ci :

Compte : stageojen@plbformation.com

Password : stageojen

Serveur sortant :smtp.plbformation.com, port 587

Avec le compte *admin*, Ajouter un utilisateur et lui donner le même email que celui de votre identité Git

Se logger avec le nouvel utilisateur et activer les notifications

Modifier un fichier source pour y ajouter une issue

Effectuer un commit avec votre utilisateur Git

Ré-exécuter une analyse

Jouer un workflow avec les compte admin et le nouveau compte

TP3 : Personnalisation projet

Objectifs

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

3.1 Configuration couverture de test

Décompresser les sources du projet fourni. Il s'agit d'un projet Java8/Angular5 utilisant Maven comme outil de build.

Initialiser un dépôt et faire le premier commit

Visualiser le *pom.xml* et la configuration du plugin Sonar

Exécuter l'analyse via le plugin Maven :

```
./mvnw -Dsonar.login=admin -Dsonar.password=<password-admin> clean
test sonar:sonar
```

Exécuter ensuite l'analyse en mode DEBUG avec

```
./mvnw -X -Dsonar.login=admin -Dsonar.password=<password-admin> -
Dsonar.verbose=true clean test sonar:sonar
```

Observez les résultats, il n'y a pas de calcul de la couverture de test.

Pour configurer la couverture des tests avec *jacoco*, Récupérer le fichier *pom.xml* fourni. Les différences avec la version précédente sont

- Ajout du plugin Maven pour générer le rapport jacoco
- Ajout des propriétés nécessaires pour Sonar

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

3.2 Exclusions

Nous voulons exclure les tâches suivantes de la couverture de test :

- Toutes les classes contenant «Views»

- `OpenAPIConfig`.java

- Toutes les classes présentes dans des packages model et dto

Vérifier vos configuration en lançant l'analyse.

Si c'est correct, committer vos changements dans votre repository git

Exclure de la couverture de test les packages *com.plb.plbsiapi*, *com.plb.util* ainsi que *com.plb.plbsiapi.cms* et ses sous-packages

Exclure la partie Angular *plbsi-ui/src*

3.3 Création d'un profil qualité

1. Créer un profil en copiant le profil SonarWay
2. 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 attributs sur la même ligne.
Désactiver la pour les classes du package *model*

Relancer une analyse, si vous êtes satisfait committer

3.4 Création d'une porte qualité

Créer un nouveau profil 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

Relancer une analyse

TP4 : Règle personnalisée

Objectifs

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

4.1 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

4.2 Règle codée

4.2.1 Reprise des exemples

Récupérer les exemples fournis par SonarQube, créer un jar via Maven : *mvn package*

Copier le jar dans le répertoire *SONAR_HOME/extensions/plugins*

Redémarrer le serveur et visualiser le nouvelles règles

4.2.2 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

TP5 : SonarLint

Dans Eclipse, importer le projet Maven des Tps précédents.

Installer SonarLint via le MarketPlace

Configurer SonarLint :

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

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

TP6 : 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

6.1 Installation Jenkins

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

6.2 Intégration Sonar

6.2.1 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)

6.2.2 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

6.2.3 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