Sommaire

[! NOTE] Ce fichier est une documentation pour la mise en place de pipelines CI/CD avec Docker, des GitActions via GitHub et l'analyse de code via Sonar Cloud

1 - Introduction

Pour mettre en place le pipeline j'ai utilisé les outils suivants :

- ## GitHub Actions

GitHub Actions est un service d'intégration et de déploiement continu (CI/CD) qui permet d'automatiser les tâches de développement telles que la création, les tests et le déploiement de code directement depuis GitHub. Les actions sont des scripts qui s'exécutent automatiquement en réponse à des événements spécifiques sur un dépôt GitHub (comme la création d'une pull request, le push de code, la création d'une release, etc.).

- ## Docker Hub

Docker est une plateforme open-source qui permet de créer, de déployer et de gérer des applications dans des conteneurs. Les conteneurs sont des environnements isolés qui contiennent tout ce dont une application a besoin pour s'exécuter (code, dépendances, bibliothèques, etc.). Docker permet de créer des conteneurs légers et portables qui peuvent s'exécuter sur n'importe quel système d'exploitation.

- ## SonarCloud

SonarCloud est un service d'analyse de code en ligne qui permet de détecter les problèmes de qualité du code et de les corriger avant qu'ils ne deviennent des problèmes plus graves. SonarCloud analyse le code source d'un projet et identifie les erreurs, les bugs, les vulnérabilités, les duplications de code, les mauvaises pratiques, etc. SonarCloud fournit des rapports détaillés sur la qualité du code et des recommandations pour l'améliorer.

2 – Post installation pipeline CI/CD

Pour mettre en place un pipeline CI/CD avec Docker, Git Actions et Sonar Cloud, suivez les étapes suivantes :

Créer un compte Docker Hub

Pour créer un compte Docker Hub, rendez-vous sur le site Docker Hub et cliquez sur le bouton "Sign Up" pour créer un compte.

Créer un compte Sonar Cloud

Pour créer un compte Sonar Cloud, rendez-vous sur le site Sonar Cloud et cliquez sur le bouton "Log In" pour créer un compte.

Créer un dossier contenant les GitHub actions

Créez un dossier. GitHub/workflows à la racine de votre projet et ajoutez-y les fichiers ****.yml contenant les actions à exécuter lors des événements spécifiques sur votre dépôt GitHub.

3 - Configuration du pipeline

Configuration des secrets dans GitHub

Pour que les actions puissent accéder à Sonar Cloud, on doit ajouter les secrets: Accéder aux paramètres de votre repository GitHub : Dans le menu de gauche, cliquez sur "Secrets and variables" puis sur "Actions". Cliquez sur "New repository secret" (Nouveau secret de repository). Ajoutez les secrets suivants : DOCKER_USERNAME : Votre nom d'utilisateur Docker Hub. DOCKER_PASSWORD : Votre mot de passe Docker Hub. SONAR_TOKEN_FRONTEND : Votre jeton d'accès Sonar Cloud pour le frontend (généré dans Sonar Cloud sous "My Account" > "Security"). SONAR_TOKEN_BACKEND : Votre jeton d'accès Sonar Cloud pour le backend (généré dans Sonar Cloud sous "My Account" > "Security"). SONAR_HOST_URL : L'URL de votre instance Sonar Cloud (par exemple, https://sonarcloud.io).

Configuration de sonar cloud :

a. Frontend

- Fichiers utilisés :

- .GitHub/workflows/CI-sonar Cloud-frontend.yml
- front/sonar-project.properties

- Fichier GitHub Actions : CI-sonar Cloud-frontend. Yml

Ce fichier permet de lancer l'analyse Sonar Cloud pour le frontend (Angular). Il est déclenché automatiquement à chaque push ou pull request.

- Principles étapes:

```
name: SonarCloud - Frontend test and analysis (CI)

on:
  push:
    branches:
      - main
  pull request:
    types: [opened, synchronize, reopened]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Setup Node.js
        uses: actions/setup-node@v4
        with:
          node-version: '16'
```

```
- name: Install dependencies
  run: npm install
  working-directory: front

- name: Run tests and generate coverage
  run: npm run test: coverage
  working-directory: front

- name: Sonar Cloud Scan
  uses: sonar source/sonarcloud-github-action@v2
  with:
    projectable: front
  env:
    SONAR_TOKEN: ${{secrets. SONAR_TOKEN_FRONTEND}}
```

- 🔍 Explication :

- Checkout : récupère le code.
- Setup-Node : installe Node.js (v16).
- Npm Install : installe les dépendances Angular.
- Test : coverage : lance les tests et génère le rapport de couverture (lcov.info).
- Sonar cloud-GitHub-action : déclenche l'analyse via l'API Sonar Cloud.

- Fichier sonar-Project. Properties (dans le dossier front/)

```
Sonar. Project Key=bobappfront
Sonar. Organisation=jelalisameh
Sonar. Project Name=BobappFront
Sonar. Sources=src
Sonar. Language=js
sonar.host.url=https://sonarcloud.io
sonar.javascript.lcov.reportPaths=coverage/bobapp/lcov.info
sonar. Token=SONAR_TOKEN_FRONTEND
```

- 🔍 Explication :

- Sonar. Project Key : identifiant unique de ton projet sur Sonar Cloud.
- Sonar. Sources : dossier analysé (src).
- Lcov. ReportPaths : chemin vers le rapport de couverture des tests.
- Sonar. Token : clé d'accès, à stocker dans les secrets GitHub.

b. Backend

- Fichiers utilisés :

- GitHub/workflows/CI-sonar Cloud-backend
- back/sonar-project.properties

- Fichier GitHub Actions : CI-sonar Cloud-backend. Yml

Ce fichier permet d'analyser le code Java backend avec Sonar Cloud.

- Principles étapes:

```
name: Sonar Cloud - Backend test and analysis (CI)

on:
  push:
    branches:
      - main
  pull request:
    types: [opened, synchronize, reopened]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Set up JDK 11
        uses: actions/setup-java@v4
```

```
  with:
    java-version: '11'
    distribution: 'adopt'

- name: Build with Maven (with tests)
  run: mvn clean verify
  working-directory: back

- name: Sonar Cloud Scan
  uses: sonar source/sonarcloud-github-action@v2
  with:
    projectBaseDir: back
  env:
    SONAR_TOKEN: ${{secrets. SONAR_TOKEN_BACKEND}}
```

- 🔍 Explication :

- Setup-java : configure l'environnement Java 11.
- mvn clean verify : compile le code, exécute les tests, et génère le rapport de couverture.
- Sonarcloud-github-action : analyse le code et envoie le rapport à Sonar Cloud.

- Fichier sonar-Project. Properties (dans le dossier back/)

```
sonar. project Key=bobappback
sonar. Organization=jelalisameh
sonar. project Name=Bobappback
sonar. Sources=src
sonar. Language=java
sonar.host.url=https://sonarcloud.io
sonar. Token=SONAR_TOKEN_BACKEND
```

- 🔍 Explication :

- Le fichier est similaire à celui du frontend mais adapté à Java.
- Sonar. Sources=src indique à Sonar Cloud d'analyser tous les fichiers Java sous src.
- Le sonar. Token permet l'authentification sécurisée (variable secrète GitHub).

Configuration de Docker

a. Front

Fichier GitHub Actions: Deploy Frontend to Docker Hub (CD)

Ce workflow est déclenché **automatiquement** quand le workflow **CI de Sonar Cloud pour le frontend** est **terminé avec succès** (workflow_run).

Étapes détaillées :

1. **Checkout code**
   - Récupère le code source de la branche.
2. **Set up Node.js**
   - Configure l'environnement Node.js avec la version 16 (Angular a besoin de Node pour builder le projet).
3. **Install dependencies**
   - Installe les dépendances avec npm install dans le dossier front.
4. **Build project**
   - Construit l'application Angular avec npm run build.
5. **Login to Docker Hub**
   - Se connecte à Docker Hub avec les identifiants stockés dans les secrets GitHub (DOCKER_USERNAME et DOCKER_PASSWORD).
6. **Build and push Docker image**
   - Construit une image Docker à partir du Docker file et pousse l'image sur Docker Hub sous le nom :
     bobapp-frontend : latest.

Docker file – Frontend

```
# Étape 1: Builder application Angular
FROM node:16-alpine AS build
WORKDIR /app
COPY package. Json package-lock. Json. /
RUN npm install --ignore-scripts
```

COPY src. /Src
COPY angular. json. /
COPY tsconfig. json. /
COPY tsconfig.app. json ./
RUN npm run build

- **Objectif** : Construire l'app Angular dans un environnement optimisé (image Node :16-alpine).
- Les fichiers dist/bobapp seront générés à cette étape.

Étape 2 : Déployer avec Nginx
FROM nginx: alpine
COPY --from=build /app/dist/bobapp /usr/share/nginx/html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

- **Objectif** : Servir les fichiers Angular via Nginx.
- **Nginx** est utilisé comme serveur HTTP pour afficher l'app en production.
- Le port 80 est exposé (classique pour HTTP).

b. Backend

- Fichier GitHub Actions: Deploy Backend to Docker Hub (CD)

Déclenché automatiquement après que **le workflow CI de Sonar Cloud pour le backend soit terminé**.

- Étapes détaillées :

1. **Checkout code**
   - Récupère le code source dans le dossier back.
2. **Set up JDK**
   - Configure Java 11 via AdoptOpenJDK.
3. **Build with Maven**
   - Exécute la commande mvn clean install -DskipTests pour construire le JAR (sans lancer les tests).
4. **Login to Docker Hub**

- Authentifie l'accès à Docker Hub.

5. **Build and push Docker image**
   - Construit une image Docker depuis back/Docker file et la pousse sous le tag bobapp-backend : latest.

Dockerfile – Backend

```
# Étape 1 : Builder le projet Java avec Maven
FROM maven:3.6.3-jdk-11-slim AS build
RUN mkdir -p /workspace
WORKDIR /workspace
COPY pom.xml /workspace
COPY src /workspace/src
RUN mvn -B -f pom.xml clean package -DskipTests
```

- **Objectif** : Builder le JAR du projet Java avec Maven (environnement propre).

```
# Étape 2 : Exécuter le backend avec Java
FROM openjdk:11-jdk-slim
COPY --from=build /workspace/target/*.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

- **Objectif** : Créer une image plus légère uniquement avec le runtime Java.
- Le fichier app.jar est le livrable du projet Spring Boot.
- L'application sera disponible sur le port 8080 (par défaut pour Spring Boot).

4 - Ajout des KPIs (via Sonar Cloud et des Quality Gates)

Ajout de KPIs (Key Performance Indicators) au projet via des Quality Gates

KPI num 1: Coverage

Le KPI num 1 est la couverture de code. Il est mesuré par JaCoCo et affiché dans Sonar Cloud. Le Quality Gate pour ce KPI est de 80%. Ça signifie que le code doit avoir une couverture de code de 80% ou plus pour passer le Quality Gate.

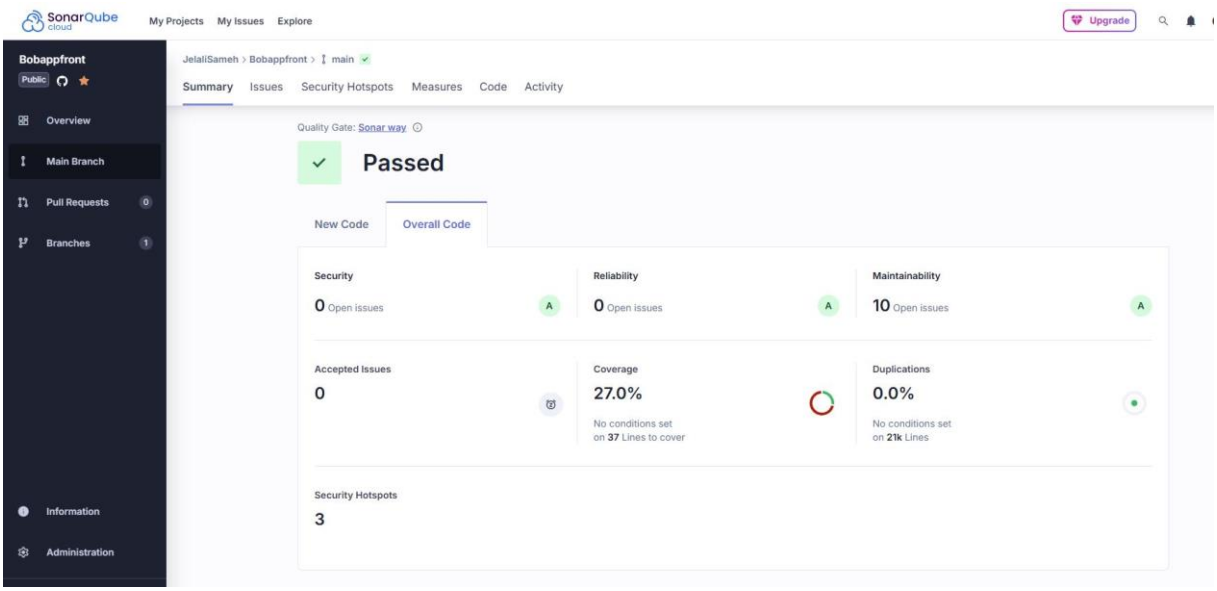
o KPI num 2: Security hotspots Reviewed

Le KPI num 3 est le nombre de Security Hotspots Reviewed. Un Security Hotspot est un indicateur de risque potentiel dans le code qui nécessite une attention particulière pour garantir qu'il ne devienne pas une vulnérabilité de sécurité. Il est mesuré par Sonar Cloud et affiché dans le tableau de bord. Le Quality Gate pour ce KPI est de moins de 100%. Ca signifie qu'il ne doit pas y avoir de Security Hotspots Reviewed dans le code pour passer le Quality Gate.

5 - Analyse des métriques et retours utilisateurs

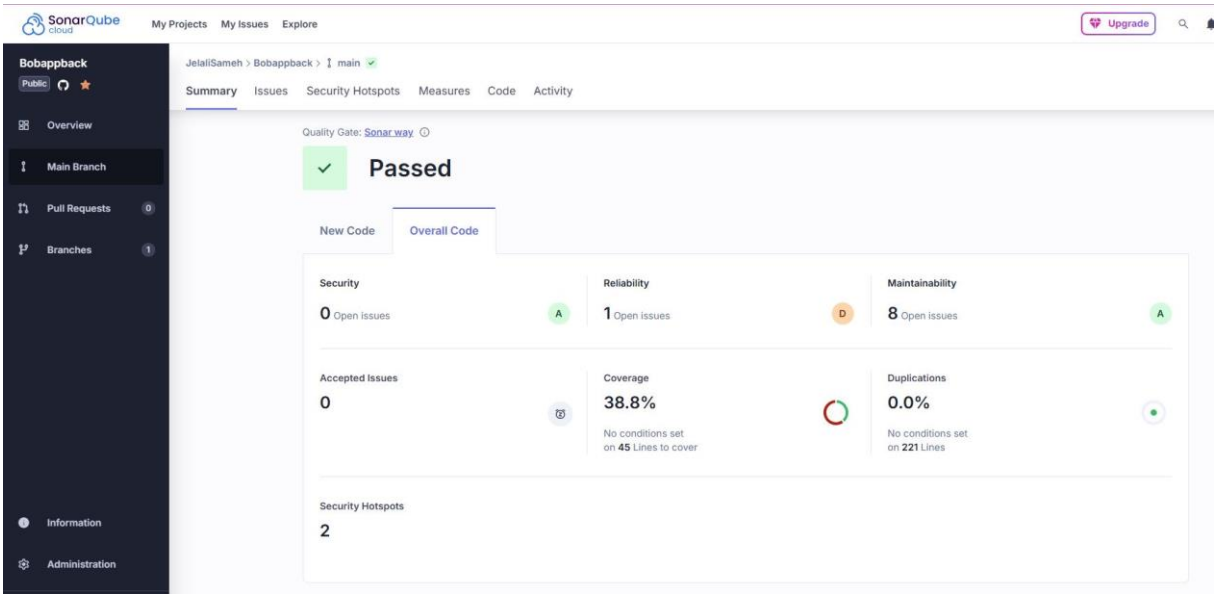
o Métriques

a. Couverture de code frontend



Le coverage est ici de 27% ce qui est très faible. Il faudrait augmenter la couverture de code pour passer le Quality Gate. Il y a aussi 1 Security hotspot reviewed qui doit être corrigé pour passer le Quality Gate.

b. Couverture de code backend



Le coverage est de 38,8% ce qui est également faible. Il faudrait augmenter la couverture de code pour passer le Quality Gate. Il y a aussi 2 Security hotspots reviewed qui doivent être corrigés pour passer le Quality Gate.

c. Retours utilisateurs.

- Avis 1

"Je mets une étoile car je ne peux pas en mettre zéro ! Impossible de poster une suggestion de blague, le bouton tourne et fait planter mon navigateur !"

- **Problème identifié** : Blocage de l'interface utilisateur lors de l'envoi d'une suggestion, entraînant un crash navigateur.

- **Solution proposée** : Mise en place de tests end-to-end automatisés (via Cypress) pour détecter les erreurs de comportement et les boucles infinies. Surveillance du front via des outils de performance pour améliorer la stabilité.

- Avis 2

"#BobApp j'ai remonté un bug sur le post de vidéo il y a deux semaines et il est encore présent ! Les devs vous faites quoi ???? "

- **Problème identifié** : Bug non corrigé malgré un signalement antérieur, signe d'un manque de réactivité de l'équipe technique.
- **Solution proposée** : Intégration continue (CI) avec analyse de code statique (SonarQube) pour détecter les anomalies. Mise en place d'un processus de tri, de suivi et de résolution des bugs via un backlog géré en Agile.

- Avis 3

"Ça fait une semaine que je ne reçois plus rien, j'ai envoyé un email il y a 5 jours mais toujours pas de nouvelles…"

- **Problème identifié** : Défaut de communication ou panne du service de notification/email.
- **Solution proposée** : Mise en place de KPIs pour surveiller l'état du service en temps réel (via un dashboard ou Grafana). Ajout de tests automatisés pour vérifier la disponibilité de l'API d'envoi d'e-mails à chaque déploiement.

- Avis 4

"J'ai supprimé ce site de mes favoris ce matin, dommage, vraiment dommage."

- **Problème identifié** : Insatisfaction globale de l'utilisateur, possiblement liée à une accumulation de bugs non résolus ou à une mauvaise expérience.

- Solution proposée : Mise en place d'un formulaire de retour ou d'un outil de feedback utilisateur intégré à l'application. Suivi de la satisfaction utilisateur avec des indicateurs (Net Promoter Score, taux de rétention, etc.).

⇨ Analyse générale :

Les premiers avis reflètent des problèmes de performance et de stabilité : ils soulignent l'importance de mettre en place une chaîne CI/CD robuste avec des tests automatisés (unitaires, d'intégration et e2e), et une bonne gestion des bugs en amont.

Les autres avis montrent un manque de communication entre l'équipe technique et les utilisateurs. Pour y répondre, il est essentiel de mettre en place une stratégie de feedback utilisateur, un suivi des incidents, et des indicateurs de qualité de service.

> [! NOTE] Pour mettre en place une stratégie CI/CD efficace, il est important de suivre les KPIs, d'analyser les retours utilisateurs et de corriger les problèmes rapidement. Il serait judicieux de modifier les GitHub actions de CD pour que le déploiement ne se fasse que si les Quality Gates sont passés.