ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Rapport Architecture Microservices
Année 2023/2024

Conception et Implémentation de Blog Personnel en utilisant l'architectutre microservice.

Ingénierie Informatique et Réseaux

Réalisé par :

KANBOUA Khaoula

BOUOUGRI Reda

NEJMEDDINE Abdellah

PLAN

1-Introduction

1. Aperçu du projet :

L'objectif ultime de ce blog est de créer une atmosphère accueillante et interactive où les lecteurs peuvent se connecter, réagir et peut-être même trouver une source d'inspiration dans les articles.

2. Importance de l'architecture ùicroservices:

L'architecture microservices est devenue de plus en plus importante dans le développement logiciel en raison de plusieurs avantages qu'elle offre par rapport aux architectures monolithiques traditionnelles. Voici quelques points clés sur l'importance de l'architecture microservices :

1. **Scalabilité :** Les microservices permettent une meilleure scalabilité. Chaque service peut être déployé, mis à l'échelle et mis à jour indépendamment des autres. Cela permet de gérer efficacement les charges de travail variables et de faire évoluer les parties spécifiques du système qui nécessitent plus de ressources sans affecter l'ensemble du système.

2. **Déploiement continu :** L'architecture microservices favorise les pratiques de déploiement continu. En raison de leur indépendance, les microservices peuvent être déployés de manière séparée, facilitant ainsi la livraison continue et la mise en production plus fréquente de nouvelles fonctionnalités.

3. **Flexibilité technologique :** Chaque microservice peut être développé, testé et déployé indépendamment, ce qui permet l'utilisation de différentes technologies et frameworks pour chaque service. Cela offre une flexibilité technique qui peut être bénéfique pour l'innovation et l'adaptation aux changements technologiques.

4. **Isolation des erreurs :** En cas d'échec d'un microservice, les autres services peuvent continuer à fonctionner normalement. Cela isole les erreurs et améliore la résilience du système dans son ensemble.

5. **Développement par équipes distribuées :** Les microservices facilitent le travail d'équipes distribuées. Chaque équipe peut être responsable d'un ou plusieurs microservices, ce qui permet une gestion efficace des équipes géographiquement dispersées.

6. **Réutilisation des composants :** Les microservices peuvent être conçus comme des services réutilisables, favorisant ainsi la modularité et la réutilisation des composants logiciels. Cela peut accélérer le développement de nouvelles fonctionnalités en tirant parti des services existants.

7. **Évolutivité indépendante :** Les microservices offrent la possibilité de faire évoluer différentes parties du système indépendamment les unes des autres. Cela permet d'ajuster la capacité et les fonctionnalités spécifiques sans nécessiter une modification globale du système.

8. **Meilleure gestion des données** : Les microservices peuvent avoir leurs propres bases de données, ce qui simplifie la gestion des données spécifiques à chaque service. Cela peut également améliorer les performances en évitant les goulots d'étranglement liés à une base de données monolithique.

En résumé, l'architecture microservices apporte une agilité, une scalabilité et une résilience accrue, ce qui en fait une option attrayante pour les entreprises cherchant à développer des applications flexibles et évolutives dans un environnement en constante évolution. Cependant, il est important de noter que l'adoption des microservices présente également des défis tels que la gestion de la complexité accrue et la nécessité d'une infrastructure appropriée.

2 - Architecture Microservices

1. Architecture Microservices :

Notre architecture repose sur le modèle de microservices, une approche modulaire où chaque composant, appelé microservice, est conçu pour exécuter une tâche spécifique de manière autonome. Cette architecture offre la flexibilité nécessaire pour développer, déployer et évoluer chaque service indépendamment.

2. Description des services :

a. Service d'authentification avec Spring Security :

Le microservice d'authentification utilise Spring Security pour gérer l'authentification des utilisateurs. Il stocke les informations d'identification et gère les sessions utilisateur, garantissant un accès sécurisé à notre application.

b. Microservice des posts :

Ce microservice est dédié à la gestion des publications sur le blog, permettant des opérations telles que la création, la modification et la suppression de publications. Il assure également la récupération des publications pour les afficher sur le blog.

c. Microservice des commentaires :

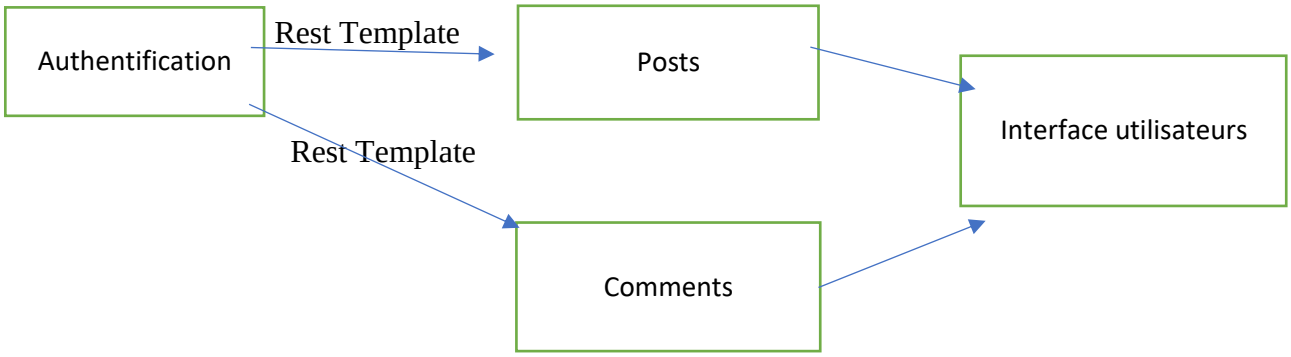
Responsable de la gestion des commentaires associés aux publications du blog, ce microservice offre des fonctionnalités de création, modification et suppression de commentaires. Il assure également la récupération des commentaires liés à une publication spécifique.

3. Mécanismes de communication :

Nous avons choisi d'adopter RestTemplate pour faciliter la communication entre nos microservices. Cette classe de Spring Framework simplifie les interactions avec les services RESTful en permettant l'envoi de requêtes HTTP synchrones. Bien que cette approche soit efficace, nous envisageons également des mécanismes asynchrones, tels que l'utilisation de la messagerie avec des files d'attente, pour améliorer la résilience et la scalabilité du système.

3- Conception des Microservices

La conception des microservices nécessite une approche réfléchie pour garantir une architecture efficace, évolutive et maintenable. Chaque microservice doit être conçu de manière à fonctionner de manière autonome tout en collaborant avec les autres services pour fournir la fonctionnalité globale de l'application. Voici une approche de conception générale pour chaque service dans votre cas spécifique avec trois microservices : authentification, publication de blogs et commentaires.



Microservice d'Authentification avec Spring Security:

Objectif : Gérer l'authentification et l'autorisation des utilisateurs.
Points clés :

- Utilisez Spring Security pour gérer l'authentification et l'autorisation.
- Stockez les informations d'identification de manière sécurisée (hachage de mot de passe).
- Émettez et gérez les jetons JWT (JSON Web Tokens) pour l'authentification stateless.
- Intégrez des fonctionnalités de gestion des sessions si nécessaire.

Microservice de Publication de Blog:

Objectif : Gérer la publication et la récupération des articles de blog personnels.
Points clés :

- Définissez une API REST pour les opérations de publication, récupération, mise à jour et suppression de blogs.

- Utilisez une base de données adaptée pour stocker les articles de blog (par exemple, MongoDB, MySQL).
- Intégrez la logique métier pour valider les autorisations avant de permettre la publication ou la modification d'un blog.
- Gérez les événements asynchrones si nécessaire (par exemple, notifications de nouveaux blogs).

Microservice de Commentaires pour les Blogs:

Objectif : Gérer les commentaires pour les articles de blog.
Points clés :

- Définissez une API REST pour ajouter, récupérer et supprimer des commentaires.
- Utilisez une base de données spécifique pour stocker les commentaires associés aux blogs.
- Assurez-vous que les commentaires sont liés de manière appropriée aux articles de blog.
- Implémentez des mécanismes de gestion des autorisations pour les commentaires.
- Communication entre Microservices :

Sécurité :

Mettez en œuvre des mécanismes de sécurité pour protéger les microservices, y compris la validation des données d'entrée, la protection contre les attaques par injection, etc.

Déploiement et Évolutivité :

Utilisez des conteneurs (Docker) pour garantir une portabilité et une gestion plus faciles. Mettez en place un système de déploiement automatisé pour simplifier le processus de mise à jour. Concevez les microservices de manière à pouvoir évoluer indépendamment les uns des autres.

4-Conteneurisation avec Docker

1. Création de docker file pour chaque Microservice :

a) docker file de MicroService d'authentification

```
FROM maven:3.8.4-openjdk-17 AS builder
WORKDIR /app
COPY ./src ./src
COPY ./pom.xml .
RUN mvn clean package

FROM openjdk:17-jdk-alpine
VOLUME /tmp
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} devoire.jar
ENTRYPOINT ["java","-jar","/devoire.jar"]
```

b) docker file de MicroService des posts

```
FROM maven:3.8.4-openjdk-17 AS builder
WORKDIR /app
COPY ./src ./src
COPY ./pom.xml .
RUN mvn clean package

FROM openjdk:17-jdk-alpine
VOLUME /tmp
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} BlogComment.jar
ENTRYPOINT ["java","-jar","/BlogComment.jar"]
```

c) docker file de MicroService des comment

```
FROM maven:3.8.4-openjdk-17 AS builder
WORKDIR /app
COPY ./src ./src
COPY ./pom.xml .
RUN mvn clean package

FROM openjdk:17-jdk-alpine
VOLUME /tmp
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} blogProject.jar
ENTRYPOINT ["java","-jar","/blogProject.jar"]
```

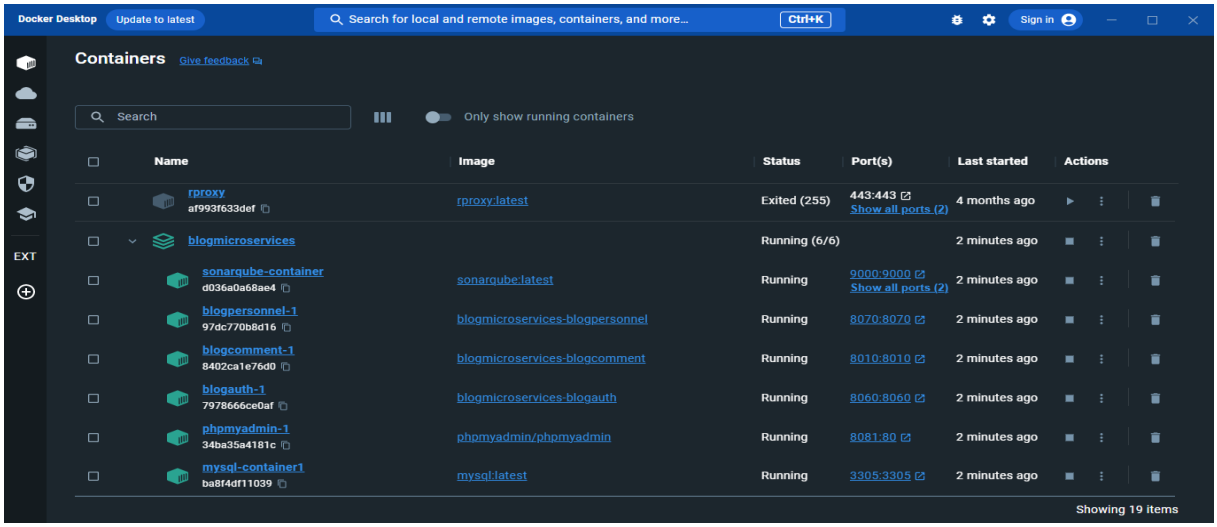
d) docker compose qui contien les images des microservices Mysql, phpmyadmin et sonarQube

```
version: '3'
services:
  sonarqube:
    image: sonarqube:latest
    container_name: sonarqube-container
    ports:
      - "9000:9000"
      - "9092:9092"
    networks:
      - microservices-network2
    environment:
      - SONARQUBE_JDBC_URL=jdbc:mysql://sonarqube-
db:3306/sonar?useUnicode=true&characterEncoding=utf8&rewriteBatchedStatemen
ts=true&useConfigs=maxPerformance
      - SONARQUBE_JDBC_USERNAME=sonar
      - SONARQUBE_JDBC_PASSWORD=sonar

  mysql:
    image: mysql:latest
    container_name: mysql-container1
    environment:
      MYSQL_ROOT_PASSWORD: root
    ports:
      - "3305:3305"
    networks:
      - microservices-network2
  blogauth:
    build:
```

```
      context: ./blogAuth
    ports:
      - "8060:8060"
    depends_on:
      - mysql
    networks:
      - microservices-network2
    environment:
      SPRING_DATASOURCE_URL:
jdbc:mysql://mysql:3306/blogauth?createDatabaseIfNotExist=true&characterEnc
oding=utf-8
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
      BEZKODER_APP_JWTSECRET: bezKoderSecretKey
      BEZKODER_APP_JWTEXPIRATIONMS: 86400000
    healthcheck:
      test: "/usr/bin/mysql --user=root --password=root --execute \"SHOW
DATABASES;\""
      interval: 5s
      timeout: 2s
      retries: 100
  blogcomment:
    build:
      context: ./BlogComment
    ports:
      - "8010:8010"
    depends_on:
      - mysql
    networks:
      - microservices-network2
    environment:
      SPRING_DATASOURCE_URL:
jdbc:mysql://mysql:3306/blogComment?createDatabaseIfNotExist=true
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
    healthcheck:
      test: "/usr/bin/mysql --user=root --password=root --execute \"SHOW
DATABASES;\""
      interval: 5s
      timeout: 2s
      retries: 100
  blogpersonnel:
    build:
      context: ./blogPersonnel
    ports:
      - "8070:8070"
    depends_on:
      - mysql
    networks:
      - microservices-network2
    environment:
      SPRING_DATASOURCE_URL:
jdbc:mysql://mysql:3306/blogPost?createDatabaseIfNotExist=true
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
    healthcheck:
      test: "/usr/bin/mysql --user=root --password=root --execute \"SHOW
DATABASES;\""
      interval: 5s
      timeout: 2s
      retries: 100

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
```

```
    environment:
      PMA_HOST: mysql
      PMA_PORT: 3304
      MYSQL_ROOT_PASSWORD: root
    ports:
      - "8081:80"
    networks:
      - microservices-network2
networks:
  microservices-network2:
    driver: bridge
```

2. Les images déployées dans docker:



5. CI/CD avec Jenkins

1. Processus CI/CD :

Développement du Code : Les développeurs travaillent sur leurs fonctionnalités ou correctifs de bugs localement.

Gestion de Code Source : Utilisation d'un système de gestion de code source (comme Git) pour versionner le code et le rendre accessible à toute l'équipe.
Push du Code : Les développeurs poussent leurs modifications de code sur la branche principale du dépôt.

Déclencheur de Build : Jenkins est configuré pour surveiller les changements dans le dépôt. Tout push déclenche un processus de build.

Build : Jenkins récupère le code source, compile les artefacts, exécute les tests unitaires et produit des binaires exécutables.

Tests Automatisés : Exécution de tests automatisés pour s'assurer que les modifications n'ont pas introduit de régressions.

Rapports de Build : Les rapports de build, y compris les résultats des tests, sont générés et archivés.

Notification : Notification d'état du build aux développeurs et à l'équipe en cas de succès ou d'échec.

2. Configuration du pipeline :

```
pipeline {
    agent any

    tools {
        maven 'maven'
    }

    stages {
        stage('Git Clone') {
            steps {
                script {
                    checkout([$class: 'GitSCM', branches: [[name: 'main']],
userRemoteConfigs: [[url:
'https://github.com/RedaBouougri/blogMicroservices.git']]])
                }
            }
        }

        stage('Build Auth') {
            steps {
                script {
                    dir('blogAuth') {
                        bat 'mvn clean install -DskipTests'
                    }
                }
            }
        }

        stage('Build Comment') {
            steps {
                script {
                    dir('BlogComment') {
                        bat 'mvn clean install -DskipTests'
                    }
                }
```

```
            }
        }

        stage('SonarQube Analysis Comment') {
            steps {
                script {
                    dir('BlogComment') {
                        // Run SonarQube analysis with default configuration
                        bat 'mvn sonar:sonar'
                    }
                }
            }
        }

        stage('Build Post') {
            steps {
                script {
                    dir('blogPersonnel') {
                        bat 'mvn clean install -DskipTests'
                    }
                }
            }
        }

        stage('Run') {
            steps {
                script {
                    bat "docker-compose up -d"
                }
            }
    }
    }
    }
```

Ce script de pipeline Jenkins déclare un pipeline de construction (build) pour un projet Java basé sur Maven, avec des étapes spécifiques pour cloner un référentiel Git, construire plusieurs modules du projet, effectuer une analyse SonarQube, et enfin exécuter l'application à l'aide de Docker Compose. Voici une explication détaillée du pipeline :

1. Agent : `agent any` indique que le pipeline peut être exécuté sur n'importe quel agent disponible dans Jenkins.

2. Outils : Utilisation de l'outil Maven. La section `tools` spécifie que le pipeline utilise Maven et que la version nommée 'maven' doit être utilisée.

3. Stages : Le pipeline est divisé en plusieurs étapes (stages), chacune représentant une phase spécifique du processus de construction et de déploiement.

- Stage 'Git Clone' : Cette étape consiste à cloner le référentiel Git. Le script utilise la commande `checkout` pour récupérer le code depuis le référentiel Git spécifié.

- Stage 'Build Auth' : Construit le module 'blogAuth' du projet à l'aide de Maven. Le script utilise la commande `mvn clean install -DskipTests` pour nettoyer le projet, le construire, et ignorer l'exécution des tests pendant cette phase.

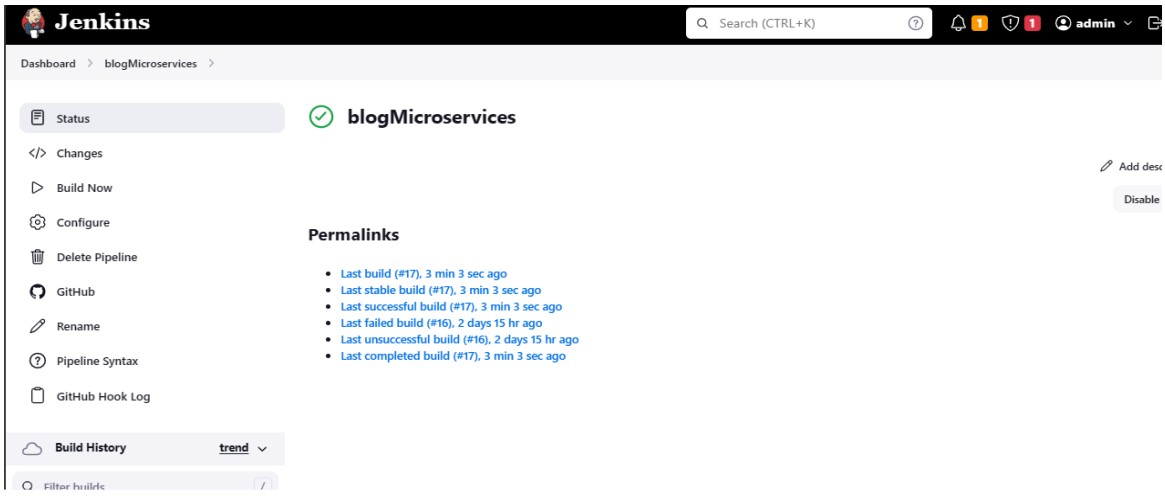
- Stage 'Build Comment' : Construit le module 'BlogComment' du projet à l'aide de Maven. De manière similaire, cela utilise la commande `mvn clean install -DskipTests`.

- Stage 'SonarQube Analysis Comment' : Effectue une analyse SonarQube sur le module 'BlogComment'. Cette étape utilise la commande `mvn sonar:sonar` pour lancer l'analyse avec la configuration par défaut.

- Stage 'Build Post' : Construit le module 'blogPersonnel' du projet à l'aide de Maven, avec la commande `mvn clean install -DskipTests`.

- Stage 'Run' : Démarre l'application à l'aide de Docker Compose. La commande `docker-compose up -d` est utilisée pour lancer les conteneurs Docker en arrière-plan.

Chaque étape est exécutée séquentiellement, et si l'une d'entre elles échoue, le pipeline s'arrête, marquant le build comme échoué. Ce script assume que les outils nécessaires (Maven, Docker Compose) sont disponibles sur l'agent Jenkins où le pipeline est exécuté. De plus, les étapes spécifiques au projet (noms de modules, commandes Maven) peuvent nécessiter des ajustements en fonction de la structure réelle du projet.



6. L’intégration de sonarQube

1. Intégration de SonarQube dans le pipeline CI/CD de notre projet de blog:

Configuration de SonarQube :

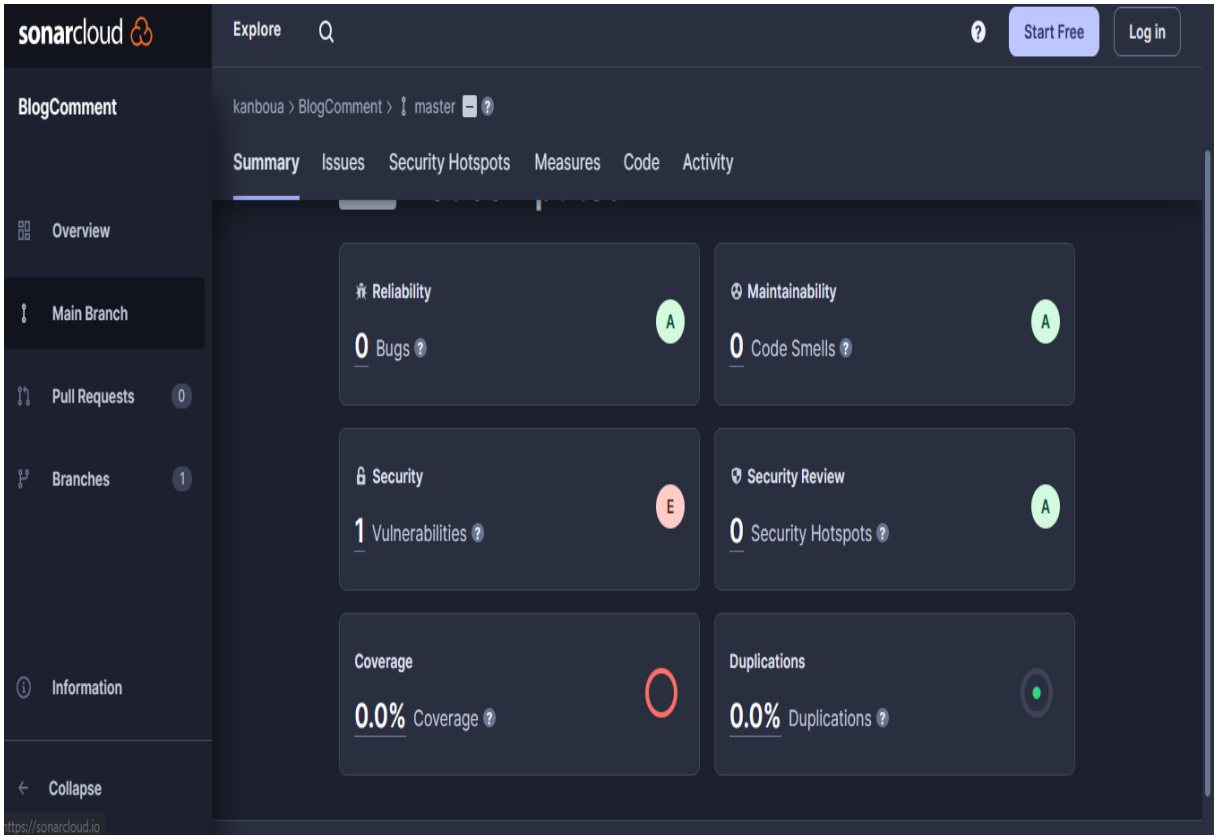
SonarQube est étroitement intégré à notre processus CI/CD via Jenkins. À chaque exécution du pipeline, des analyses statiques du code sont automatiquement déclenchées, fournissant ainsi des rapports détaillés sur la qualité du code.

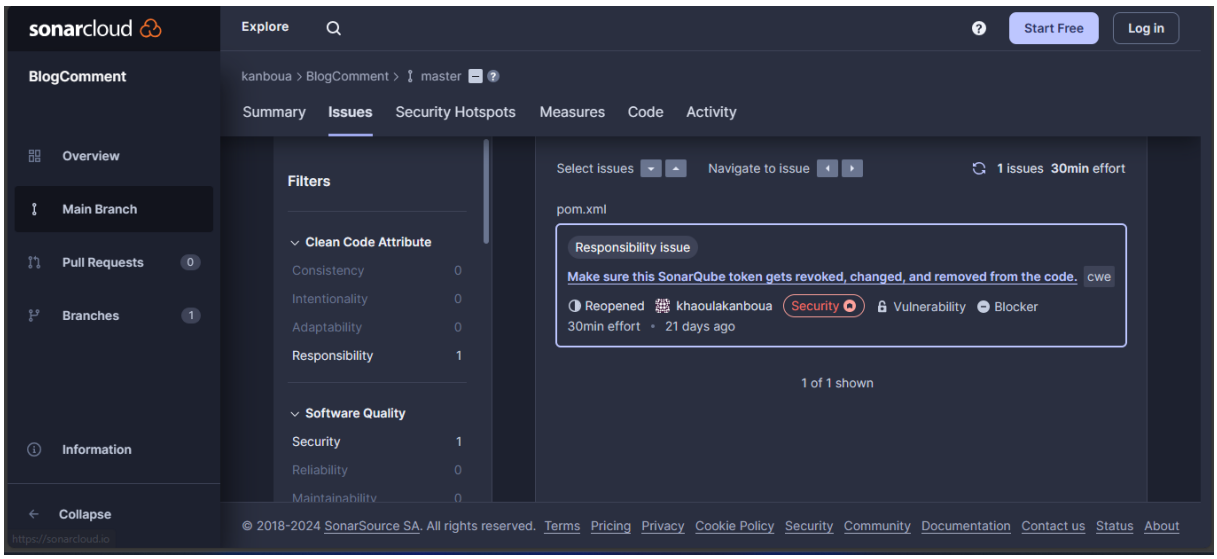
Configuration des Règles :

Nous avons personnalisé les règles de SonarQube pour refléter précisément nos normes internes de codage spécifiques au projet de blog. Les règles par défaut ont été ajustées pour s'aligner avec les meilleures pratiques et les standards particuliers à notre application.

Utilisation des Plugins :

Dans le cadre de notre configuration, nous avons exploité des plugins complémentaires de SonarQube pour couvrir un large éventail de langages de programmation et de technologies utilisés dans notre application de blog. Cette configuration garantit une analyse complète de tout le code source, qu'il s'agisse du Microservice Comment, du Microservice Blog, ou d'autres composants, assurant ainsi une évaluation exhaustive de la qualité du code.





2. Les plugins utilisés pour le vouverage :

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.7</version>
    <executions>
       <execution>
          <goals>
             <goal>prepare-agent</goal>
          </goals>
       </execution>
       <execution>
          <id>report</id>
          <phase>prepare-package</phase>
          <goals>
             <goal>report</goal>
          </goals>
       </execution>
    </executions>
</plugin>
```

7. Conclusion

Blog Personnel a franchi des étapes significatives en réussissant l'implémentation réussie d'une architecture basée sur les microservices, la containerisation avec Docker, l'établissement d'une CI/CD grâce à Jenkins, l'intégration de SonarQube, et l'utilisation judicieuse de Consul pour la découverte de services dans le contexte de notre projet de blog.

En matière de perspectives futures, notre vision comprend le développement d'une application mobile dédiée, des améliorations notables de l'expérience utilisateur, l'optimisation des performances, l'expansion des fonctionnalités, et la continuité de l'évolutivité et de la résilience de notre architecture.