



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

MINI PROJET

2023

Conception de Blog Collaboratif

Ingénierie Informatique et Réseaux

Réalisé par :

Ait El Bhiri Aymane
Behloul Anas

Encadré par :

M LACHGAR Mohamed

Table des matières

1	Introduction.....	1
1.1	Aperçu du projet.....	1
1.2	Définition de l'architecture micro-service.....	1
1.3	Importance de l'architecture micro-service	1
2	Architecture micro-service	2
2.1	Architecture	3
2.2	Description des services	3
2.2.1	Service utilisateur.....	3
2.2.2	Service poste	4
2.3	Mécanisme de communication.....	5
3	Conception des micro-services	5
3.1	Micro-service utilisateur.....	5
3.1.1	Diagramme de classe.....	6
3.2	Micro-service poste	6
3.2.1	Diagramme de classe.....	6
4	Conteneurisation avec Docker	8
4.1	Implémentation	8
4.1.1	Création d'une Image Docker	8
4.2	Avantage	11
5	CI/CD avec Jenkins	11
5.1	Processus et configuration	11
5.1.1	Création d'un nouveau item.....	12
6	Conclusion	17
6.1	Accomplissements:.....	17
6.2	Perspectives Futures:.....	18

Liste de figure :

Figure 1: Architecture micro-service.....	3
Figure 2: Eureka Server	3
Figure 3: Diagramme de classe : Utilisateur.....	6
Figure 4: Diagramme de séquence	6
Figure 5: Diagramme de classe : Poste.....	7
Figure 8: Fichier docker pour le Frontend	9
Figure 9: Fichier docker pour le Backend.....	9
Figure 10: Pipeline partie1	10
Figure 11: Pipeline partie2.....	10
Figure 12: Pipeline partie3.....	10
Figure 13:Création de l'item.....	12
Figure 14: Configuration	12
Figure 15: Code pipeline.....	13
Figure 16: Résultat du lancement de pipeline	14
Figure 17: Création des images docker	14

1 Introduction :

1.1 Aperçu du projet :

L'avènement des blogs collaboratifs marque une étape importante dans l'évolution des plateformes en ligne, offrant une expérience interactive et dynamique pour les passionnés du voyage et de la découverte. Notre projet, basé sur une architecture de microservices utilisant Java côté serveur et ReactJS côté client, vise à créer un espace virtuel où divers voyageurs peuvent partager leurs expériences, perspectives et conseils.

Cette convergence technologique permet une interactivité accrue, une gestion modulaire efficace et une expérience utilisateur fluide. Dans ce rapport, nous explorerons les détails de l'implémentation, les choix architecturaux, les fonctionnalités clés et les défis rencontrés tout au long du développement de ce blog collaboratif. L'objectif est de créer une plateforme qui transcende les limites traditionnelles des blogs de voyage en offrant une immersion totale dans la diversité des récits, suscitant ainsi l'intérêt et l'engagement d'une communauté mondiale d'explorateurs virtuels.

1.2 Définition de l'architecture micro-service :

L'architecture micro services est un style architectural qui structure une application comme une collection de services indépendants, déployables et évolutifs. Chaque service, souvent appelé micro service, est conçu pour accomplir une tâche spécifique et communique avec d'autres services au moyen d'API (Interfaces de Programmation d'Applications) bien définies. Contrairement à l'approche monolithique, où une application est construite comme une seule unité, l'architecture micro services favorise la décomposition d'une application en services autonomes, facilitant ainsi le développement, le déploiement et la maintenance.

1.3 Importance de l'architecture micro-service :

Scalabilité facilitée : Les micro-services permettent de faire évoluer chaque composant individuellement, ce qui facilite la mise à l'échelle des parties spécifiques de l'application en fonction des besoins.

Indépendance et flexibilité : Chaque micro-service est une entité indépendante, ce qui permet aux équipes de développement de travailler sur des services distincts sans affecter les autres. Cela favorise la flexibilité et la mise en œuvre de changements plus rapidement.

Déploiement continu : Les micro-services facilitent la mise en place de pipelines de déploiement continu, permettant ainsi des mises à jour fréquentes et un déploiement plus rapide des nouvelles fonctionnalités.

Technologies adaptées : Chaque micro-service peut être développé en utilisant la technologie la plus appropriée pour sa tâche spécifique, ce qui permet d'utiliser une variété de langages de programmation et de technologies au sein d'une même application.

Réparabilité améliorée : En cas de défaillance d'un micro-service, le reste de l'application peut continuer à fonctionner normalement. Cela améliore la résilience et facilite la détection et la résolution des problèmes.

Évolutivité du développement : Les équipes de développement peuvent travailler de manière indépendante sur des micro-services spécifiques, ce qui accélère le développement global du projet.

Gestion simplifiée : La gestion des micro-services peut être facilitée en utilisant des outils d'orchestration de conteneurs tels que Kubernetes, ce qui simplifie la gestion des versions, la mise à l'échelle et la surveillance.

Réutilisation des services : Les micro-services bien conçus peuvent être réutilisés dans différentes parties de l'application ou même dans d'autres projets, favorisant ainsi l'efficacité et la modularité.

2 Architecture micro-service :

2.1 Architecture :

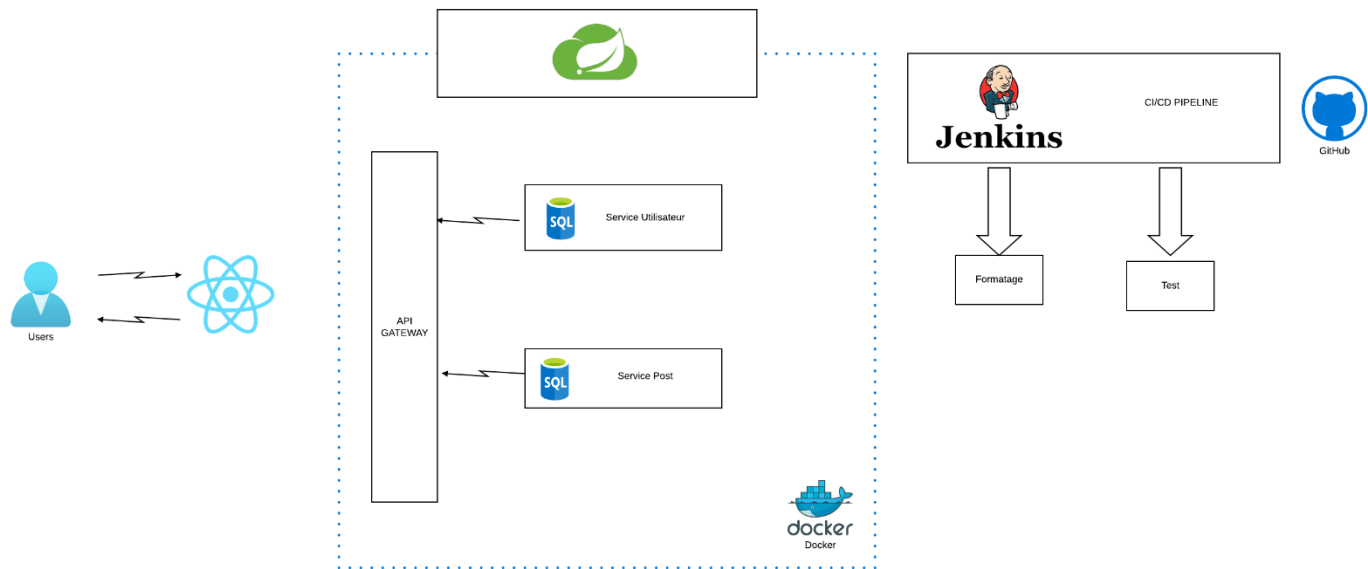



Figure 1: Architecture micro-service

2.2 Description des services :

HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2024-01-21T10:22:22 +0100
Data center	default	Uptime	00:02
		Lease expiration enabled	false
		Renews threshold	6
		Renews (last min)	6

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
GATEWAY	n/a (1)	(1)	UP (1) - localhost:Gateway:8888
SERVICE-POST	n/a (1)	(1)	UP (1) - localhost:service-post:8089
SERVICE-UTILISATEUR	n/a (1)	(1)	UP (1) - localhost:service-utilisateur:8088

General Info

Figure 2: Eureka Server

2.2.1 Service utilisateur :

Fonctionnalité :

➤ Authentification :

Vérification des identifiants de connexion (nom d'utilisateur ou e-mail et mot de passe).

➤ Enregistrement(Registre) :

Validation des données utilisateur unicité de l'e-mail, complexité du mot de passe.

Création d'un nouveau profil utilisateur avec des informations de base.

2.2.2 Service poste :

Fonctionnalité :

- Gestion complète du cycle de vie des postes y compris la création, la lecture, la mise à jour et la suppression (CRUD).
- Possibilité d'ajouter des commentaires aux postes.
- Ajout de la possibilité pour les utilisateurs d'ajouter des commentaires à n'importe quel poste.
- Consultation des commentaires associés à un poste spécifique.
- Fonctionnalité de suppression des commentaires par leurs auteurs ou par des administrateurs.

Communication avec d'autres Micro-services :

- Gateway : Utilisation d'une gateway API pour gérer les communications entre micro-services.
- Utilisateurs : Interaction avec le micro-service utilisateur pour récupérer des informations sur les auteurs de postes, vérifier l'authentification.
- Gestion des Postes : Interaction pour récupérer des informations sur le poste lié au commentaire.

2.3 Mécanisme de communication :

➤ Description :

Les services exposent des API RESTful et interagissent entre eux via des requêtes HTTP.

➤ Avantage :

Simplicité, standardisation, compatibilité avec de nombreuses technologies.

➤ Considérations :

Peut-être synchrone (requête-réponse) ou asynchrone avec l'utilisation de webhooks.

3 Conception des micro-services :

3.1 Micro-service utilisateur :

3.1.1 Diagramme de classe :

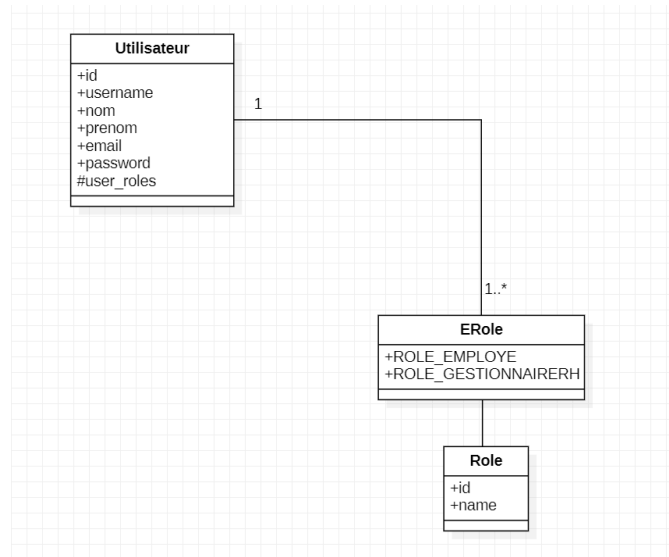


Figure 3: Diagramme de classe : Utilisateur

3.2 Micro-service poste :

3.2.1 Diagramme de classe :

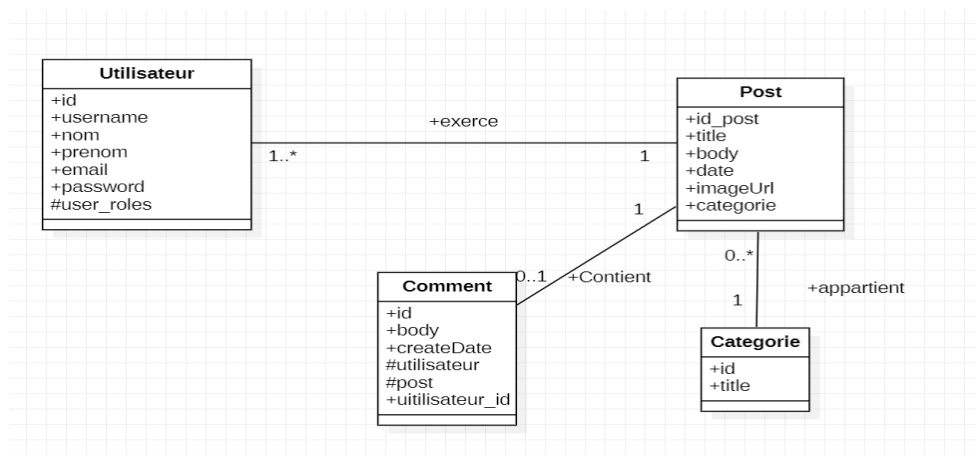


Figure 5: Diagramme de classe : Poste

4 Conteneurisation avec Docker :

4.1 Implémentation :

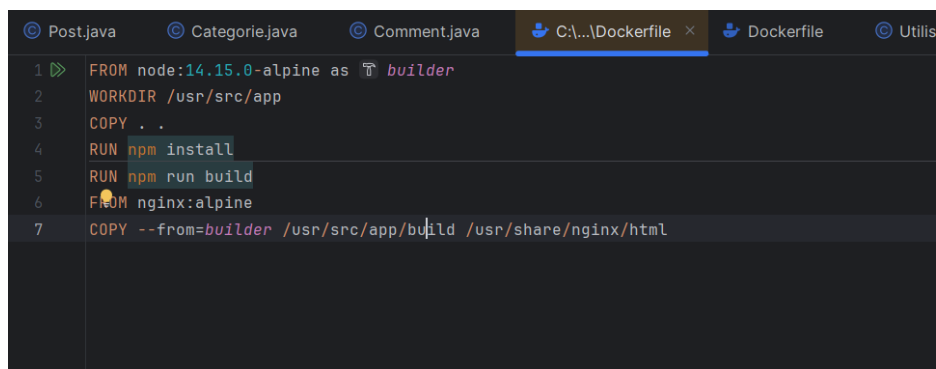
La conteneurisation avec Docker est un processus permettant d'encapsuler une application et ses dépendances dans un conteneur léger et portable. Voici comment cela peut être mis en œuvre :

4.1.1 Création d'une Image Docker :

- Dockerfile : Un fichier de configuration qui décrit les étapes nécessaires à la création d'une image Docker.

Nous avons créé deux fichiers docker :

Pour le Frontend :

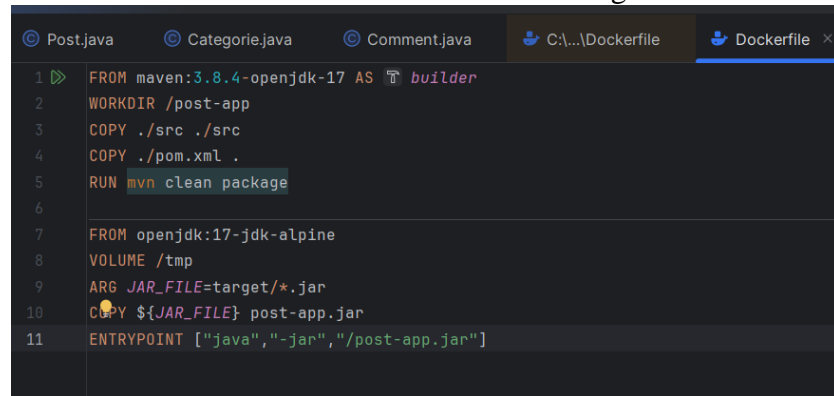


```
1 FROM node:14.15.0-alpine as builder
2 WORKDIR /usr/src/app
3 COPY . .
4 RUN npm install
5 RUN npm run build
6 FROM nginx:alpine
7 COPY --from=builder /usr/src/app/build /usr/share/nginx/html
```

Figure 8: Fichier docker pour le Frontend

Pour le Backend :

Pour chaque micro-service nous avons crée un fichier docker. La figure et comme exemple.

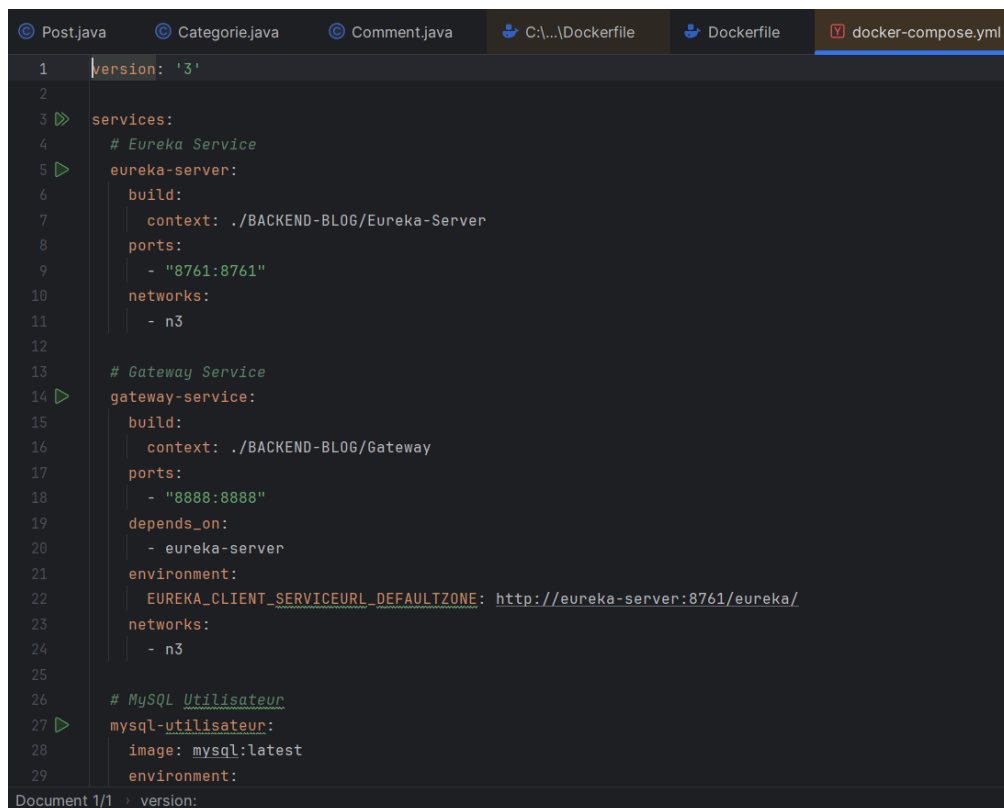


```
1 FROM maven:3.8.4-openjdk-17 AS builder
2 WORKDIR /post-app
3 COPY ./src ./src
4 COPY ./pom.xml .
5 RUN mvn clean package
6
7 FROM openjdk:17-jdk-alpine
8 VOLUME /tmp
9 ARG JAR_FILE=target/*.jar
10 COPY ${JAR_FILE} post-app.jar
11 ENTRYPOINT ["java","-jar","/post-app.jar"]
```

Figure 9: Fichier docker pour le Backend

- Construction de l'image : Utilisation de la commande 'docker build' pour créer l'image en fonction du Dockerfile.
- Docker Compose: Le fichier Docker Compose est un fichier YAML qui permet de définir et de configurer des services Docker, des réseaux, et des volumes. Il facilite le déploiement et la gestion d'applications multi-conteneurs.

Pour notre cas nous avons crée un fichier docker compose avec une syntaxe d'architecture micro-service :



```
1 version: '3'
2
3 services:
4   # Eureka Service
5   eureka-server:
6     build:
7       context: ./BACKEND-BLOG/Eureka-Server
8     ports:
9       - "8761:8761"
10    networks:
11      - n3
12
13   # Gateway Service
14   gateway-service:
15     build:
16       context: ./BACKEND-BLOG/Gateway
17     ports:
18       - "8888:8888"
19     depends_on:
20       - eureka-server
21     environment:
22       EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://eureka-server:8761/eureka/
23     networks:
24       - n3
25
26   # MySQL Utilisateur
27   mysql-utilisateur:
28     image: mysql:latest
29     environment:
```

Figure 10: Pipeline partiel

```

30     MYSQL_ROOT_PASSWORD: root
31     MYSQL_DATABASE: service-utilisateur
32     MYSQL_PASSWORD: root
33     ports:
34     - "3306:3306"
35     networks:
36     - n3
37
38     # MySQL Post
39     mysql-post:
40     image: mysql:latest
41     environment:
42     MYSQL_ROOT_PASSWORD: root
43     MYSQL_DATABASE: service-post
44     MYSQL_PASSWORD: root
45     ports:
46     - "3307:3306"
47     networks:
48     - n3
49
50     # Utilisateur Service
51     utilisateur-service:
52     build:
53     context: ./BACKEND-BLOG/Utilisateur
54     ports:
55     - "8088:8088"
56     depends_on:
57     - eureka-server
58     - mysql-utilisateur

```

Document 1/1 | version:

Figure 11: Pipeline partie2

```

60     SPRING_DATASOURCE_URL: jdbc:mysql://mysql-utilisateur:3306/service-utilisateur
61     SPRING_DATASOURCE_USERNAME: root
62     SPRING_DATASOURCE_PASSWORD: root
63     eureka.client.serviceUrl.defaultZone: http://eureka-server:8761/eureka/
64     healthcheck:
65     test: "/usr/bin/mysql --user=root --password=root --execute \"SHOW DATABASES;\""
66     interval: 5s
67     timeout: 2s
68     retries: 100
69     networks:
70     - n3
71
72     # Post Service
73     post-service:
74     build:
75     context: ./BACKEND-BLOG/Post
76     ports:
77     - "8089:8089"
78     depends_on:
79     - eureka-server
80     - mysql-post
81     environment:
82     SPRING_DATASOURCE_URL: jdbc:mysql://mysql-post:3306/service-post
83     SPRING_DATASOURCE_USERNAME: root
84     SPRING_DATASOURCE_PASSWORD: root
85     eureka.client.serviceUrl.defaultZone: http://eureka-server:8761/eureka/
86     healthcheck:
87     test: "/usr/bin/mysql --user=root --password=root --execute \"SHOW DATABASES;\""
88     interval: 5s

```

Figure 12: Pipeline partie3

```
94 frontend:
95   build:
96     context: ../FRONT-END-BLOG
97   ports:
98     - "3000:80"
99   depends_on:
100     - post-service
101     - utilisateur-service
102
103 phpmyadmin-utilisateur:
104   image: phpmyadmin/phpmyadmin
105   environment:
106     PMA_HOST: mysql-utilisateur
107     PMA_PORT: 3306
108     MYSQL_ROOT_PASSWORD: root
109   ports:
110     - "8082:80"
111   networks:
112     - n3
113
114 phpmyadmin-post:
115   image: phpmyadmin/phpmyadmin
116   environment:
117     PMA_HOST: mysql-post
118     PMA_PORT: 3306
119     MYSQL_ROOT_PASSWORD: root
120   ports:
121     - "8083:80"
122   networks:
```

Document 1/1 > services: > phpmyadmin-post: > networks: > Item 1/1 > n3

Figure 13: Pipeline partie4

Avantage :

➤ Isolation et portabilité :

Les conteneurs isolent les applications et leurs dépendances, garantissant une portabilité entre différents environnements.

➤ Léger et rapide :

Les conteneurs partagent le noyau de l'hôte, ce qui les rend plus légers et plus rapides à démarrer par rapport aux machines virtuelles.

➤ Gestion des dépendances :

Docker permet de définir les dépendances et configurations nécessaires dans le fichier Dockerfile, assurant une reproduction cohérente de l'environnement.

➤ Évolutivité :

Les conteneurs peuvent être facilement mis à l'échelle horizontalement pour gérer une charge croissante en utilisant des orchestrateurs comme Docker Swarm ou Kubernetes.

➤ Facilité de déploiement :

La distribution des conteneurs avec Docker Hub facilite le déploiement et la mise à jour des applications.

➤ Sécurité :

Les conteneurs utilisent des mécanismes de sécurité tels que les espaces de noms et les cgroup pour isoler les processus, renforçant la sécurité des applications.

➤ Gestion des versions :

Les images Docker peuvent être versionnées, facilitant le suivi des changements et le retour en arrière si nécessaire.

5 CI/CD avec Jenkins

5.1 Processus et configuration :

5.1.1 Création d'un nouveau item :

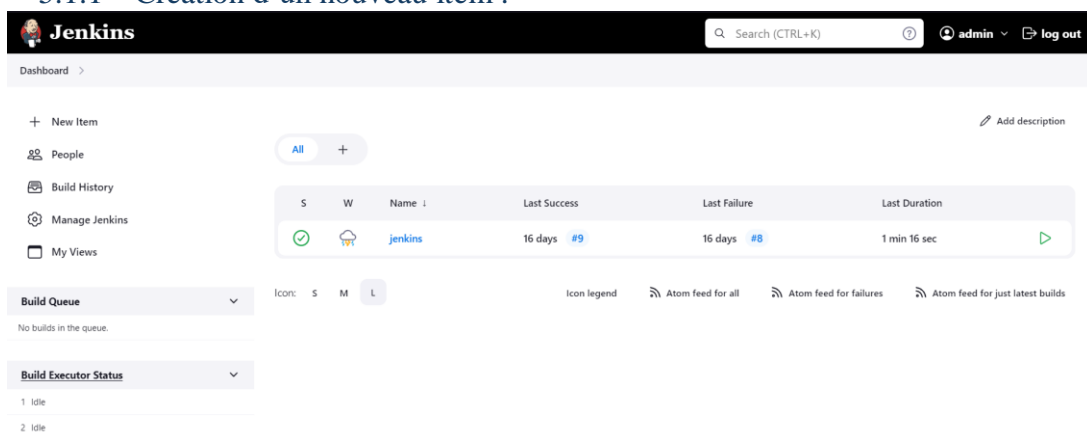


Figure 13:Création de l'item

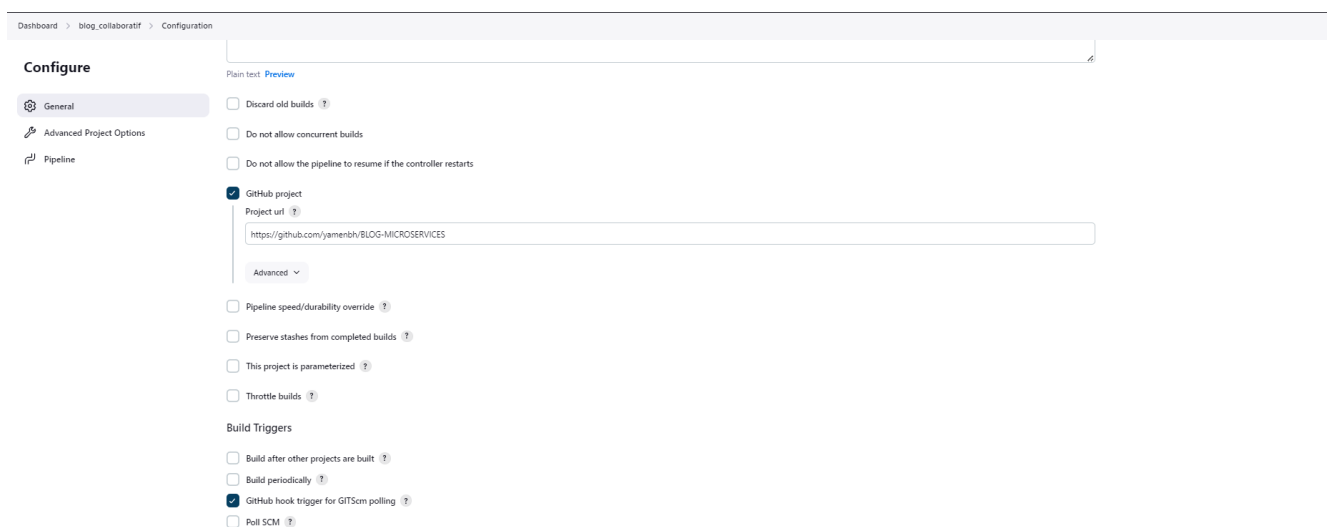


Figure 14: Configuration

Notre pipeline et comme suit :



```
1 pipeline {
2   agent any
3
4   tools {
5     // Specify the Maven tool named 'maven'
6     maven 'maven'
7   }
8
9   stages {
10    stage('Git Clone') {
11      steps {
12        script {
13          checkout([$class: 'GitSCM', branches: [[name: 'main']],
14            userRemoteConfigs: [[url: 'https://github.com/yamenbh/BLOG-MICROSERVICES.git']]])
15        }
16      }
17    }
18
19    stage('Build Backend') {
20      steps {
21        script {
22          dir('BACKEND-BLOG/Eureka-Server') {
23            bat 'mvn clean install'
24          }
25          dir('BACKEND-BLOG/Gateway') {
26            bat 'mvn clean install'
27          }
28          dir('Back_blog/Utilisateur') {
29            bat 'mvn clean install'
30          }
31          dir('BACKEND-BLOG/Post') {
32            bat 'mvn clean install'
33          }
34        }
35      }
36    }
37
38    stage('Create Docker Image (Backend)') {
39      steps {
40        dir('BACKEND-BLOG/Eureka-Server') {
41          bat 'docker build -t blog/eurekaserver .'
42        }
43        dir('BACKEND-BLOG/Gateway') {
44          bat 'docker build -t blog/gateway .'
45        }
46        dir('BACKEND-BLOG/Utilisateur') {
47          bat 'docker build -t blog/utilisateur .'
48        }
49        dir('BACKEND-BLOG/Post') {
50          bat 'docker build -t blog/post .'
51        }
52      }
53    }
54
55    stage('Build Frontend') {
56      steps {
57        script {
58          dir('FRONT-END-BLOG') {
59            bat 'npm install'
60          }
61        }
62      }
63    }
64
65    stage('Create Docker Image (Frontend)') {
66      steps {
67        dir('FRONT-END-BLOG') {
68          bat 'docker build -t blog/front .'
69        }
68    }
69  }
```

Figure 15: Code pipeline

Après le lancement du pipeline :

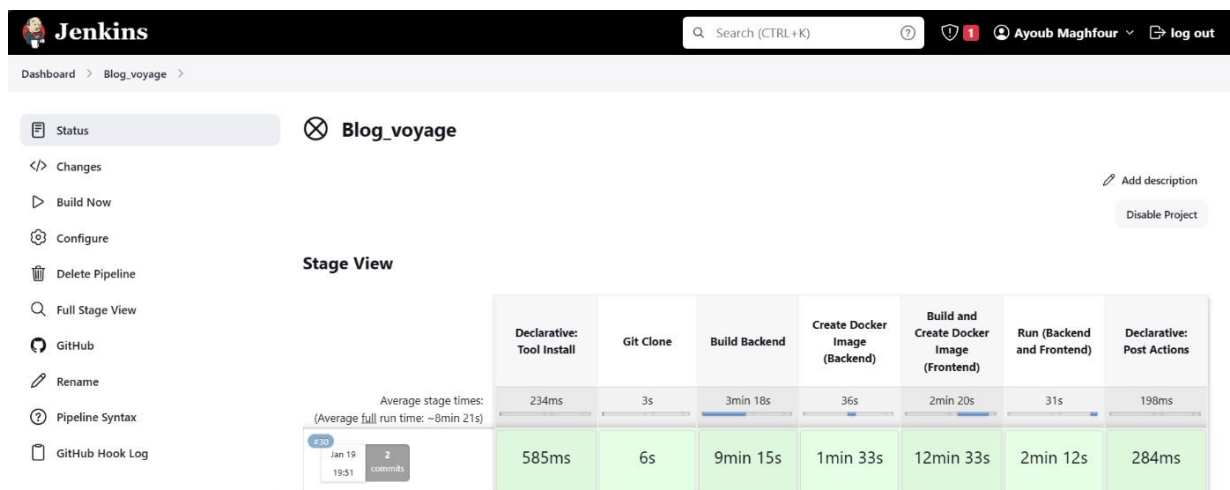


Figure 16: Résultat du lancement de pipeline

Création des images dans docker :

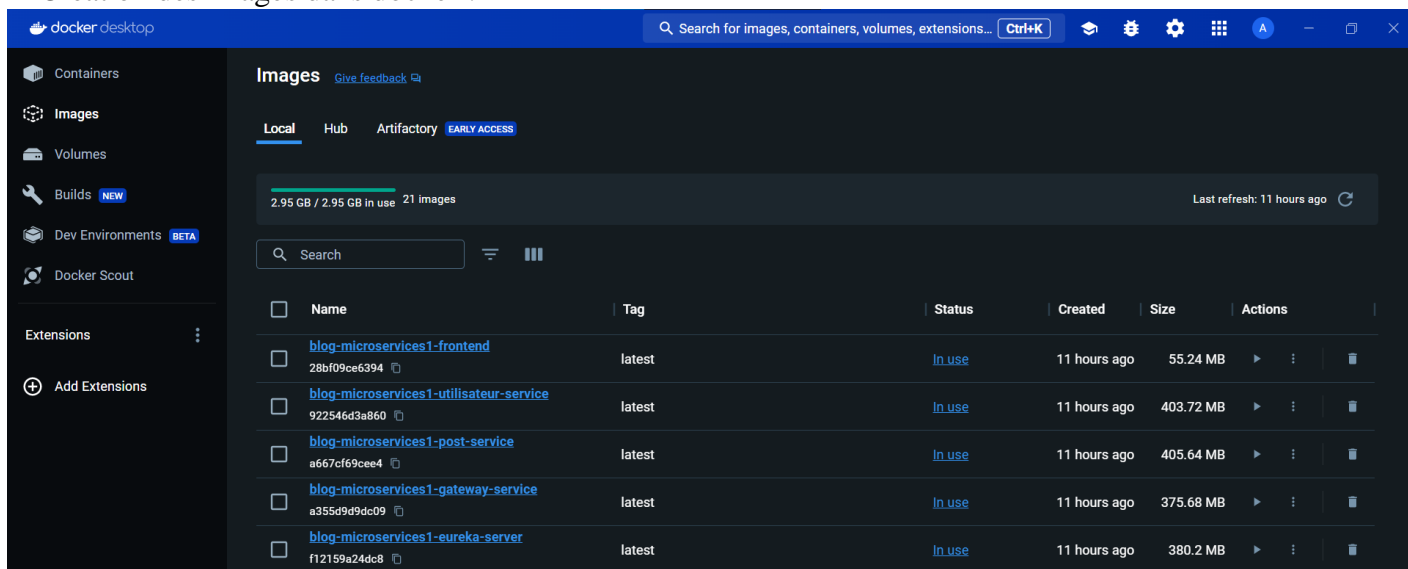


Figure 17: Création des images docker

6 Conclusion :

6.1 Accomplissements:

- Architecture Micro-services:

Mise en place d'une architecture micro-services pour le blog collaboratif, permettant une meilleure scalabilité et une gestion modulaire des fonctionnalités.

- Développement du Blog:

Conception et développement des micro-services pour différentes fonctionnalités du blog telles que la publication d'articles, la gestion des commentaires, et la gestion des utilisateurs.

- Intégration de Jenkins:

Mise en œuvre de l'intégration continue (CI) avec Jenkins pour automatiser le processus de build à chaque modification de code.

Configuration de pipelines Jenkins pour effectuer des tests automatiques, garantissant la qualité du code à chaque itération.

- Utilisation de Docker:

Intégration de Docker pour la conteneurisation des micro-services, facilitant le déploiement et l'orchestration dans différents environnements.

- Déploiement Continu:

Implémentation de la livraison continue (CD) avec Jenkins pour automatiser le déploiement des micro-services dans les environnements de test et de production.

➤ Gestion des Versions:

Utilisation de Git pour la gestion de version, permettant le suivi des changements et la collaboration efficace au sein de l'équipe de développement.

6.2 Perspectives Futures:

➤ Optimisation des Performances:

Continuer à optimiser les performances des micro-services pour assurer une expérience utilisateur fluide, même avec une croissance significative du trafic.

➤ Ajout de Fonctionnalités:

Étendre le blog en ajoutant de nouvelles fonctionnalités, telles que la recherche avancée, la géolocalisation des articles, et l'intégration de médias enrichis.

➤ Internationalisation:

Mettre en œuvre la prise en charge de plusieurs langues pour atteindre un public plus large.

➤ Automatisation des Tests:

Renforcer les suites de tests automatisés pour garantir la stabilité du système à mesure qu'il évolue.