

MINI PROJET

2023

Conception de Blog de voyage

Ingénierie Informatique et Réseaux

Réalisé par :

Boussouna Salma

Maghfour Ayoub

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
2.2.3	Service favorite :	4
2.2.4	Service Commentaire :	5
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
3.1.2	Diagramme de séquence :	6
3.2	Micro-service poste :	6
3.2.1	Diagramme de classe :	6
3.3	Micro-service Commentaire :	7
3.3.1	Diagramme de classe :	7
3.4	Micro-service favorite :	8
3.4.1	Diagramme de classe :	8
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	Déploiement Automatique :	15
6.1	Création du host avec Ngrok	15
6.2	Création du webhook dans Github :	15
6.3	Ajouter WebHook Github dans Jenkins :	16
6.4	Test :	16
7	Conclusion :	17
7.1	Accomplissements:	17
7.2	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 6: Diagramme de classe: Commentaire	7
Figure 7: Diagramme de classe : Favorite.....	8
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
Figure 18: Création du host.....	15
Figure 19: Webhook.....	16
Figure 20: Configuration Plugins	16
Figure 21: Configuration 2	16
Figure 22::Push dans github.....	16
Figure 23: Déploiement automatique	17

1 Introduction :

1.1 Aperçu du projet :

Les blogs de voyage ont émergé comme des plateformes captivantes qui transcendent les frontières physiques, permettant aux voyageurs de partager leurs expériences uniques, leurs découvertes culturelles et leurs aventures à travers le monde. En ces temps de connectivité numérique, un blog de voyage n'est pas simplement un récit en ligne, mais plutôt une fenêtre ouverte sur des destinations lointaines, des traditions exotiques, et des expériences personnelles fascinantes.

Un blog de voyage offre bien plus que des itinéraires et des conseils pratiques. Il raconte des histoires, capture des moments inoubliables à travers des images et offre un aperçu authentique des destinations explorées. Ces récits numériques transportent les lecteurs au-delà des pages virtuelles, éveillant leur curiosité, suscitant l'inspiration, et parfois même les incitant à explorer le monde par eux-mêmes.

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 :

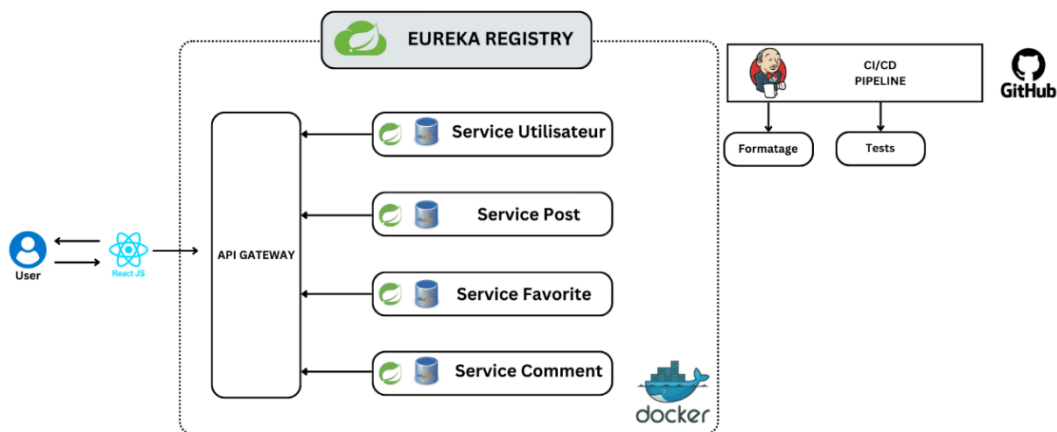



Figure 1: Architecture micro-service

2.2 Description des services :



HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2024-01-19T19:36:40 +0100
Data center	default	Uptime	00:04
		Lease expiration enabled	true
		Renews threshold	10
		Renews (last min)	16

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
GATEWAY	n/a (1)	(1)	UP (1) - DESKTOP-C0E57BI:Gateway:8888
SERVICE-COMMENTAIRE	n/a (1)	(1)	UP (1) - DESKTOP-C0E57BI:SERVICE-COMMENTAIRE:8084
SERVICE-FAVORITE	n/a (1)	(1)	UP (1) - DESKTOP-C0E57BI:SERVICE-FAVORITE:8083
SERVICE-POST	n/a (1)	(1)	UP (1) - DESKTOP-C0E57BI:SERVICE-Post:8082
SERVICE-UTILISATEUR	n/a (1)	(1)	UP (1) - DESKTOP-C0E57BI:SERVICE-UTILISATEUR:8081

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(Register) :

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.

➤ Récupération du mot de passe :

Génération et envoi d'un lien de réinitialisation de mot de passe par e-mail.

Vérification du lien de réinitialisation et mise à jour du mot de passe.

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.

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.

2.2.3 Service favorite :

Fonctionnalité :

- Ajout d'une fonctionnalité permettant aux utilisateurs d'ajouter un poste à leur liste de favoris.
- Possibilité de consulter la liste des postes favoris d'un utilisateur.
- Fonctionnalité de suppression d'un poste des favoris.

Communication avec d'autres Micro-services :

- Gateway : Utilisation de la Gateway API pour gérer les communications entre micro-services.

- Micro-service de Gestion des Postes : Interaction pour récupérer les informations nécessaires sur le poste à ajouter aux favoris.
- Micro-service Utilisateur : Validation de l'authentification de l'utilisateur pour s'assurer qu'il est autorisé à ajouter des favoris.

2.2.4 Service Commentaire :

Fonctionnalité :

- 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'autre Micro-services :

- Gateway : Utilisation de la Gateway API pour gérer les communications entre micro-services.
- Micro-service de Gestion des Postes : Interaction pour récupérer des informations sur le poste lié au commentaire.
- Micro-service Utilisateur : Validation de l'authentification de l'utilisateur pour s'assurer qu'il est autorisé à ajouter des commentaires.

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 :

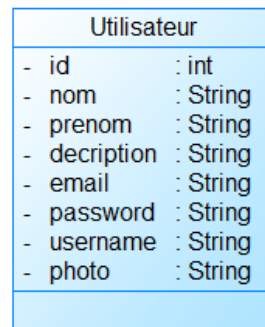


Figure 3: Diagramme de classe : Utilisateur

3.1.2 Diagramme de séquence :

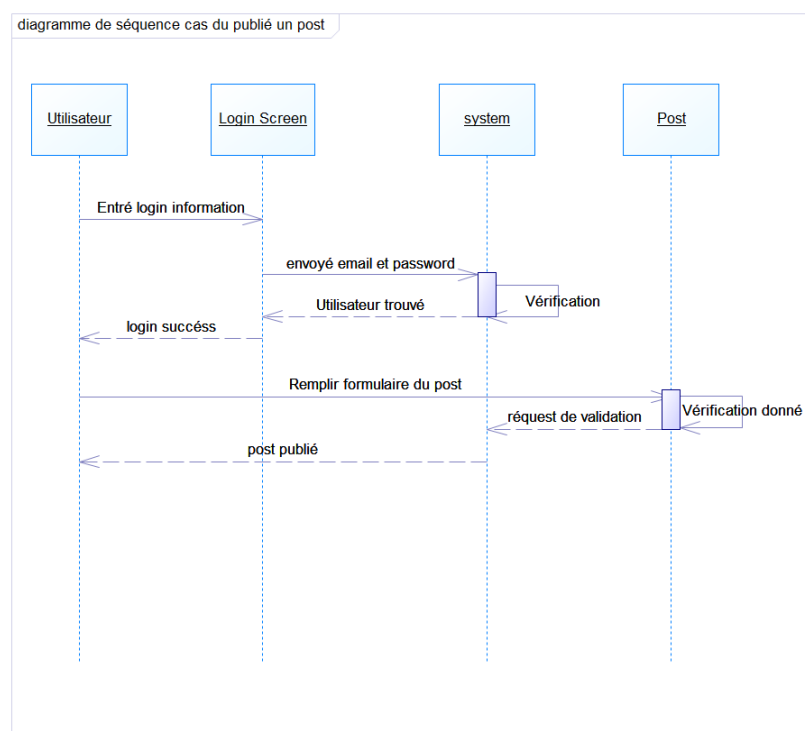


Figure 4: Diagramme de séquence

3.2 Micro-service poste :

3.2.1 Diagramme de classe :

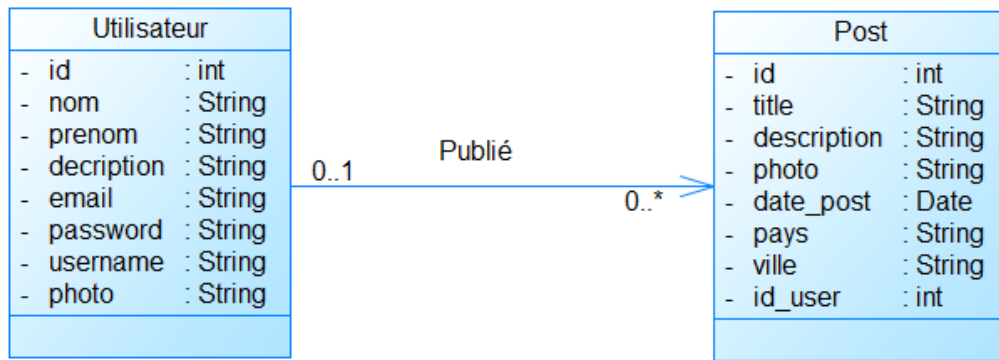


Figure 5: Diagramme de classe : Poste

3.3 Micro-service Commentaire :

3.3.1 Diagramme de classe :

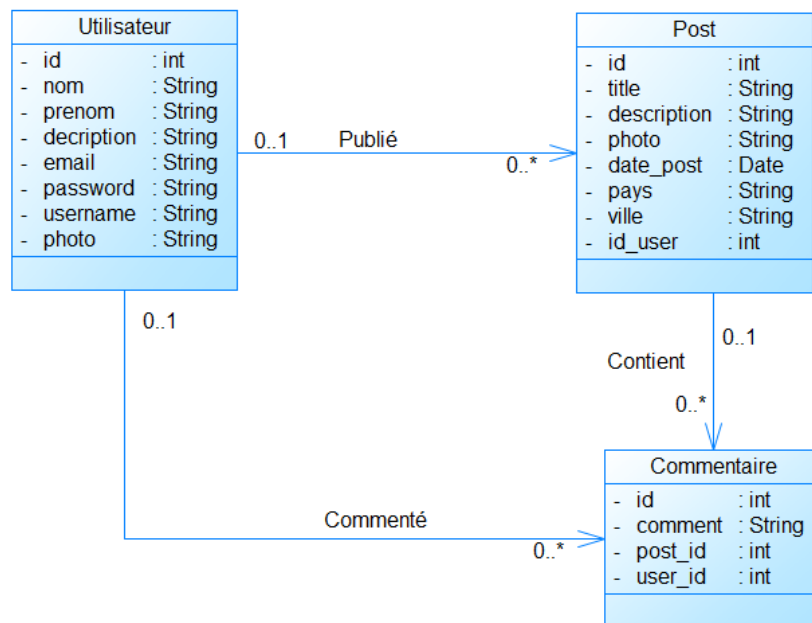


Figure 6: Diagramme de classe: Commentaire

3.4 Micro-service favorite :

3.4.1 Diagramme de classe :

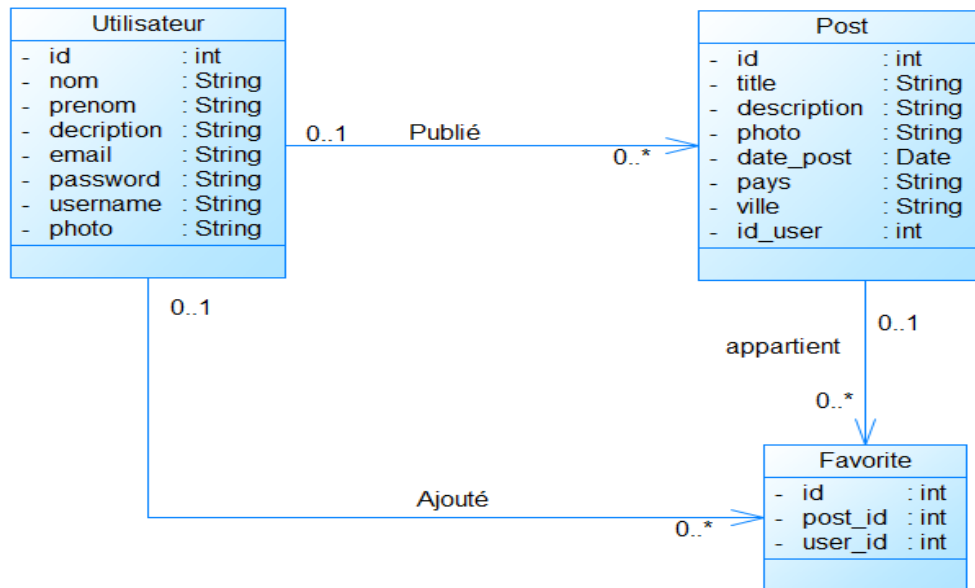


Figure 7: Diagramme de classe : Favorite

4 Conteneurisation avec Docker :

4.1 Implémentation :

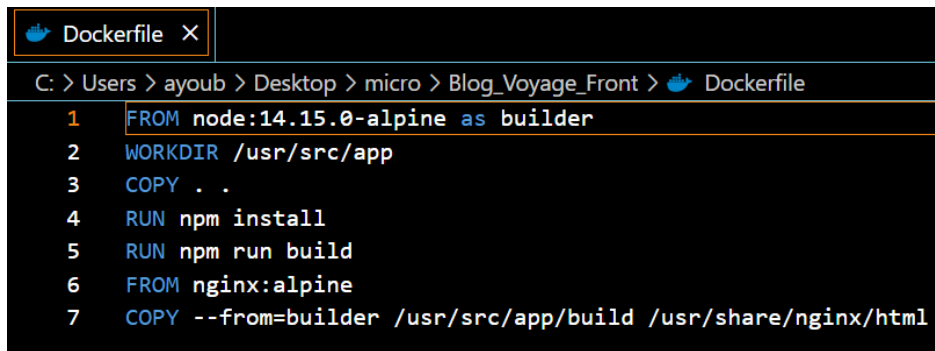
La conteneurisation avec Docker est un processus permettant d'encapsuler une application et ses dépendances dans un conteneur léger et portable. Voici comment cela peut être mis en œuvre :

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 :



```

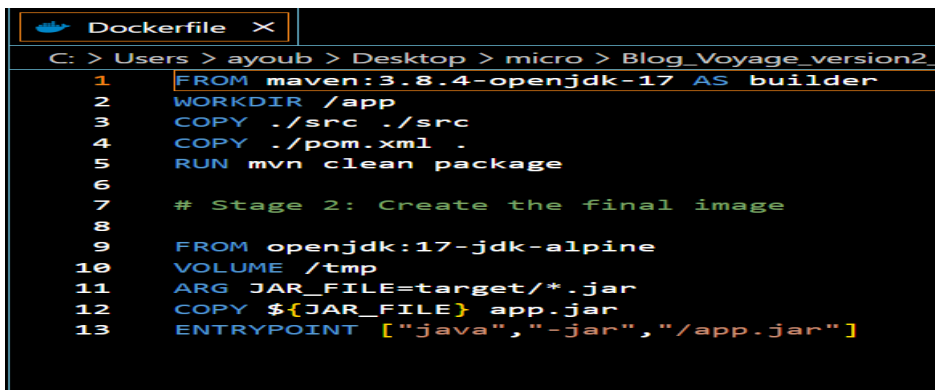
1 FROM node:14.15.0-alpine as builder
2 WORKDIR /usr/src/app
3 COPY . .
4 RUN npm install
5 RUN npm run build
6 FROM nginx:alpine
7 COPY --from=builder /usr/src/app/build /usr/share/nginx/html

```

Figure 8: Fichier docker pour le Frontend

Pour le Backend :

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



```

1 FROM maven:3.8.4-openjdk-17 AS builder
2 WORKDIR /app
3 COPY ./src ./src
4 COPY ./pom.xml .
5 RUN mvn clean package
6
7 # Stage 2: Create the final image
8
9 FROM openjdk:17-jdk-alpine
10 VOLUME /tmp
11 ARG JAR_FILE=target/*.jar
12 COPY ${JAR_FILE} app.jar
13 ENTRYPOINT ["java", "-jar", "/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 créé un fichier docker compose avec une syntaxe d'architecture micro-service :

```

version: '3'
services:
  eurekaserver:
    image: ayoub/eurekaserver:latest
    ports:
      - "8761:8761"
  gateway:
    image: ayoub/gateway:latest
    ports:
      - "8888:8888"
    depends_on:
      - eurekaserver
  mysql_user:
    image: mysql:latest
    environment:
      MYSQL_DATABASE: userblog
      MYSQL_ROOT_PASSWORD: root
    ports:
      - "3306:3306"
  mysql_post:
    image: mysql:latest
    environment:
      MYSQL_DATABASE: postdb
      MYSQL_ROOT_PASSWORD: root
    ports:
      - "3307:3306"
  mysql_commente:
    image: mysql:latest
    environment:
      MYSQL_DATABASE: commentdb
      MYSQL_ROOT_PASSWORD: root
    ports:
      - "3308:3306"
  mysql_favorite:
    image: mysql:latest

```

Figure 10: Pipeline partie1

```

    image: mysql:latest
    environment:
      MYSQL_DATABASE: favoritedb
      MYSQL_ROOT_PASSWORD: root
    ports:
      - "3309:3306"

  user:
    image: ayoub/user:latest
    ports:
      - "8081:8081"
    depends_on:
      - eurekaservers
      - mysql_user
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql_user:3306/userblog?createDatabaseIfNotExist=true
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root

  post:
    image: ayoub/post:latest
    ports:
      - "8082:8082"
    depends_on:
      - eurekaserver
      - mysql_post
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql_post:3306/postdb?createDatabaseIfNotExist=true
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root

  commente:
    image: ayoub/commentaire:latest
    ports:
      - "8084:8084"
    depends_on:
      - eurekaserver
      - mysql_commente
    environment:

```

Figure 11: Pipeline partie2

```

    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql_commente:3306/commentdb?createDatabaseIfNotExist=true
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root

  favorite:
    image: ayoub/favorite:latest
    ports:
      - "8083:8083"
    depends_on:
      - eurekaserver
      - mysql_favorite
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql_favorite:3306/favoritedb?createDatabaseIfNotExist=true
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root

  frontend:
    image: ayoub/front:latest
    ports:
      - "80:80"
    depends_on:
      - gateway

```

Figure 12: Pipeline partie3

4.2 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.

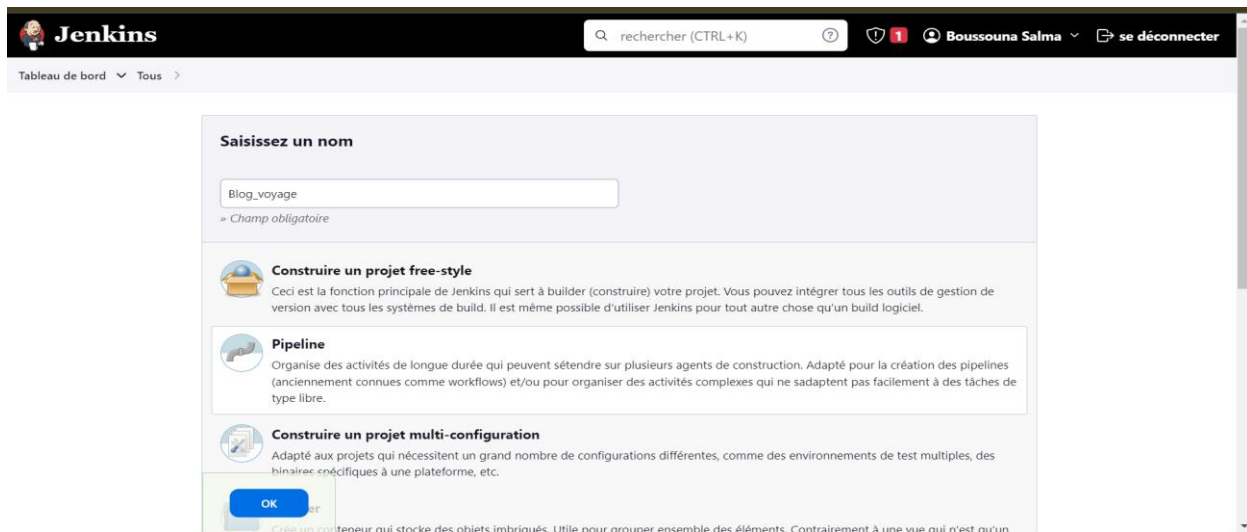
➤ Gestion des versions :

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

5 CI/CD avec Jenkins

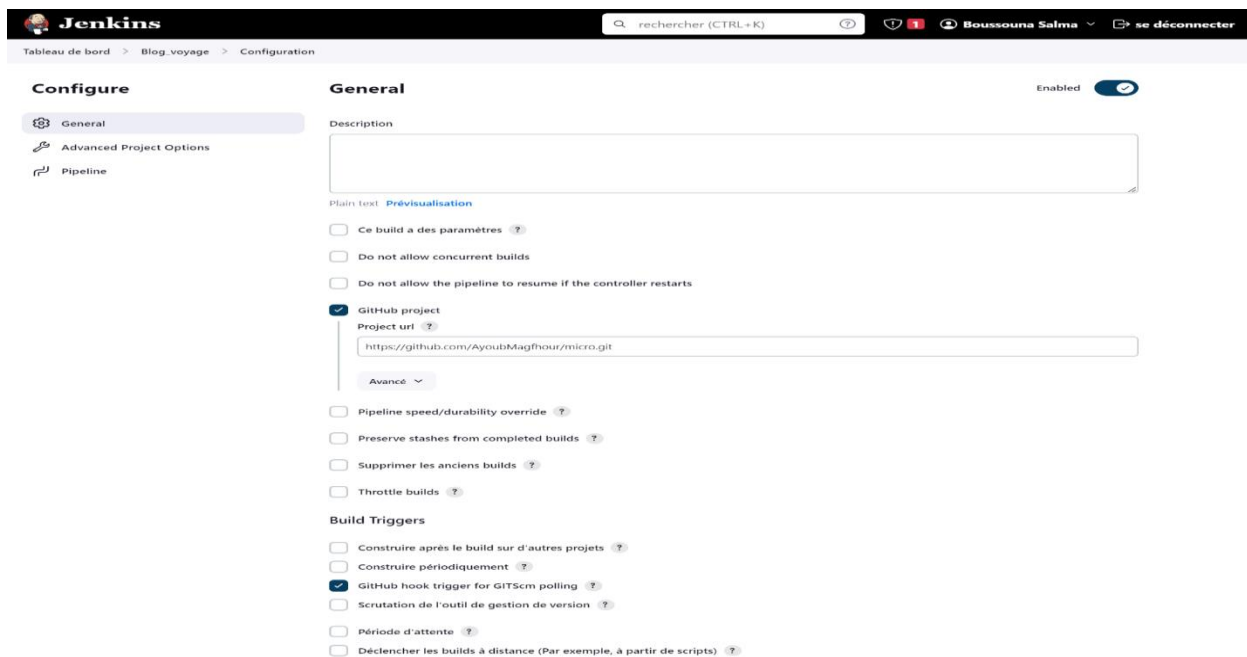
5.1 Processus et configuration :

5.1.1 Création d'un nouveau item :



The screenshot shows the Jenkins 'Create New Item' page. At the top, there's a header with the Jenkins logo, a search bar, and user information. Below the header, a breadcrumb trail shows 'Tableau de bord > Tous >'. The main content area is titled 'Saisissez un nom' and contains a text input field with 'Blog_voyage' entered. Below the input field, there's a note '» Champ obligatoire'. Underneath, there are three options for creating a new item: 'Construire un projet free-style', 'Pipeline', and 'Construire un projet multi-configuration'. Each option has a brief description. At the bottom left, there's a blue 'OK' button.

Figure 13:Création de l'item



The screenshot shows the Jenkins 'Configure' page for the 'Blog_voyage' item. The page has a sidebar on the left with 'Configure' at the top and three sub-items: 'General', 'Advanced Project Options', and 'Pipeline'. The 'General' tab is selected. The main content area is titled 'General' and has an 'Enabled' toggle switch. Below this, there's a 'Description' text area. Underneath, there's a 'Plain text' label and a 'Prévisualisation' link. A list of checkboxes follows: 'Ce build a des paramètres', 'Do not allow concurrent builds', 'Do not allow the pipeline to resume if the controller restarts', 'GitHub project' (checked), 'Project url' (with a value 'https://github.com/AyoubMaghbour/micro.git'), 'Avancé' (dropdown), 'Pipeline speed/durability override', 'Preserve stashes from completed builds', 'Supprimer les anciens builds', and 'Throttle builds'. Below this is a 'Build Triggers' section with checkboxes: 'Construire après le build sur d'autres projets', 'Construire périodiquement', 'GitHub hook trigger for GITScm polling' (checked), 'Scrutation de l'outil de gestion de version', 'Période d'attente', and 'Déclencher les builds à distance (Par exemple, à partir de scripts)'.

Figure 14: Configuration

Notre pipeline et comme suit :

```
pipeline {
  agent any
  tools {
    maven 'Maven'
  }

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

    stage('Build Backend') {
      steps {
        script {
          // Build Maven project for each microservice
          dir('Blog_Voyage_version2_Backend/Backend/User') {
            bat 'mvn clean install'
          }
          dir('Blog_Voyage_version2_Backend/Backend/Favorite') {
            bat 'mvn clean install'
          }
          dir('Blog_Voyage_version2_Backend/Backend/Post') {
            bat 'mvn clean install'
          }
          dir('Blog_Voyage_version2_Backend/Backend/Commentaire') {
            bat 'mvn clean install'
          }
          dir('Blog_Voyage_version2_Backend/Backend/EurekaServer') {
            bat 'mvn clean install'
          }
          dir('Blog_Voyage_version2_Backend/Backend/Gateway') {
            bat 'mvn clean install'
          }
        }
      }
    }

    stage('Create Docker Image (Backend)') {
      steps {
        script {
          // Build Docker image for each microservice
          dir('Blog_Voyage_version2_Backend/Backend/EurekaServer') {
            bat 'docker build -t ayoub/eurekaserver .'
          }
          dir('Blog_Voyage_version2_Backend/Backend/Gateway') {
            bat 'docker build -t ayoub/gateway .'
          }
          dir('Blog_Voyage_version2_Backend/Backend/Post') {
            bat 'docker build -t ayoub/post .'
          }
          dir('Blog_Voyage_version2_Backend/Backend/Commentaire') {
            bat 'docker build -t ayoub/commentaire .'
          }
          dir('Blog_Voyage_version2_Backend/Backend/Favorite') {
            bat 'docker build -t ayoub/favorite .'
          }
          dir('Blog_Voyage_version2_Backend/Backend/User') {
            bat 'docker build -t ayoub/user .'
          }
        }
      }
    }

    stage('Build and Create Docker Image (Frontend)') {
      steps {
        script {
          // Build frontend project
          dir('Blog_Voyage_Front') {
            bat 'npm install'
          }

          // Build Docker image for frontend
          bat 'docker build -t ayoub/front ./Blog_Voyage_Front'
        }
      }
    }

    stage('Run (Backend and Frontend)') {
      steps {
        script {
          // Stop and remove the previous containers if exist
          bat 'docker-compose down'

          // Run the new Docker containers for both backend and frontend
          bat 'docker-compose up -d'
        }
      }
    }
  }

  post {
    success {
      echo 'Pipeline succeeded! Your services are deployed.'
    }
    failure {
      echo 'Pipeline failed! Check the logs for errors.'
    }
  }
}
```

Figure 15: Code pipeline

Après le lancement du pipeline :

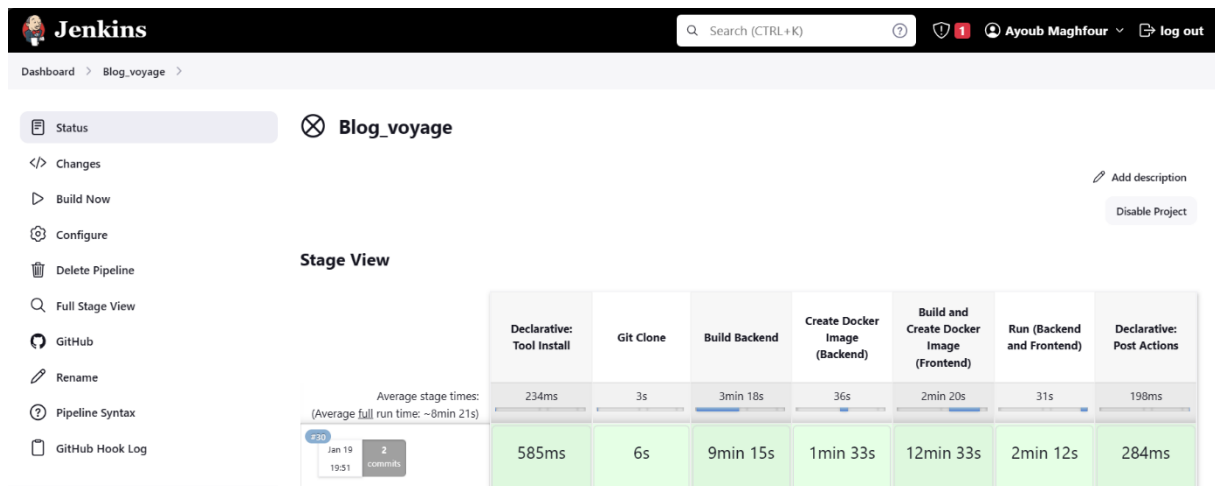


Figure 16: Résultat du lancement de pipeline

Création des images dans docker :

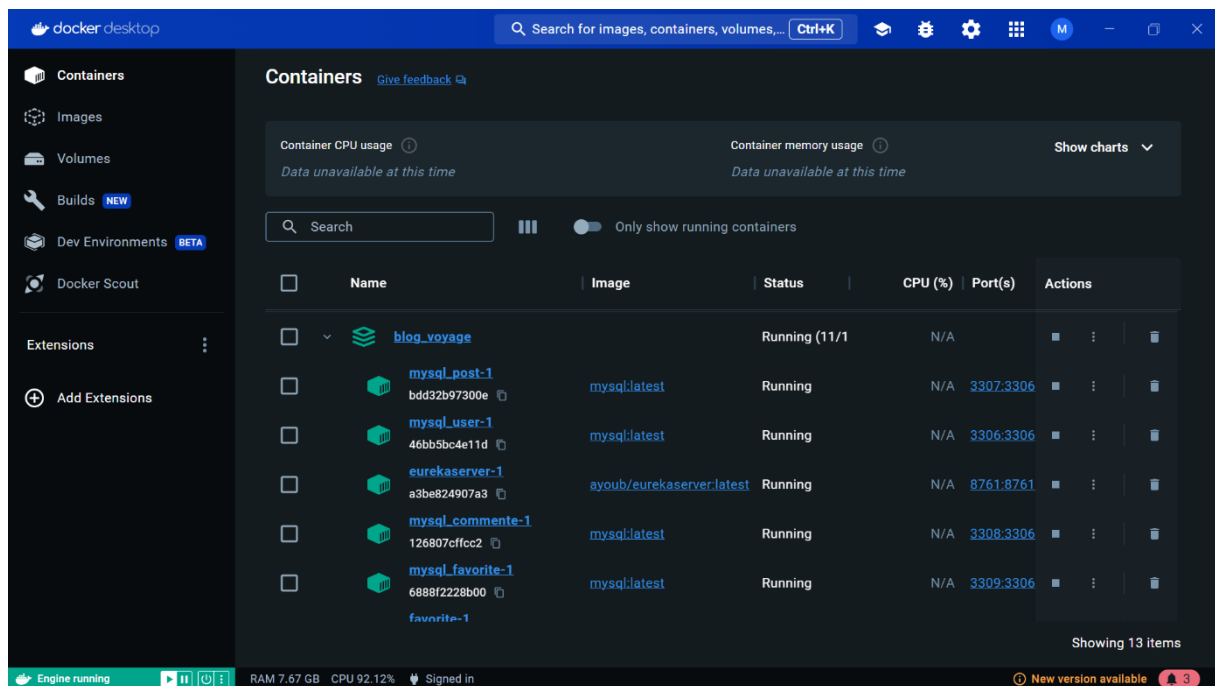


Figure 17: Création des images docker

6 Déploiement Automatique :

6.1 Création du host avec Ngrok

Créer le host avec la commande « ngrok http 5555 ».

```
Administrator: C:\Users\ayoub\Downloads\ngrok.exe - ngrok http 5555
ngrok
Build better APIs with ngrok. Early access: ngrok.com/early-access

Session Status      online
Account             ayoub (Plan: Free)
Version             3.5.0
Region              Europe (eu)
Latency             85ms
Web Interface       http://127.0.0.1:4040
Forwarding           https://c1aa-105-67-131-87.ngrok-free.app -> http://localhost:5555

Connections          ttl    opn    rt1    rt5    p50    p90
                   80     2     0.29   0.19   0.35   12.12

HTTP Requests
-----
GET /job/Blog_voyage/wfapi/runs          200 OK
GET /job/Blog_voyage/wfapi/runs          200 OK
GET /job/Blog_voyage/wfapi/runs          200 OK
GET /job/Blog_voyage/wfapi/runs          200 OK
GET /job/Blog_voyage/25/wfapi/changesets 200 OK
GET /job/Blog_voyage/28/wfapi/changesets 200 OK
GET /job/Blog_voyage/26/wfapi/changesets 200 OK
GET /job/Blog_voyage/27/wfapi/changesets 200 OK
GET /job/Blog_voyage/29/wfapi/changesets 200 OK
GET /job/Blog_voyage/wfapi/runs          200 OK
```

Figure 18: Création du host

6.2 Création du webhook dans Github :

1. Accédez à votre dépôt de projet.
2. Allez dans "Paramètres" dans le coin supérieur droit.
3. Cliquez sur "Webhooks".
4. Cliquez sur "Ajouter un webhook".
5. Indiquez l'URL de charge utile (Payload URL) comme :
URL_générée_par_ngrok/github-
6. webhook.
7. Type de contenu : application/json.

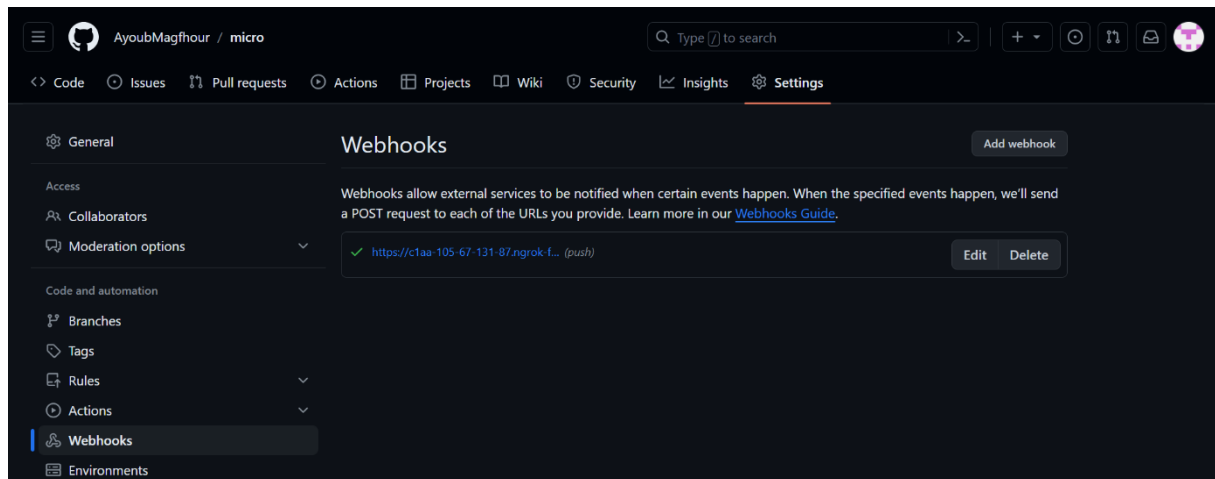


Figure 19: Webhook

6.3 Ajouter WebHook Github dans Jenkins :

Accédez à Manage jenkins / Plugins / Available plugins



Figure 20: Configuration Plugins

Allez a Manage jenkins / System



Figure 21: Configuration 2

6.4 Test :

En ajoute modification dans le code après en push au github repo

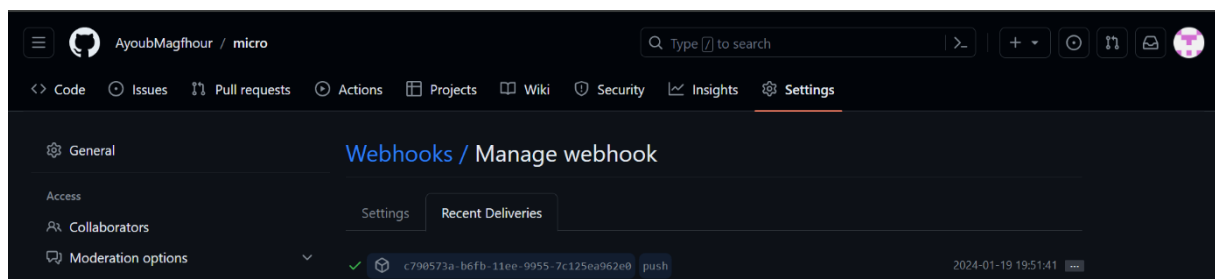


Figure 22::Push dans github

Après le déploiement automatique activé :

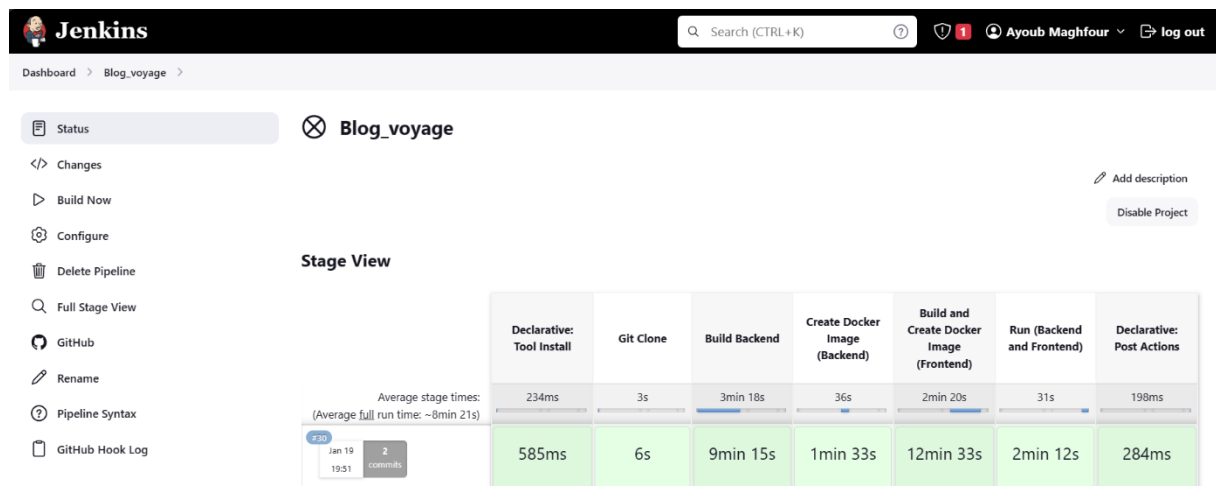


Figure 23: Déploiement automatique

7 Conclusion :

7.1 Accomplissements:

➤ Architecture Micro-services:

Mise en place d'une architecture micro-services pour le blog de voyage, 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.

7.2 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.