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

Réaliser Par :

OTHMAN MABROUR / NAIFI Youssef / ABRIL Yahia

5IIR G3

Table des matières

Introduction Générale

Les systèmes informatiques complexes requièrent une architecture pensée, évolutive et capable de répondre aux demandes croissantes de performance, de flexibilité et de facilité de gestion. Le développement de solutions logicielles, telles que les plateformes de réservation pour des événements, des spectacles et des cinémas, nécessite une approche architecturale solide. Dans ce contexte, l'architecture microservices émerge comme une solution innovante pour répondre à ces défis. Cette approche divise une application en composants autonomes et indépendants, appelés microservices, permettant une évolution et une maintenance agiles.

Notre projet se focalise sur la création d'une plateforme de réservation de billets, s'appuyant sur une architecture microservices bien pensée. Cette plateforme se compose de plusieurs microservices tels que le serveur, la passerelle (gateway), l'administration, le client, les microservices dédiés aux événements et aux réservations. La communication entre ces microservices est orchestrée avec l'utilisation du modèle Rest Template, favorisant ainsi une intégration légère et efficace basée sur les principes REST (Representational State Transfer).

Pour assurer la portabilité, la gestion des dépendances simplifiée, et faciliter le déploiement et la mise à l'échelle des microservices, nous avons choisi d'implémenter la conteneurisation avec Docker. Cette approche permet également une répétabilité des environnements, garantissant que chaque instance fonctionne de manière cohérente, indépendamment de l'environnement d'exécution.

Le déploiement automatique, facilité par l'utilisation de Jenkins en conjonction avec Ngrok et des webhooks, représente un pilier essentiel de la disponibilité continue de notre plateforme. Cette combinaison offre un déploiement automatisé à chaque mise à jour du code source, garantissant une réactivité accrue aux changements et une expérience utilisateur sans interruption.

Enfin, l'intégration de SonarQube avec Jenkins assure la qualité du code, détectant et résolvant les problèmes potentiels, contribuant ainsi à la création d'une base de code propre, maintenable et conforme aux normes établies.

Ce rapport explore en détail chaque composant de notre architecture microservices, du choix de l'architecture à la mise en œuvre des technologies clés, tout en mettant en lumière les avantages et les perspectives de cette approche novatrice.

1. Introduction

- **Aperçu du projet :**

Le projet vise à développer une application web de réservation de billets pour des événements, des spectacles, des cinémas... Notre application permettra aux utilisateurs de rechercher, consulter des informations sur les événements, et effectuer des réservations en ligne. L'objectif est de créer une solution robuste, scalable et facile à maintenir.

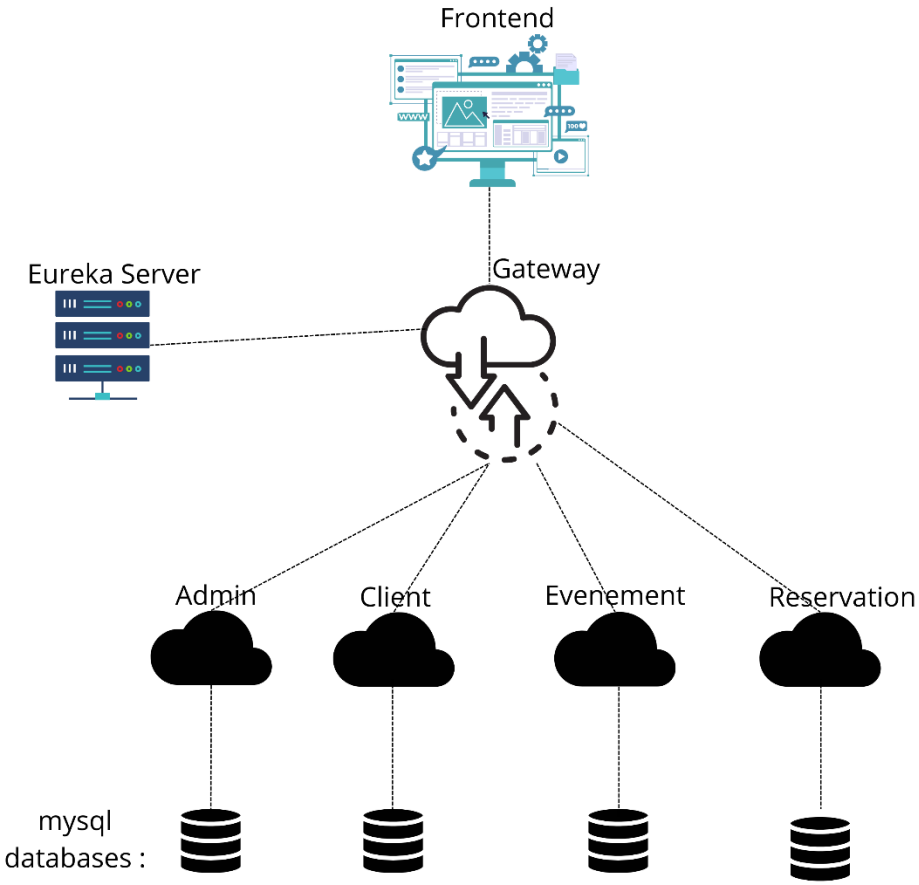
- **Importance de l'architecture microservices :**

L'architecture microservices a été choisie pour ce projet en raison de ses nombreux avantages. Elle favorise la scalabilité, la flexibilité et la facilité de déploiement. Chaque microservice est indépendant, ce qui permet une évolution et une maintenance aisées. De plus, cette architecture facilite la gestion des différentes fonctionnalités de la plateforme.

2. Architecture Microservices

- **Architecture :**

L'architecture microservices est composée des éléments suivants : serveur, gateway, admin, client, événements et réservation.

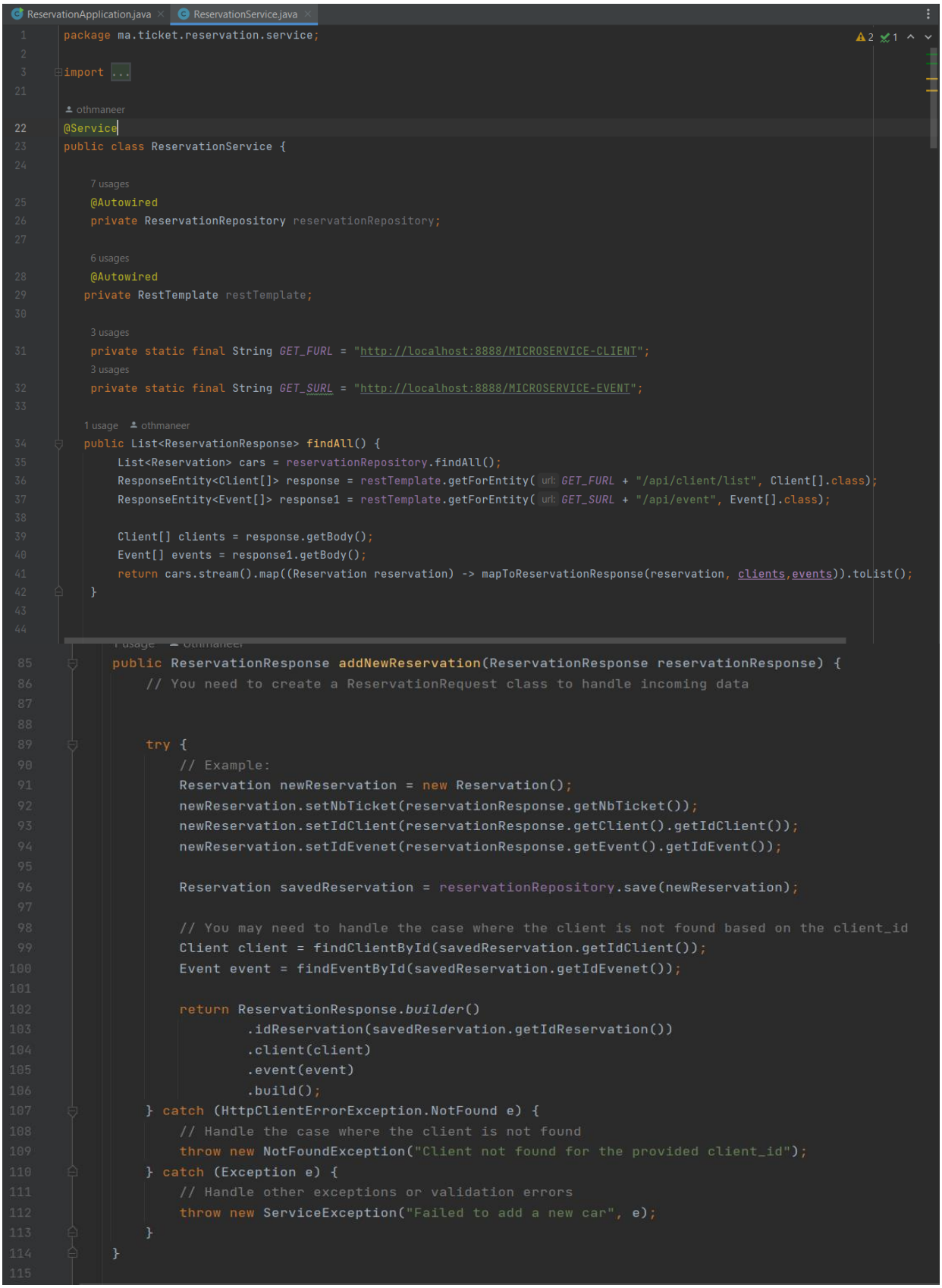


- **Description des services :**

Serveur : Héberge tous les microservices et assure la communication entre eux.
Gateway : Routage des requêtes clients vers les services appropriés.
Admin : Gestion des fonctionnalités d'administration telles que la gestion des utilisateurs, des événements et des réservations.
Client : Interface utilisateur fournissant la recherche et la réservation de billets.
Événements : Stockage et gestion des informations relatives aux événements tels que les détails, les horaires et les emplacements.
Réservation : Gestion des opérations de réservation, y compris la disponibilité des billets et la confirmation des réservations.

- **Mécanismes de communication : (Rest Template) :**

La communication entre les microservices est réalisée à l'aide du modèle Rest Template, basé sur les principes REST (Representational State Transfer).


ReservationApplication.java
ReservationService.java
package ma.ticket.reservation.service;
import ...
othmaneer
@Service
public class ReservationService {
7 usages
@Autowired
private ReservationRepository reservationRepository;
6 usages
@Autowired
private RestTemplate restTemplate;
3 usages
private static final String GET_FURL = "http://localhost:8888/MICROSERVICE-CLIENT";
3 usages
private static final String GET_SURL = "http://localhost:8888/MICROSERVICE-EVENT";
1 usage othmaneer
public List<ReservationResponse> findAll() {
List<Reservation> cars = reservationRepository.findAll();
ResponseEntity<Client[]> response = restTemplate.getForEntity(url: GET_FURL + "/api/client/list", Client[].class);
ResponseEntity<Event[]> response1 = restTemplate.getForEntity(url: GET_SURL + "/api/event", Event[].class);
Client[] clients = response.getBody();
Event[] events = response1.getBody();
return cars.stream().map((Reservation reservation) -> mapToReservationResponse(reservation, clients,events)).toList();
}
public ReservationResponse addNewReservation(ReservationResponse reservationResponse) {
// You need to create a ReservationRequest class to handle incoming data
try {
// Example:
Reservation newReservation = new Reservation();
newReservation.setNbTicket(reservationResponse.getNbTicket());
newReservation.setIdClient(reservationResponse.getClient().getIdClient());
newReservation.setIdEvenet(reservationResponse.getEvent().getIdEvent());
Reservation savedReservation = reservationRepository.save(newReservation);
// You may need to handle the case where the client is not found based on the client_id
Client client = findClientById(savedReservation.getIdClient());
Event event = findEventById(savedReservation.getIdEvenet());
return ReservationResponse.builder()
.idReservation(savedReservation.getIdReservation())
.client(client)
.event(event)
.build();
} catch (HttpClientErrorException.NotFound e) {
// Handle the case where the client is not found
throw new NotFoundException("Client not found for the provided client_id");
} catch (Exception e) {
// Handle other exceptions or validation errors
throw new ServiceException("Failed to add a new car", e);
}
}

Un niveau de la couche service du microservice on trouve la communication entre microservice ou utilisant le gateway. Un exemple concret de cette interconnexion se produit lors de l'ajout d'une réservation.
Lorsqu'une réservation doit être effectuée, la couche service du microservice responsable de la gestion des réservations intervient. Pour ce faire, elle communique avec d'autres microservices, en l'occurrence le microservice client et le microservice événements.
Tout d'abord, pour associer une réservation à un client spécifique, le service de réservation effectue une requête vers le microservice client pour récupérer les informations du client concerné. Cette étape permet de garantir que la réservation est correctement liée au profil du client, assurant ainsi une traçabilité et une personnalisation appropriées.
En parallèle, pour déterminer les détails de l'événement pour lequel la réservation est effectuée, la couche service de réservation communique avec le microservice événements. Cette communication permet de récupérer les informations nécessaires sur l'événement en question, telles que la date, l'emplacement et d'autres détails pertinents.
L'ajout de la réservation ne peut aboutir si le cas de réservation est déjà enregistré. Cela garantit l'intégrité du processus de réservation en évitant les doublons et en assurant une gestion efficace des réservations.

3. Conception des Microservices

- **Approche de conception pour chaque service :**

Chaque microservice est conçu de manière indépendante, avec une approche spécifique pour ses fonctionnalités.
Serveur : Déployé en tant qu'hôte principal, gérant la communication entre les services.
Gateway : Gère le routage des requêtes clients vers les microservices appropriés.
Admin : Fonctionnalités d'administration avec des modules distincts pour les utilisateurs, les événements et les réservations.
Client : Interface utilisateur simple pour une expérience utilisateur optimale.
Événements : Stockage des données sur les événements avec des API pour la récupération et la modification.
Réservation : Gestion des opérations de réservation.

4. Conteneurisation avec Docker

- **Implémentation et avantages :**

La solution est conteneurisée à l'aide de Docker pour assurer une portabilité et une gestion des dépendances simplifiées. Cela facilite le déploiement et la mise à l'échelle des microservices de manière efficace.

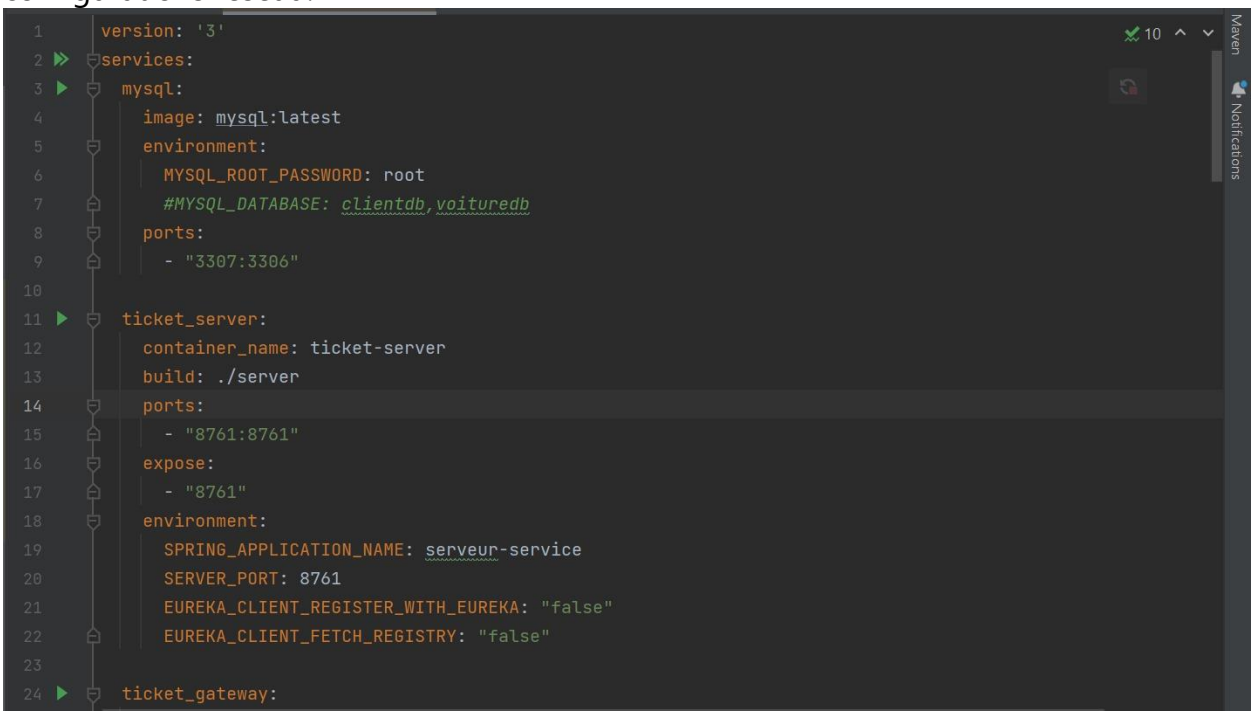
Étapes d'Implémentation :

1. Création des Dockerfiles :

Pour chaque microservice, un fichier Dockerfile est créé pour définir les dépendances, les étapes d'installation et la configuration nécessaire.

2. Configuration du Docker Compose :

Un fichier docker-compose.yml est créé pour définir les services, les dépendances et les configurations réseau.



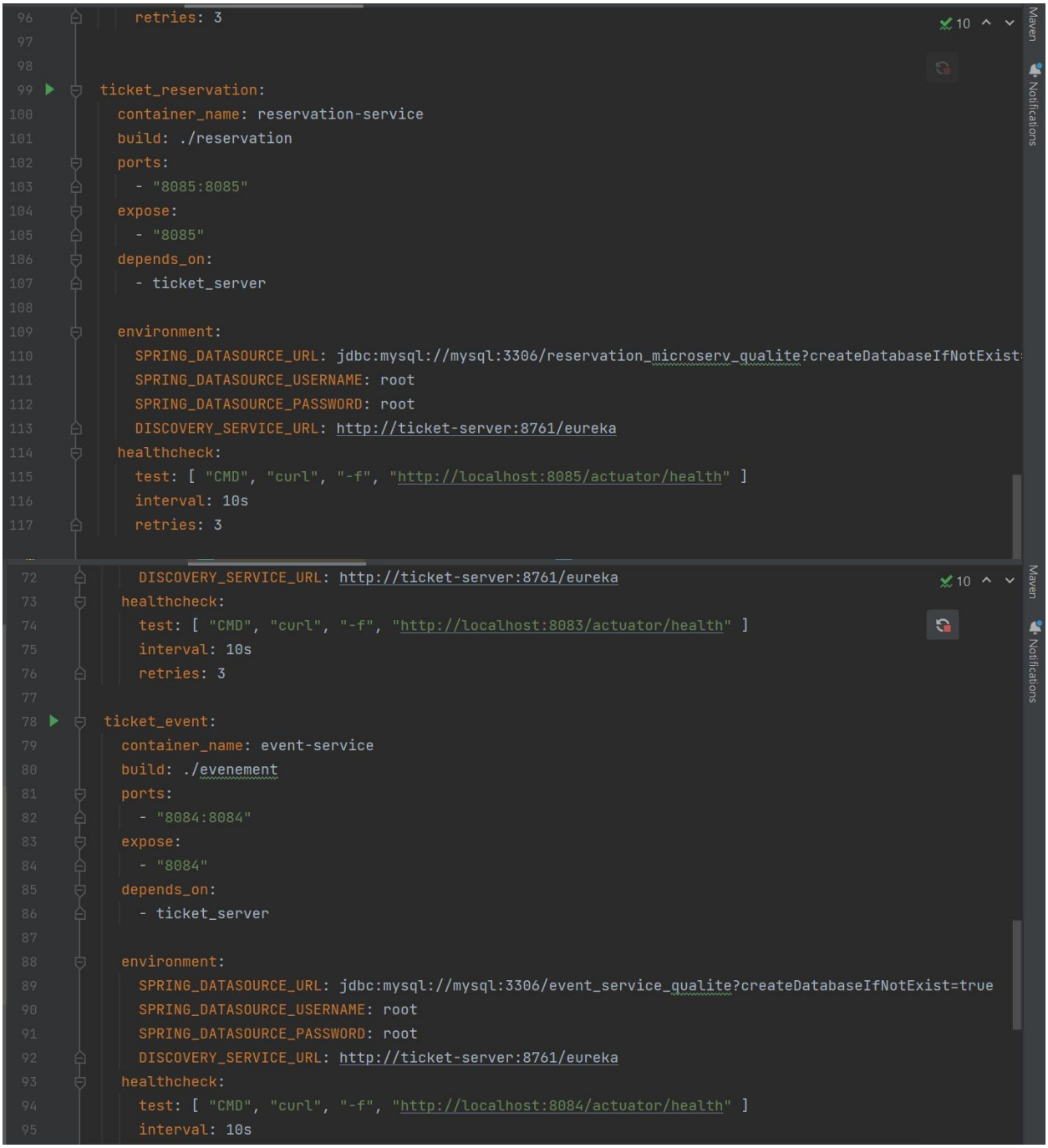
```yaml
    container_name: gateway-server
    build: ./gate
    ports:
      - "8888:8888"
    expose:
      - "8888"
    environment:
      SPRING_APPLICATION_NAME: Gateway
      EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://ticket-server:8761/eureka
    depends_on:
      - ticket_server


  ticket_client:
    container_name: client-service
    build: ./client
    ports:
      - "8080:8080"
    expose:
      - "8080"
    depends_on:
      - ticket_server

    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/clientdb?createDatabaseIfNotExist=true
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
      DISCOVERY_SERVICE_URL: http://ticket-server:8761/eureka
    healthcheck:
      test: [ "CMD", "curl", "-f", "http://localhost:8080/actuator/health" ]
      interval: 10s
      retries: 3

  ticket_admin:
    container_name: admin-service
    build: ./admin
    ports:
      - "8083:8083"
    expose:
      - "8083"
    depends_on:
      - ticket_server

    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/admin_service?createDatabaseIfNotExist=true
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
```



```
    retries: 3

ticket_reservation:
  container_name: reservation-service
  build: ./reservation
  ports:
    - "8085:8085"
  expose:
    - "8085"
  depends_on:
    - ticket_server

  environment:
    SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/reservation_microserv_qualite?createDatabaseIfNotExist
    SPRING_DATASOURCE_USERNAME: root
    SPRING_DATASOURCE_PASSWORD: root
    DISCOVERY_SERVICE_URL: http://ticket-server:8761/eureka
  healthcheck:
    test: [ "CMD", "curl", "-f", "http://localhost:8085/actuator/health" ]
    interval: 10s
    retries: 3
```

```
    DISCOVERY_SERVICE_URL: http://ticket-server:8761/eureka
  healthcheck:
    test: [ "CMD", "curl", "-f", "http://localhost:8083/actuator/health" ]
    interval: 10s
    retries: 3

ticket_event:
  container_name: event-service
  build: ./evenement
  ports:
    - "8084:8084"
  expose:
    - "8084"
  depends_on:
    - ticket_server

  environment:
    SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/event_service_qualite?createDatabaseIfNotExist=true
    SPRING_DATASOURCE_USERNAME: root
    SPRING_DATASOURCE_PASSWORD: root
    DISCOVERY_SERVICE_URL: http://ticket-server:8761/eureka
  healthcheck:
    test: [ "CMD", "curl", "-f", "http://localhost:8084/actuator/health" ]
    interval: 10s
```

3. Exécution de la construction avec Maven :

Dans chaque répertoire de microservice, on execute la commande Maven pour construire le fichier JAR.

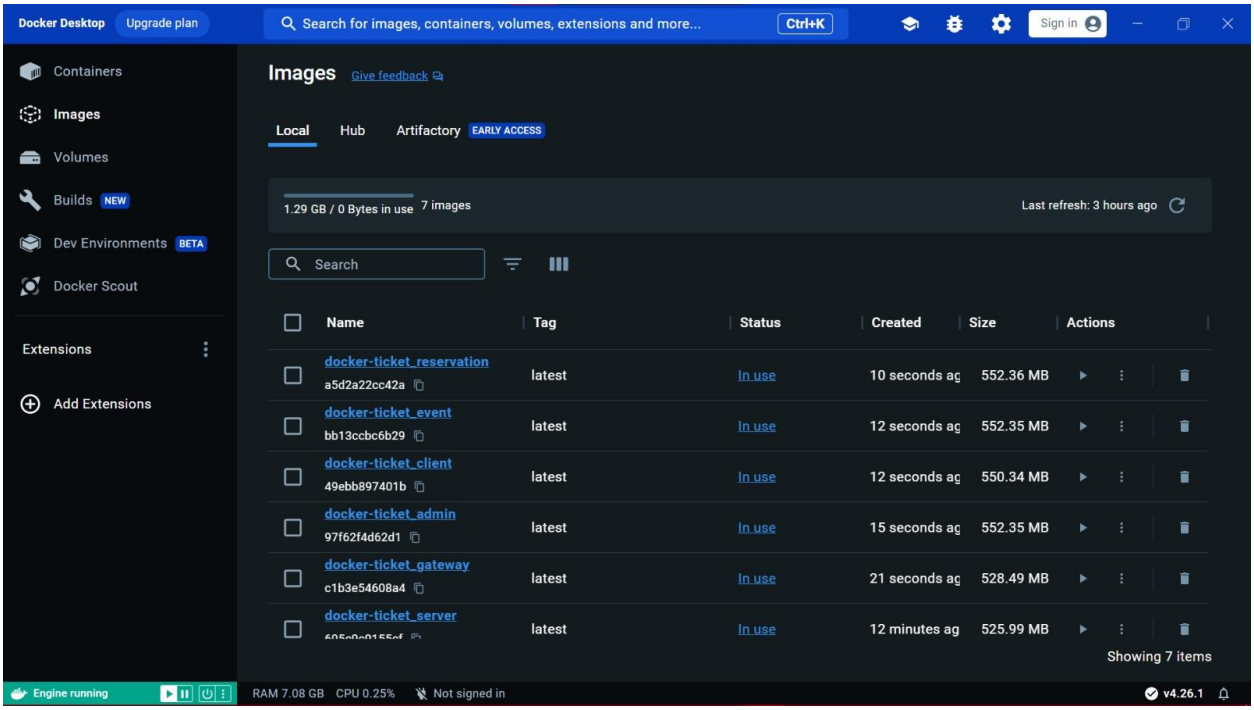
```
mvn clean install
```

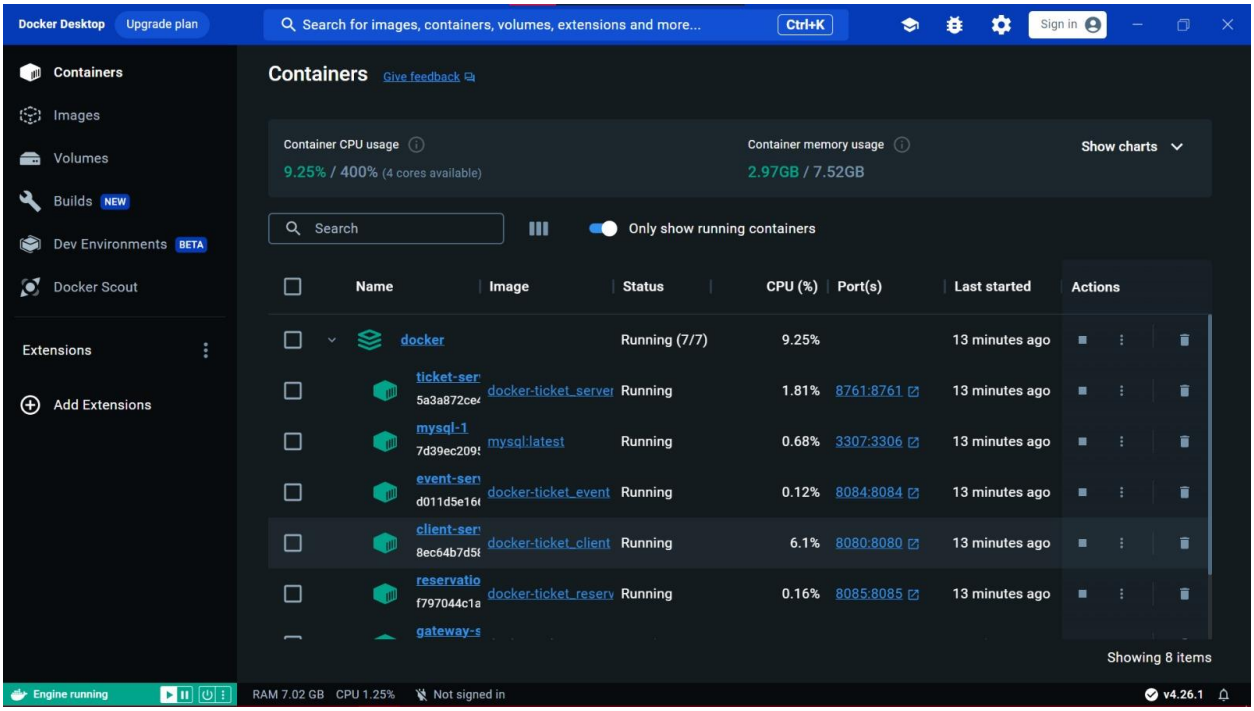
4. Exécution de Docker Compose :

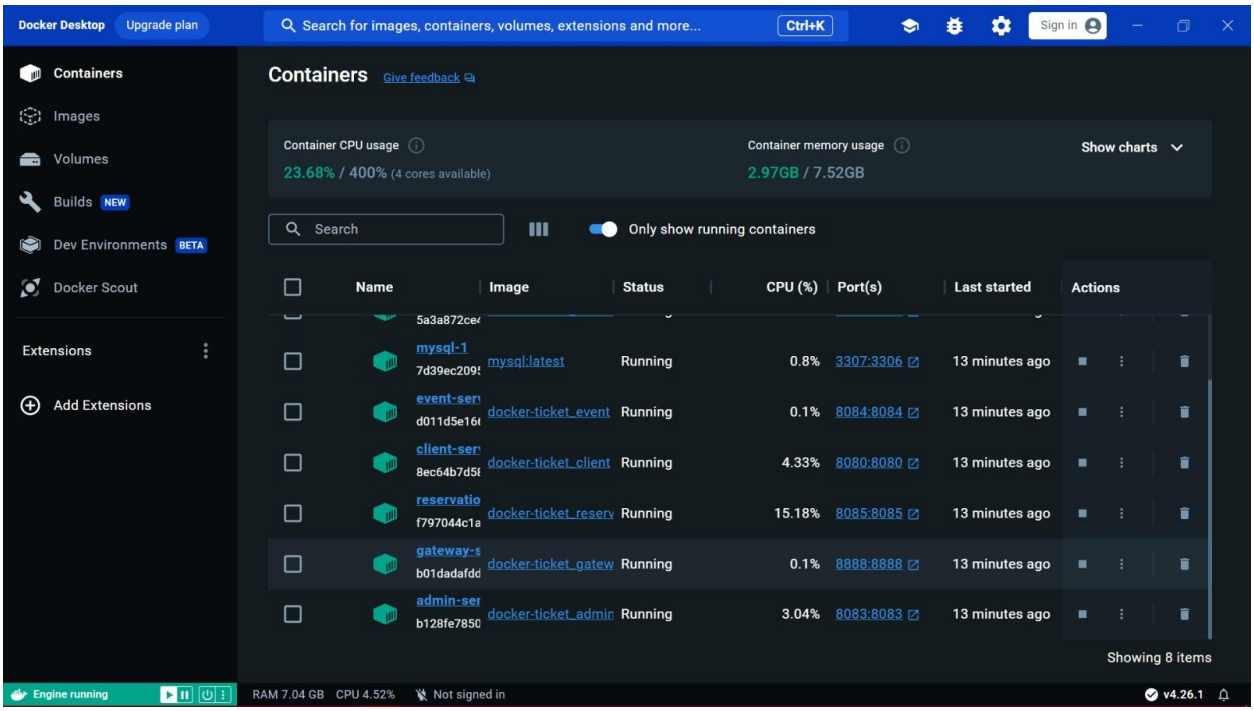
À la racine du projet, on exécute la commande Docker Compose pour démarrer tous les microservices.

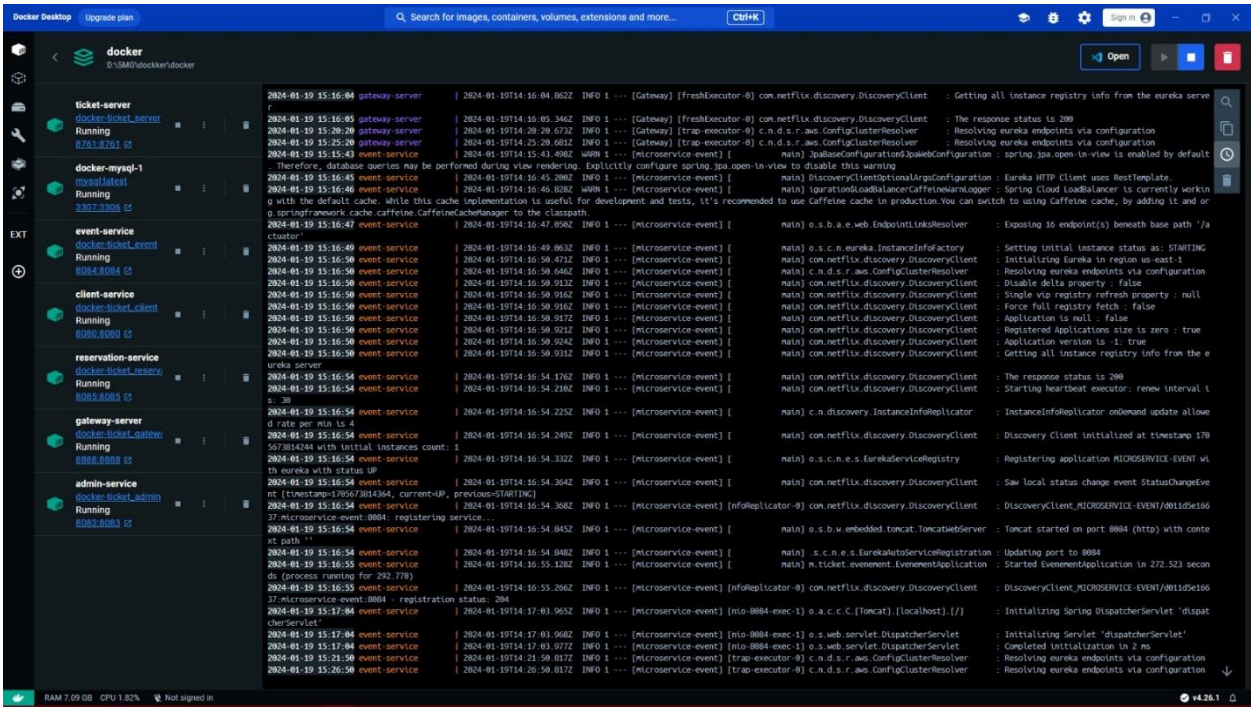
```
docker-compose up --build -d
```

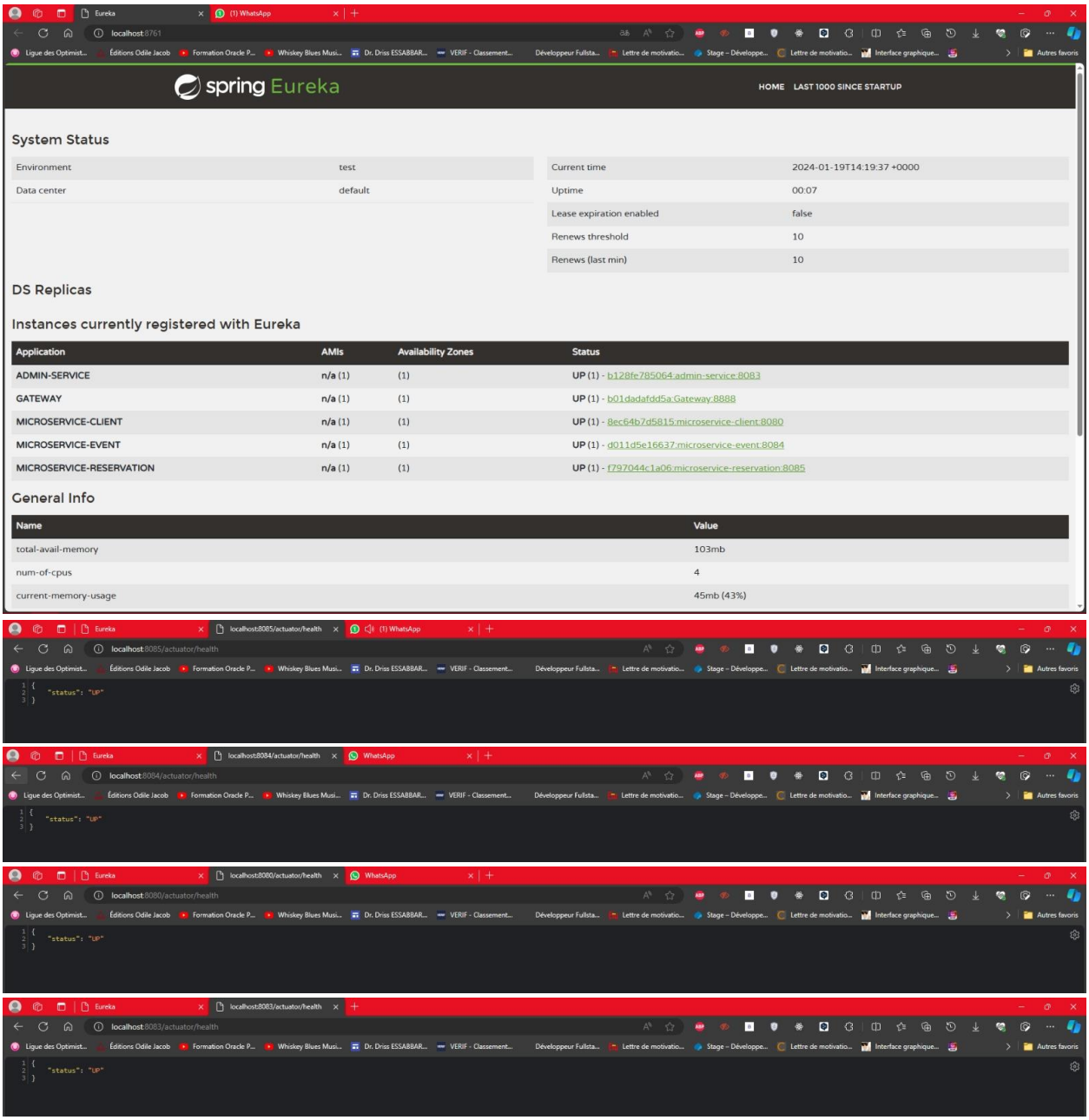
Résultat d'implémentation :






Docker Desktop
Upgrade plan
Search for images, containers, volumes, extensions and more...
Ctrl+K
Sign in
Containers
Images
Volumes
Builds NEW
Dev Environments BETA
Docker Scout
Extensions
Add Extensions
Containers
Give feedback
Container CPU usage
23.68% / 400% (4 cores available)
Container memory usage
2.97GB / 7.52GB
Show charts
Search
Only show running containers
Name
Image
Status
CPU (%)
Port(s)
Last started
Actions
mysql-1
mysql:latest
Running
0.8%
3307:3306
13 minutes ago
event-ser
docker-ticket_event
Running
0.1%
8084:8084
13 minutes ago
client-ser
docker-ticket_client
Running
4.33%
8080:8080
13 minutes ago
reservatio
docker-ticket_reserv
Running
15.18%
8085:8085
13 minutes ago
gateway-s
docker-ticket_gatew
Running
0.1%
8888:8888
13 minutes ago
admin-ser
docker-ticket_admin
Running
3.04%
8083:8083
13 minutes ago
Showing 8 items
Engine running
RAM 7.04 GB CPU 4.52%
Not signed in
v4.26.1


Docker Desktop
Upgrade plan
Search for images, containers, volumes, extensions and more...
Ctrl+K
Sign in
docker
D:\SMO\dockker\docker
Open
ticket-server
docker-ticket_server
Running
8761:8761
docker-mysql-1
mysql:latest
Running
3307:3306
event-service
docker-ticket_event
Running
8084:8084
client-service
docker-ticket_client
Running
8080:8080
reservation-service
docker-ticket_reserv
Running
8085:8085
gateway-server
docker-ticket_gatew
Running
8888:8888
admin-service
docker-ticket_admin
Running
8083:8083
EXT
2024-01-19 15:16:04 gateway-server | 2024-01-19T14:16:04.862Z INFO 1 --- [Gateway] [freshExecutor-0] com.netflix.discovery.DiscoveryClient : Getting all instance registry info from the eureka serve
r
2024-01-19 15:16:05 gateway-server | 2024-01-19T14:16:05.346Z INFO 1 --- [Gateway] [freshExecutor-0] com.netflix.discovery.DiscoveryClient : The response status is 200
2024-01-19 15:20:20 gateway-server | 2024-01-19T14:20:20.673Z INFO 1 --- [Gateway] [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration
2024-01-19 15:25:20 gateway-server | 2024-01-19T14:25:20.681Z INFO 1 --- [Gateway] [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration
2024-01-19 15:15:43 event-service | 2024-01-19T14:15:43.498Z WARN 1 --- [microservice-event] [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default
. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2024-01-19 15:16:45 event-service | 2024-01-19T14:16:45.200Z INFO 1 --- [microservice-event] [main] DiscoveryClientOptionalArgsConfiguration : Eureka HTTP Client uses RestTemplate.
2024-01-19 15:16:46 event-service | 2024-01-19T14:16:46.828Z WARN 1 --- [microservice-event] [main] iguration$LoadBalancerCaffeineWarnLogger : Spring Cloud LoadBalancer is currently workin
g with the default cache. While this cache implementation is useful for development and tests, it's recommended to use Caffeine cache in production.You can switch to using Caffeine cache, by adding it and or
g.springframework.cache.caffeine.CaffeineCacheManager to the classpath.
2024-01-19 15:16:47 event-service | 2024-01-19T14:16:47.850Z INFO 1 --- [microservice-event] [main] o.s.b.a.e.web.EndpointLinksResolver : Exposing 16 endpoint(s) beneath base path '/a
ctuator'
2024-01-19 15:16:49 event-service | 2024-01-19T14:16:49.863Z INFO 1 --- [microservice-event] [main] o.s.c.n.eureka.InstanceInfoFactory : Setting initial instance status as: STARTING
2024-01-19 15:16:50 event-service | 2024-01-19T14:16:50.471Z INFO 1 --- [microservice-event] [main] com.netflix.discovery.DiscoveryClient : Initializing Eureka in region us-east-1
2024-01-19 15:16:50 event-service | 2024-01-19T14:16:50.646Z INFO 1 --- [microservice-event] [main] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration
2024-01-19 15:16:50 event-service | 2024-01-19T14:16:50.913Z INFO 1 --- [microservice-event] [main] com.netflix.discovery.DiscoveryClient : Disable delta property : false
2024-01-19 15:16:50 event-service | 2024-01-19T14:16:50.916Z INFO 1 --- [microservice-event] [main] com.netflix.discovery.DiscoveryClient : Single vip registry refresh property : null
2024-01-19 15:16:50 event-service | 2024-01-19T14:16:50.916Z INFO 1 --- [microservice-event] [main] com.netflix.discovery.DiscoveryClient : Force full registry fetch : false
2024-01-19 15:16:50 event-service | 2024-01-19T14:16:50.917Z INFO 1 --- [microservice-event] [main] com.netflix.discovery.DiscoveryClient : Application is null : false
2024-01-19 15:16:50 event-service | 2024-01-19T14:16:50.921Z INFO 1 --- [microservice-event] [main] com.netflix.discovery.DiscoveryClient : Registered Applications size is zero : true
2024-01-19 15:16:50 event-service | 2024-01-19T14:16:50.924Z INFO 1 --- [microservice-event] [main] com.netflix.discovery.DiscoveryClient : Application version is -1: true
2024-01-19 15:16:50 event-service | 2024-01-19T14:16:50.931Z INFO 1 --- [microservice-event] [main] com.netflix.discovery.DiscoveryClient : Getting all instance registry info from the e
ureka server
2024-01-19 15:16:54 event-service | 2024-01-19T14:16:54.176Z INFO 1 --- [microservice-event] [main] com.netflix.discovery.DiscoveryClient : The response status is 200
2024-01-19 15:16:54 event-service | 2024-01-19T14:16:54.216Z INFO 1 --- [microservice-event] [main] com.netflix.discovery.DiscoveryClient : Starting heartbeat executor: renew interval i
s: 30
2024-01-19 15:16:54 event-service | 2024-01-19T14:16:54.225Z INFO 1 --- [microservice-event] [main] c.n.discovery.InstanceInfoReplicator : InstanceInfoReplicator onDemand update allowe
d rate per min is 4
2024-01-19 15:16:54 event-service | 2024-01-19T14:16:54.249Z INFO 1 --- [microservice-event] [main] com.netflix.discovery.DiscoveryClient : Discovery Client initialized at timestamp 170
5673814244 with initial instances count: 1
2024-01-19 15:16:54 event-service | 2024-01-19T14:16:54.332Z INFO 1 --- [microservice-event] [main] o.s.c.n.e.s.EurekaServiceRegistry : Registering application MICROSERVICE-EVENT wi
th eureka with status UP
2024-01-19 15:16:54 event-service | 2024-01-19T14:16:54.364Z INFO 1 --- [microservice-event] [main] com.netflix.discovery.DiscoveryClient : Saw local status change event StatusChangeEve
nt [timestamp=1705673814364, current=UP, previous=STARTING]
2024-01-19 15:16:54 event-service | 2024-01-19T14:16:54.368Z INFO 1 --- [microservice-event] [nfoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_MICROSERVICE-EVENT/d011d5e166
37.microservice-event:8084: registering service...
2024-01-19 15:16:54 event-service | 2024-01-19T14:16:54.845Z INFO 1 --- [microservice-event] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8084 (http) with conte
xt path ''
2024-01-19 15:16:54 event-service | 2024-01-19T14:16:54.848Z INFO 1 --- [microservice-event] [main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to 8084
2024-01-19 15:16:55 event-service | 2024-01-19T14:16:55.128Z INFO 1 --- [microservice-event] [main] m.ticket.evenement.EvenementApplication : Started EvenementApplication in 272.523 secon
ds (process running for 292.778)
2024-01-19 15:16:55 event-service | 2024-01-19T14:16:55.266Z INFO 1 --- [microservice-event] [nfoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_MICROSERVICE-EVENT/d011d5e166
37.microservice-event:8084 - registration status: 204
2024-01-19 15:17:04 event-service | 2024-01-19T14:17:03.965Z INFO 1 --- [microservice-event] [nio-8084-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispat
cherServlet'
2024-01-19 15:17:04 event-service | 2024-01-19T14:17:03.968Z INFO 1 --- [microservice-event] [nio-8084-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-01-19 15:17:04 event-service | 2024-01-19T14:17:03.977Z INFO 1 --- [microservice-event] [nio-8084-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
2024-01-19 15:21:50 event-service | 2024-01-19T14:21:50.817Z INFO 1 --- [microservice-event] [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration
2024-01-19 15:26:50 event-service | 2024-01-19T14:26:50.817Z INFO 1 --- [microservice-event] [trap-executor-0] c.n.d.s.r.aws.ConfigClusterResolver : Resolving eureka endpoints via configuration
RAM 7.09 GB CPU 1.82%
Not signed in
v4.26.1



Avantages :

- Portabilité :

Les conteneurs Docker encapsulent toutes les dépendances et configurations nécessaires, assurant ainsi la portabilité entre différents environnements.

- Gestion Simplifiée des Dépendances :

Docker permet d'isoler les dépendances de chaque microservice, évitant les conflits et simplifiant la gestion des dépendances.

- Facilité de Déploiement :

La conteneurisation simplifie le déploiement en garantissant que chaque microservice fonctionne dans son propre conteneur, indépendamment des autres.

- Mise à l'Échelle Efficace :

La nature légère des conteneurs Docker permet une mise à l'échelle efficace en ajoutant ou en retirant des instances de microservices selon les besoins.

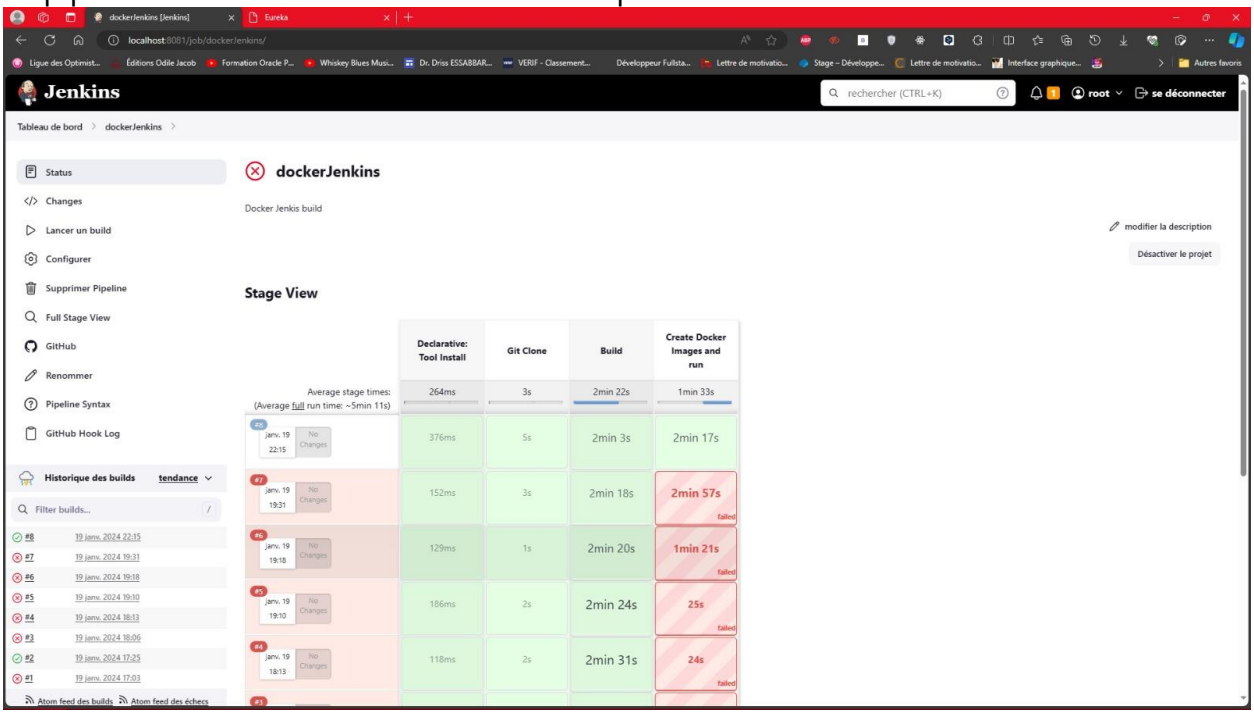
- Répétabilité des Environnements :

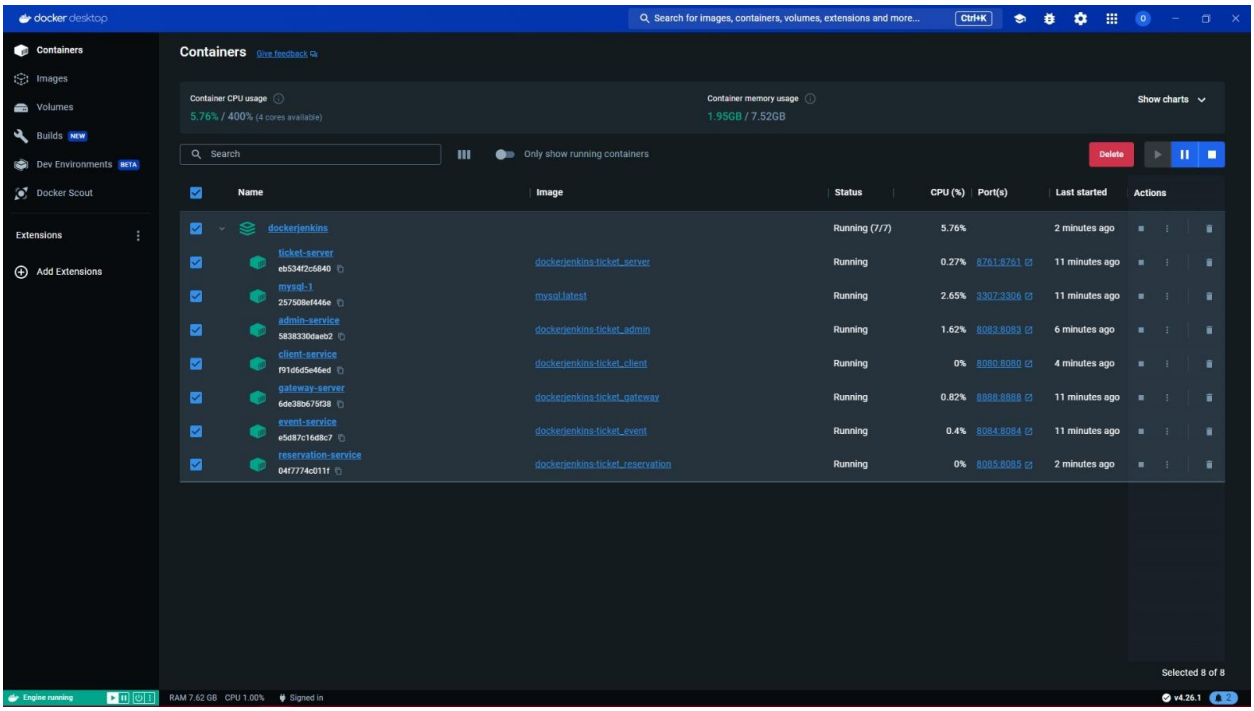
Les conteneurs Docker garantissent que chaque environnement, du développement à la production, est cohérent et répétable.

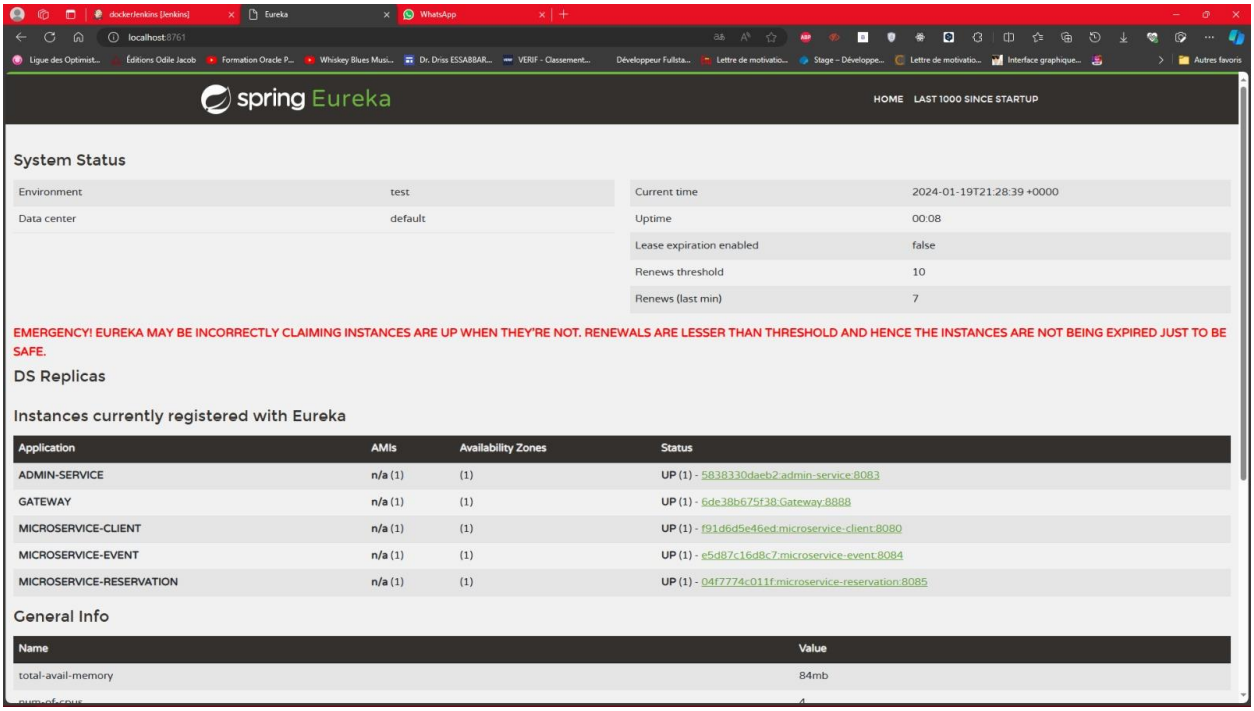
5. CI/CD avec Jenkins

- **Processus et configuration :**

Le processus CI/CD est mis en place à l'aide de Jenkins, automatisant le processus de construction, de test et de déploiement. Cela garantit une intégration continue et une livraison rapide des nouvelles fonctionnalités.
Le pipeline va être affiché dans le 7eme chapitre.




docker desktop
Search for images, containers, volumes, extensions and more...
Ctrl+K
Containers
Images
Volumes
Builds
Dev Environments
Docker Scout
Extensions
Add Extensions
Containers
Give feedback
Container CPU usage
5.76% / 400% (4 cores available)
Container memory usage
1.95GB / 7.52GB
Show charts
Search
Only show running containers
Delete
Name
Image
Status
CPU (%)
Port(s)
Last started
Actions
dockerjenkins
Running (7/7)
5.76%
2 minutes ago
ticket-server
eb534f2c6840
dockerjenkins-ticket_server
Running
0.27%
8761:8761
11 minutes ago
mysql-1
257508ef446e
mysql:latest
Running
2.65%
3307:3306
11 minutes ago
admin-service
5838330daeb2
dockerjenkins-ticket_admin
Running
1.62%
8083:8083
6 minutes ago
client-service
f91d6d5e46ed
dockerjenkins-ticket_client
Running
0%
8080:8080
4 minutes ago
gateway-server
6de38b675f38
dockerjenkins-ticket_gateway
Running
0.82%
8888:8888
11 minutes ago
event-service
e5d87c16d8c7
dockerjenkins-ticket_event
Running
0.4%
8084:8084
11 minutes ago
reservation-service
04f7774c011f
dockerjenkins-ticket_reservation
Running
0%
8085:8085
2 minutes ago
Selected 8 of 8
Engine running
RAM 7.62 GB CPU 1.00%
Signed in
v4.26.1

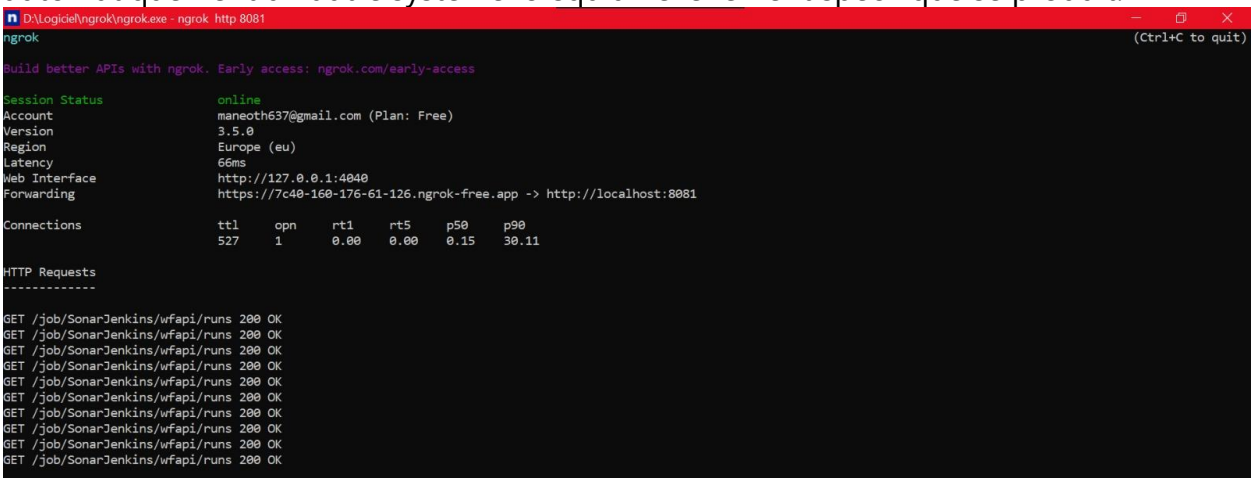

dockerJenkins [Jenkins]
Eureka
WhatsApp
localhost:8761
spring Eureka
HOME
LAST 1000 SINCE STARTUP
System Status
Environment
test
Data center
default
Current time
2024-01-19T21:28:39 +0000
Uptime
00:08
Lease expiration enabled
false
Renews threshold
10
Renews (last min)
7
EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.
DS Replicas
Instances currently registered with Eureka
Application
AMIs
Availability Zones
Status
ADMIN-SERVICE
n/a (1)
(1)
UP (1) - 5838330daeb2:admin-service:8083
GATEWAY
n/a (1)
(1)
UP (1) - 6de38b675f38:Gateway:8888
MICROSERVICE-CLIENT
n/a (1)
(1)
UP (1) - f91d6d5e46ed:microservice-client:8080
MICROSERVICE-EVENT
n/a (1)
(1)
UP (1) - e5d87c16d8c7:microservice-event:8084
MICROSERVICE-RESERVATION
n/a (1)
(1)
UP (1) - 04f7774c011f:microservice-reservation:8085
General Info
Name
Value
total-avail-memory
84mb

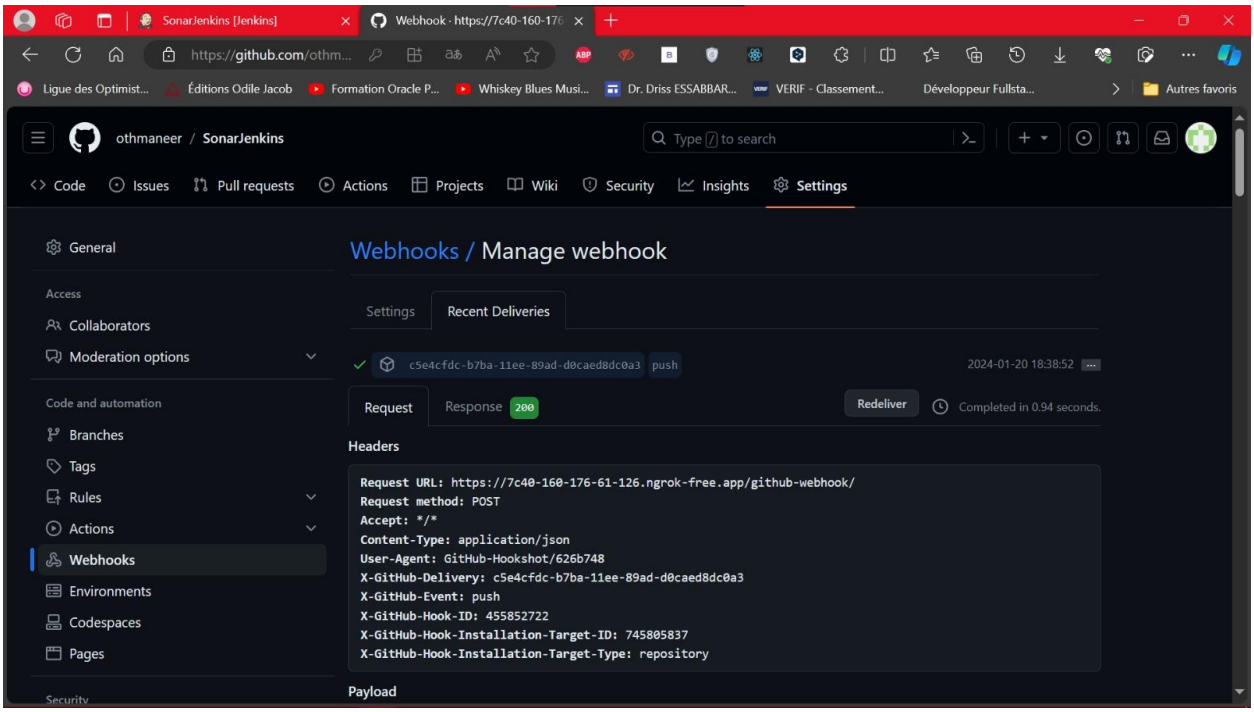
6. Déploiement Automatique

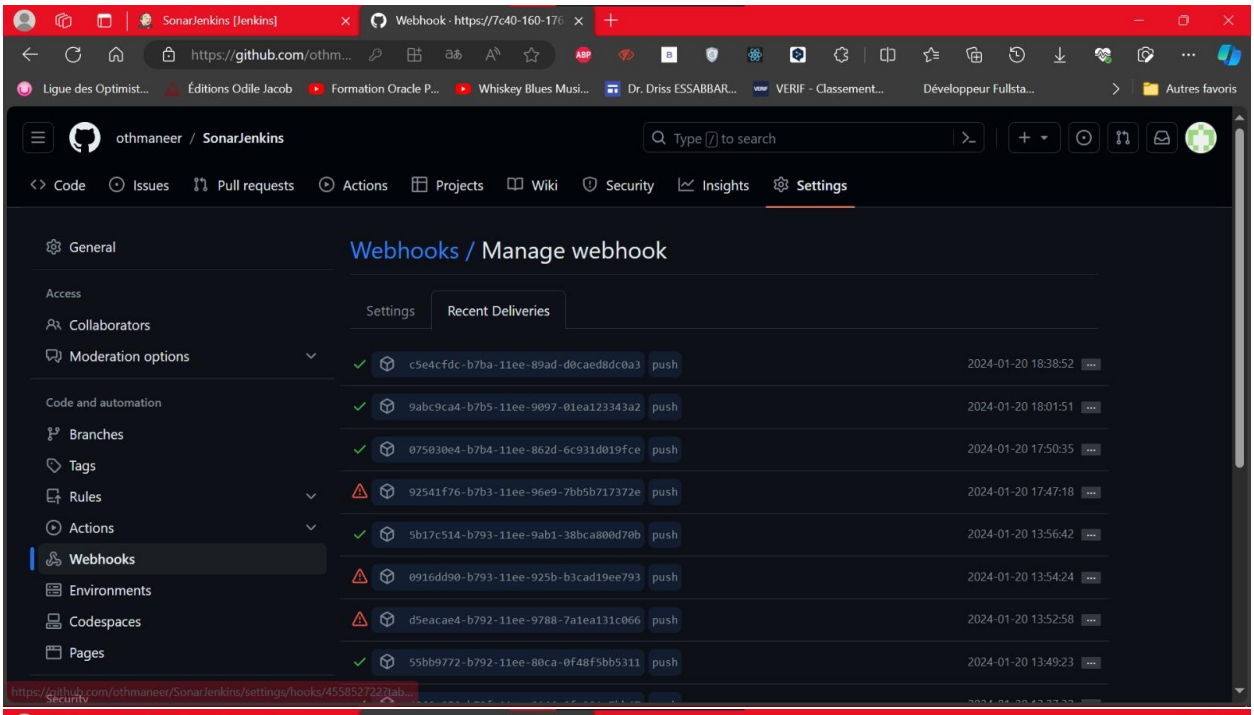
- **Utilisation de Ngrok :**

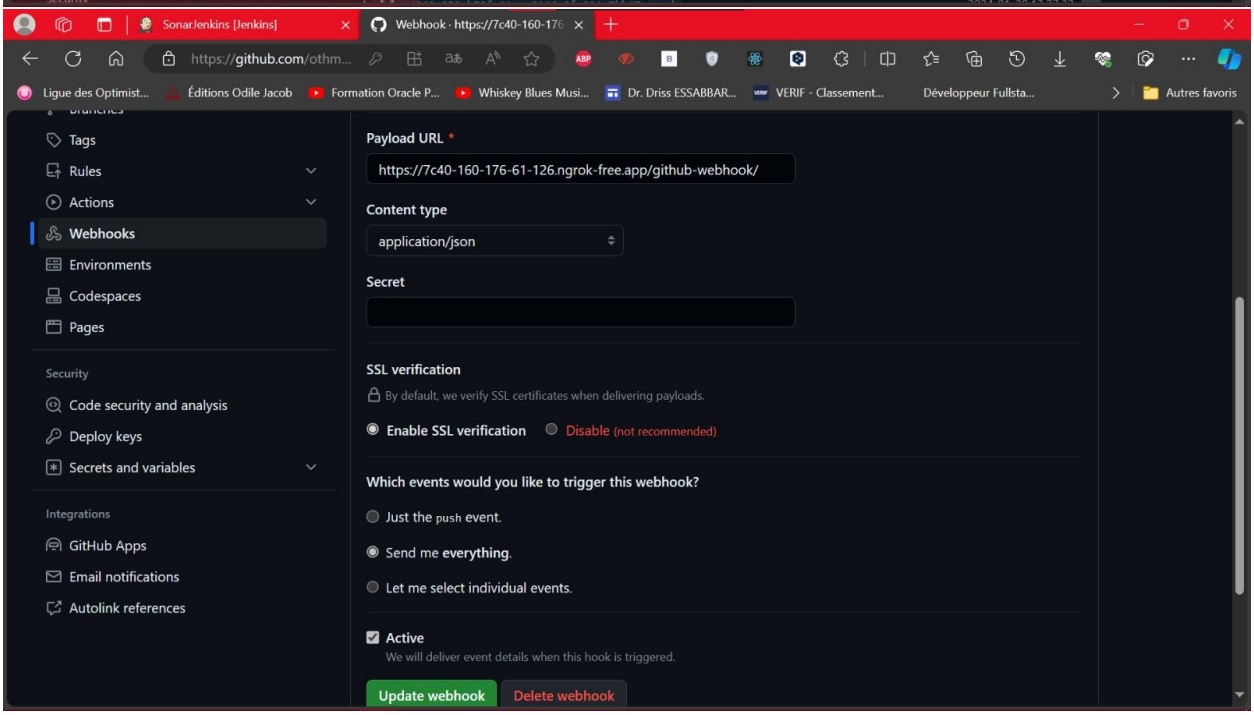
L'utilisation de Ngrok permet un déploiement automatique et efficace des microservices, assurant ainsi une disponibilité constante de la plateforme.

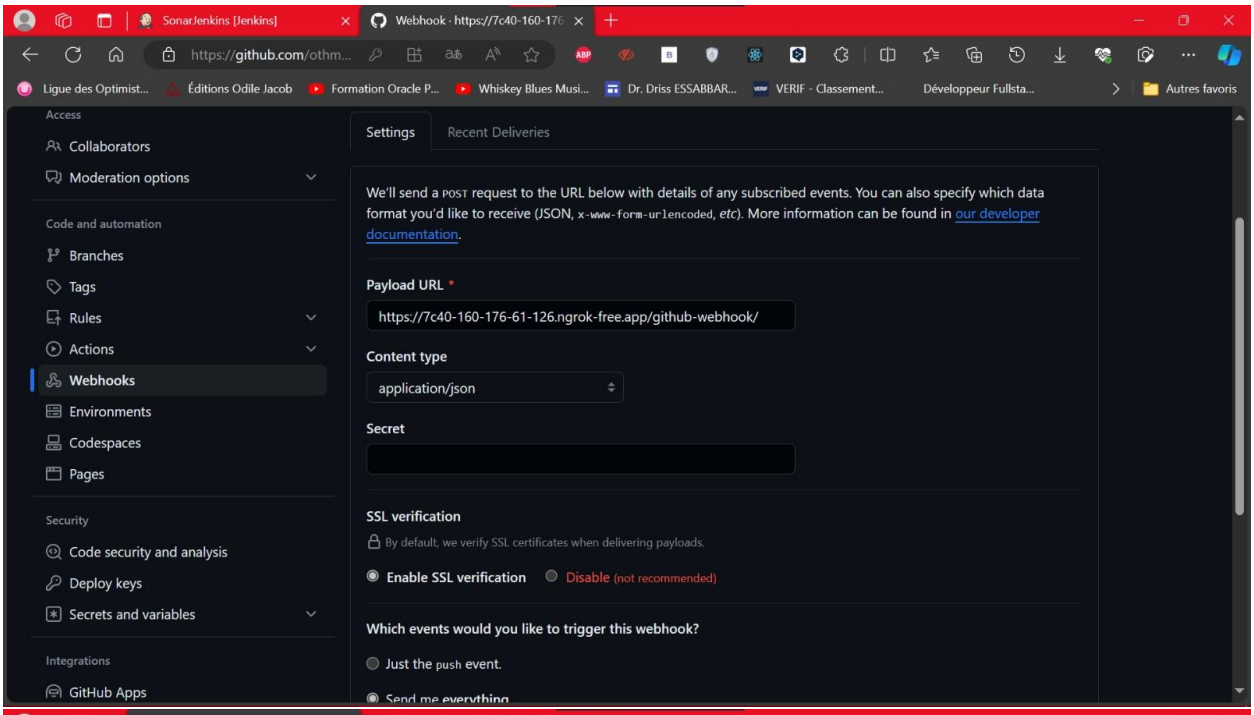
L'utilisation de Ngrok est particulièrement bénéfique lorsqu'elle est associée à des webhooks. Les webhooks sont des mécanismes permettant à un système d'informer automatiquement un autre système lorsqu'un événement spécifique se produit.

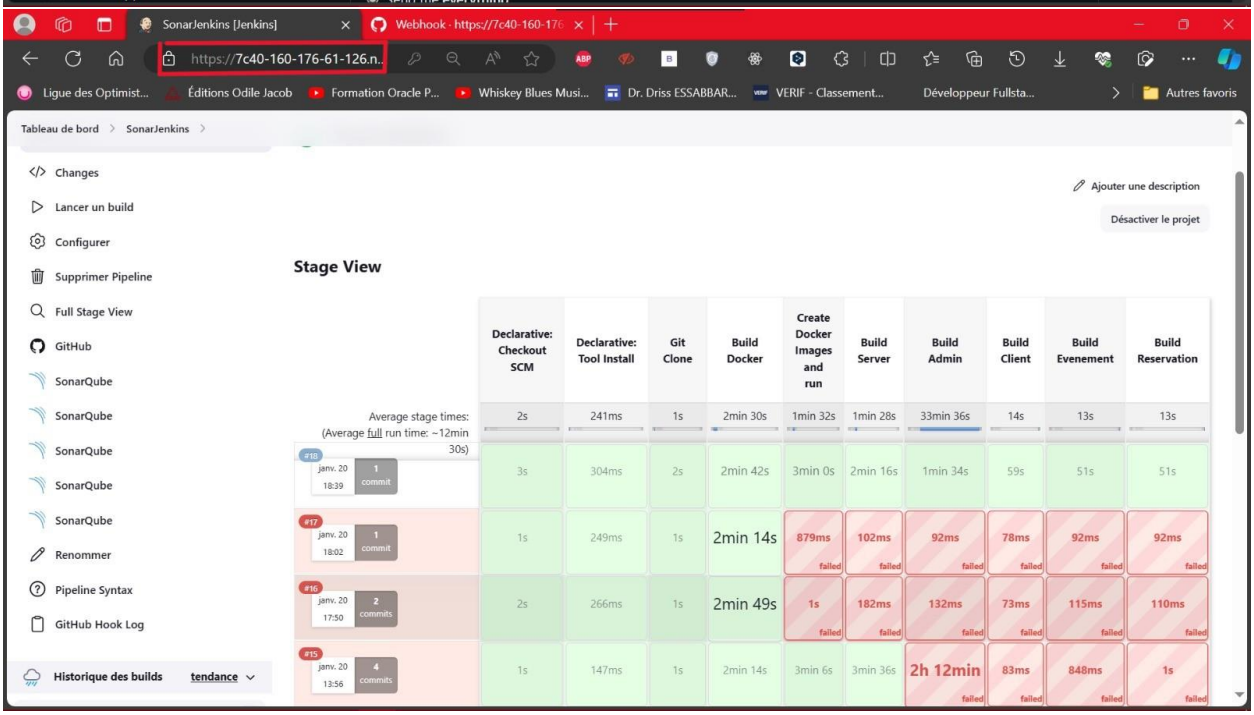





SonarJenkins [Jenkins]
Webhook · https://7c40-160-176
https://github.com/othm...
Ligue des Optimist...
Éditions Odile Jacob
Formation Oracle P...
Whiskey Blues Musi...
Dr. Driss ESSABBAR...
VERIF - Classement...
Développeur Fullsta...
Autres favoris
othmaneer / SonarJenkins
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
General
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Webhooks / Manage webhook
Settings
Recent Deliveries
c5e4cfdc-b7ba-11ee-89ad-d0caed8dc0a3 push 2024-01-20 18:38:52
9abc9ca4-b7b5-11ee-9097-01ea123343a2 push 2024-01-20 18:01:51
075030e4-b7b4-11ee-862d-6c931d019fce push 2024-01-20 17:50:35
92541f76-b7b3-11ee-96e9-7bb5b717372e push 2024-01-20 17:47:18
5b17c514-b793-11ee-9ab1-38bca800d70b push 2024-01-20 13:56:42
0916dd90-b793-11ee-925b-b3cad19ee793 push 2024-01-20 13:54:24
d5eacae4-b792-11ee-9788-7a1ea131c066 push 2024-01-20 13:52:58
55bb9772-b792-11ee-80ca-0f48f5bb5311 push 2024-01-20 13:49:23


SonarJenkins [Jenkins]
Webhook · https://7c40-160-176
https://github.com/othm...
Ligue des Optimist...
Éditions Odile Jacob
Formation Oracle P...
Whiskey Blues Musi...
Dr. Driss ESSABBAR...
VERIF - Classement...
Développeur Fullsta...
Autres favoris
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Security
Code security and analysis
Deploy keys
Secrets and variables
Integrations
GitHub Apps
Email notifications
Autolink references
Payload URL *
https://7c40-160-176-61-126.ngrok-free.app/github-webhook/
Content type
application/json
Secret
SSL verification
By default, we verify SSL certificates when delivering payloads.
Enable SSL verification
Disable (not recommended)
Which events would you like to trigger this webhook?
Just the push event.
Send me everything.
Let me select individual events.
Active
We will deliver event details when this hook is triggered.
Update webhook
Delete webhook

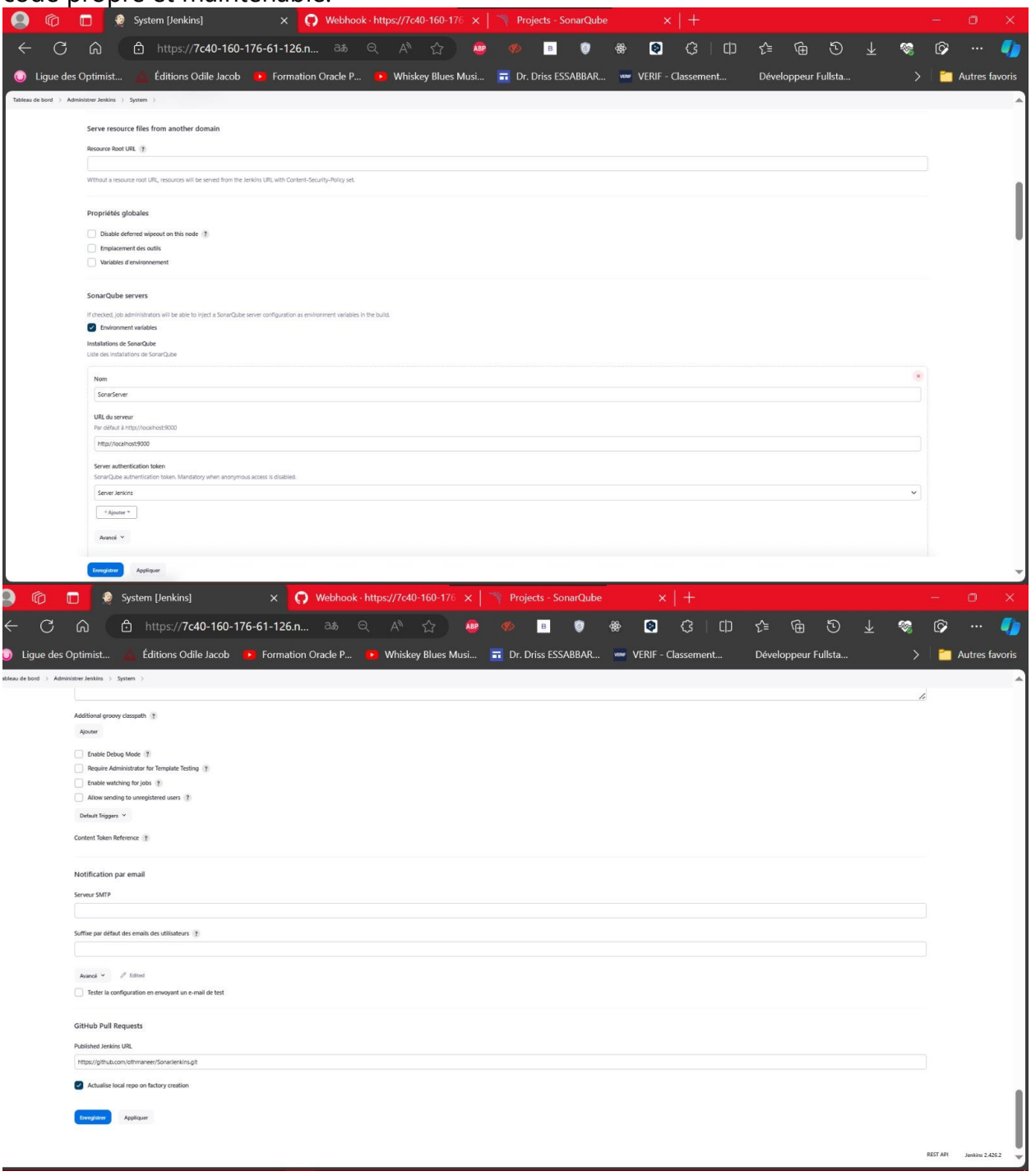

SonarJenkins [Jenkins]
Webhook · https://7c40-160-176
https://github.com/othm...
Ligue des Optimist...
Éditions Odile Jacob
Formation Oracle P...
Whiskey Blues Musi...
Dr. Driss ESSABBAR...
VERIF - Classement...
Développeur Fullsta...
Autres favoris
Access
Collaborators
Moderation options
Code and automation
Branches
Tags
Rules
Actions
Webhooks
Environments
Codespaces
Pages
Security
Code security and analysis
Deploy keys
Secrets and variables
Integrations
GitHub Apps
Settings
Recent Deliveries
We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our developer documentation.
Payload URL *
https://7c40-160-176-61-126.ngrok-free.app/github-webhook/
Content type
application/json
Secret
SSL verification
By default, we verify SSL certificates when delivering payloads.
Enable SSL verification
Disable (not recommended)
Which events would you like to trigger this webhook?
Just the push event.
Send me everything.

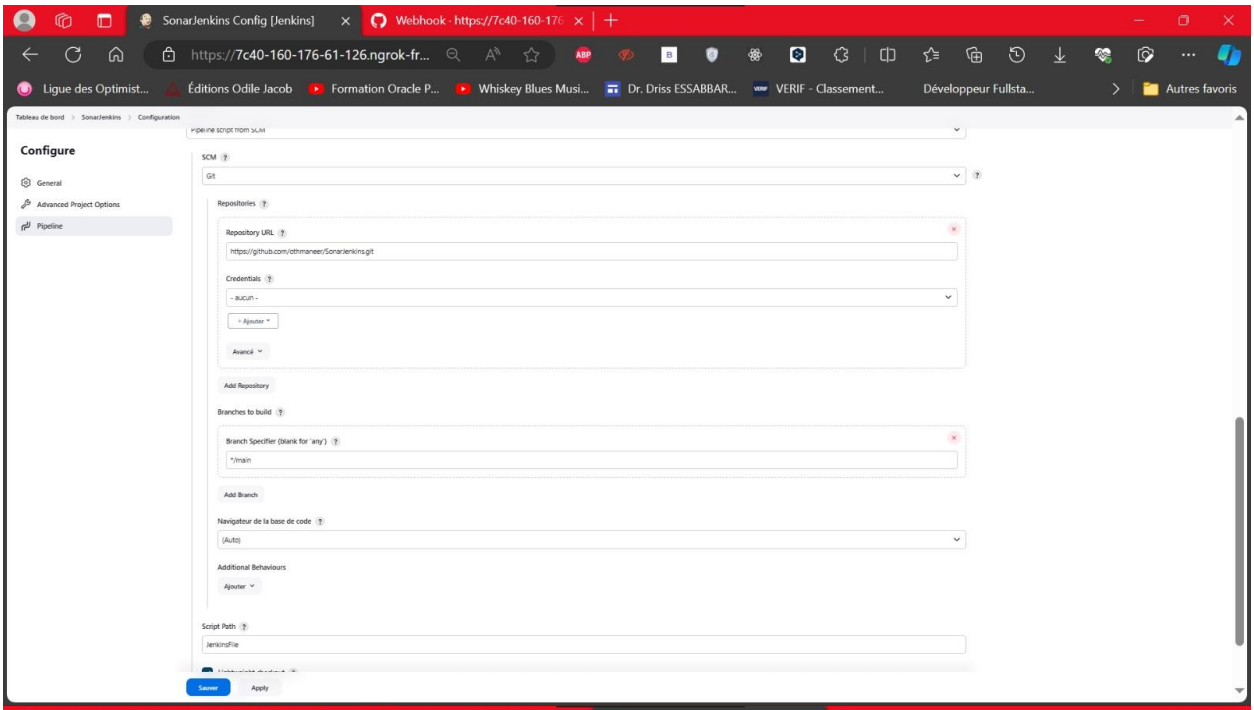

SonarJenkins [Jenkins]
Webhook · https://7c40-160-176
https://7c40-160-176-61-126.n...
Ligue des Optimist...
Éditions Odile Jacob
Formation Oracle P...
Whiskey Blues Musi...
Dr. Driss ESSABBAR...
VERIF - Classement...
Développeur Fullsta...
Autres favoris
Tableau de bord
SonarJenkins
Changes
Lancer un build
Configurer
Supprimer Pipeline
Full Stage View
GitHub
SonarQube
SonarQube
SonarQube
SonarQube
SonarQube
Renommer
Pipeline Syntax
GitHub Hook Log
Historique des builds
tendance
Ajouter une description
Désactiver le projet
Stage View
Declarative: Checkout SCM
Declarative: Tool Install
Git Clone
Build Docker
Create Docker Images and run
Build Server
Build Admin
Build Client
Build Evenement
Build Reservation
Average stage times:
(Average full run time: ~12min 30s)
2s
241ms
1s
2min 30s
1min 32s
1min 28s
33min 36s
14s
13s
13s
#18
janv. 20 18:39
1 commit
3s
304ms
2s
2min 42s
3min 0s
2min 16s
1min 34s
59s
51s
51s
#17
janv. 20 18:02
1 commit
1s
249ms
1s
2min 14s
879ms
102ms
92ms
78ms
92ms
92ms
failed
#16
janv. 20 17:50
2 commits
2s
266ms
1s
2min 49s
1s
182ms
132ms
73ms
115ms
110ms
failed
#15
janv. 20 13:56
4 commits
1s
147ms
1s
2min 14s
3min 6s
3min 36s
2h 12min
83ms
848ms
1s
failed

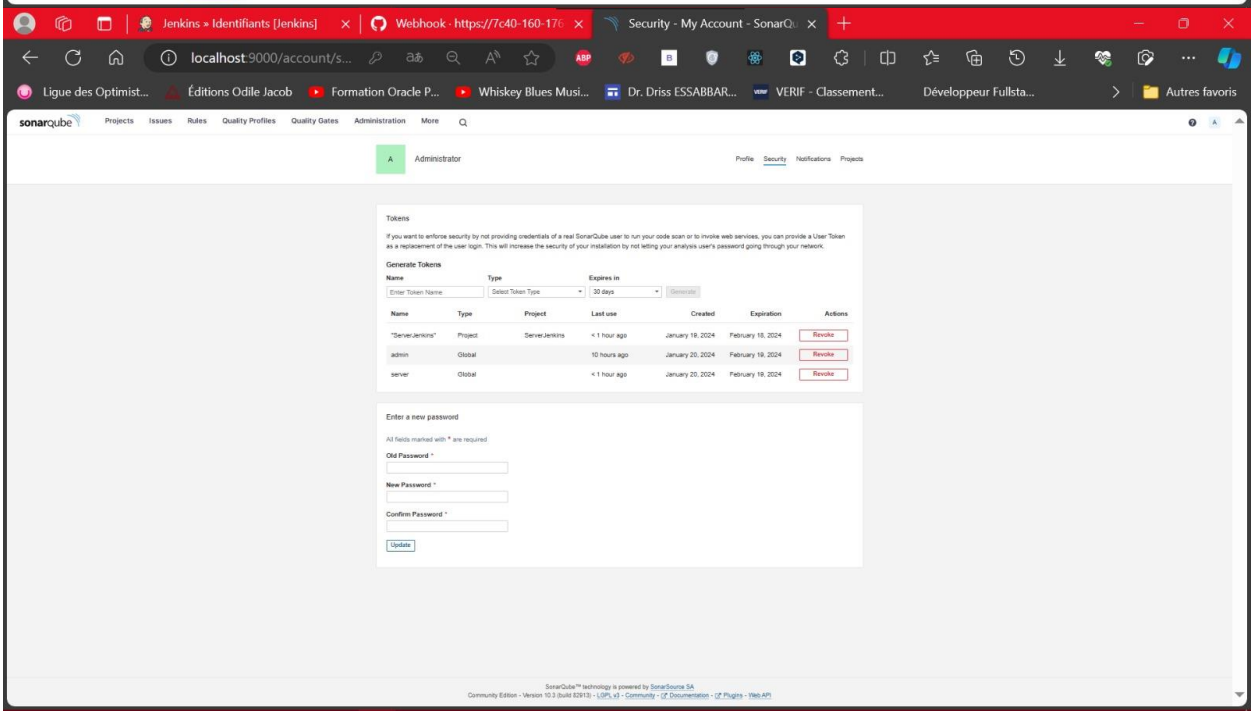
7. Intégration de SonarQube

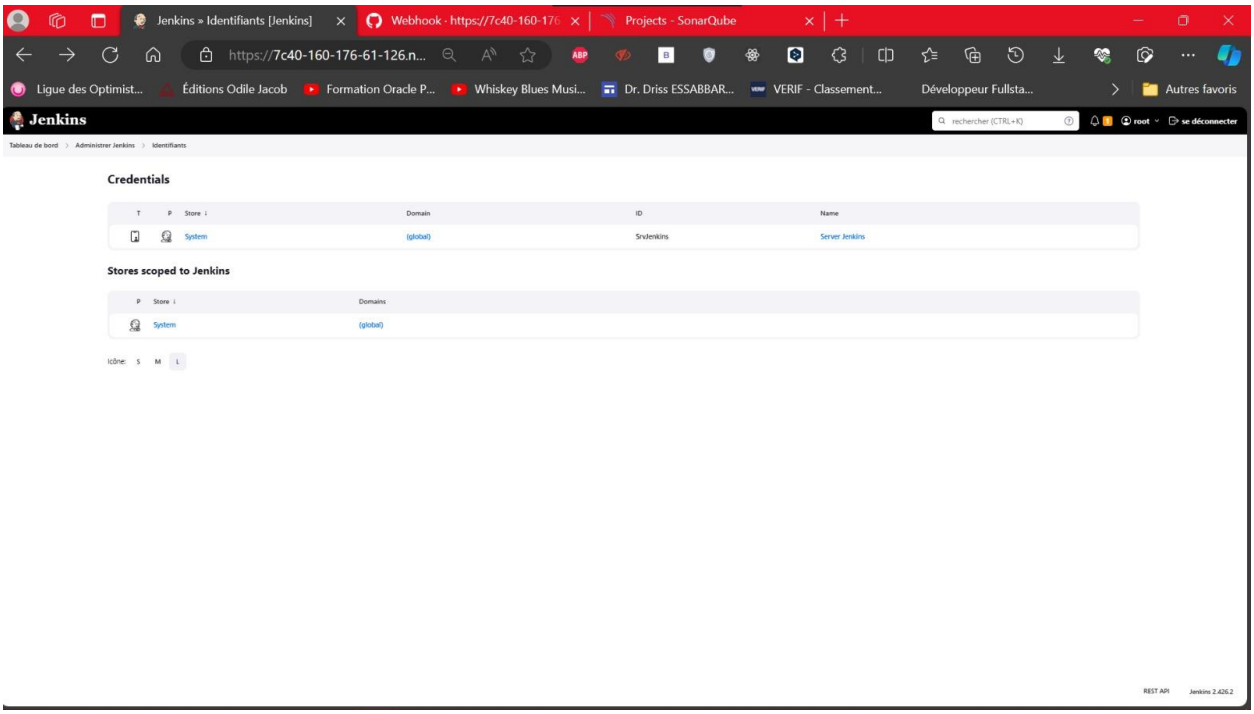
- **Configuration et bénéfices pour la qualité du code :**

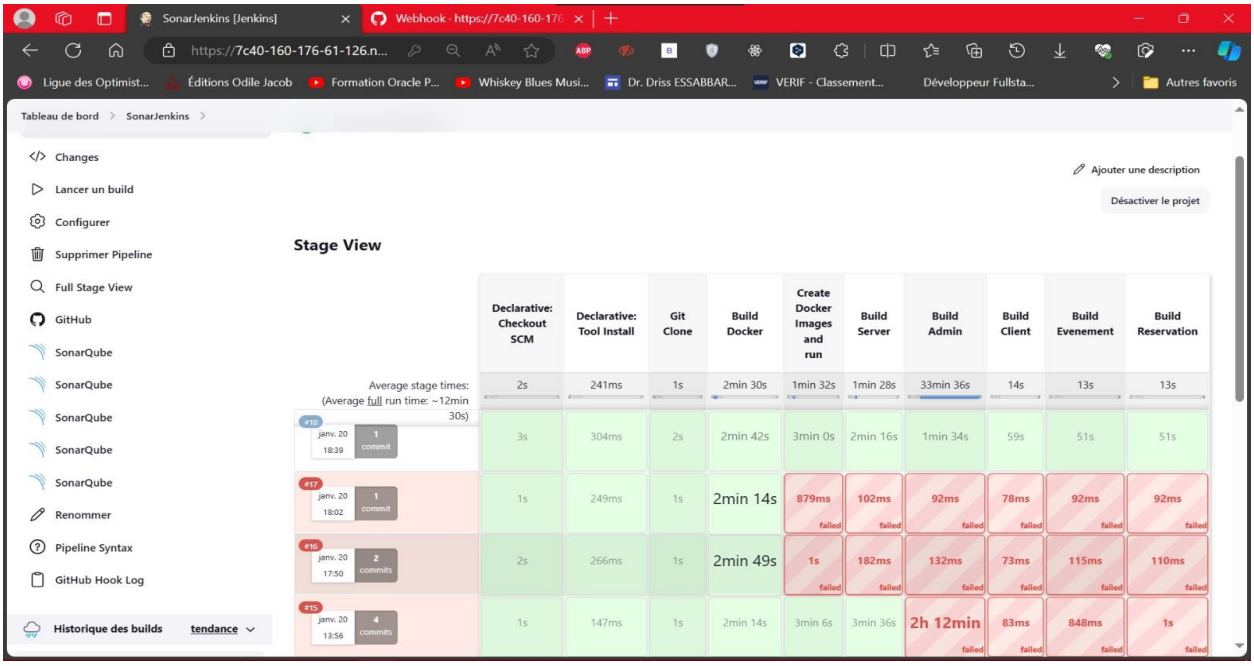
SonarQube est intégré pour assurer la qualité du code. La configuration de SonarQube permet de détecter et de résoudre les problèmes de code, garantissant ainsi une base de code propre et maintenable.

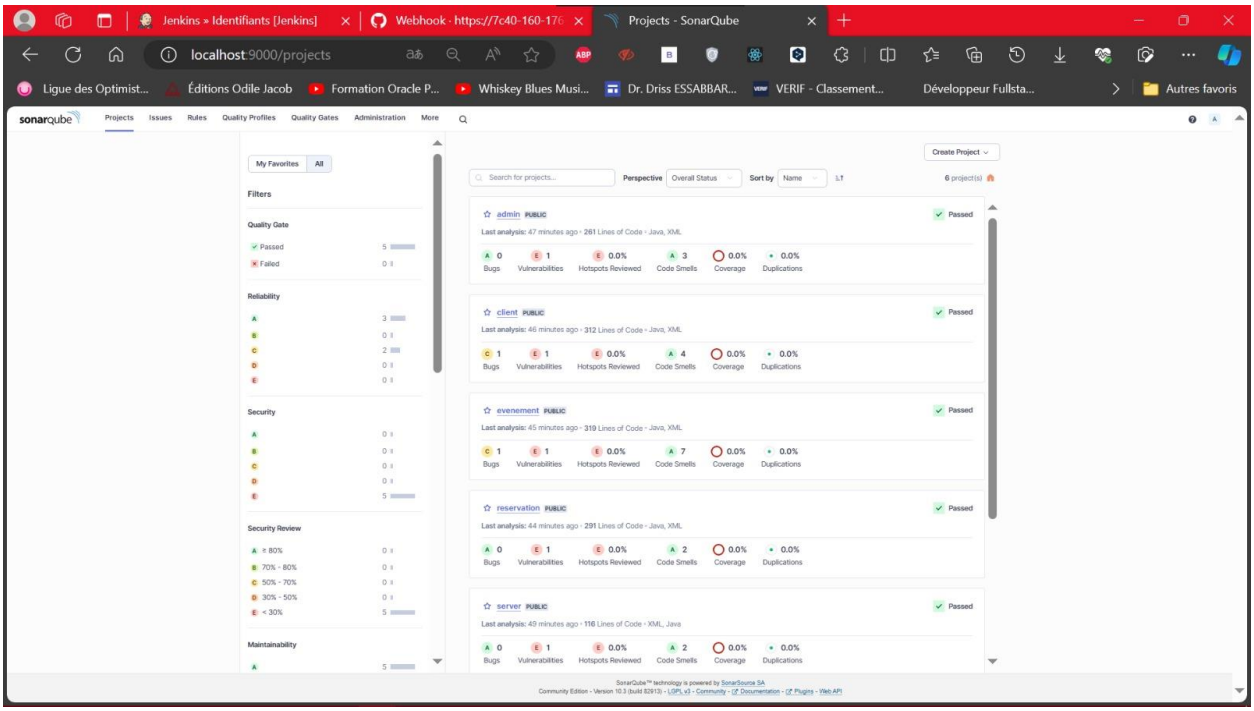



SonarJenkins Config [Jenkins]
Webhook · https://7c40-160-176
https://7c40-160-176-61-126.ngrok-fr...
Configure
General
Advanced Project Options
Pipeline
SCM
Git
Repositories
Repository URL
https://github.com/othmaneer/SonarJenkins.git
Credentials
- aucun -
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/main
Add Branch
Navigateur de la base de code
(Auto)
Additional Behaviours
Script Path
JenkinsFile
Sauver
Apply


Jenkins » Identifiants [Jenkins]
Webhook · https://7c40-160-176
Security - My Account - SonarQu
localhost:9000/account/s...
sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
Administrator
Profile
Security
Notifications
Projects
Tokens
Generate Tokens
Name
Type
Expires in
Enter Token Name
Select Token Type
30 days
Generate
Name
Type
Project
Last use
Created
Expiration
Actions
"ServerJenkins"
Project
ServerJenkins
< 1 hour ago
January 19, 2024
February 18, 2024
Revoke
admin
Global
10 hours ago
January 20, 2024
February 19, 2024
Revoke
server
Global
< 1 hour ago
January 20, 2024
February 19, 2024
Revoke
Enter a new password
All fields marked with * are required
Old Password *
New Password *
Confirm Password *
Update


Jenkins » Identifiants [Jenkins]
Webhook · https://7c40-160-176
Projects - SonarQube
https://7c40-160-176-61-126.n...
Ligue des Optimist...
Éditions Odile Jacob
Formation Oracle P...
Whiskey Blues Musi...
Dr. Driss ESSABBAR...
VERIF - Classement...
Développeur Fullsta...
Autres favoris
Jenkins
root
se déconnecter
Tableau de bord
Administrer Jenkins
Identifiants
Credentials
Store
Domain
ID
Name
System
(global)
Server Jenkins
Stores scoped to Jenkins
Domains
Icône:
REST API
Jenkins 2.426.2


SonarJenkins [Jenkins]
Webhook · https://7c40-160-176
https://7c40-160-176-61-126.n...
Ligue des Optimist...
Éditions Odile Jacob
Formation Oracle P...
Whiskey Blues Musi...
Dr. Driss ESSABBAR...
VERIF - Classement...
Développeur Fullsta...
Autres favoris
Tableau de bord
SonarJenkins
Changes
Lancer un build
Configurer
Supprimer Pipeline
Full Stage View
GitHub
SonarQube
SonarQube
SonarQube
SonarQube
SonarQube
Renommer
Pipeline Syntax
GitHub Hook Log
Historique des builds
tendance
Ajouter une description
Désactiver le projet
Stage View
Declarative: Checkout SCM
Declarative: Tool Install
Git Clone
Build Docker
Create Docker Images and run
Build Server
Build Admin
Build Client
Build Evenement
Build Reservation
Average stage times:
(Average full run time: ~12min 30s)
2s
241ms
1s
2min 30s
1min 32s
1min 28s
33min 36s
14s
13s
13s
#18
janv. 20 18:39
1 commit
3s
304ms
2s
2min 42s
3min 0s
2min 16s
1min 34s
59s
51s
51s
#17
janv. 20 18:02
1 commit
1s
249ms
1s
2min 14s
879ms
failed
102ms
failed
92ms
failed
78ms
failed
92ms
failed
92ms
failed
#16
janv. 20 17:50
2 commits
2s
266ms
1s
2min 49s
1s
failed
182ms
failed
132ms
failed
73ms
failed
115ms
failed
110ms
failed
#15
janv. 20 13:56
4 commits
1s
147ms
1s
2min 14s
3min 6s
3min 36s
2h 12min
failed
83ms
failed
848ms
failed
1s
failed



Pipline :

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

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

                }
            }
        }

                stage('Build Docker') {
                    steps {
                        script {
                            // Navigate into the project directory before running Maven commands
                            dir('admin') {
                                bat 'mvn clean install package'
                            }
                            dir('client') {
                                bat 'mvn clean install package'
                            }
                            dir('evenement') {
                                bat 'mvn clean install package'
                            }

                            dir('reservation') {
                                bat 'mvn clean install package'
                            }
                            dir('gate') {
                                bat 'mvn clean install package'
```

```
                            }
                            dir('server') {
                                bat 'mvn clean install package'
                            }
                        }
                    }
                }



                     stage('Create Docker Images and run') {
                    steps {
                        script {

                                bat "docker-compose down"

                                bat "docker-compose up --build -d"



                        }
                    }
                }

    stage('Build Server') {
        steps {
        script{
            dir('server') {

                withSonarQubeEnv('SonarServer') {
                    bat "mvn clean verify sonar:sonar"
                }
```

```
                }


        }
        }
    }


    stage('Build Admin') {
        steps {
        script{
            dir('admin') {

                withSonarQubeEnv('SonarServer') {
                    bat "mvn clean verify sonar:sonar"
                }

                }


        }
        }
    }

    stage('Build Client') {
        steps {
        script{
            dir('client') {

                withSonarQubeEnv('SonarServer') {
                    bat "mvn clean verify sonar:sonar"
                }
```

```
                }


        }
        }
    }

    stage('Build Evenement') {
        steps {
        script{
            dir('evenement') {

                withSonarQubeEnv('SonarServer') {
                    bat "mvn clean verify sonar:sonar"
                }

                }


        }
        }
    }

    stage('Build Reservation') {
        steps {
        script{
            dir('reservation') {

                withSonarQubeEnv('SonarServer') {
                    bat "mvn clean verify sonar:sonar"
                }

                }
```

```
                }


        }
        }
    }

        }
    }
```

8. Conclusion

- **Résumé des accomplissements :**

Le projet a abouti à la création d'une plateforme de réservation de billets robuste, basée sur une architecture microservices. L'utilisation de Docker, Jenkins et SonarQube a permis d'assurer une mise en œuvre efficace, une intégration et déploiement continue et une qualité de code élevée.

- **Perspectives :**

Pour l'avenir, des améliorations continues, l'ajout de nouvelles fonctionnalités et l'exploration de technologies émergentes sont envisagés. L'évolutivité de l'architecture microservices permettra une adaptation aisée aux besoins changeants de la plateforme.