

Rapport ACE

**Naja Mohamed et
Mouhoubi Meryem**

20/01/2024

—

ACE

—

Pr. LACHGAR

Aperçu

Notre projet repose sur une application de blog développée en utilisant l'architecture microservices. Cette approche permet une conception modulaire et évolutive, favorisant la flexibilité et la maintenabilité de l'application.



Importance de l'architecture microservices

L'architecture microservices offre plusieurs avantages, tels que la scalabilité indépendante, la facilité de déploiement, et la réduction des dépendances entre les composants. Elle permet également une gestion plus efficace des équipes de développement et une meilleure résilience.

Scalabilité Indépendante

L'un des principaux avantages de l'architecture microservices réside dans la capacité à faire évoluer indépendamment chaque composant du système. Chaque microservice peut être déployé, mis à l'échelle et mis à jour de manière autonome. Cela signifie que les parties spécifiques de l'application qui nécessitent une plus grande capacité peuvent être agrandies sans affecter les autres parties. Cette modularité offre une flexibilité considérable pour répondre aux fluctuations de la demande, garantissant une utilisation efficace des ressources.

Facilité de Déploiement

L'architecture microservices favorise un déploiement continu et une livraison fréquente. En raison de leur indépendance, les microservices peuvent être déployés séparément, accélérant le cycle de développement. Les équipes peuvent publier des fonctionnalités, des correctifs ou des améliorations sans perturber l'intégrité de l'ensemble du système. Cela réduit les risques associés aux mises à

jour, permettant une mise en production plus rapide et plus fiable.

Réduction des Dépendances entre les Composants

Contrairement aux architectures monolithiques, où tous les composants sont fortement interconnectés, l'architecture microservices minimise les dépendances entre les différents services. Chaque microservice communique généralement via des API bien définies, permettant aux équipes de développement de travailler de manière plus autonome. Cette réduction des dépendances facilite l'évolution et la maintenance, car les changements dans un microservice n'affectent pas nécessairement les autres.

Gestion Efficace des Équipes de Développement

Les équipes de développement peuvent être organisées de manière à ce que chaque équipe soit responsable d'un ou plusieurs microservices spécifiques. Cela favorise la spécialisation et permet aux équipes de se concentrer sur des domaines spécifiques de l'application. Chaque équipe peut opérer de manière autonome, accélérant le développement et améliorant la productivité. La communication entre les équipes se fait généralement par des interfaces clairement définies, minimisant les conflits.

Meilleure Résilience

En cas d'échec d'un microservice, l'ensemble de l'application peut continuer de fonctionner, car les autres services restent opérationnels. Cette architecture distribuée contribue à une meilleure résilience du système. De plus, la détection des pannes et la gestion de la charge peuvent être optimisées pour chaque microservice individuellement, garantissant une disponibilité plus élevée.

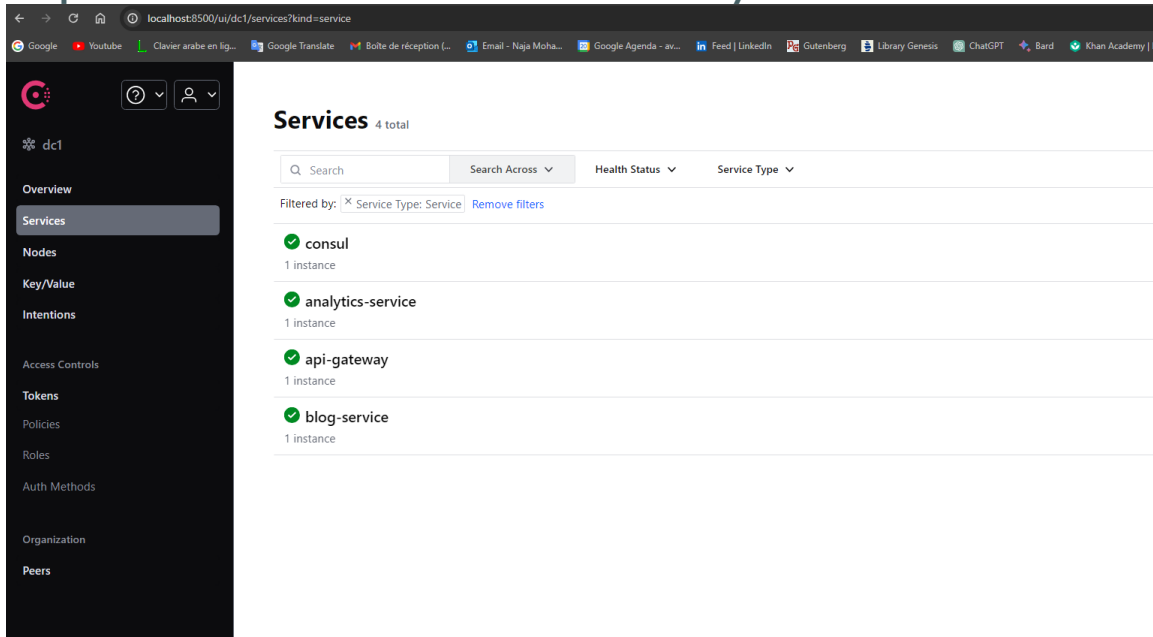
ARCHITECTURE

Nous avons décomposé notre application en différents services interconnectés, chacun se concentrant sur une fonctionnalité spécifique. Cela favorise la modularité et permet le développement parallèle des différentes parties de l'application.

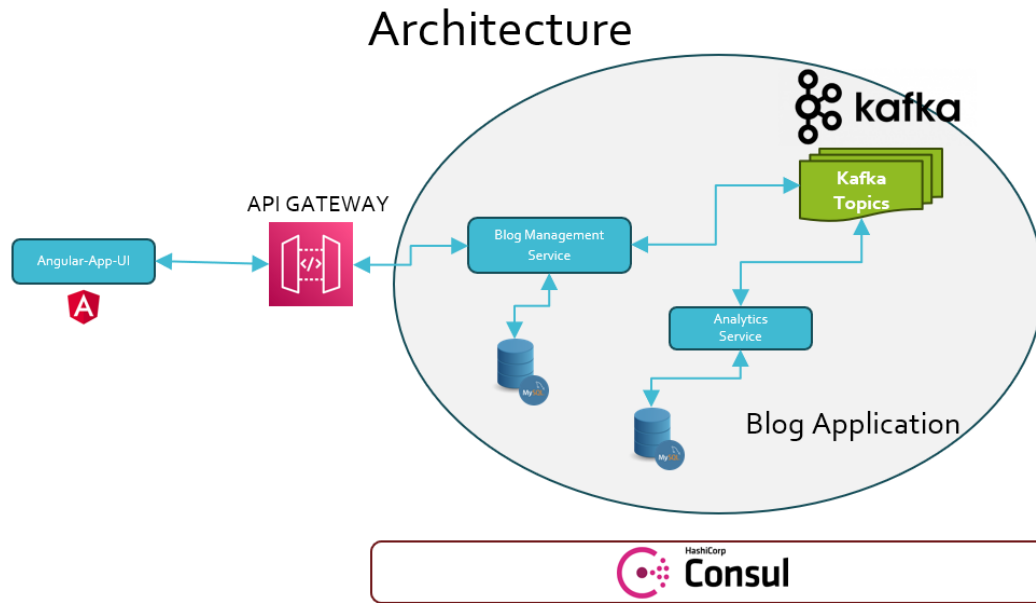
Description des services

- **mysql**: Service de base de données MySQL pour stocker les données de l'application.
- **app**: Service principal de l'application, gérant la logique métier.
- **app-ui**: Interface utilisateur de l'application, développée en Angular.
- **consul**: Service de découverte de services pour la gestion des microservices.
- **analyticsMicroservice**: Microservice dédié à l'analyse des données.
- **api_gateway**: Point d'entrée central pour l'accès aux microservices.
- **kafka**: Service de messagerie pour la communication asynchrone entre les microservices.
- **zookeeper**: Service de coordination pour Kafka.

Capture Consul Pour le Service Discovery



Architecture des microservices :



Conception des Microservices :

1. Conception du Blog Service Backend

Le service backend du blog est conçu comme une entité indépendante, responsable de la gestion de la logique métier liée aux publications, commentaires, et autres fonctionnalités du blog. Il fonctionne de manière autonome, permettant des évolutions spécifiques sans impact sur d'autres parties de l'application.

2. Interface Distincte et Clair pour le Blog Service Backend

Le blog service expose une interface claire, définissant les points d'entrée pour la création, la modification et la récupération des publications et des commentaires. Cette interface bien définie simplifie l'intégration avec d'autres services et garantit une communication efficace au sein de l'architecture microservices.

3. Fonctionnalités Spécifiques au Blog Service Backend


Le blog service se spécialise dans la gestion des publications et des interactions associées. Il encapsule la logique métier liée aux opérations de blog, permettant une organisation du code claire et une maintenance facile des fonctionnalités spécifiques au blog.


4. Communication avec d'Autres Services via API REST et Kafka

Le blog service communique avec d'autres services, tels que le service d'analyse, à l'aide d'API REST et de messages Kafka. Les API REST permettent des interactions directes, tandis que Kafka facilite la communication asynchrone pour des scénarios où des événements importants doivent être propagés de manière déconnectée.

Exemple : Publication d'un Commentaire

La publication d'un commentaire déclenche un événement kafka
le microservice de service de blog publie l'événement dans une file d'attente
appelée commentEventTopic où les microservices Analytics sont abonnés, lors de
l'ajout d'un commentaire à un article comme l'exemple ci-dessous, un nouvel
événement kafka est immédiatement ajouté au sujet.

 Home Tags Categories Submit

Logout  brian123

Unveiling the Wonders of Java Programming

brian123 / Monday, Jan 22, 2024

Explore the fascinating world of Java programming! Learn about the latest features, best practices, and advanced techniques to level up your coding skills. 🚀 Dive into the magic of object-oriented programming and conquer the challenges of building robust and scalable applications. Discover the power of Java libraries and frameworks, from Spring to Hibernate, that can supercharge your development journey. 🌐 Join a vibrant community of Java enthusiasts and stay updated on the latest industry trends and innovations. Whether you're a beginner or an experienced developer, this space is your gateway to mastering the art of Java!

Categories:

Tags:

this is a comment

Save

Cancel

criado por - 22-01-2024

ezzeg

reply

Delete

criado por - 22-01-2024

ezzeg

Capture d'écran de Kafka

```
Close tab (Ctrl+F4)
PowerShell
Id not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-01-22 00:20:03,833] WARN [AdminClient clientId=adminclient-1] Connection to node -1 (localhost/127.0.0.1:9094) could not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)
^Cbash-4.4# kafka-topics.sh --bootstrap-server=kafka:9094 --list
CommentTopic
__consumer_offsets
commentEventTopic
bash-4.4# kafka-console-consumer.bat --topic commentEventTopic --from-beginning --bootstrap-server localhost:9094
bash: kafka-console-consumer.bat: command not found
bash-4.4# kafka-console-consumer.sh --topic commentEventTopic --from-beginning --bootstrap-server localhost:9094
[2024-01-22 00:21:11,030] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9459] Connection to node -1 (localhost/127.0.0.1:9094) could not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-01-22 00:21:11,084] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9459] Connection to node -1 (localhost/127.0.0.1:9094) could not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-01-22 00:21:11,185] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9459] Connection to node -1 (localhost/127.0.0.1:9094) could not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-01-22 00:21:11,388] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9459] Connection to node -1 (localhost/127.0.0.1:9094) could not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-01-22 00:21:11,895] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9459] Connection to node -1 (localhost/127.0.0.1:9094) could not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)
[2024-01-22 00:21:12,603] WARN [Consumer clientId=consumer-1, groupId=console-consumer-9459] Connection to node -1 (localhost/127.0.0.1:9094) could not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)
^CProcessed a total of 0 messages
bash-4.4# kafka-console-consumer.sh --topic commentEventTopic --from-beginning --bootstrap-server kafka:9094
{"createdAt": [2024, 1, 22, 0, 14, 9, 160060000], "createdBy": "brian123", "updatedAt": [2024, 1, 22, 0, 14, 9, 160060000], "updatedBy": "brian123", "id": 2, "content": "ezzeg", "parentComment": null, "comments": [], "user": null}
{"createdAt": [2024, 1, 22, 0, 19, 8, 627060000], "createdBy": "brian123", "updatedAt": [2024, 1, 22, 0, 19, 8, 627060000], "updatedBy": "brian123", "id": 4, "content": "fazf\n", "parentComment": null, "comments": [], "user": null}
{"createdAt": [2024, 1, 22, 0, 24, 21, 381590000], "createdBy": "brian123", "updatedAt": [2024, 1, 22, 0, 24, 21, 381590000], "updatedBy": "brian123", "id": 6, "content": "this is a comment ", "parentComment": null, "comments": [], "user": null}
```

5. Gestion des Erreurs et Tolérance aux Pannes dans le Blog Service Backend

Le service backend du blog intègre des mécanismes de gestion des erreurs pour garantir la robustesse face à d'éventuels problèmes. Il est conçu pour tolérer les pannes partielles, permettant au reste de l'application de continuer à fonctionner même en cas de défaillance spécifique au blog service.

6. Tests Spécifiques pour le Blog Service Backend

Le blog service est accompagné d'une suite de tests dédiée, comprenant des tests unitaires et d'intégration. Ces tests valident la fonctionnalité spécifique du service, garantissant que toutes les opérations liées au blog sont correctement implémentées et fonctionnent de manière fiable.

7. Conception Spécifique pour l'Analytics Service

Le service d'analyse est conçu pour gérer spécifiquement les données analytiques de l'application. Il fonctionne de manière indépendante, traitant et analysant les données générées par le blog service et d'autres parties de l'application.

8. Interface Distincte pour l'Analytics Service

L'interface du service d'analyse est clairement définie pour permettre l'accès aux données d'analyse. Elle expose des points d'entrée spécifiques pour récupérer des informations analytiques, favorisant ainsi une utilisation cohérente par d'autres services.

9. Communication Asynchrone avec le Blog Service Backend via Kafka

Le service d'analyse communique de manière asynchrone avec le blog service backend via Kafka. Cela lui permet de traiter les données analytiques de manière déconnectée, évitant tout impact sur les performances du blog service lors de l'analyse des données.

10. Tests Spécifiques pour l'Analytics Service

De même, le service d'analyse est accompagné de tests spécifiques qui valident sa capacité à traiter, analyser et générer des informations analytiques de manière précise.

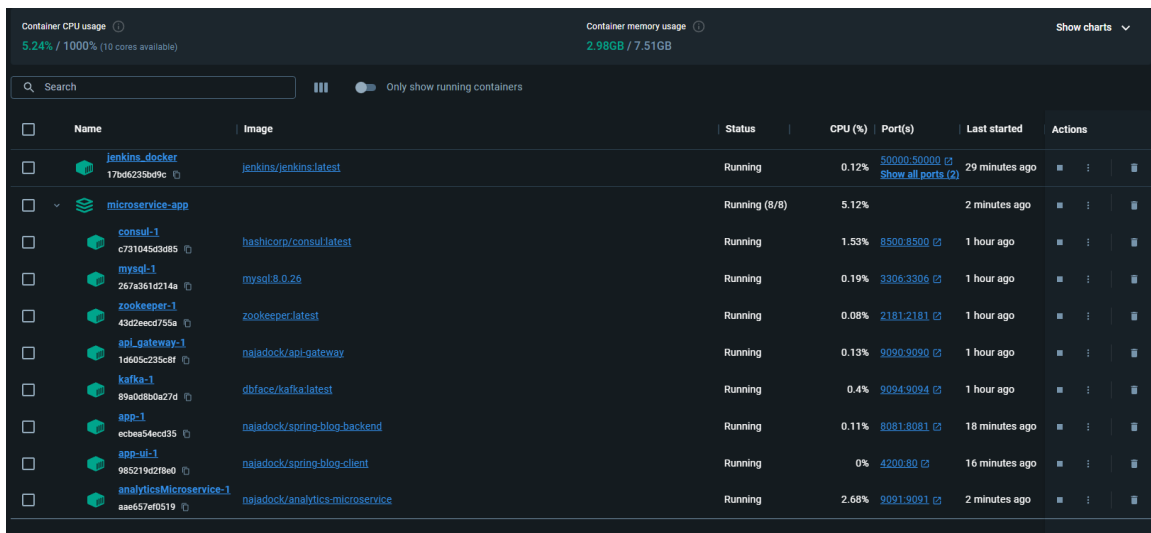
En adoptant cette approche spécifique pour le service backend du blog et le service d'analyse, nous avons créé des microservices distincts, autonomes et spécialisés, contribuant chacun de manière significative à l'ensemble de l'application. Cette approche facilite la maintenance, l'évolutivité et l'intégration au sein de l'architecture microservices.

Conteneurisation avec Docker

Implémentation et avantages

Nous utilisons Docker pour la conteneurisation de nos microservices. Cela simplifie le déploiement en garantissant que chaque service et ses dépendances sont encapsulés dans un conteneur, assurant une portabilité élevée et une isolation des environnements.

Capture D'écran de Docker Desktop :



The screenshot shows the Docker Desktop interface. At the top, it displays 'Container CPU usage' at 5.24% / 1000% (10 cores available) and 'Container memory usage' at 2.98GB / 7.51GB. Below this is a search bar and a toggle for 'Only show running containers'. The main table lists the following containers:

Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
jenkins_docker	jenkins/jenkins:latest	Running	0.12%	50000-50000	29 minutes ago	
microservice-app		Running (8/8)	5.12%		2 minutes ago	
consul-1	hashicorp/consul:latest	Running	1.53%	8500-8500	1 hour ago	
mysql-1	mysql:8.0.26	Running	0.19%	3306-3306	1 hour ago	
zookeeper-1	zookeeper:latest	Running	0.08%	2181-2181	1 hour ago	
api_gateway-1	najadock/api-gateway	Running	0.13%	9090-9090	1 hour ago	
kafka-1	dbface/kafka:latest	Running	0.4%	9094-9094	1 hour ago	
app-1	najadock/spring-blog-backend	Running	0.11%	8081-8081	18 minutes ago	
app-ui-1	najadock/spring-blog-client	Running	0%	4200-80	16 minutes ago	
analyticsMicroservice-1	najadock/analytics-microservice	Running	2.68%	9091-9091	2 minutes ago	

Capture d'écran terminal docker ps :



```
PowerShell 7.4.1
PS C:\Users\Mohamed> docker ps
CONTAINER ID   IMAGE                                COMMAND                  NAMES                  CREATED          STATUS          PORTS
8c0d2e8ff735   najadock/spring-blog-backend        "java -jar /app.jar"    microservice-app-app-1 13 minutes ago   Up 9 minutes    0.0.0.0:8081->8081/tcp
89a0d8b0a27d   dbface/kafka:latest                 "start-kafka.sh"        microservice-app-kafka-1 13 minutes ago   Up 13 minutes   0.0.0.0:9094->9094/tcp
1d605c235c8f   najadock/api-gateway                 "java -jar app.jar"     microservice-app-api_gateway-1 13 minutes ago   Up 13 minutes   0.0.0.0:9090->9090/tcp
43d2eecd755a   zookeeper:latest                    "/docker-entrypoint..." microservice-app-zookeeper-1 13 minutes ago   Up 13 minutes   2888/tcp, 3888/tcp
267a361d214a   mysql:8.0.26                        "docker-entrypoint.s..." microservice-app-mysql-1 13 minutes ago   Up 13 minutes   0.0.0.0:3306->3306/tcp
c731045d3d85   hashicorp/consul:latest              "docker-entrypoint.s..." microservice-app-consul-1 13 minutes ago   Up 13 minutes   8300-8302/tcp, 8600/udp, 0.0.0.0:8500->8500/tcp
PS C:\Users\Mohamed>
```

Code dockerCompose :

```
version: '3.6'

services:
  mysql:
    image: mysql:8.0.26
```

```

environment:
  MYSQL_DATABASE: springblog
  MYSQL_ROOT_USERNAME: root
  MYSQL_ROOT_PASSWORD: root
ports:
  - "3306:3306"
volumes:
  - /home/braians/desenvolvimento/Docker/Volumes/Mysql/lib/mysql
networks:
  - app-network
app:
  image: najadock/spring-blog-backend
  build:
    context: ./spring-blog-management
    dockerfile: Dockerfile
  environment:
    SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/springblog
    SPRING_DATASOURCE_USERNAME: root
    SPRING_DATASOURCE_PASSWORD: root
  ports:
    - 8081:8081
  networks:
    - app-network
  depends_on:
    - mysql
    - consul
app-ui:
  image: najadock/spring-blog-client
  build: ./spring-blog-client
  ports:
    - 4200:80
  depends_on:
    - app
consul:
  image: hashicorp/consul:latest
  ports:
    - "8500:8500"
  networks:
    - app-network

analyticsMicroservice:
  image: najadock/analytics-microservice
  build:
    context: ./analyticsMicroservice
    dockerfile: Dockerfile
  ports:
    - 9091:9091
  networks:
    - app-network
  depends_on:
    - consul
api_gateway:
  image: najadock/api-gateway
  build:
    context: ./apiGateway
    dockerfile: Dockerfile
  ports:
    - 9090:9090
  networks:

```

```

    - app-network
  depends_on:
    - consul

  kafka:
    image: dbface/kafka:latest
    environment:
      KAFKA_ADVERTISED_LISTENERS:
INSIDE://kafka:9093,OUTSIDE://localhost:9094
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
INSIDE:PLAINTEXT,OUTSIDE:PLAINTEXT
      KAFKA_LISTENERS: INSIDE://0.0.0.0:9093,OUTSIDE://0.0.0.0:9094
      KAFKA_INTER_BROKER_LISTENER_NAME: INSIDE
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    ports:
      - "9094:9094"
    networks:
      - app-network
  depends_on:
    - zookeeper

  zookeeper:
    image: zookeeper:latest
    ports:
      - "2181:2181"
    networks:
      - app-network

networks:
  app-network:
    driver: bridge

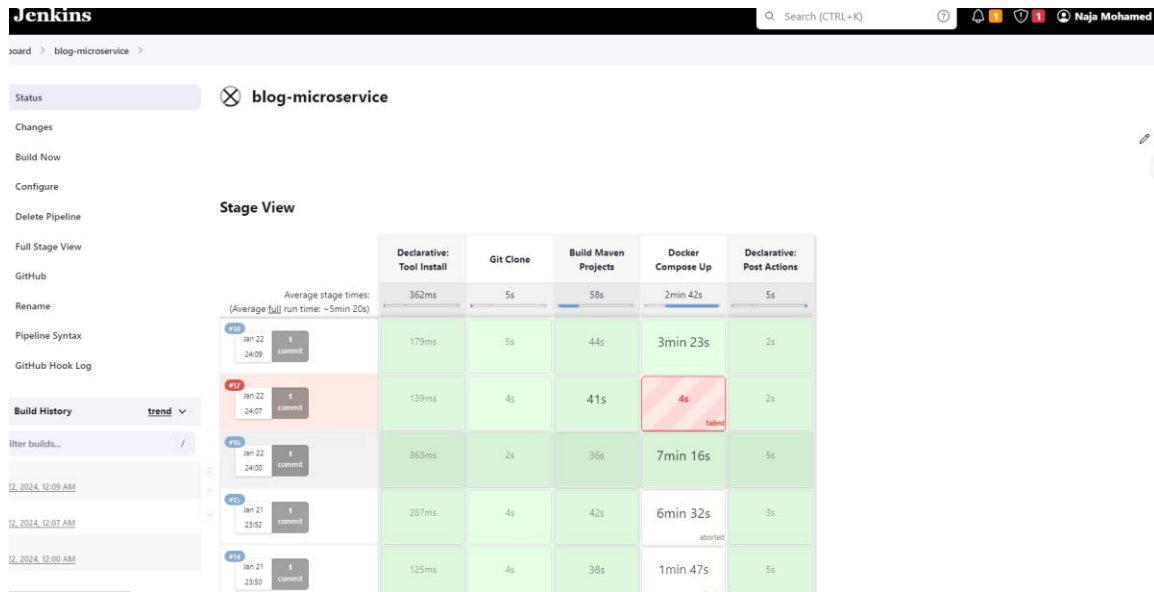
```

CI/CD avec Jenkins

Processus et configuration

Jenkins est utilisé pour l'intégration continue et le déploiement continu. À chaque modification du code source, Jenkins automatise les tests et déploie la nouvelle version de l'application de manière transparente.

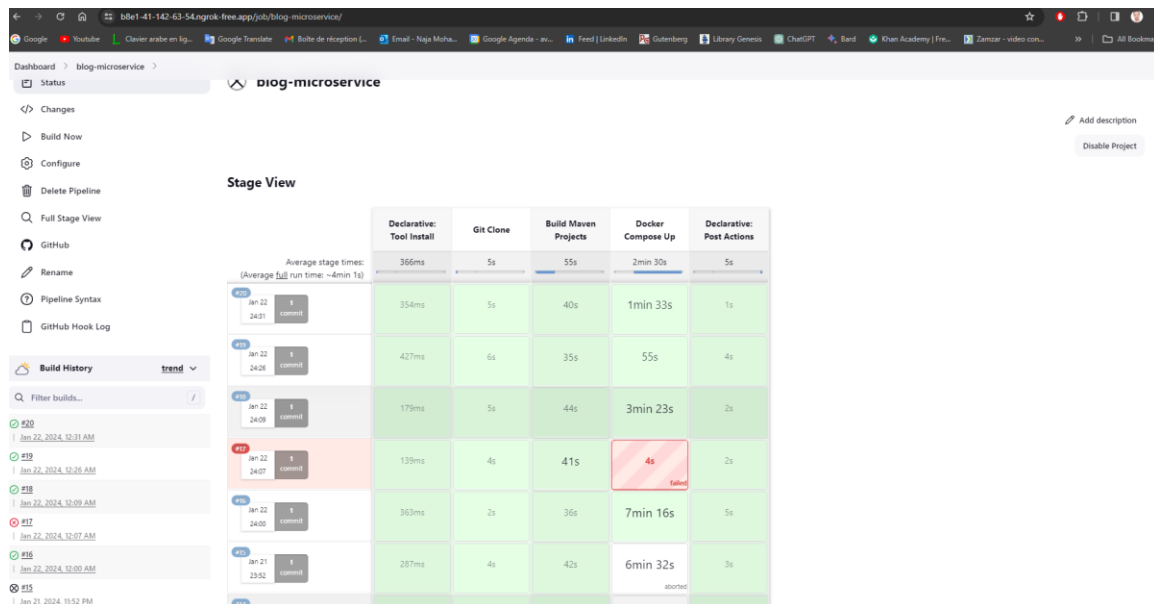
Capture d'écran Jenkins :



Video de CI CD avec jenkins :



20240121-2330-38.2
252970.mp4



Ngrok :

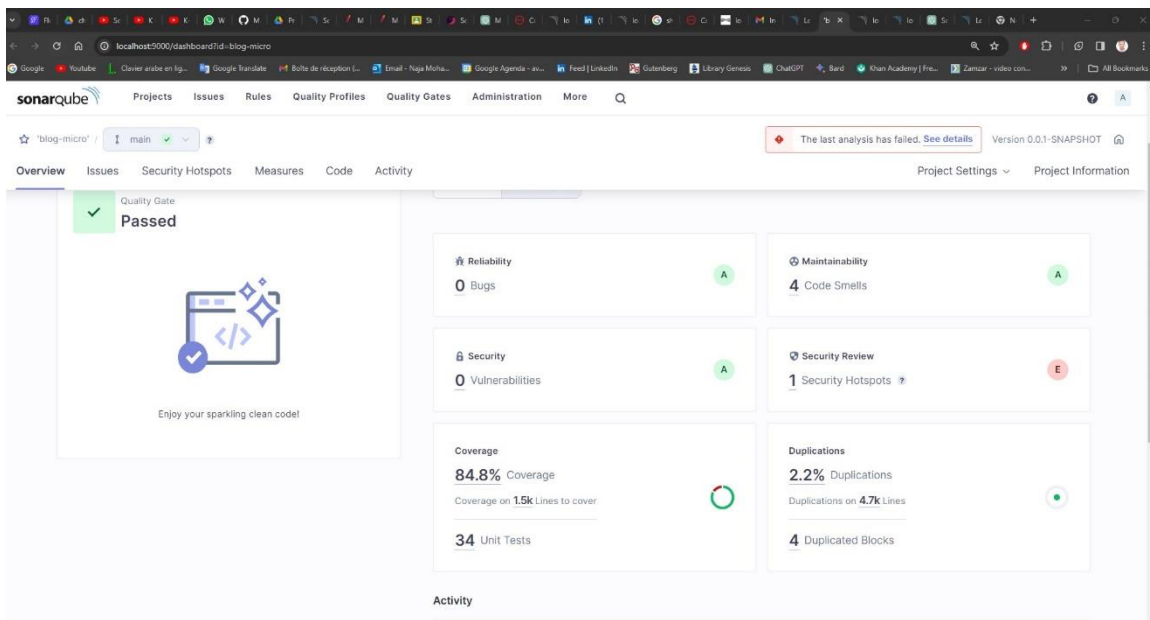
```
cmd in C:\ - ngrok http 8080 x PowerShell x + v
ngrok (Ctrl+C to quit)
Build better APIs with ngrok. Early access: ngrok.com/early-access

Session Status      online
Account             med.nj1999@gmail.com (Plan: Free)
Version             3.5.0
Region              Europe (eu)
Latency              75ms
Web Interface        http://127.0.0.1:4040
Forwarding            https://b8e1-41-142-63-54.ngrok-free.app -> http://localhost:8080

Connections          ttl      opn      rt1      rt5      p50      p90
                    1602     1        0.41     0.55     0.14     0.32

HTTP Requests
-----
GET /job/blog-microservice/wfapi/runs 200 OK
GET /job/blog-microservice/wfapi/runs 200 OK
GET /job/blog-microservice/buildHistory/ajax 200 OK
GET /job/blog-microservice/wfapi/runs 200 OK
GET /job/blog-microservice/wfapi/runs 200 OK
GET /job/blog-microservice/wfapi/runs 200 OK
GET /job/blog-microservice/wfapi/runs 200 OK
GET /job/blog-microservice/buildHistory/ajax 200 OK
GET /job/blog-microservice/wfapi/runs 200 OK
GET /job/blog-microservice/buildHistory/ajax 200 OK
GET /job/blog-microservice/wfapi/runs 200 OK
```

SonarQube :



Conclusion

Résumé des accomplissements

Notre adoption de l'architecture microservices, la conteneurisation avec Docker, l'automatisation du déploiement, l'intégration continue, et l'utilisation de SonarQube ont considérablement amélioré le développement et la qualité de notre application.

Perspectives futures

Nous envisageons d'explorer davantage les technologies émergentes et d'étendre nos services pour répondre aux besoins futurs de l'application. L'architecture microservices offre une base solide pour l'évolutivité et l'innovation continue.

Ce rapport résume notre voyage vers une architecture moderne et agile, soulignant notre engagement envers l'excellence technique et la satisfaction des utilisateurs.
