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

Projet de class

Conception et réalisation d'une application de Bloc d'opération

IIR5 – Group 4

Ingénierie Informatique et Réseaux

Réalisé par :

Nom de l'étudiant : **Anass Hajjouj – Hamza Maataoui**

Encadré par : **M. Mohamed Lachgar**

Remerciements

Nous tenons à exprimer nos sincères remerciements à notre professeur M.**LCHGER** pour son précieux soutien et ses conseils éclairés tout au long de la réalisation de notre Projet de class. Sa disponibilité, son expertise et son dévouement ont grandement contribué à la réussite de ce projet.

Ses retours constructifs et ses orientations ont été d'une importance capitale pour affiner nos idées, surmonter les défis et atteindre les objectifs fixés. Nous sommes reconnaissants pour l'inspiration qu'il a apportée à notre travail et pour avoir partagé son savoir d'une manière qui a profondément enrichi notre expérience académique collective.

Merci infiniment, Professeur M.**LCHGER**, pour avoir été un guide exceptionnel et pour avoir joué un rôle crucial dans la concrétisation de ce projet. Votre mentorat a été une source d'inspiration et a grandement contribué à notre développement professionnel en tant qu'équipe.

Table de matière

Table de matière III
Partie I : Contexte générale du projet 1
Introduction : 1
I. Micro-services 1
1.1 Introduction : 1
1.2 L'importance : 1
Partie II : Architecture Micro-services 3
2.2 Mécanismes de communication : 4
Partie III : Conception des Micro-services 4
Partie IV: Conteneurisation avec Docker 6
4.1 Implémentation et avantages : 6
Partie V: CI/CD avec Jenkins 7
5.1 Configuration et bénéfices pour la qualité du code : 7
Partie VII : Conclusion 8
6.1 Résumé des accomplissements : 8
6.2 Perspectives : 9

Partie I : Contexte générale du projet

Introduction :

Dans un contexte où les avancées technologiques transforment la façon dont les services sont proposés et consommés, notre projet prend place avec pour objectif de répondre à un besoin croissant dans le secteur de la réservation d'hôtels. Cette introduction pose les bases du contexte général de notre initiative, mettant en lumière les défis et les opportunités inhérents à ce domaine dynamique.

I. Micro-services

1.1Introduction :

Les micro-services sont une approche architecturale dans le développement logiciel où une application monolithique traditionnelle est décomposée en un ensemble de services autonomes, indépendants et spécialisés. Chaque micro-services représente une unité fonctionnelle distincte, déployée et gérée de manière indépendante. Ces services interagissent entre eux au travers d'interfaces bien définies, souvent des API (Interfaces de Programmation Applicatives), favorisant la scalabilité, la flexibilité et la maintenance facilitée des systèmes logiciels. L'architecture micro-services permet une meilleure résilience, une évolutivité aisée, ainsi qu'une agilité accrue dans le développement, le déploiement et la maintenance des applications.

1.2L’importance :

L'importance de l'architecture micro-services repose sur plusieurs avantages clés qui répondent aux défis modernes du développement logiciel et de la gestion des systèmes informatiques. Voici quelques points cruciaux qui soulignent l'importance de l'architecture micro-services:

- **Scalabilité et Flexibilité** : Les micro-services permettent une scalabilité granulaire, où chaque service peut être développé, déployé, et évolué indépendamment des autres. Cela offre une flexibilité accrue pour adapter les composants du système en fonction des besoins spécifiques.

- **Facilité de Maintenance :** En découpant une application en petits services, la maintenance devient plus aisée. Les mises à jour, les correctifs, et les modifications peuvent être effectués sur un service sans impacter l'ensemble de l'application. Cela facilite également la localisation et la résolution des problèmes.

- **Développement Agile** : L'architecture micro-services favorise les méthodologies de développement agile. Des équipes autonomes peuvent travailler sur des services spécifiques, accélérant le processus de développement, facilitant la collaboration, et permettant des itérations plus rapides.

- **Technologies Diverses :** Chaque micro-service peut être développé en utilisant des technologies spécifiques adaptées à son contexte. Cela permet d'adopter la meilleure technologie pour chaque fonctionnalité, sans être contraint par les choix technologiques du reste de l'application.

- **Résilience et Isolation** :En cas de panne d'un service, les autres services continuent à fonctionner normalement, assurant une meilleure résilience. De plus, l'isolation des services permet de minimiser les impacts négatifs en cas de défaillance d'un composant.

- **Déploiement Continu :** Les micro-services facilitent la mise en place de pipelines CI/CD (Continuous Integration/Continuous Deployment), permettant des déploiements fréquents et rapides. Cela réduit le temps entre le développement d'une fonctionnalité et sa mise en production.

- **Meilleure Gestion des Complexités** : La modularité des micro-services simplifie la gestion des complexités. Chaque service étant spécialisé, la compréhension, le développement, et la maintenance sont plus simples par rapport à un monolithe complexe.

- **Adaptabilité aux Technologies Cloud** : L'architecture microservices est bien adaptée aux environnements cloud, permettant une distribution efficace des services et une utilisation optimale des ressources cloud, favorisant ainsi la scalabilité horizontale.

Partie II : Architecture Micro-services

2.1 Architecture Micro-service :

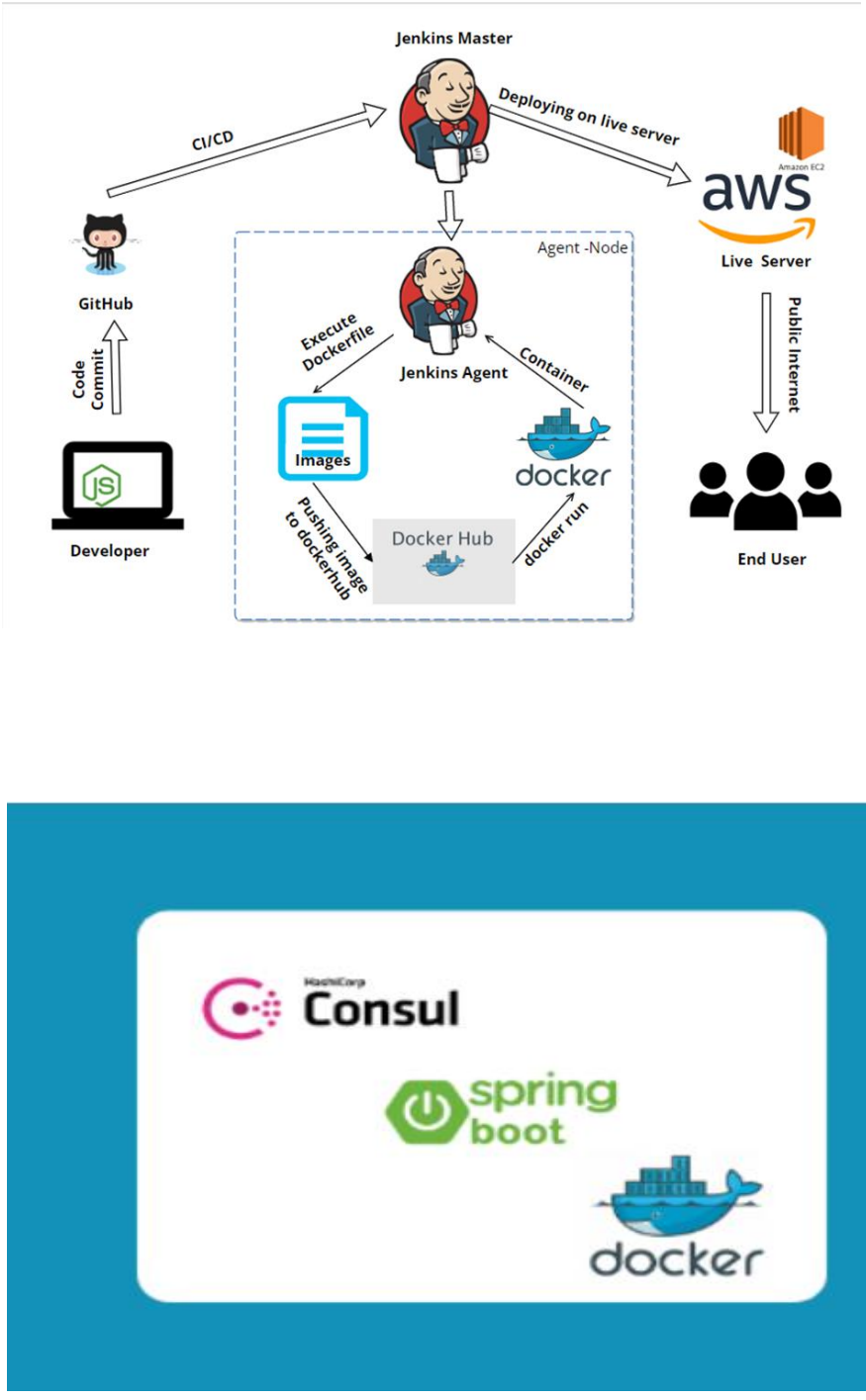

Figure 1 : Architecture Micro-services

2.2 Mécanismes de communication :

Dans le cadre de notre architecture microservices, les mécanismes de communication jouent un rôle crucial dans la création d'un écosystème cohérent et interconnecté. La communication entre les différents microservices est réalisée principalement via des mécanismes basés sur les API RESTful. Cette approche a été délibérément choisie pour favoriser l'indépendance entre les services et faciliter leur intégration harmonieuse.

Les API RESTful offrent une interface standardisée et flexible pour la communication entre les microservices. Elles sont basées sur les principes du protocole HTTP, permettant une interaction stateless et simplifiant la gestion des échanges de données. Ce choix permet aux services de communiquer de manière uniforme, qu'ils soient développés en utilisant Spring, Angular, ou d'autres technologies.

Consul, en tant qu'outil central de notre architecture, joue un rôle essentiel dans la facilitation de la communication entre microservices. Il agit comme un registre de services, permettant la découverte dynamique des services disponibles dans l'écosystème. Lorsqu'un microservice doit interagir avec un autre, Consul offre un mécanisme de découverte efficace, garantissant ainsi une connectivité fluide sans dépendance rigide sur des adresses IP statiques.

La combinaison de l'approche RESTful et de l'utilisation de Consul crée un environnement où chaque microservice peut évoluer indépendamment tout en restant connecté au reste du système. Cette flexibilité dans la communication entre les microservices contribue à la résilience et à la scalabilité de notre architecture, permettant une gestion efficace des interactions complexes au sein de l'écosystème.

Partie III : Conception des Micro-services

3.1 Approche de conception pour chaque service :

Dans le cadre de la conception des microservices au sein de notre projet, nous avons adopté une approche rigoureuse basée sur les principes SOLID. Ces principes fournissent une base solide pour le développement de logiciels robustes, faciles à maintenir et évolutifs. Ci-dessous, nous détaillons notre approche de conception pour chaque service:

Principes SOLID appliqués à la Conception des Microservices

1. Single Responsibility Principle (SRP):

Chaque microservice est conçu avec une responsabilité unique, correspondant à un domaine spécifique du système. Cette spécialisation garantit que chaque service a une mission claire, favorisant une compréhension précise de son rôle dans l'ensemble du système.

2. Open/Closed Principle (OCP):

Les microservices sont conçus pour être ouverts à l'extension mais fermés à la modification. Cela signifie que l'ajout de nouvelles fonctionnalités se fait par le biais d'extensions plutôt que de modifications directes du code existant, facilitant ainsi l'évolutivité sans compromettre la stabilité.

3. Liskov Substitution Principle (LSP):

Les microservices sont conçus de manière à respecter la substitution de Liskov, garantissant que chaque service peut être substitué par un autre sans affecter la cohérence globale du système. Cela favorise l'interopérabilité et la possibilité de faire évoluer les services individuellement.

4. Interface Segregation Principle (ISP):

Lorsque cela est approprié, les interfaces des microservices sont conçues de manière à être spécifiques à leurs besoins. Cela évite la surcharge d'interfaces avec des méthodes non pertinentes pour un service donné, favorisant ainsi une conception légère et ciblée.

5. Dependency Inversion Principle (DIP):

Les dépendances entre les microservices sont inversées pour minimiser les couplages directs. Les services dépendent d'abstractions plutôt que d'implémentations concrètes, permettant une flexibilité accrue lors des mises à jour et des évolutions.

Avantages de l'Approche SOLID dans la Conception des Microservices

Extensibilité Maximale: Les microservices peuvent être étendus facilement pour répondre aux nouveaux besoins sans altérer les fonctionnalités existantes.

Maintenabilité Optimale: La clarté de responsabilité unique et la modularité facilitent la maintenance du code, réduisant les risques d'effets indésirables lors de modifications.

Modularité Prononcée: Chaque microservice fonctionne comme une entité autonome, permettant le remplacement ou l'ajout de services sans perturber l'équilibre de l'ensemble du système.

Évolutivité Facilitée: L'approche SOLID offre une base solide pour faire évoluer chaque microservice indépendamment, répondant ainsi aux exigences évolutives sans compromettre la stabilité.

Partie IV: Conteneurisation avec Docker

4.1 Implémentation et avantages :

Implémentation de Docker dans notre Architecture Microservices :

La conteneurisation avec Docker a été soigneusement intégrée dans notre projet, offrant une solution efficace pour encapsuler chaque microservice. Chaque composant du système, du backend développé avec Spring aux services frontend en Angular, est maintenant encapsulé dans des conteneurs Docker autonomes.

L'implémentation a suivi un processus structuré, où chaque microservice a été dockerisé individuellement. Les fichiers Dockerfile ont été créés avec une attention particulière aux dépendances et aux configurations spécifiques à chaque service. L'utilisation de Docker Compose a été privilégiée pour orchestrer le déploiement simultané de plusieurs conteneurs, créant ainsi une infrastructure cohérente et facilement reproductible.

Avantages de la Conteneurisation avec Docker

Simplification du Déploiement: La conteneurisation avec Docker simplifie considérablement le déploiement des microservices. Chaque conteneur encapsule toutes les dépendances nécessaires, éliminant ainsi les problèmes potentiels liés aux variations d'environnements.

Portabilité Accrue: Les conteneurs Docker sont portables et indépendants de l'environnement, garantissant une exécution cohérente sur n'importe quelle machine. Cela facilite le déploiement sur différentes infrastructures, qu'il s'agisse de serveurs locaux, de cloud public, ou de clusters Kubernetes.

Environnement Reproductible: Docker fournit un moyen de décrire précisément l'environnement d'exécution d'une application à l'aide du fichier Dockerfile. Cela permet de reproduire facilement l'environnement de développement sur n'importe quelle machine, facilitant ainsi le travail collaboratif et la résolution des problèmes.

Gestion des Dépendances Simplifiée: Chaque conteneur contient son propre ensemble de dépendances, isolant ainsi les microservices les uns des autres. Cela simplifie la gestion des versions de dépendances et réduit les risques de conflits.

Évolutivité Facilitée: Docker permet le déploiement et la mise à l'échelle rapide des microservices, offrant ainsi une solution efficace pour répondre aux demandes croissantes de trafic ou de nouvelles fonctionnalités

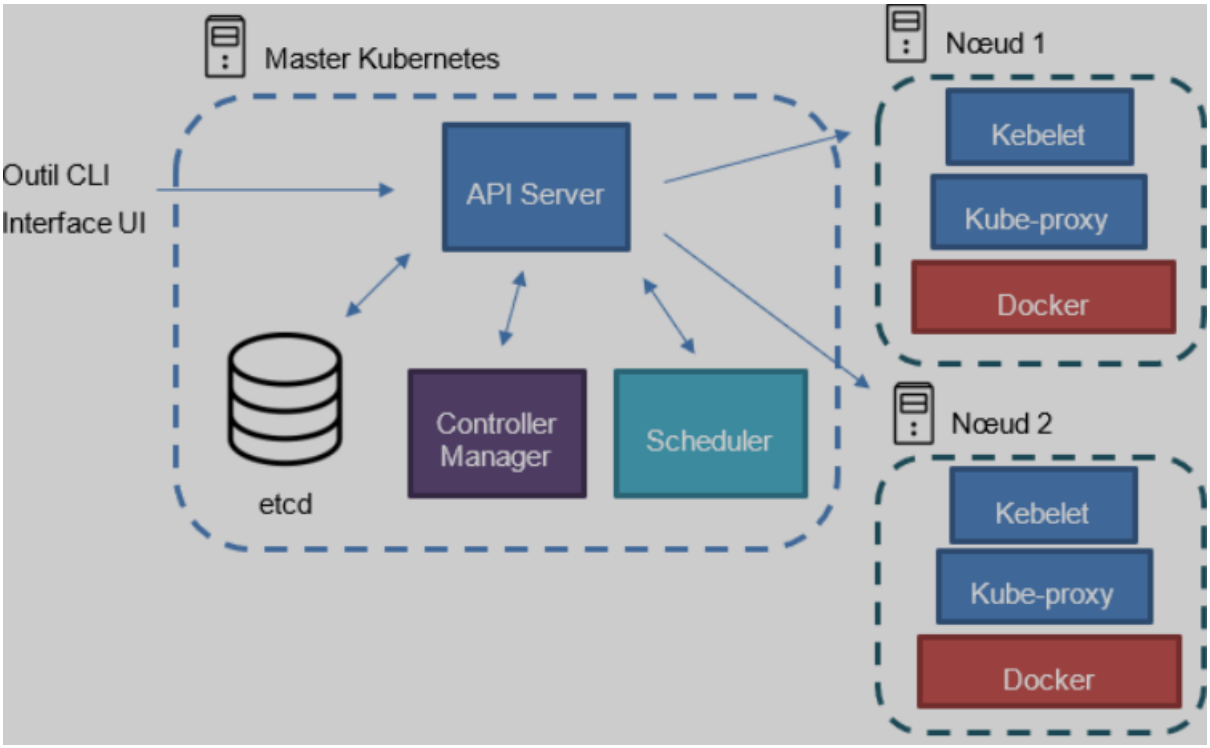

Figure 2 : Architecture Docker

Partie V: CI/CD avec Jenkins

5.1 Configuration et bénéfices pour la qualité du code :

La mise en place de Jenkins dans notre projet pour automatiser le processus de CI/CD a considérablement amélioré notre efficacité de développement et garantit des déploiements fiables. Voici une vue d'ensemble du processus et de la configuration que nous avons adoptés :

Processus de CI/CD avec Jenkins :

Déclenchement Automatique : Le processus de CI/CD est déclenché automatiquement à chaque commit sur le référentiel de code source. Cela garantit une intégration continue à chaque changement, facilitant la détection rapide d'éventuels problèmes.

Étapes de Tests Unitaires : Le pipeline commence par l'exécution de tests unitaires pour chaque microservice. Ces tests vérifient l'intégrité des fonctionnalités individuelles et s'assurent qu'aucune modification n'affecte négativement le comportement existant.

Construction des Images Docker: Après des tests unitaires réussis, le pipeline procède à la construction des images Docker pour chaque microservice. Cette étape encapsule le microservice avec toutes ses dépendances dans un conteneur Docker, garantissant une portabilité et une reproductibilité maximales.

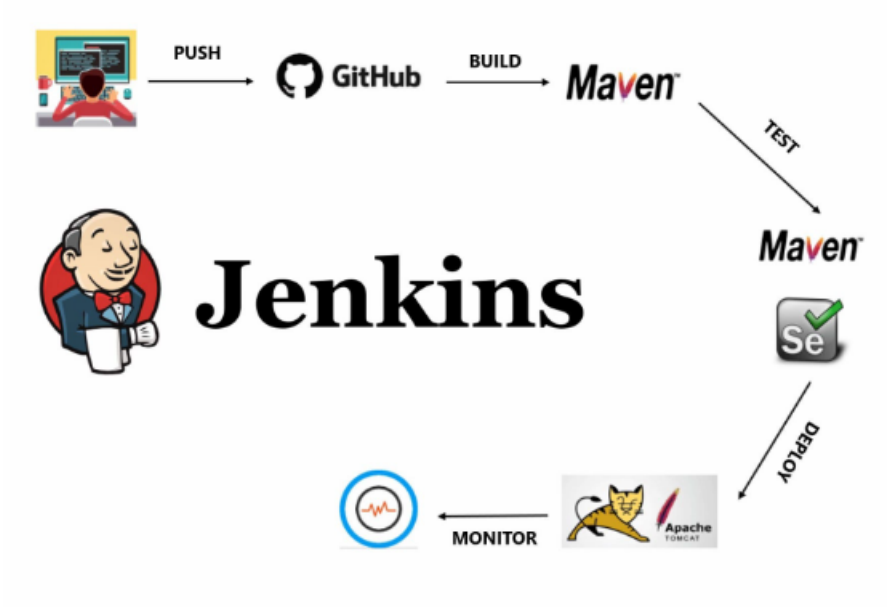

Figure 3 : Architecture Jenkins

Partie VII : Conclusion

6.1 Résumé des accomplissements :

Notre projet a atteint avec succès ses objectifs en mettant en place une architecture de microservices moderne et innovante. En combinant des technologies de pointe telles que Consul, Spring, Angular, et Jenkins, nous avons créé un écosystème informatique agile et hautement efficace. Les accomplissements clés de ce projet peuvent être résumés comme suit:

Architecture Microservices Bien Établie: Implementation de l'architecture microservices a été soigneusement planifiée et réalisée, avec Consul en tant que composant central pour la gestion de la configuration et la découverte des services. Cette approche favorise la modularité, l'indépendance, et la facilité d'évolutivité.

Intégration de Docker pour la Conteneurisation : L'utilisation de Docker a simplifié le déploiement des microservices, offrant une encapsulation efficace avec des avantages en termes de portabilité, de gestion des dépendances, et de reproductibilité de l'environnement.

CI/CD Optimisée avec Jenkins : Jenkins a été intégré avec succès pour automatiser les processus de CI/CD. Cette automatisation a permis des déploiements continus et fiables, accélérant ainsi le cycle de développement et garantissant l'intégrité du code.

6.2 Perspectives :

Le succès de notre projet actuel crée une base solide pour les évolutions futures et les améliorations continues. Les perspectives futures s'orientent vers plusieurs axes stratégiques pour garantir l'efficacité opérationnelle à long terme:

1. Exploration de Tests d'Intégration Approfondis :

La prochaine étape consistera à approfondir nos pratiques de tests en incorporant des tests d'intégration complets. Cela permettra de garantir le bon fonctionnement et la cohésion de l'ensemble du système, en mettant l'accent sur les interactions complexes entre les microservices.

2. Optimisation Continue des Performances :

L'optimisation des performances demeurera une priorité constante. Nous prévoyons d'analyser en profondeur les performances de chaque microservice, d'identifier les points de congestion potentiels, et de mettre en œuvre des améliorations continues pour garantir une réactivité maximale du système.

3. Adoption de Nouvelles Technologies :

La technologie évolue rapidement, et notre projet restera à la pointe en explorant de nouvelles technologies pertinentes. L'adoption de frameworks, bibliothèques, ou outils émergents permettra d'optimiser davantage notre architecture microservices et de rester aligné sur les meilleures pratiques de l'industrie.

4. Sécurisation Avancée :

La sécurité sera renforcée avec des stratégies avancées pour protéger nos microservices contre les menaces potentielles. L'exploration de solutions de gestion des identités, l'intégration de pratiques de sécurité DevSecOps, et l'audit régulier des vulnérabilités seront au cœur de nos préoccupations.

5. Analyse de Données et Intelligence Artificielle (IA) :

L'exploration de solutions d'analyse de données et d'intelligence artificielle sera envisagée pour tirer des insights précieux de nos systèmes. L'intégration de modèles d'IA peut apporter des avantages

significatifs, que ce soit pour la prise de décision, la personnalisation des expériences utilisateur, ou l'optimisation des processus.

6. Formation et Développement Continu :

La formation continue des membres de l'équipe sur les nouvelles technologies et les meilleures pratiques sera encouragée. Un programme de développement continu visant à renforcer les compétences existantes et à acquérir de nouvelles connaissances sera mis en place pour maintenir l'équipe à la pointe de l'innovation.