**Rapport Projet Gestion RH : Architecture des composants d’entreprise.**

**Réalisé par :**

ELHAMMOUI Mousaab

ELKABIR Hamza

5IIR G2

1. **Introduction**

* **Aperçu du projet**

Le projet de gestion de portfolio vise à développer une solution permettant de créer des portfolios personnalisés en fonction des données fournies. Il se compose de deux microservices, "Profile" et "Portfolio", qui interagissent avec un serveur Eureka pour la découverte des services, ainsi qu'avec une passerelle (Gateway) pour la gestion des requêtes et des réponses.

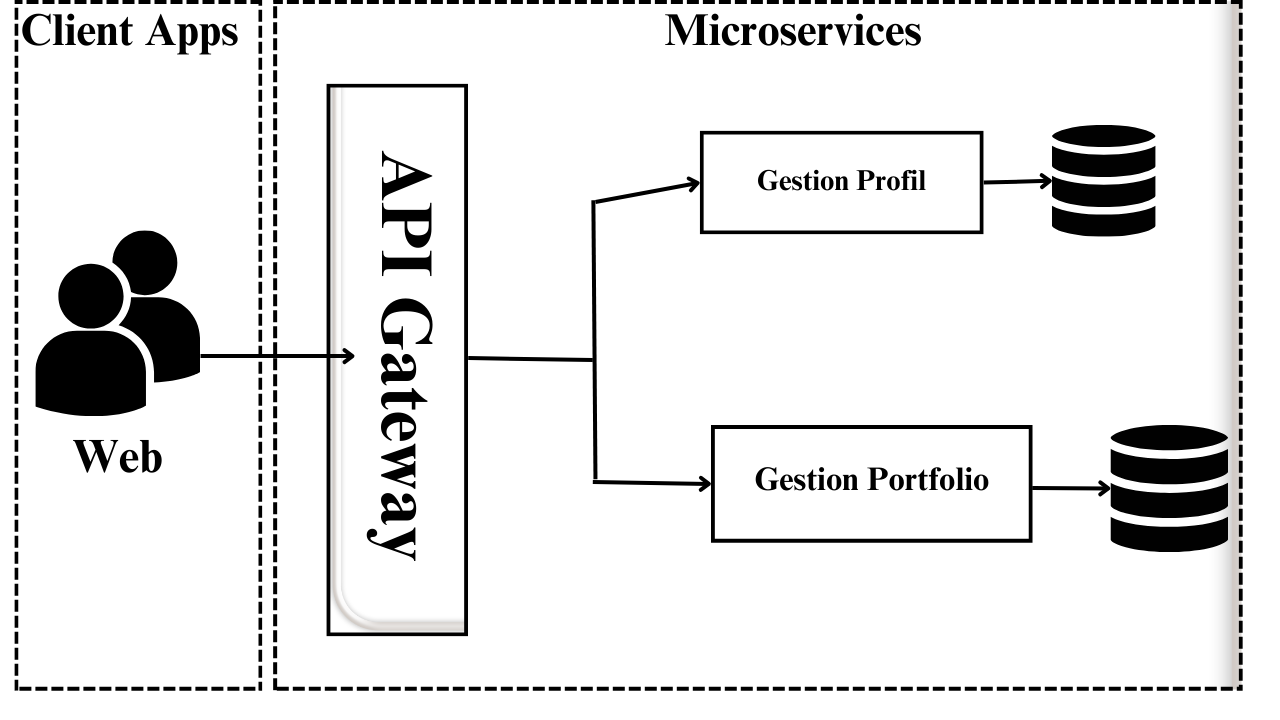
* **Importance de l’architecture microservices**

L'architecture microservices a été choisie pour sa capacité à offrir une solution modulaire et évolutive. En adoptant cette approche, le système peut être développé de manière indépendante, permettant une maintenance facilitée, une évolutivité accrue et une agilité dans l'ajout de nouvelles fonctionnalités.

1. **Architecture Microservices**

* **Architecture**

L'architecture du système est basée sur le modèle microservices, composée de deux microservices principaux, "Profile" et "Portfolio". Ces microservices sont interconnectés via un serveur Eureka pour la découverte des services, et une passerelle (Gateway) pour la gestion centralisée des requêtes et des réponses.



* **Description des services**
* **Microservice ‘Profile’**

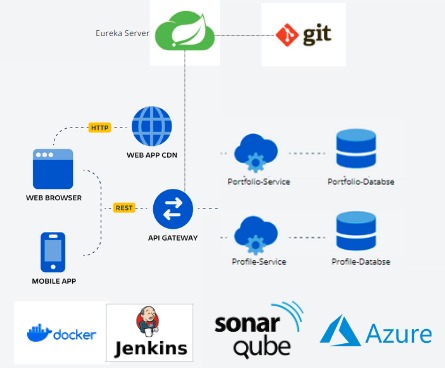
Le microservice "Profile" est responsable de la gestion des informations personnelles des utilisateurs. Cela inclut les détails tels que le nom, l'expérience, les compétences et d'autres données pertinentes pour la construction d'un portfolio personnalisé.

* **Microservice ‘Portfolio’**

Le microservice "Portfolio" gère les informations spécifiques au portfolio, telles que les projets, les réalisations, les certifications et autres éléments qui contribuent à la création d'un portfolio complet et personnalisé.

* **Mécanismes de communication**

Les microservices communiquent entre eux via des API RESTful sécurisées. La passerelle (Gateway) agit comme un point d'entrée unique pour les clients, acheminant les requêtes vers les microservices appropriés. Le serveur Eureka facilite la découverte des services, permettant une communication dynamique et évolutive.

1. **Conception des Microservices**

* **Microservice ‘Profil’**

- **Base de données :** Utilisation d'une base de données MySQL dédiée pour stocker les informations personnelles des utilisateurs.

- **API REST :** Exposition d'API RESTful pour la gestion des profils, permettant la création, la mise à jour et la récupération des données.

* **Microservice ‘Portfolio’**

**- Base de données :** Utilisation d'une base de données MYSQL dédiée pour stocker les informations spécifiques au portfolio.

- **API REST :** Exposition d'API RESTful pour la gestion des portfolios, permettant la création, la mise à jour et la récupération des informations liées aux projets, réalisations, certifications, etc.

1. **Conteneurisation avec Docker**

* **Implémentation**

La conteneurisation avec Docker est une étape essentielle dans le déploiement et la gestion efficace des microservices dans un environnement de gestion de portfolio. Docker permet d'encapsuler chaque microservice et ses dépendances dans des conteneurs légers et autonomes.

* **Avantages**
* **Isolation**

Les conteneurs Docker isolent chaque microservice, garantissant qu'ils fonctionnent indépendamment sans interférence les uns avec les autres. Cela facilite le déploiement et la gestion.

* **Portabilité**

Les conteneurs Docker encapsulent tout ce dont un microservice a besoin pour s'exécuter, ce qui rend l'application portable. Les conteneurs peuvent être exécutés de manière cohérente sur n'importe quel environnement compatible Docker.

* **Facilite de déploiement**

La conteneurisation simplifie le processus de déploiement. Les microservices peuvent être déployés rapidement et de manière cohérente, que ce soit localement pour le développement, sur des serveurs de production ou dans le cloud.

* **Gestion de dépendances**

Docker permet de gérer efficacement les dépendances de chaque microservice. Les dépendances spécifiques à un microservice sont incluses dans son conteneur, éliminant ainsi les conflits de versions.

* **Evolutivité**

La conteneurisation avec Docker facilite l'évolutivité horizontale en permettant le déploiement de plusieurs instances de chaque microservice, selon les besoins de charge.

* **Gestion des versions**

Les images Docker peuvent être taguées avec des versions spécifiques, permettant un contrôle précis des versions et des mises à jour.

* **Facilite de collaboration**

Les développeurs peuvent travailler de manière plus cohérente en utilisant les mêmes images Docker, garantissant que l'environnement de développement est similaire à celui de la production.