Examen pratique pour les étudiants de Master 2 Numres 🎓

Gestion d'un cabinet médical 🏥 avec microservices

Objectifs

1. Implémenter des microservices en utilisant Hystrix (ou Resilience4j), Retry (OP), Eureka (ou Grafana) et Swagger (ou openAPI).
2. (OP) Intégrer un calendrier externe gratuit via une API pour la prise de rendez-vous.
3. Créer des relations entre les microservices pour assurer une cohérence des données.
4. (OP) Mettre en place un dossier médical partagé entre les praticiens et les patients.
5. Configurer Eureka pour la découverte dynamique des services.
6. Créer un Dockerfile pour chaque microservice.
7. Écrire les fichiers YAML pour le déploiement sur Kubernetes.

Description du projet

Le cabinet médical doit comporter les microservices suivants :

1. **Service Patient** : Gestion des patients (ajout, modification, suppression, consultation).
2. **Service Praticien** : Gestion des praticiens (ajout, modification, suppression, consultation).
3. (OP) **Service Rendez-vous** : Gestion des rendez-vous et intégration avec un calendrier externe.
4. (OP) **Service Dossier Médical** : Gestion d'un dossier médical partagé entre un praticien et un patient.
5. **Service Gateway** : Gateway pour centraliser les appels aux microservices.
6. **Service Eureka** : Mise en place d'un serveur Eureka pour la découverte des services.

Instructions

Partie 1 : Implémentation des microservices

1. **Service Patient**
   - Gérer les opérations liées aux patients, notamment l'ajout, la modification, la suppression et la consultation.
   - Documenter les endpoints avec Swagger.
2. **Service Praticien**
   - Gérer les opérations liées aux praticiens, notamment l'ajout, la modification, la suppression et la consultation.
   - Documenter les endpoints avec Swagger.
3. **(OP) Service Rendez-vous**
   - Gérer la prise de rendez-vous et intégrer un calendrier externe gratuit via une API.

    - Assurer la résilience des appels externes avec des mécanismes de tolérance aux pannes (Hystrix) et de Retry.
    - Documenter les endpoints avec Swagger.
4. **(OP) Service Dossier Médical**
    - Mettre en place un système de dossier médical partagé entre un praticien et un patient.
    - Assurer la synchronisation des données avec les services Patient et Praticien.
    - Documenter les endpoints avec Swagger.
5. **Service Gateway**
    - Centraliser les appels aux microservices via un service Gateway.
    - Configurer la Gateway pour utiliser Eureka comme service de découverte.
6. **Service Eureka**
    - Créer un serveur Eureka pour permettre la découverte dynamique des services.
    - Configurer chaque microservice pour s'enregistrer automatiquement auprès du serveur Eureka.

Partie 2 : Conteneurisation

1. Créer un fichier Dockerfile pour chaque microservice.
    - Utiliser un environnement JDK pour exécuter les applications.
    - Inclure les commandes pour copier et exécuter les applications.
2. Tester chaque image en démarrant les conteneurs localement avec Docker.

Partie 3 : Déploiement Kubernetes

1. Écrire un fichier YAML pour chaque microservice contenant :
    - Les déploiements Kubernetes (Deployment).

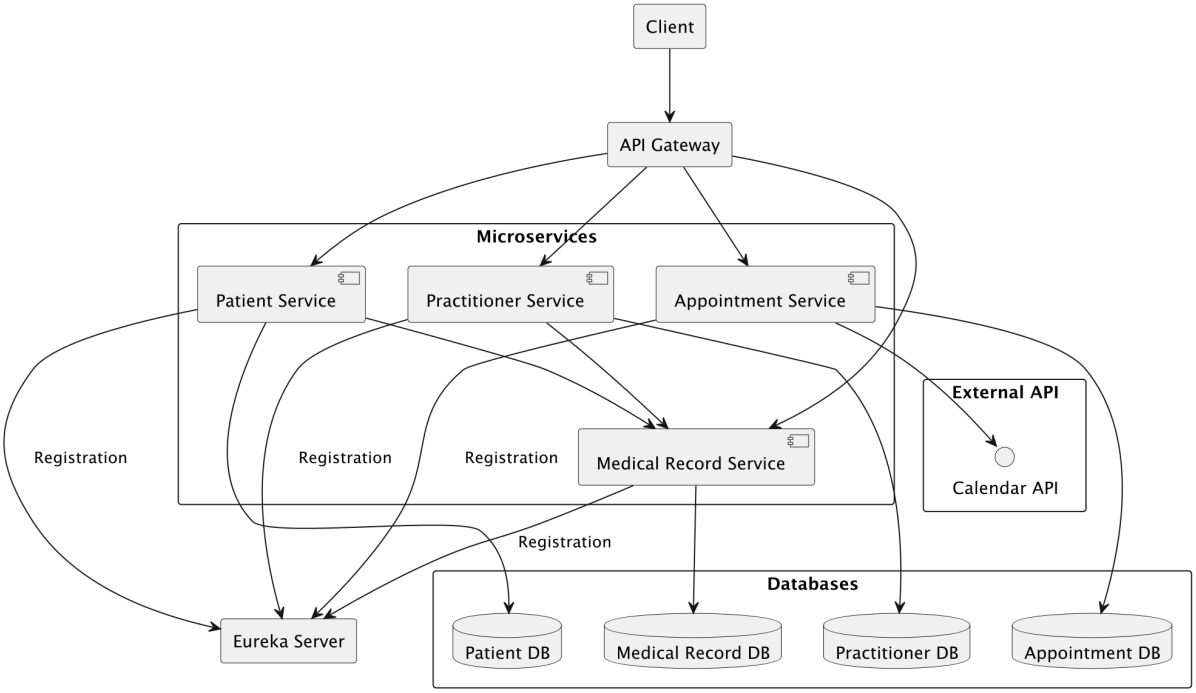
Livrables

1. Code source des microservices avec un README(OP) expliquant comment les exécuter.
2. Dockerfiles pour chaque microservice.
3. Fichiers YAML pour le déploiement Kubernetes.
4. un répo github avec l'historique des commits

Barème de notation

- Implémentation correcte des microservices (25%) et relations entre eux (15%) : 40%
- Documentation Swagger : 5%
- Tolérance aux pannes (Hystrix) : 20%
- Intégration avec Eureka : 20%
- Conteneurisation avec Docker : 10%
- Déploiement Kubernetes et connectivité des services : 5%

Architecture :



Voici une description textuelle du schéma d'architecture de l'application :

Annexe :

1. **Client/API Gateway** :
    - **Rôle** : Interface unique pour tous les appels effectués par les clients (front-end ou applications externes).
    - **Technologie** : Spring Cloud Gateway.
    - **Fonctionnalités** :
        - Redirection des requêtes vers les microservices appropriés.
        - Gestion des erreurs globales (fallback via Hystrix si un service est indisponible).
2. **Service Eureka** :
    - **Rôle** : Service de découverte centralisé pour enregistrer et découvrir dynamiquement les microservices.
    - **Technologie** : Spring Cloud Netflix Eureka Server.
    - **Fonctionnalités** :
        - Les microservices (Patient, Praticien, Rendez-vous, Dossier médical) s'enregistrent automatiquement ici.
        - Permet à chaque service de communiquer sans avoir besoin de connaître directement les adresses réseau.
3. **Microservices principaux** :
    - **Service Patient** :

        - Gestion des informations des patients (ajout, modification, suppression, consultation).
        - Connecté au service Dossier médical pour récupérer et mettre à jour les dossiers partagés.
    - **Service Praticien** :
        - Gestion des informations des praticiens (ajout, modification, suppression, consultation).
        - Connecté au service Dossier médical pour accéder aux dossiers des patients suivis.
    - (OP) **Service Rendez-vous** :
        - Gestion des rendez-vous.
        - Intégration avec un calendrier externe gratuit via une API tierce (par exemple, Google Calendar ou autre).
        - Mécanismes de résilience pour les appels externes avec Hystrix et Retry.
    - (OP) **Service Dossier médical** :
        - Gestion des dossiers médicaux partagés entre praticiens et patients.
        - Synchronisation avec les services Patient et Praticien pour assurer la cohérence des données.
4. (OP) **Base de données** :
    - Chaque microservice utilise sa propre base de données pour respecter le principe de séparation (Database per service).
    - Types de bases de données :
        - Service Patient et Praticien : Bases relationnelles (MySQL, PostgreSQL).
        - Service Dossier médical : Base relationnelle ou NoSQL (MongoDB pour stocker les documents médicaux).
        - Service Rendez-vous : Base relationnelle avec intégration des informations externes.
5. **Infrastructure de déploiement** :
   Conteneurs Docker : Chaque microservice est empaqueté dans son propre conteneur Docker.
    - **Cluster Kubernetes** :
        - Déploiement des conteneurs Docker sur Kubernetes.
        - Utilisation des fichiers YAML pour définir les déploiements Kubernetes.
6. **Communication et résilience** :
    - **Hystrix** : Implémenté sur le service Gateway pour gérer les scénarios de défaillance (circuit breaker).
    - **Retry** : Implémenté dans le service Rendez-vous pour les appels vers l'API de calendrier externe.
    - **Service-to-Service Communication** : Les microservices communiquent entre eux via des appels HTTP (ou gRPC) et utilisent Eureka pour résoudre les adresses réseau.

Relations entre les services :

- (OP) Le **Service Dossier médical** joue un rôle central en étant lié à la fois au **Service Patient** et au **Service Praticien**. Il agit comme un référentiel partagé.
- (OP) Le **Service Rendez-vous** utilise les informations des patients et praticiens pour organiser les rendez-vous et met à jour les dossiers médicaux lorsque nécessaire.)
- Le **Service Gateway** agit comme un point d'entrée unique pour accéder à tous les autres services.

Exemple de squelette 🩻

```
medical-office-management/
├── eureka-server/
│   ├── src/
│   │   ├── main/java/com/example/eurekaserver/
│   │   │   └── EurekaServerApplication.java
│   │   ├── main/resources/
│   │   │   └── application.yml
│   ├── Dockerfile
│   ├── deployment.yaml
│   ├── pom.xml
│
├── gateway-service/
│   ├── src/
│   │   ├── main/java/com/example/gateway/
│   │   │   └── GatewayApplication.java
│   │   ├── main/resources/
│   │   │   └── application.yml
│   ├── Dockerfile
│   ├── deployment.yaml
│   ├── pom.xml
│
├── patient-service/
│   ├── (idem avec PatientController, PatientService, etc.)
│   ├── Dockerfile
│   ├── deployment.yaml
│   ├── pom.xml
│
├── practitioner-service/
│   ├── (idem avec PractitionerController, PractitionerService, etc.)
│   ├── Dockerfile
│   ├── deployment.yaml
│   ├── pom.xml
│
├── appointment-service/
│   ├── (idem avec AppointmentController, AppointmentService, etc.)
│   ├── Dockerfile
│   ├── deployment.yaml
│   ├── pom.xml
```

```
│
├── medical-record-service/
│   ├── (idem avec MedicalRecordController, MedicalRecordService, etc.)
│   ├── Dockerfile
│   ├── deployment.yaml
│   ├── pom.xml
├── docker-compose.yml
└── README.md
```