⚽Examen de Développement de Microservices pour la Gestion d'Équipes de Football ⚽

Durée de l'examen : 6 heures

Instructions : Répondez à toutes les questions dans un environnement de développement approprié en utilisant Spring Boot avec les bibliothèques Ribbon, Eureka, Circuit Breaker, Swagger et Actuator. Vous pouvez choisir le langage de programmation de votre choix (Java ou Kotlin). Veillez à suivre les bonnes pratiques de développement et à commenter votre code lorsque cela est nécessaire.

Contexte : Vous développez un système de gestion d'équipes de football. Vous devez créer quatre microservices pour gérer différentes fonctionnalités liées au football.

Partie 1 : Service d'Équipes

Créez un microservice appelé "TeamService" qui gère les équipes de football. Le service doit avoir les points de terminaison suivants :
- `GET /teams/{id}` : Renvoie les détails d'une équipe par son identifiant.
- `POST /teams` : Permet d'ajouter une nouvelle équipe.
- `PUT /teams/{id}` : Met à jour les informations d'une équipe existante.
- `DELETE /teams/{id}` : Supprime une équipe par son identifiant.
Assurez-vous que le service s'enregistre auprès d'Eureka.

Partie 2 : Service de Joueurs

Créez un microservice appelé "PlayerService" qui gère les joueurs de football. Le service doit avoir les points de terminaison suivants :
- `GET /players/{id}` : Renvoie les détails d'un joueur par son identifiant.
- `POST /players` : Permet d'ajouter un nouveau joueur.
- `PUT /players/{id}` : Met à jour les informations d'un joueur existant.
- `DELETE /players/{id}` : Supprime un joueur par son identifiant.
Assurez-vous que le service s'enregistre auprès d'Eureka.

Partie 3 : Service de Matchs

Créez un microservice appelé "MatchService" qui gère les matchs de football. Le service doit avoir les points de terminaison suivants :
- `GET /matches/{id}` : Renvoie les détails d'un match par son identifiant.
- `POST /matches` : Permet d'ajouter un nouveau match.
- `PUT /matches/{id}` : Met à jour les informations d'un match existant.
- `DELETE /matches/{id}` : Supprime un match par son identifiant.
Assurez-vous que le service s'enregistre auprès d'Eureka.

Partie 4 : Service de Statistiques

Créez un microservice appelé "StatsService" qui gère les statistiques des équipes et des joueurs dans le contexte des matchs de football. Le service doit avoir les points de terminaison suivants :
- `GET /team-stats/{teamId}` : Renvoie les statistiques d'une équipe pour la saison.
- `GET /player-stats/{playerId}` : Renvoie les statistiques d'un joueur pour la saison.
Assurez-vous que le service s'enregistre auprès d'Eureka.

Partie 5 : Documentation Swagger

Utilisez Swagger pour documenter les points de terminaison de tous les microservices. Assurez-vous que la documentation est accessible via des points de terminaison Swagger distincts pour chaque microservice.

Partie 6 : Tolérance de Panne

Intégrez un circuit breaker (par exemple, Hystrix) dans les microservices "TeamService," "PlayerService," et "MatchService" pour gérer la tolérance de panne. Créez un point de terminaison Swagger pour surveiller l'état du circuit breaker. Implémentez également des méthodes de fallback pour gérer les cas où le circuit est ouvert.

Partie 7 : Monitoring

Intégrez Actuator dans tous les microservices pour surveiller leur santé, leurs métriques et d'autres informations pertinentes. Assurez-vous que ces informations sont accessibles via des points de terminaison Actuator distincts pour chaque microservice.

Partie 8 : Load Balancing

Faites en sorte que le microservices “MatchService” se déploie sur 3 instances en mode Actif/Actif/Actif

Partie 9 : Dockorisation et déploiement sur Kubernetes

Créez des dockerfile pour tous les microservices en veillant à spécifier le bon chemin des .jar et les bonnes dépendances dans le dockerfile (Version de Java, répertoire copié …)
Créez des manifestes Kubernetes de kind : “Deployment” pour tous les microservices en référençant la bonne image docker dans le bon manifest .yaml

Évaluation :

Les candidats seront évalués en fonction de la qualité du code, du respect des bonnes pratiques de développement, de la documentation, de la gestion de la tolérance de panne, et de la mise en œuvre des mécanismes de surveillance avec Actuator. Assurez-vous que les microservices communiquent correctement via Eureka et que les points de terminaison Swagger et Actuator sont accessibles.

Remarque : N'oubliez pas de commenter votre code pour expliquer les choix de conception et les étapes clés de l'implémentation.