Reporting de la POC Medhead

Table des matières

Auteur	HENRICK AGNAME
Version du document	V1

Introduction :

MedHead est un regroupement de grandes institutions médicales œuvrant au sein du système de santé britannique et assujetti à la réglementation et aux directives locales (NHS). Les organisations membres du Consortium utilisent une grande variété de plateformes, de technologies et d'appareils qui souhaite utiliser Java comme langage principal.

Justification de l'environnement

Spring Boot

- Notre application améliore considérablement la productivité grâce à sa configuration automatique. En adoptant une approche de convention sur la configuration, Spring Boot permet un démarrage rapide des projets, ce qui est particulièrement bénéfique dans les environnements où la vitesse de mise sur le marché est essentielle.
- Avec notre application, nous avons simplifié le déploiement grâce à la création d'exécutables autonomes. Ces applications peuvent être lancées comme des jars indépendants, ce qui facilite grandement le déploiement et la gestion des environnements, en particulier pour les architectures de microservices et sur les plateformes cloud.
- Notre application assure une intégration transparente au sein d'un écosystème étendu. Grâce à un vaste choix de starters et de modules qui intègrent des bibliothèques courantes, le temps nécessaire pour développer des fonctionnalités supplémentaires est réduit, permettant aux développeurs de se concentrer sur les aspects métier spécifiques sans se préoccuper de la compatibilité et de la configuration des composants.

Liste des dependance spring boot pour le POC :

- Spring Boot Starter Data JPA
- Spring Boot Starter Security
- Spring Boot Starter Web
- Spring Boot Starter Test
- Spring Boot Starter JDBC

Maven

- Simplicité et efficacité : En adoptant Maven, nous avons privilégié la philosophie de "convention sur la configuration", ce qui a permis de démarrer nos projets Spring Boot rapidement. Cet outil s'aligne avec notre objectif de maintenir une structure de projet épurée et de faciliter l'intégration continue dès le début.
- Gestion des dépendances fiable : Notre choix de Maven s'est avéré judicieux pour la gestion des dépendances, une composante clé de notre projet Spring Boot qui s'appuie sur un ensemble complexe de bibliothèques. Le fichier pom.xml de Maven simplifie la déclaration et la résolution des dépendances, assurant ainsi la stabilité de notre environnement de build.
- Interopérabilité avec l'écosystème Spring : La synergie entre Maven et Spring Boot est fructueuse, avec des starters et configurations prêts à l'emploi qui nous permettent de minimiser le temps de configuration manuelle. Cette interopérabilité favorise un développement plus fluide, nous permettant de nous concentrer sur la logique métier.
- Communauté robuste et ressources abondantes : Nous avons choisi Maven aussi pour sa communauté mature et ses ressources abondantes. En cas de problèmes ou pour optimiser notre processus de build, nous pouvons nous appuyer sur une documentation complète et le soutien d'une vaste communauté.
- Cycle de vie du projet standardisé : L'utilisation de Maven garantit l'adhésion à un cycle de vie de projet standardisé, avantageux pour la maintenance et l'évolutivité à long terme. Les phases bien définies du cycle de vie de build de Maven assurent que les tests, le packaging et le déploiement de notre application sont exécutés de manière cohérente et automatisée.

JAVA 17 :

- Long-Term Support (LTS) : Java 17 étant une version LTS, nous bénéficions d'un support à long terme de la part d'Oracle et de la communauté OpenJDK. Cela nous assure des mises à jour de sécurité et de performances pour les années à venir, minimisant les risques liés à l'utilisation de versions obsolètes. Pour notre

entreprise et nos projets de développement, Java 17 offre une plateforme stable et sûre, garantissant la viabilité et la maintenance du code à long terme.

- Nouvelles fonctionnalités et améliorations de performances : L'adoption de Java 17 nous apporte d'importantes innovations, y compris des améliorations de performance, de nouvelles API et des fonctionnalités de langage avancées, comme le pattern matching pour les expressions switch, qui simplifient et clarifient notre code. Java 17 propose également des optimisations au niveau du ramasse-miettes (garbage collector) et de la JVM, qui peuvent améliorer considérablement les performances de notre application Spring Boot. L'utilisation de Java 17 nous permet donc de tirer pleinement parti de ces progrès et d'optimiser l'exécution de notre application.

Coordination des classes :

@RestController dans HospitalController:

- Notre **HospitalController** facilite la création d'interfaces API RESTful grâce à l'annotation **@RestController**, qui combine **@Controller** et **@ResponseBody**. Cela nous permet de renvoyer des réponses directement dans le corps des réponses HTTP sans avoir besoin de vues.
- En déléguant les aspects de la logique métier aux services, **HospitalController** améliore la séparation des préoccupations, se focalisant uniquement sur le traitement des requêtes HTTP. Cette approche clarifie la responsabilité de notre contrôleur.
- Nous gérons finement les réponses HTTP en utilisant **ResponseEntity<?>** au sein de **HospitalController**, ce qui nous donne un contrôle précis sur le statut HTTP et le contenu de la réponse, assurant ainsi que nous répondons de manière adéquate aux demandes des clients.
- Le mapping des requêtes est grandement simplifié grâce aux annotations déclaratives telles que **@GetMapping** et **@RequestParam**. Cela rend les associations entre les endpoints et les méthodes de traitement explicites, claires et faciles à maintenir dans notre projet.

@Service dans HospitalService:

- Notre approche promeut une séparation stricte des préoccupations, isolant la logique métier de la logique de présentation. Nous avons centralisé les opérations spécifiques au domaine dans le service, ce qui a clarifié et structuré notre base de code.
- Nous facilitons la gestion des transactions grâce à l'annotation **@Transactional**, ce qui garantit l'intégrité des données à travers des opérations atomiques. Ceci est particulièrement important lors des interactions complexes avec la base de données, où la cohérence des données est essentielle.

- En structurant notre logique métier de manière à ce qu'elle soit réutilisable et modulaire, nous avons permis qu'elle soit invoquée depuis différents points de l'application. Cela a réduit la duplication du code et a facilité la maintenance, contribuant à une base de code plus propre et plus gérable.
- Nous avons également amélioré la testabilité du système en isolant la logique métier, ce qui a permis de réaliser des tests unitaires et d'intégration plus efficaces.

DTO (Data Transfer Object):

- Sécurité et encapsulation : Dans notre architecture API, l'utilisation de DTOs tels que **HospitalDTO** est essentielle pour sécuriser les interactions. Ils nous permettent de partager uniquement les données nécessaires avec les clients de l'API, protégeant ainsi contre l'exposition d'informations sensibles qui peuvent résider dans nos entités.
- Flexibilité de l'API : Les DTOs nous offrent également la flexibilité de façonner les données que nous exposons à travers notre API sans impacter la structure des entités de base. Cette approche est cruciale pour assurer la stabilité de notre API, même en cas de modifications internes du modèle de données, garantissant ainsi une meilleure expérience pour les utilisateurs de l'API et une maintenance simplifiée du système.

Mapper:

- Automatisation de la conversion : L'intégration de **HospitalMapper** dans notre projet automatise le processus de transformation entre les entités et les DTOs. Cela réduit considérablement le besoin d'écrire du code redondant et diminue le risque d'erreurs qui pourraient survenir avec une conversion manuelle, rendant le processus plus efficace et fiable.
- Cohérence et clarté : En centralisant la logique de conversion avec **HospitalMapper**, nous avons simplifié la gestion des transformations de données au sein de l'application. Cela est particulièrement utile lorsque nous travaillons avec un grand nombre d'entités et de DTOs, garantissant que la logique de conversion est maintenue en un seul endroit pour une meilleure cohérence et une plus grande clarté du code.

Entité:

- Représentation de la base de données : Les entités de notre modèle, comme **Hospital**, servent de reflet direct à la structure des tables dans notre base de données. Grâce au mappage ORM (Object-Relational Mapping), nous pouvons interagir avec la base de données de manière efficace, en traduisant nos opérations orientées objet en opérations de base de données sans effort manuel excessif.
- Isolation de la couche de persistance : En confinant la logique d'accès aux données au sein des entités, nous avons assuré que notre couche métier

demeure indépendante des complexités et des détails techniques de la persistance des données. Cela permet une meilleure abstraction et une plus grande flexibilité lors de la modification ou de l'extension de notre logique métier, sans être contraint par la structure de la base de données.

Repository (HospitalRepository):

Dans notre projet, nous utilisons le repository comme une couche d'abstraction de haut niveau avec JPA qui simplifie les interactions avec la base de données. Il cache les complexités des opérations CRUD (Create, Read, Update, Delete) et des requêtes, permettant ainsi à d'autres parties de l'application de manipuler les données sans avoir à connaître les détails des commandes SQL ou de la logique de la base de données sous-jacente. Cela contribue à la propreté et à la maintenabilité du code en séparant les préoccupations et en réduisant la duplication.

Classes métier :

Hospital: Cette classe sert de représentation d'un hôpital au sein de notre application. Elle encapsule les détails tels que le nom de l'hôpital, sa latitude, sa longitude, son adresse, entre autres informations pertinentes. Chaque instance de **Hospital** correspond à un hôpital unique dans notre base de données.
Reservation: La classe **Reservation** représente une réservation faite par ou pour un patient. Elle contient des détails tels que le nom du patient, son âge, l'heure d'arrivée prévue, les symptômes présentés, et d'autres informations nécessaires pour gérer efficacement la réservation d'un rendez-vous ou d'un traitement.
Specialization: Cette classe représente une spécialité médicale au sein de l'hôpital. Elle inclut des détails tels que le nom de la spécialité et l'identifiant du groupe de spécialités auquel elle appartient. Cela permet d'associer les médecins et les services de l'hôpital aux spécialités correspondantes, facilitant ainsi la recherche et le référencement par les patients et le personnel de l'hôpital.

Mysql :

- **Popularité** : Forte communauté et support étendu.
- **Facilité d'utilisation** : Installation et configuration simples, accélérant le développement.
- **Interopérabilité** : Intégration aisée avec java et springboot.
- **Performances** : Gestion efficace de transactions importantes pour les requêtes de service.

- **Fonctionnalités Géospatiales** : Outils intégrés pour le calcul de proximité.
- **Gratuité** : Coût nul, favorable pour un budget de POC limité.
- **Outils** : Écosystème riche pour la gestion et l'optimisation de la performance.

Procédure Stockée (find_nearest_hospital):

Dans notre système de gestion des données hospitalières, l'utilisation de procédures stockées représente une stratégie efficace pour encapsuler et améliorer les performances. En concentrant la logique complexe directement dans la base de données, par exemple pour la recherche géospatiale d'hôpitaux, nous optimisons les performances et réduisons la charge sur l'application. Les procédures stockées permettent des opérations spécialisées qui s'exécutent plus rapidement car elles sont plus proches des données, diminuant ainsi le temps de réponse pour les utilisateurs et allégeant la quantité de traitement nécessaire côté serveur de l'application.

Angular 17.2

- **Modernité et Mises à Jour** : Angular 17.2 est une plateforme à la pointe de la technologie, incorporant les dernières optimisations de performance, les renforcements de sécurité et les fonctionnalités innovantes.
- **Écosystème Complet** : Angular offre un environnement de développement holistique avec des outils intégrés pour une expérience de développement, de test et de déploiement d'applications fluides et efficaces.
- **Productivité** : Grâce à des fonctionnalités avancées telles que le data binding, l'injection de dépendances et un système de routage sophistiqué, Angular booste la productivité en accélérant le processus de développement.
- **TypeScript** : L'adoption de TypeScript au sein d'Angular renforce la maintenabilité et la robustesse de notre code, profitant du typage statique et d'outils de refactoring avancés.
- **Composants Matériels** : Avec Angular Material, nous disposons d'une bibliothèque riche de composants d'interface utilisateur prêts à l'emploi qui facilitent la création rapide d'interfaces attrayantes et uniformes.
- **Support des PWA** : Angular simplifie le développement de Progressive Web Apps (PWA), ce qui est particulièrement bénéfique pour un prototype de concept (POC) démontrant des fonctionnalités opérationnelles en mode connecté et déconnecté.
- **Documentation et Communauté** : Une documentation exhaustive et une communauté dynamique soutiennent Angular, permettant une résolution rapide des problèmes et un apprentissage continu des meilleures pratiques.

Architecture microservices

Front-end :

Frontend:

- **Framework :** Angular 17.2, qui est une plateforme de développement riche et moderne pour la création d'applications web.
- **Langage :** TypeScript et JavaScript, utilisés pour structurer le code de façon robuste et maintenable avec l'avantage du typage statique de TypeScript.
- **Serveur :** Webpack Dev Server, qui permet un rechargement en direct (live reloading) pour un développement efficace et rapide.

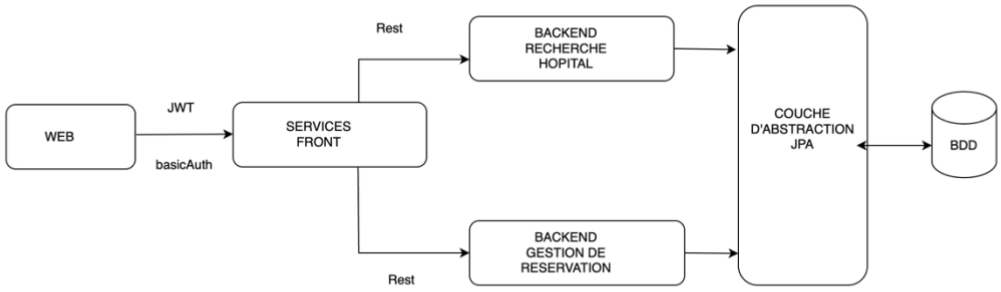
Backend:

- **Framework :** Spring et Spring Boot, qui offrent un démarrage rapide pour le développement d'applications d'entreprise avec un minimum de configuration requise.
- **Langage :** Java 17, la dernière version LTS qui apporte des améliorations de performance, de nouvelles fonctionnalités de langage et des mises à jour de sécurité régulières.
- **Serveur :** Tomcat, un serveur d'applications web léger et puissant utilisé pour exécuter nos applications Spring Boot.

Base de Données (BDD):

- **MySQL**, un système de gestion de base de données relationnelle robuste et largement utilisé.

Schéma d'architecture :

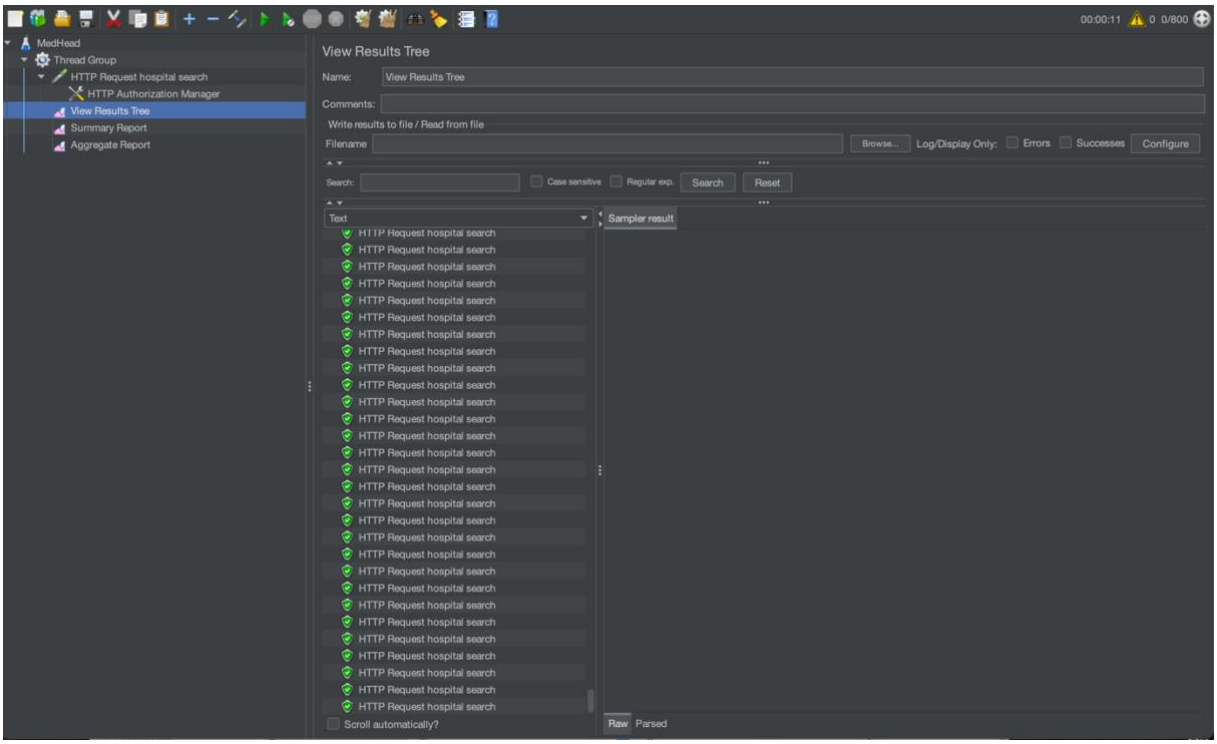


Rapport sur le POC :

Notre application a été soigneusement conçue suivant les principes d'une architecture microservices, où chaque service est doté d'une responsabilité unique et emploie une pile technologique cohérente. Cela simplifie non seulement l'intégration dans des systèmes plus vastes mais accélère également les processus grâce à des procédures stockées en base de données. Par exemple, la fonction de recherche d'hôpitaux par distance, spécialité et disponibilité de lits est gérée efficacement via une procédure stockée, optimisant ainsi les temps de réponse.

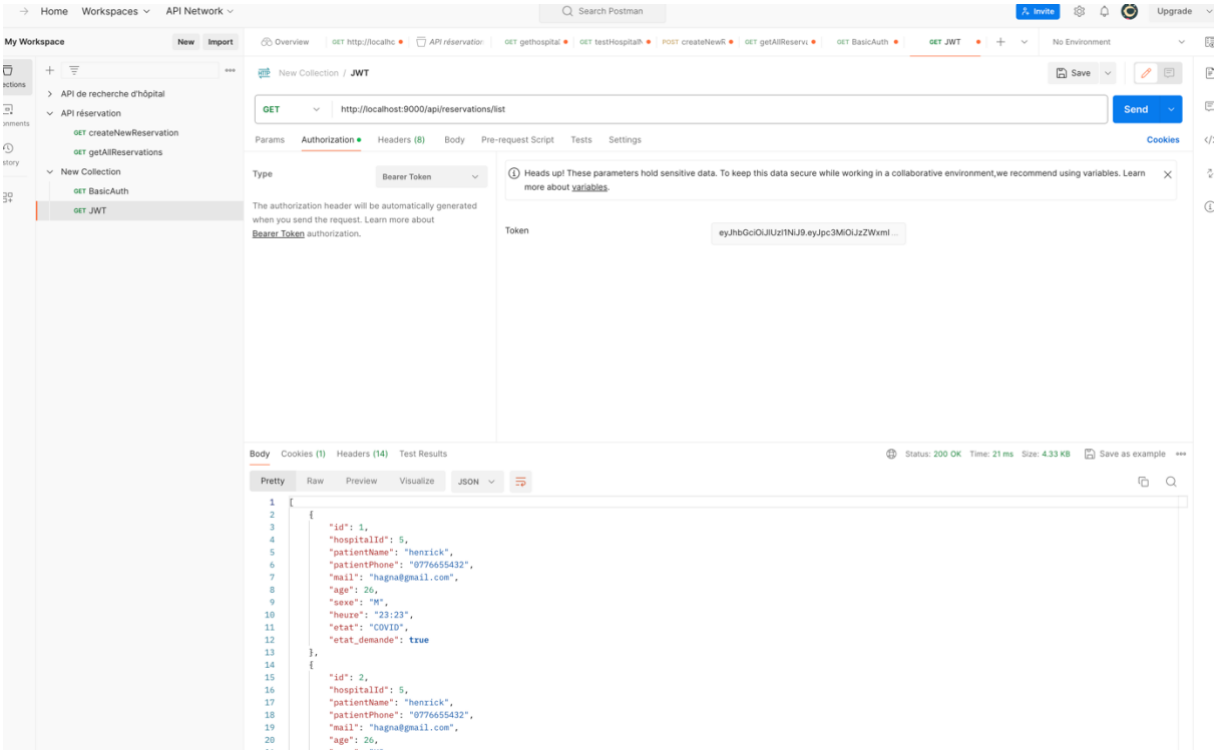
Le processus de test de l'application a été conduit avec précision et attention. Nous avons utilisé Postman pour tester de façon exhaustive tous les points d'accès, assurant une couverture complète des fonctionnalités et une intégration transparente avec l'API. En parallèle, des tests de charge avec JMeter ont permis d'évaluer la robustesse de l'application sous de fortes contraintes, garantissant sa stabilité et sa performance en conditions réelles. En complément, des tests d'intégration et unitaires ont été menés pour confirmer la fiabilité fonctionnelle et la solidité du code à chaque étape du développement.

JMeter :



Dans notre scénario de test de stress, nous avons simulé une situation où l'application reçoit 800 requêtes en l'espace d'une seconde pour localiser l'hôpital le plus proche. Les résultats ont démontré que notre système est parfaitement capable de gérer une telle charge sans compromettre la performance ou la stabilité, attestant ainsi de la robustesse et de l'efficacité de notre solution dans des conditions de trafic élevé.

Postman :



Le test de connexion à notre système a été concluant lorsqu'il a été effectué avec un token valide. Cela confirme que notre processus d'authentification fonctionne correctement, permettant aux utilisateurs autorisés d'accéder à l'application en toute sécurité.

Les test unitaire et d'intégration on été réaliser

```
2024-03-01T15:27:21.343Z  INFO 1982 --- [api] [           main] c.m.api.ApiApplicationTests
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.769 s -- in com.medHeadRese
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  8.144 s
[INFO] Finished at: 2024-03-01T15:27:21Z
[INFO] ------------------------------------------------------------------------
```

Notre application Angular intègre le package **GooglePlaceModule**, qui exploite l'API Google Places pour fournir des capacités de recherche d'adresses et de lieux. Cela, couplé avec le consentement du patient, nous permet de récupérer sa position géographique actuelle. À partir de ces données, nous pouvons calculer la distance qui le sépare des services d'urgence et lui indiquer l'hôpital le plus proche.
Le calcul de cette distance et la récupération de l'hôpital le plus proche sont effectués directement dans une procédure stockée au sein de notre base de données. Cette méthode d'implémentation optimise la vitesse de calcul de la distance et la récupération

des informations de l'hôpital, rendant ainsi le processus beaucoup plus rapide et efficace pour l'utilisateur final.

Automatisation des builds et conteneurisation :

Outils : GitHub Actions

Nous avons établi une chaîne d'intégration continue et de déploiement continu (CI/CD) en utilisant GitHub Actions, ce qui nous permet d'automatiser notre pipeline de développement logiciel directement depuis notre dépôt GitHub. Cette stratégie d'automatisation comprend la compilation du code, l'exécution des tests, et le déploiement des applications, en profitant de la souplesse et de l'intégration approfondie avec notre code source sur GitHub. L'emploi de GitHub Actions pour la CI/CD optimise les processus de développement, minimise les erreurs humaines, accélère le cycle de mise en production et assure une qualité logicielle supérieure en fournissant des retours d'information en temps réel et en identifiant les problèmes à un stade précoce.
CICD :

Dans notre projet, nous avons configuré un pipeline d'intégration et de déploiement continu (CI/CD) avec GitHub Actions, qui est activé par toute activité sur la branche "develop" ou par les pull requests sur la branche "master", à condition que les modifications affectent des fichiers dans le dossier "api/**" par exemple. Le workflow automatisé comprend plusieurs étapes clés :

1. **Construction et test de l'API** : Lorsqu'un push ou une pull request est détecté, GitHub Actions démarre une machine virtuelle Ubuntu et récupère le code source. Java 17 est installé pour exécuter Maven, qui compile le code de l'API et lance les tests unitaires. Cela assure que les changements récents n'affectent pas négativement l'API existante.
2. **Construction et publication de l'image Docker** : En cas de succès des tests, le workflow procède à la construction de l'image Docker de l'API. Le code source est de nouveau récupéré, Docker Buildx est configuré, et une connexion sécurisée est établie avec DockerHub. Une nouvelle image Docker est construite et poussée vers DockerHub avec le tag "latest".
3. **Notification** : Si le workflow échoue à n'importe quelle étape, GitHub Actions envoie automatiquement un email de notification au responsable du workflow. Cela permet une intervention rapide pour résoudre tout problème qui aurait pu survenir durant le processus de build ou de test, assurant ainsi une maintenance efficace et continue de la qualité de l'API.

L'adoption prochaine de Kubernetes en synergie avec DockerHub marquera un jalon important dans l'évolution de notre infrastructure CI/CD. Cette intégration vise à automatiser le déploiement des images Docker récemment mises à jour sur DockerHub vers un cluster Kubernetes. Ce processus automatisé réduira considérablement la nécessité d'interventions manuelles lors des déploiements, assurant ainsi que la version

la plus récente de notre API soit continuellement déployée et disponible. Avec Kubernetes orchestrant le déploiement et la gestion des services, nous pouvons nous attendre à une gestion simplifiée, à des déploiements sans heurts et à une scalabilité dynamique de notre API, correspondant aux exigences actuelles et futures de charge et de disponibilité.

Avec l'intégration de Kubernetes et DockerHub, nous nous apprêtons à transformer notre processus de déploiement d'API de plusieurs façons significatives :

1. **Automatisation du déploiement** : Kubernetes sera configuré pour surveiller DockerHub et déployer automatiquement les nouvelles versions de l'API dès qu'elles sont disponibles. Cela garantit une mise à jour continue et sans interruption des services API.
2. **Mise à l'échelle efficace** : Kubernetes optimisera l'utilisation des ressources en ajustant le nombre de réplicas de l'API en fonction de la demande. Cette capacité de mise à l'échelle dynamique assure une réponse adaptée aux pics de trafic, garantissant ainsi la performance et la disponibilité de l'API.
3. **Résilience accrue** : L'architecture conçue avec Kubernetes est résiliente par nature. Elle possède des mécanismes d'auto-réparation qui redémarrent automatiquement les conteneurs en cas d'échec, et maintient la santé de l'API en vérifiant et en respectant les sondes de vivacité et de préparation.
4. **Gestion unifiée des ressources** : L'association de DockerHub pour la distribution des images et de Kubernetes pour la gestion des conteneurs nous permettra d'avoir une vue centralisée et uniforme de la gestion des ressources. Toutes les instances de l'API s'exécuteront avec des configurations et des versions de logiciel identiques, ce qui simplifie la gestion et réduit les risques d'incohérences.

Conclusion

Le POC de MedHead a validé que l'adoption de technologies ciblées et l'application de méthodes de développement agiles constituent une base solide et évolutive pour répondre aux standards du National Health Service (NHS) du Royaume-Uni. En prévoyant l'intégration de Kubernetes et DockerHub, MedHead se positionne stratégiquement pour améliorer et étendre son infrastructure technologique. Cette évolution est essentielle pour soutenir l'efficacité et la fiabilité des services de santé dans le cadre britannique, en garantissant une plateforme capable de s'adapter aux besoins changeants et de gérer efficacement les données de santé à grande échelle.