Dossier d’architecture technique

Diagrammes, décisions d'architecture et guide de démonstration

Louis ZEPHIR
MEDHEAD

Historique

Auteur	Remarques	Date	n° version
Louis ZEPHIR	Création du documents, début de rédaction	01/12/2025	1.0
Louis ZEPHIR	Mise à jour, correction orthographe	03/01/2026	*1.1*

Auteurs

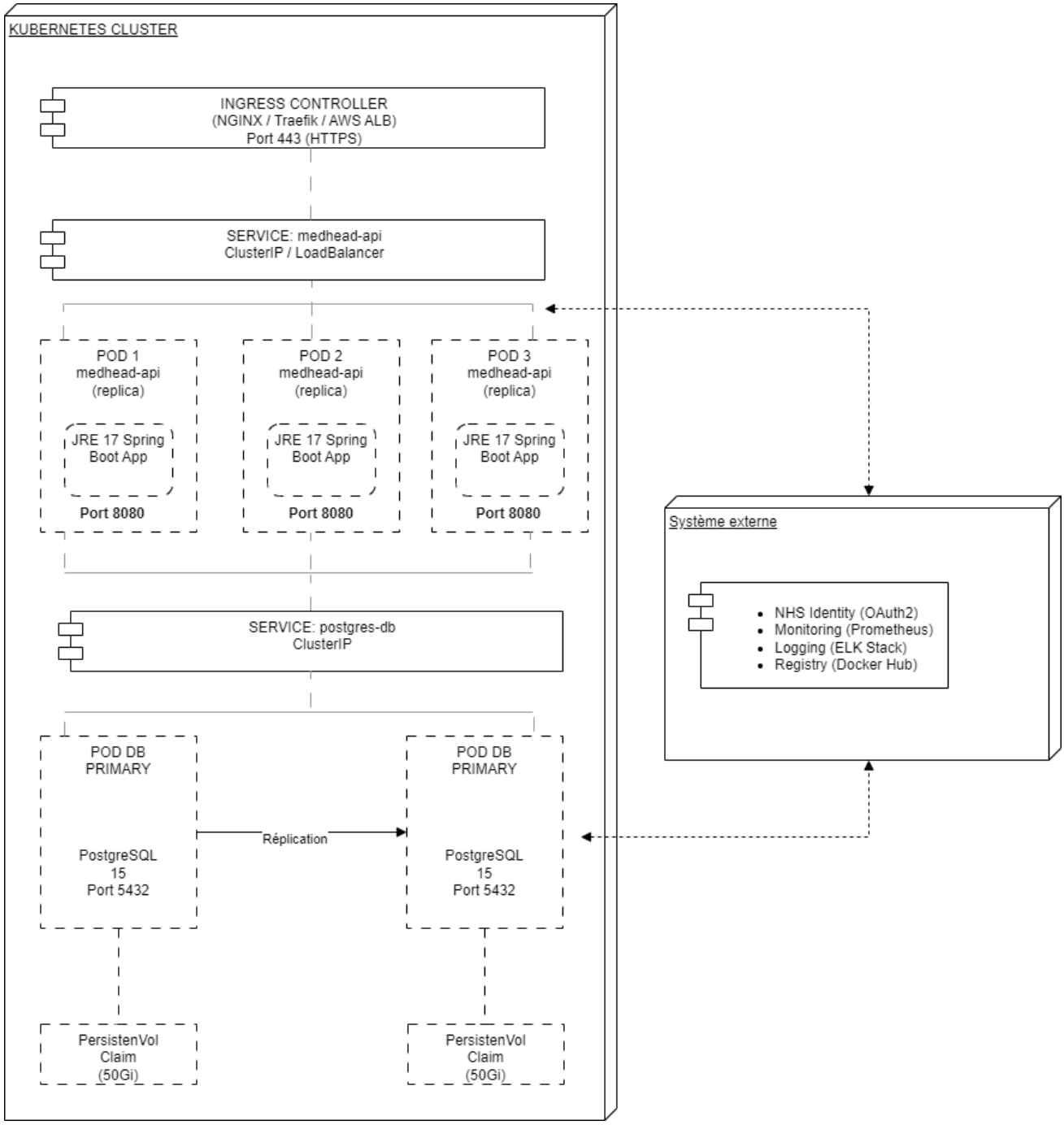
Auteur	Fonction	Contact
Louis ZEPHIR	Architecte Logiciel	louis.zephir@medhead.fr

Table des matières

Diagramme de déploiement

Le diagramme ci-dessous illustre l'architecture de déploiement cible sur Kubernetes, avec réplication de l'application et haute disponibilité de la base de données.

Vue d'ensemble du déploiement



Composants du déploiement

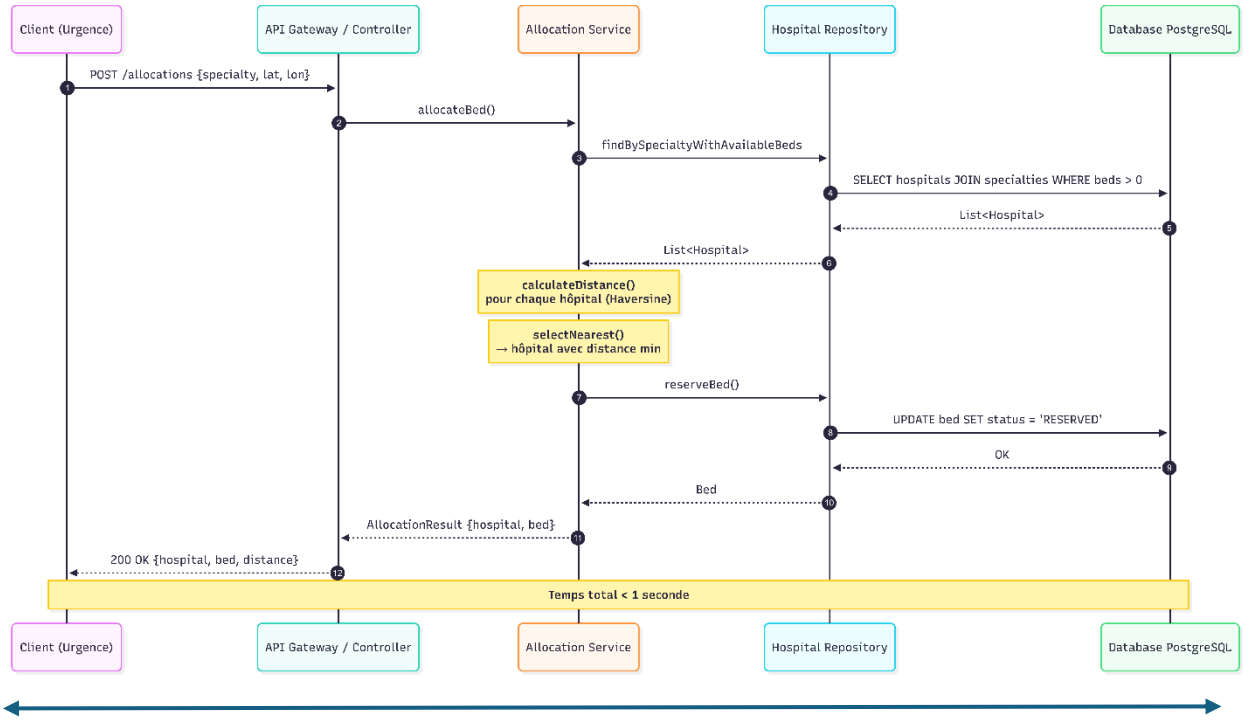
Composant	Type	Réplicas	Ressources
Ingress Controller	NGINX / AWS ALB	2	512Mi / 0.5 CPU
medhead-api	Deployment	3 (min) - 10 (max)	1Gi / 1 CPU
postgres-primary	StatefulSet	1	2Gi / 2 CPU
postgres-replica	StatefulSet	1	2Gi / 1 CPU
PersistentVolumeClaim	Storage	2 x 50Gi	SSD

Configuration réseau

- **Ingress :** Expose le service sur HTTPS (port 443) avec certificat TLS
- **Service API :** ClusterIP interne, load balancing vers les pods applicatifs
- **Service DB :** ClusterIP interne, non exposé à l'extérieur
- **Network Policy :** Seuls les pods API peuvent accéder à la base de données

Diagramme de séquence - Allocation de lit

Le diagramme de séquence ci-dessous détaille le flux d'exécution lors d'une demande d'allocation de lit, de la requête client jusqu'à la réponse (voir le diagramme).



Description des étapes

#	Étape	Composant	Durée estimée
1	Réception requête HTTP	Controller	5 ms
2	Validation des paramètres	Controller	2 ms
3	Appel service allocation	Service	1 ms
4	Requête SQL hôpitaux + spécialités	Repository	50-100 ms
5	Calcul distances (Haversine)	Service	1-5 ms
6	Sélection hôpital le plus proche	Service	< 1 ms
7	Réservation du lit (UPDATE)	Repository	20-50 ms
8	Construction réponse JSON	Controller	2 ms
	TOTAL		**< 200 ms**

Architecture Decision Records (ADR)

Les ADR documentent les décisions architecturales significatives prises durant le projet, leur contexte, les alternatives considérées et les conséquences.

ADR-001 : Choix de Spring Boot comme framework principal

Date	2025-10-01	Statut	Accepté

Contexte

Le consortium MedHead nécessite un framework robuste pour développer des microservices performants et maintenables. Le principe C3 recommande de favoriser les langages exécutables sur JVM.

Décision

Utiliser Spring Boot 3.x avec Spring Web, Spring Data JPA et Spring Actuator comme stack technique principale.

Justification

- ✓ Maturité et stabilité du framework (15+ ans d'écosystème Spring)
- ✓ Large communauté et documentation extensive
- ✓ Intégration native avec les outils de monitoring (Actuator, Micrometer)
- ✓ Support natif de la conteneurisation et du cloud
- ✓ Conformité avec le principe C3 (technologies JVM)

Alternatives considérées

- • Quarkus - Plus performant au démarrage mais écosystème moins mature
- • Micronaut - Bon compromis mais moins de ressources disponibles
- • Node.js/Express - Non conforme au principe C3 (JVM)

Conséquences

- → Équipe doit maîtriser l'écosystème Spring
- → Temps de démarrage plus long que Quarkus (acceptable pour la PoC)
- → Facilite le recrutement (compétences répandues)

ADR-002 : Choix de PostgreSQL comme base de données

Date	2025-10-05	Statut	Accepté

Contexte

Le système nécessite une base de données relationnelle fiable pour stocker les données des hôpitaux, spécialités et lits avec des contraintes d'intégrité fortes.

Décision

Utiliser PostgreSQL 15+ comme système de gestion de base de données relationnelle.

Justification

- ✓ Open source avec support entreprise disponible
- ✓ Excellentes performances sur les requêtes complexes
- ✓ Support natif des types géographiques (PostGIS) pour les extensions futures
- ✓ Fonctionnalités avancées (JSONB, indexation partielle)
- ✓ Réplication et haute disponibilité matures

Alternatives considérées

- MySQL/MariaDB - Moins de fonctionnalités avancées
- Oracle - Coût de licence prohibitif
- MongoDB - Non adapté aux contraintes d'intégrité fortes requises

Conséquences

→ Nécessite une expertise DBA pour la production
→ Configuration du cluster pour la haute disponibilité en Phase 1

ADR-003 : Architecture REST synchrone pour la PoC

Date	2025-10-10	Statut	Accepté

Contexte

Le principe B6 recommande une architecture événementielle, mais la PoC doit valider rapidement les hypothèses de performance sans complexité excessive.

Décision

Implémenter une API REST synchrone pour la PoC, avec préparation pour l'évolution vers une architecture événementielle.

Justification

✓ Simplicité de développement et de test
✓ Validation rapide des temps de réponse
✓ Principe C4 autorise l'assouplissement pour les PoC
✓ L'API REST restera nécessaire comme point d'entrée

Alternatives considérées

- Architecture événementielle complète dès le départ - Trop complexe pour la PoC
- GraphQL - Overhead non justifié pour les cas d'usage simples

Conséquences

→ Évolution vers Kafka planifiée en Phase 2
→ Design des services compatible avec l'ajout d'événements

ADR-004 : Algorithme de distance Haversine pour la géolocalisation

Date	2025-10-15	Statut	Accepté

Contexte

Le système doit sélectionner l'hôpital le plus proche d'une position GPS donnée. La précision et la performance sont critiques.

Décision

Utiliser la formule de Haversine pour calculer la distance entre deux points géographiques.

Justification

✓ Précision suffisante pour les distances < 500 km
✓ Calcul rapide (O(1) par paire de points)
✓ Pas de dépendance externe requise
✓ Standard de l'industrie pour ce type de calcul

Alternatives considérées

- Vincenty - Plus précis mais 10x plus lent
- PostGIS ST_Distance - Nécessite extension, overhead réseau
- API Google Maps - Dépendance externe, coût, latence

Conséquences

→ Implémentation en Java pure dans le service
→ Tests de validation avec cas réels (Paris, Lyon, etc.)

ADR-005 : Authentification basique pour la PoC

Date	2025-10-20	Statut	Accepté (temporaire)

Contexte

Le principe de sécurité exige OAuth2/OpenID Connect pour la production, mais la PoC doit avancer rapidement.

Décision

Implémenter une authentification HTTP Basic pour la PoC, avec migration OAuth2 planifiée en Phase 1.

Justification

✓ Principe C4 autorise l'assouplissement de la conformité pour les PoC
✓ Permet de valider les autres hypothèses sans blocage
✓ L'API est conçue pour supporter OAuth2 (stateless, tokens)

Alternatives considérées

- OAuth2 dès la PoC - Retard estimé de 2 semaines
- Pas d'authentification - Inacceptable même pour une PoC

Conséquences

→ Risque R003 documenté dans le registre des risques
→ Migration OAuth2/JWT prioritaire en Phase 1
→ Aucune exposition externe de la PoC

ADR-006 : Conteneurisation Docker avec orchestration Kubernetes cible

Date	2025-10-25	Statut	Accepté

Contexte

Le déploiement doit être reproductible, scalable et compatible avec l'infrastructure cloud du consortium.

Décision

Conteneuriser l'application avec Docker, déploiement Kubernetes prévu pour la production.

Justification

✓ Portabilité entre environnements (dev, test, prod)
✓ Scalabilité horizontale native avec Kubernetes
✓ Standard de l'industrie pour les microservices
✓ Compatible avec les offres cloud (EKS, GKE, AKS)

Alternatives considérées

- Déploiement VM classique - Moins scalable, plus de maintenance
- Serverless (Lambda) - Contraintes de temps d'exécution incompatibles

Conséquences

→ Dockerfile multi-stage pour optimiser les images
→ Helm charts à développer pour la Phase 4 (pilote)

Guide de démonstration

Ce guide décrit les scénarios de démonstration pour la soutenance, avec les commandes à exécuter et les résultats attendus.

Préparation de l'environnement

1. Démarrer les services : `docker-compose up -d`
2. Vérifier le statut : `curl http://localhost:8080/actuator/health`
3. Charger les données de test : `./scripts/load-test-data.sh`
4. Ouvrir Swagger UI : `http://localhost:8080/swagger-ui.html`

Scénarios de démonstration

DEMO-01 : Allocation de lit réussie - Cas nominal

Objectif : Démontrer le cas d'usage principal : allocation d'un lit dans l'hôpital le plus proche

Prérequis

- Base de données initialisée avec données de test
- Au moins 2 hôpitaux avec des lits disponibles en Cardiologie
- API démarrée et accessible

Étapes de démonstration

1. Afficher la liste des hôpitaux avec lits disponibles

```
GET /api/v1/hospitals
```

Résultat attendu : Liste des hôpitaux avec leurs spécialités et lits disponibles

2. Effectuer une demande d'allocation pour Cardiologie

```
POST /api/v1/allocations
{
  "specialtyCode": "CARDIO",
  "latitude": 48.8566,
  "longitude": 2.3522
}
```

Résultat attendu : Réponse 200 avec l'hôpital le plus proche et le lit réservé

3. Vérifier la mise à jour du stock de lits

```
GET /api/v1/hospitals/{id}
```

Résultat attendu : Nombre de lits disponibles décrémenté de 1

Points clés à souligner

- ★ **Temps de réponse < 1 seconde**
- ★ **L'hôpital retourné est le plus proche géographiquement**
- ★ **Le stock est mis à jour atomiquement**

DEMO-02 : Aucun lit disponible - Cas d'erreur

Objectif : Démontrer la gestion du cas où aucun lit n'est disponible

Prérequis

- Aucun lit disponible pour la spécialité Neurologie

Étapes de démonstration

1. Effectuer une demande d'allocation pour Neurologie

```
POST /api/v1/allocations
{
  "specialtyCode": "NEURO",
  "latitude": 48.8566,
  "longitude": 2.3522
}
```

Résultat attendu : Réponse 404 avec message explicatif

Points clés à souligner

- ★ **Code erreur approprié (404 Not Found)**
- ★ **Message d'erreur clair pour l'appelant**
- ★ **Aucune modification en base**

DEMO-03 : Consultation des spécialités NHS

Objectif : Démontrer l'intégration du référentiel NHS des 57 spécialités

Prérequis

- Données de référence NHS chargées

Étapes de démonstration

1. Lister toutes les spécialités

```
GET /api/v1/specialties
```

Résultat attendu : Liste des 57 spécialités avec leurs groupes

2. Filtrer par groupe de spécialité

```
GET /api/v1/specialties?group=Groupe%20chirurgical
```

Résultat attendu : Liste des spécialités chirurgicales uniquement

Points clés à souligner

- ★ **57 spécialités conformes au référentiel NHS**
- ★ **Groupes de spécialités correctement associés**

DEMO-04 : Performance sous charge

Objectif : Démontrer que le système répond en moins d'une seconde

Prérequis

- Outil de test de charge (ex: Apache JMeter, k6)
- Base de données avec volume réaliste

Étapes de démonstration

1. Lancer un test de charge avec 50 requêtes/seconde

```
k6 run --vus 50 --duration 30s load-test.js
```

Résultat attendu : P95 < 1000ms, aucune erreur

2. Afficher les métriques de performance

```
GET /actuator/metrics/http.server.requests
```

Résultat attendu : Métriques détaillées par endpoint

Points clés à souligner

- ★ **Temps de réponse P95 < 1 seconde**
- ★ **Taux d'erreur = 0%**
- ★ **Throughput stable**

DEMO-05 : Pipeline CI/CD

Objectif : Démontrer le pipeline d'intégration et déploiement continus

Prérequis

- Accès à Jenkins ou GitHub Actions
- Commit récent sur la branche develop

Étapes de démonstration

1. Montrer le déclenchement automatique du pipeline

```
git push origin develop
```

Résultat attendu : Pipeline déclenché automatiquement

2. Parcourir les étapes du pipeline

```
(Interface Jenkins/GitHub)
```

Résultat attendu : 8 étapes : Build → Tests → Quality → Docker → Deploy

3. Afficher le rapport de tests

```
(Rapport Surefire/JaCoCo)
```

Résultat attendu : 100% tests passants, 82% couverture

Points clés à souligner

- ★ **Pipeline automatisé de bout en bout**
- ★ **Rapport de couverture de code**
- ★ **Durée totale < 15 minutes**

FAQ

Cette section prépare les réponses aux questions susceptibles d'être posées.

Architecture

Q : Pourquoi avoir choisi une architecture synchrone plutôt qu'événementielle ?

R : Le principe C4 autorise l'assouplissement pour les PoC. L'architecture synchrone permet de valider rapidement les hypothèses de performance. L'évolution vers Kafka est planifiée en Phase 2, et le design actuel est compatible avec cette évolution.

Q : Comment le système gère-t-il la scalabilité pour 2000 hôpitaux ?

R : La PoC valide les performances pour un périmètre régional. La Phase 5 prévoit des tests de charge à l'échelle nationale avec auto-scaling Kubernetes, sharding de base de données si nécessaire, et mise en cache distribuée.

Q : Pourquoi ne pas avoir utilisé PostGIS pour les calculs géographiques ?

R : L'algorithme Haversine en Java est suffisant et évite l'overhead réseau. PostGIS serait pertinent si nous avions besoin de requêtes géospatiales complexes (zones, polygones), ce qui n'est pas le cas pour la sélection du plus proche.

Sécurité

Q : L'authentification basique n'est-elle pas un risque majeur ?

R : Oui, c'est documenté comme risque R003 (score 20/25). C'est une décision acceptée pour la PoC (principe C4). La migration OAuth2/JWT est la priorité n°1 de la Phase 1 et doit être terminée avant toute exposition externe.

Q : Comment les données patients sont-elles protégées ?

R : La PoC n'utilise aucune donnée patient réelle (principe C4). Toutes les données sont fictives ou anonymisées. En production, le chiffrement TLS, le chiffrement au repos et la conformité RGPD seront implémentés.

Tests

Q : Pourquoi 82% de couverture et pas 100% ?

R : La couverture de 100% n'est pas un objectif pertinent. Les 18% non couverts concernent principalement le code de configuration et les DTOs simples. Les composants critiques (services, repositories) dépassent 85%.

Q : Comment garantissez-vous la non-régression ?

R : Le pipeline CI exécute automatiquement tous les tests à chaque commit. Les tests BDD Cucumber documentent les comportements métier et servent de filet de sécurité. Le quality gate SonarQube bloque les PR non conformes.

Projet

Q : Quels sont les principaux risques pour la suite ?

R : Les 4 risques prioritaires sont : R003 (sécurité OAuth2), R002 (résilience), R010 (SPOF base de données), et R008 (scalabilité nationale). Tous ont des plans de mitigation dans la feuille de route.

Q : Quel est le planning pour la mise en production ?

R : La feuille de route prévoit 5 phases sur 15 mois : consolidation (Q1 2026), événements (Q2), observabilité (Q2-Q3), pilote régional (Q3), déploiement national (Q4 2026 - Q1 2027).