medx Healthcare system design

AWY

SUPINFO

5DESI

Table des matières

[**Introduction** 2](#_Toc211674533)

[**1.** **Architecture Globale** 3](#_Toc211674534)

[**1.1.** **Vu d’ensemble du système** 3](#_Toc211674535)

[**2.** **Composant détaillée** 4](#_Toc211674536)

[**2.1.** **Couche Application** 4](#_Toc211674537)

[**2.2.** **La couche donnée et Stockage** 8](#_Toc211674538)

[**2.3.** **La couche log et monitoring** 9](#_Toc211674539)

[**2.4.** **La couche serveur de notification** 10](#_Toc211674540)

[**3.** **Data Model et stockage** 11](#_Toc211674541)

[**3.1.** **Stratégie globale** 11](#_Toc211674542)

[**3.2.** **PostgreSQL - Bases Relationnelles** 12](#_Toc211674543)

[**3.3.** **Redis – Cache et Sessions** 12](#_Toc211674544)

[**3.4.** **S3 / MinIO – Stockage de Fichiers** 12](#_Toc211674545)

[**3.5.** **ElasticSearch – Recherche Médicale** 12](#_Toc211674546)

[**3.6.** **Message Queue – RabbitMQ / Kafka** 13](#_Toc211674547)

[**3.7.** **Sauvegarde et Reprise d’Activité (DR)** 13](#_Toc211674548)

[**4.** **Scalabilité et Performance** 13](#_Toc211674549)

[**4.1.** **Stratégies de Scalabilité** 14](#_Toc211674550)

[**4.2.** Scalabilité des Bases de Données 15](#_Toc211674551)

[**4.3.** **Cache et CDN** 16](#_Toc211674552)

[**4.4.** Optimisations Applicatives 16](#_Toc211674553)

# **Introduction**

**MedX** est une plateforme de santé innovante conçue pour faciliter l’accès aux soins médicaux grâce aux technologies numériques. Elle propose un ensemble de services intégrés visant à simplifier le parcours de soin des patients tout en soutenant les professionnels de santé dans leur pratique quotidienne.

Parmi ses principales fonctionnalités, MedX offre la possibilité de réaliser des téléconsultations médicales, permettant aux patients de consulter un médecin à distance, rapidement et en toute sécurité, sans avoir à se déplacer.

La plateforme intègre également un système de gestion sécurisée des dossiers médicaux, garantissant la confidentialité des données de santé tout en assurant leur accessibilité aux professionnels autorisés. Cela favorise une meilleure coordination entre les différents acteurs du système de santé.

Enfin, MedX permet le suivi des prescriptions médicales par les pharmacies, en assurant une transmission claire et instantanée des ordonnances, ce qui renforce la sécurité des traitements et facilite leur gestion pour les patients.

Grâce à cette approche connectée, MedX ambitionne de moderniser l’accès aux soins et d’optimiser l’expérience santé pour tous.

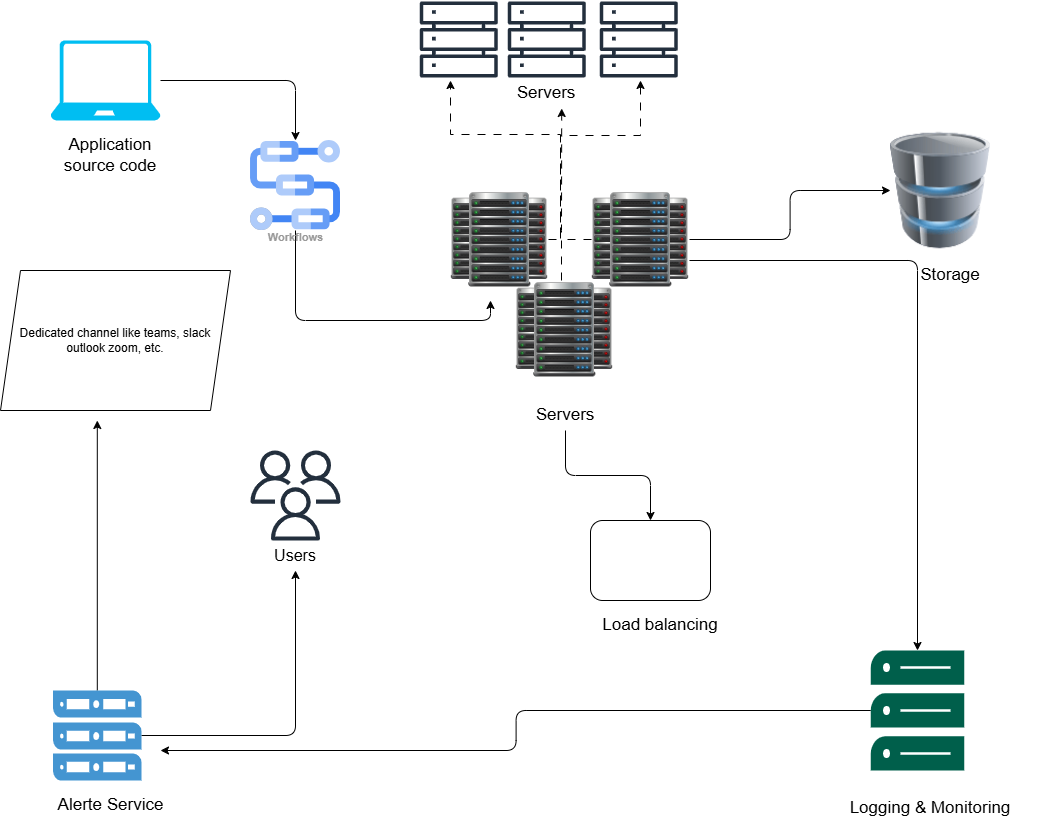
Ce document présente la solution technique complète pour MedX, incluant :

* L'architecture du système
* La conception détaillée des composants
* Les stratégies de données et stockage
* Les considérations de scalabilité et performance
* Les mesures de sécurité et conformité réglementaire.

## **Architecture Globale**

### **Vu d’ensemble du système**

Le diagramme ci-dessous présente l’application dans son ensemble, présentant ainsi comment elle fonction et les différents composant essentielles à son fonctionnement



**Figure 1**: Architecture globale de MEDX

En premier lieu, nous avons le code source de l'application, développé par l'équipe et versionné dans un repository Git, grâce à un pipeline CI/CD automatisé (via GitHub Actions et Jenkins), le code est testé, validé et déployé automatiquement sur nos serveurs de production. Les serveurs d'application ont été répliqués sur plusieurs instances pour garantir la haute disponibilité et pallier d'éventuelles interruptions de service. Un load balancer répartit intelligemment la charge entre ces serveurs. Les données fournies et produit par les utilisateurs au cours de l’utilisation du logiciel sont stockées dans une base de données sécurisée avec chiffrement. Pour la bonne gestion des erreurs, les logs sont stockés sur un serveur de monitoring dédié, séparé de la base de données principale. Un serveur de messagerie assure les notifications tant pour les développeurs (alertes techniques) que pour les utilisateurs (rappels, confirmations).

L'ensemble du système communique via des connexions chiffrées TLS pour garantir la sécurité des données médicales sensibles.

## **Composant détaillée**

### **Couche Application**

La couche application est représenté dans le diagramme ci-dessus par le « **1** » elle contient le code source de la solution, qui a suivi une découpe en micro-service. Nous avons exactement 8 microservices, dans les prochaines lignes nous tâcherons d’expliquer nous choix pour chaque microservices.

#### **User Management Service**

Cette couche est Responsable de :

* Gestion des inscriptions et profils (patients, médecins, pharmaciens)
* Authentification multi-facteurs (MFA avec TOTP)
* Autorisation basée sur les rôles (RBAC) ex :**Patient**, **Médecin,** **Pharmacien**, **Administrateur, Urgentiste**
* Gestion des sessions et token JWT

Les principales technologies utilisées sont :

* Backend : Node.js avec Express.js
* Base de données : PostgreSQL (données structurées, ACID)
* Authentification : OAuth 2.0 + JWT
* MFA : TOTP (Google Authenticator) + SMS backup

#### **Teleconsultation Scheduling Service**

Cette couche est responsable de la :

* Réservation, modification, annulation de consultations
* Gestion des disponibilités médecins
* Notifications automatiques (rappels)
* Gestion liste d'attente et priorités urgentes
* Intégration calendriers externes (Google Calendar, Outlook)

Les principales technologies utilisées sont :

* Backend : Python avec FastApi
* Base de données : PostgreSQL (intégrité transactionnelle)
* Authentification : OAuth 2.0 + JWT
* Cache : Redis pour disponibilités temps réel
* Calendrier : Intégration CalDAV / Google Calendar API

#### **Consultation Service**

Elle prend en compte la :

* Création de salles de consultation virtuelles
* Gestion streaming audio/vidéo sécurisé(WebRTC)
* Chat en temps réel pendant consultation
* Partage de documents médicaux
* Enregistrement des sessions (avec consentement)

Pour les technologies :

* Serveur média : Jitsi Meet (self-hosted) ou Twilio
* Backend : Node.js avec Socket.io pour chat
* Signaling : WebSocket sécurisé (WSS)
* Stockage enregistrements : S3 avec chiffrement (uniquement en opt-in)

En guise de mésure de sécurité les salles sont avec tokens JWT à usage unique, expiration automatique après consultation, chiffrement DTLS pour transport média, watermarking des flux vidéo (identification des captures)

#### **Patient Records Management Service**

Elle est chargée de :

* Stockage sécurisé des dossiers médicaux
* Gestion de l'historique des consultations
* Contrôle d'accès granulaire aux données patients
* Versioning des modifications
* Accès d'urgence pour professionnels autorisés
* Conformité RGPD/HIPAA

Il est attendue des patients :

* Informations démographiques (nom, date naissance, contact)
* Antécédents médicaux
* Allergies et contre-indications
* Historique des consultations
* Résultats d'examens
* Documents médicaux (images, PDF)

Pour les technologies :

* Backend : Java Spring Boot
* Base de données : PostgreSQL (données structurées)
* Stockage documents : AWS S3 avec chiffrement côté serveur
* Chiffrement : AES-256-GCM pour données au repos
* Versioning : Table d'audit avec triggers PostgreSQL

Elle a aussi des particularités comme un/une :

* **Schéma flexible** : MongoDB permet structure variable selon type de dossier
* **Auto-sauvegarde** : Intégration automatique avec Consultation Service
* **Filtrage par rôle** : Patients voient version simplifiée (non-technique)
* **Audit automatique** : Chaque accès enregistré avec timestamp et justification

#### **Prescription Management Service**

Elle est chargée de :

* Création de prescriptions par médecins
* Modification et révocation de prescriptions
* Notification automatique aux pharmacies
* Suivi du statut de délivrance
* Historique des prescriptions patient

Pour les technologies**:**

* Backend : Java Spring Boot
* Base de données : PostgreSQL
* Signature numérique : PKI avec certificats X.509
* QR codes : Bibliothèque ZXing

En guise de sécurité : les prescriptions sont non modifiables après création (sauf révocation), la signature numérique est obligatoire, audit trail complet, détection de fraudes (patterns anormaux).

#### **Pharmacy Interface Service**

Elle est chargée de :

* Confirmation réception prescriptions
* Mise à jour statuts préparation
* Vérification disponibilité stock,
* Communication avec patients

Comme techno elle utilise :

#### **Notification Service**

**Ce service fait le :**

* Envoi notifications multi-canal (SMS, Email, Push, In-app)
* Gestion préférences utilisateurs
* Tracking délivrance et lecture
* Template management

**Avec plusieurs types de notifications comme :**

* Rappels de RDV (24h, 1h avant)
* Nouvelle prescription disponible
* Prescription prête en pharmacie
* Accès d'urgence à dossier médical
* Résultats d'examens disponibles

**Et les technologies utilisées sont**

* Message Queue : RabbitMQ pour asynchrone
* Email : SendGrid / AWS SES
* SMS : Twilio
* Push : Firebase Cloud Messaging
* Backend : Node.js worker processes

#### **Billing and Payment Service**

**Cette couche est responsable du :**

* Traitement paiements consultations
* Génération factures
* Remboursements
* Intégration passerelles paiement

**Les technologies utilisées sont**

* Payment Gateway : Stripe / PayPal
* Compliance : PCI-DSS (via gateway externe)
* Base de données : PostgreSQL

### **La couche donnée et Stockage**

Cette contient toutes les ressources produites et fournit par l’utilisateur au cours de l’utilisation de MedX. Chaque microservice possède sa propre base de données pour garantir l’autonomie et déploiements indépendants, la scalabilité ciblée, l’isolation des défaillances et la flexibilité technologique. Notre approche combine **SQL et NoSQL** selon les besoins :

|  |  |  |
| --- | --- | --- |
| Type de Données | Technologie | Justification |
| Utilisateurs, authentification | PostgreSQL | ACID, relations, intégrité |
| Dossiers patients | PostgreSQL | Compliance, transactions |
| Prescriptions | PostgreSQL | Intégrité, audit |
| Documents médicaux | S3 / Azure Blob | Scalabilité, durabilité |
| Sessions consultations | Redis | Performance, TTL |
| Logs, événements | MongoDB / Elasticsearch | Flexibilité, recherche |
| Calendrier, disponibilités | PostgreSQL + Redis Cache | Cohérence + performance |

**Tableau 1:** Tableau de repartitions des bases de données entre les services

#### **Stratégie de mise en cache**

Le cache est utilisé pour alléger la charge sur les serveurs tout en garantissant une expérience de recherche fluide pour les utilisateurs. Il permet notamment de gérer :

* Les sessions utilisateurs (durée de vie : 15 min)
* Les profils utilisateurs les plus consultés (durée de vie : 30 min)
* Les disponibilités des médecins (durée de vie : 5 min)
* Le rate limiting via des compteurs par utilisateur
* La file d’attente des notifications à envoyer
* Le suivi en temps réel du statut des consultations en cours
  + 1. Chiffrement de la donnée

Afin de garantir la confidentialité des données sensibles, un mécanisme de chiffrement robuste est mis en place pour les documents.

Chiffrement des documents :

* Algorithme utilisé : AES-256-GCM
* Gestion des clés : via AWS KMS ou Azure Key Vault
* Rotation des clés : automatique tous les 90 jours
* Chiffrement en enveloppe (Envelope Encryption) :
  + Une clé de chiffrement des données (DEK) est générée de manière unique pour chaque fichier
  + La DEK est ensuite chiffrée à l’aide d’une clé maître (Master Key) stockée dans le KMS

### **La couche log et monitoring**

La solution intègre un serveur dédié à la collecte et au stockage centralisé de l’ensemble des **logs**, traces et métriques générés lors de l’utilisation de la plateforme. Ce serveur distinct assure une gestion centralisée, souveraine et sécurisée des données de monitoring.

Pour l’exploitation de ces données, le choix s’est porté sur Prometheus pour la collecte des métriques, associé à Grafana pour la visualisation à travers des tableaux de bord dynamiques et personnalisables. Ce dispositif facilite le suivi de la performance, l’observabilité du système, ainsi que la détection proactive d’anomalies.

L’analyse des données collectées permet notamment :

* D’identifier rapidement les points de congestion (bottlenecks),
* D’assurer un suivi en temps réel de l’état de santé de la solution,
* Et de définir des règles d’alerte afin de déclencher des notifications en cas de comportement anormal.

### **La couche serveur de notification**

La solution dispose d’un serveur dédié aux notifications, chargé de la gestion centralisée de tous les messages entrants et sortants liés aux événements du système.

Ce serveur remplit deux fonctions principales :

* Notifications techniques (monitoring et alertes)  
  En lien direct avec le serveur de logs, il reçoit les événements critiques ou anormaux détectés (via Prometheus ou tout autre moteur d’alerte). En fonction de la gravité ou du type d’événement, il déclenche automatiquement l’envoi de notifications aux équipes techniques via les canaux configurés (email, Slack, webhook, teams, zoom etc.).
* Notifications utilisateurs  
  Il est également utilisé pour notifier les utilisateurs finaux (patients, médecins, administrateurs) à travers différents canaux (email, SMS, push, etc.), pour des événements comme :
  + Rappels de rendez-vous
  + Modifications de créneaux
  + Disponibilité de documents
  + Statuts de consultation

Cette séparation entre le moteur de détection (serveur de logs) et le moteur d’émission (serveur de notifications) permet une architecture plus modulaire, scalable et résiliente, tout en facilitant le suivi et la traçabilité des messages envoyés.

**ICI je vais mettre le schéma détaillé alliant local et cloud.**

## **Data Model et stockage**

Au regard de la sensibilité de la solution un accent particulier a été mis sur le modèle de donnée et la politique de stockage. Dans les lignes qui suivent nous tâcheront de vous expliquer le dispositions prises pour assurer cette partie.

### **Stratégie globale**

L’architecture suit le principe « Database per Service » : chaque microservice dispose de sa propre base de données.  
Cette approche permet :

* Une **autonomie** complète des services,
* Une **scalabilité ciblée** en fonction des besoins,
* Une **isolation des défaillances**,
* Une **flexibilité technologique** accrue.

Le tableau ci-dessous montre comment la base de données est découpée :

|  |  |  |
| --- | --- | --- |
| Type de données | Technologie de stockage | Justification |
| Utilisateurs, authentification | PostgreSQL | Garantit la cohérence des données (ACID), la gestion des relations et l’intégrité des identifiants. |
| Dossiers patients | PostgreSQL | Assure la conformité réglementaire et la fiabilité des transactions médicales sensibles. |
| Prescriptions | PostgreSQL | Offre traçabilité, intégrité et auditabilité des ordonnances. |
| Documents médicaux | S3 / Azure Blob Storage | Fournit une grande capacité de stockage, haute disponibilité et durabilité des fichiers volumineux. |
| Sessions de consultation | Redis | Stockage en mémoire performant avec durée de vie limitée (TTL) pour les données éphémères. |
| Logs et événements | MongoDB / Elasticsearch | Flexibilité pour les données semi-structurées et moteur de recherche rapide pour l’analyse. |
| Calendrier et disponibilités | PostgreSQL + Redis (cache) | Combinaison entre cohérence transactionnelle (PostgreSQL) et rapidité d’accès (Redis). |

**Tableau 2**: tableau de gestion des données

### **PostgreSQL - Bases Relationnelles**

Les données relationnelles sont structurées selon la 3ᵉ Forme Normale (3NF) afin de limiter les redondances et d’assurer l’intégrité référentielle. Confère Annexe 1 pour le mcd et MLD

* **Base Utilisateurs** : stockage des comptes, profils, rôles et préférences.
* **Base Planification** : gestion des rendez-vous, disponibilités et listes d’attente.
* **Base Dossiers Médicaux** *(chiffrée)* : stockage sécurisé des diagnostics, traitements, prescriptions et consentements.
* **Base Facturation** : gestion des factures et transactions.

Des index stratégiques (clés étrangères, composites, partiels et BRIN pour les dates) optimisent les performances des requêtes et la scalabilité.

### **Redis – Cache et Sessions**

Redis est utilisé comme cache en mémoire pour réduire la charge sur la base relationnelle et accélérer les temps de réponse.  
Il stocke notamment :

* Les sessions utilisateurs,
* Les profils et disponibilités fréquemment consultés,
* Les compteurs de limitation de débit (*rate limiting*),
* Les statuts de consultations en temps réel.

Cette stratégie réduit la latence de 100 ms à environ 2 ms et diminue jusqu’à 70 % la charge sur la base principale.

### **S3 / MinIO – Stockage de Fichiers**

Les fichiers volumineux (imageries, rapports, ordonnances, pièces jointes) sont stockés dans des buckets S3/MinIO chiffrés (AES-256).

* Chaque fichier est protégé par une clé unique,
* Les accès se font via des URL présignées valables 15 minutes,
* Une réplication inter-région assure la haute disponibilité,
* Aucune donnée n’est conservée en clair.

### **ElasticSearch – Recherche Médicale**

Pour les besoins d’analyse et de recherche, un index anonymisé est maintenu dans ElasticSearch.  
Seules des données non identifiantes (hash patient, code diagnostic, symptômes, tranche d’âge, genre) y sont stockées.  
Cela permet d’effectuer des recherches et statistiques médicales sans exposer de données personnelles.

### **Message Queue – RabbitMQ / Kafka**

Un système de messagerie asynchrone est utilisé pour découpler les traitements et fluidifier les échanges entre services.  
Des échanges sont définis pour :

* Les notifications (email, SMS, push),
* Les prescriptions (création, mise à jour, dispensation),
* Les rendez-vous (planification, annulation, clôture),
* Les audits de sécurité et d’accès.

### **Sauvegarde et Reprise d’Activité (DR)**

Une politique de sauvegarde rigoureuse est mise en place :

* **PostgreSQL** : sauvegardes complètes quotidiennes, incrémentales toutes les 6 h, journaux de transactions, rétention 30 jours en ligne et 7 ans hors ligne.
* **Redis** : snapshots horaires et fichiers AOF, rétention 7 jours.
* **S3** : versioning, réplication inter-région, archivage Glacier après 90 jours.
* **Logs et audit** : stockage immuable pendant 7 ans.

En cas d’incident majeur, un **plan de reprise d’activité** (PRA) est prévu :

* **RTO** (Recovery Time Objective) : 4 heures
* **RPO** (Recovery Point Objective) : 15 minutes
* Basculement automatique vers une région secondaire (mode actif–passif),
* Tests de basculement réalisés trimestriellement.

## **Scalabilité et Performance**

L’architecture est conçue pour assurer une faible latence, une haute disponibilité, ainsi qu’une capacité d’évolution fluide avec la croissance de la plateforme, le tableau ci-dessous montre les résultats attendus :

|  |  |  |
| --- | --- | --- |
| Métrique | Cible | Critique |
| Latence API (P95) | < 200 ms | < 500 ms |
| Disponibilité | 99,9 % | 99,5 % |
| Débit | 10 000 RPS | 50 000 RPS |
| Latence vidéo | < 100 ms | < 300 ms |

**Tableau 3:** tableau des métriques

### **Stratégies de Scalabilité**

* + 1. **Scalabilité Horizontale (Scaling Out)**

L’ensemble des microservices est dimensionné pour **s’adapter dynamiquement à la charge** grâce à l’**auto-scaling Kubernetes**., elle assure un scale up immédiat si seuil dépassé, un scale down progressif avec 5 minutes de stabilisation et avec un load balancing : algorithme *least-connections* pour les services temps réel. :

|  |  |  |  |
| --- | --- | --- | --- |
| Service | Min Pods | Max Pods | Déclencheur de scaling |
| User Management | 3 | 20 | CPU 70 % ou 1000 RPS |
| Scheduling | 5 | 30 | CPU 70 % ou 2000 RPS |
| Consultation | 10 | 50 | 1000 connexions actives |
| Patient Records | 5 | 25 | CPU 80 % ou latence requêtes |
| Prescription | 3 | 15 | CPU 70 % |
| Notification | 5 | 40 | 10 000 messages en file |
| Billing | 3 | 10 | CPU 70 % |

Tableau 4: Tableau Representatif de Kubernetes

#### Scalabilité Verticale (Scaling Up)

Le Scalabilité Verticale (Scaling Up)fait référence à tout ce qui est monté en charge du hardware comme les serveurs, les VM, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| Service | Charge normale | Pic de charge | Justification |
| User Management | 2 vCPU / 4 Go RAM | 4 vCPU / 8 Go RAM | Opérations légères |
| Patient Records | 4 vCPU / 8 Go RAM | 8 vCPU / 16 Go RAM | Traitement et chiffrement intensifs |
| Consultation | 4 vCPU / 8 Go RAM | 8 vCPU / 16 Go RAM | Streaming vidéo |
| Notification | 2 vCPU / 4 Go RAM | 4 vCPU / 8 Go RAM | Workers asynchrones |

Tableau 5: tableau de scaling up

### **Scalabilité des Bases de Données**

#### **Réplication et Répartition des Charges**

Le système de base de données repose sur une architecture maître – réplicas. Toutes les opérations d’écriture sont dirigées vers une instance primaire (maître), tandis que les opérations de lecture sont réparties entre quatre réplicas grâce à un répartiteur de charge.

La réplication asynchrone permet de propager les données de la base primaire vers les réplicas en quasi-temps réel. En pratique, 90 % des requêtes de lecture sont traitées par les réplicas, ce qui réduit considérablement la charge sur la base principale et améliore la scalabilité globale.

En cas de défaillance de l’instance primaire, un mécanisme de basculement automatique (failover) peut promouvoir un réplica en nouveau maître, assurant ainsi la haute disponibilité et la continuité de service. Grâce à cette approche nous gagnons :

* Répartition de la charge : les lectures sont déléguées aux réplicas, libérant la base primaire pour les écritures.
* Amélioration des performances : temps de réponse réduit grâce à la parallélisation des accès.
* Haute disponibilité : le basculement automatique garantit une résilience face aux pannes.
* Scalabilité horizontale : possibilité d’ajouter facilement de nouveaux réplicas selon la croissance du trafic.
* Maintenance facilitée : certaines opérations lourdes (reporting, exports, analyses) peuvent être exécutées sur les réplicas sans impacter la production.

#### Partitionnement et Archivage

|  |  |  |
| --- | --- | --- |
| Table | Partitionnement | Objectif |
| audit\_logs | Mensuel (par date) | Accès rapide et nettoyage simplifié |
| Consultations | Annuel | Volumes importants |
| patient\_records | Archivage + cold storage | Alléger la base active (données > 10 ans) |

Tableau 6: Tableau de Partionnement et d'archivage

#### Optimisations Techniques

Pour optimiser les performances de la base de données et garantir sa stabilité à grande échelle, plusieurs mécanismes d’optimisation ont été mis en place.

Tout d’abord, l’utilisation de PgBouncer permet de mettre en place un pool de connexions. Ce mécanisme limite le nombre de connexions réelles à la base (25) tout en pouvant gérer jusqu’à 500 connexions client simultanées, ce qui réduit considérablement la surcharge liée à la création et à la fermeture de connexions. Les requêtes les plus courantes sont optimisées grâce à l’utilisation de prepared statements, permettant d’éviter une recompilation systématique des requêtes et d’améliorer les temps de réponse.

Des index couvrants et des index BRIN sont également mis en place, notamment sur les colonnes temporelles, afin d’accélérer les recherches et de réduire le coût des scans sur de larges volumes de données.Enfin, l’exécution d’un VACUUM automatique assure la maintenance régulière des tables et évite la fragmentation, garantissant ainsi de bonnes performances dans la durée.

### **Cache et CDN**

#### **Cache Redis**

Afin d’illustrer la stratégie de cache mise en place et les durées de conservation associées, le tableau ci-dessous présente les principaux cas d’usage ainsi que les TTL configurés pour chaque type de donnée :

|  |  |  |
| --- | --- | --- |
| Type de données | TTL | Invalidation |
| Profil utilisateur | 1 h | Modification profil |
| Disponibilités médecin | 5 min | Nouvelle réservation |
| Liste prescriptions actives | 15 min | Nouvelle prescription |
| Résultats de recherche | 10 min | — |
| Sessions JWT blacklist | 15 min | Logout |

Tableau 7: tableau justificatif du scaling de redis

#### **CDN (Content Delivery Network)**

Pour la distribution des contenus statiques tels que les images, feuilles de style CSS et fichiers JavaScript, un CDN CloudFront / Cloudflare est utilisé. Les assets versionnés bénéficient d’un cache d’une durée d’un an grâce aux en-têtes HTTP appropriés, ce qui améliore significativement les performances de chargement.  
De plus, une compression automatique (Gzip / Brotli) est appliquée afin de réduire la taille des fichiers transférés et d’optimiser la bande passante. Enfin, une invalidation automatique est déclenchée à chaque déploiement, garantissant que les utilisateurs disposent toujours de la dernière version des ressources.

### **Optimisations Applicatives**

#### **Asynchrone et File de Messages**

Afin d’optimiser le traitement des tâches asynchrones et de garantir une meilleure répartition de la charge, plusieurs files de messages ont été configurées avec des priorités et des workers dédiés. Le tableau ci-dessous présente la configuration adoptée pour chaque type de file.

|  |  |  |  |
| --- | --- | --- | --- |
| Queue | Priorité | Workers | Cas d’usage |
| notifications.email | Medium | 5 | Envoi d’e-mails |
| notifications.sms | High | 3 | Envoi de SMS |
| notifications.push | Medium | 3 | Notifications push |
| audit.logs | High | 2 | Écriture logs audit |
| prescriptions.sync | Medium | 2 | Synchronisation pharmacies |

Tableau 8: Tableau d'utilisation des queues

#### **Pagination Lazy Loading et Compresssion**

La pagination côté API consiste à découper un grand volume de résultats retournés par une requête en séquences de taille fixe (par exemple 15 éléments) avant de les envoyer à l’interface utilisateur. Cette approche permet de réduire significativement le temps de chargement et d’optimiser l’utilisation des ressources. Par ailleurs, la stratégie de lazy loading a été mise en place à la fois côté backend et frontend. Côté serveur, elle permet d’éviter les requêtes N+1 vers la base de données en ne chargeant que les données réellement nécessaires. Côté client, elle permet de charger dynamiquement les modules et composants React au moment où ils sont requis, allégeant ainsi le bundle initial. Enfin, l’utilisation du infinite scroll permet un chargement progressif des données au fil de la navigation, tandis que les skeleton screens améliorent l’expérience utilisateur en donnant un retour visuel immédiat pendant le chargement des contenus.

Ex : quand on fait GET /api/records/patient/{id}/history?page=1&limit=20

On obtiendra :

{

"data": [...],

"pagination": {

"current\_page": 1,

"total\_pages": 15,

"total\_items": 287,

"items\_per\_page": 20,

"next\_page": 2,

"prev\_page": null

}

}

Ensuite les réponses API sont compressées en Gzip offrant une réduction de 70% de la taille des données transférées. De plus, les WebSocket utilisent l’extension permessage-deflate réduisant ainsi la bande passante lors d’une communication en temps réel. Les images sont Converties en WebP qui sont 0,3 fois plus légère que JPEG rendant le chargement des pages plus rapide. Les vidéos de consultations utilisent codecs VP9/ H.2.64 adaptatifs en fonction de la bande passante disponible

#### **Monitoring et Alerting**

Pour la partie Monitoring, au vu du serveur dédié utilisée pour stocker les logs et les métriques, Prometheus est utilisé pour la collecte des métriques, Grafana pour la visualisation de lasanté globale du système, la performance par service, les métriques business (consultations/h, paiements), les métriques sécurité (tentatives connexion, accès refusés) et ELK pour les centraliser. Le tableau ci-dessous montre les seuils mis de tolérance acceptée, au-delà de cela une alerte est envoyée à l’équipe.

|  |  |
| --- | --- |
| Métrique surveillée | Seuil d’alerte |
| Latence API (P95) | > 500 ms |
| Taux d’erreurs (4xx/5xx) | > 1 % |
| CPU / RAM | > 80 % |
| Taux de cache hit | < 80 % |
| Temps réponse DB | > 300 ms |

Tableau 9: tableau des métriques

* Alertes **Critical** : DB down, échec paiement, service indisponible.
* Alertes **Warning** : CPU élevé, latence élevée.
* Alertes **Info** : cache hit faible, backup échoué.

#### **Capacity Planning**

Le tableau ci-dessous présente les prévisions de croissance de la plateforme MedX sur cinq ans, en indiquant le nombre d’utilisateurs, les utilisateurs actifs quotidiens (DAU), le nombre de consultations quotidiennes et le pic de requêtes par seconde (RPS), ainsi que les principales mesures d’infrastructure prévues pour accompagner cette croissance.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Année | Utilisateurs | DAU | Consultations/jour | Peak RPS |
| An 1 | 100 K | 20 K | 5 000 | 500 |
| An 3 | 1 M | 200 K | 50 000 | 5 000 |
| An 5 | 5 M | 1 M | 250 000 | 25 000 |

Tableau 10: Tableau prévisionnel de L'utilisation de MEDX

## **Sécurité et conformité**