RAPPORT D'ALTERNANCE

Industrialisation et Optimisation d'une usine logicielle cloud-native chez GE HealthCare

Réalisé par DASSI MANUEL

Sous la supervision de :

Référent Institut :

M. Marc BENNATAR

Référent Entreprise :

M. Abdelghani Achibane

Année Académique

2024-2025

Fiche de confidentialite

Contents

Fait à Buc, le …….

Mr Abdelghani Achibane

Dassi Manuel

Remerciements

Je souhaite exprimer ma profonde gratitude à toutes les personnes qui m'ont accompagné et soutenu tout au long de cette année d'alternance, sur les plans professionnel et personnel.

Je remercie tout d'abord M. Abdelghani Achibane, mon référent en entreprise chez GE Healthcare, pour son encadrement rigoureux, sa disponibilité et la confiance qu'il m'a accordée. Son expertise et ses conseils ont été déterminants dans ma progression technique.

Je remercie également M. Marc Bennatar, mon référent pédagogique à l'institut PMN, pour son accompagnement constant et la qualité de ses conseils tout au long de mon cursus.

J'adresse aussi ma reconnaissance à l'ensemble de l'équipe de GE Healthcare pour leur accueil, leur esprit d'équipe et leur professionnalisme. Leur environnement de travail bienveillant et stimulant m'a permis de développer mes compétences dans les meilleures conditions.

Un remerciement particulier à mes collègues pour leur soutien quotidien, leur patience et leur bonne humeur. Grâce à leurs échanges et à leur esprit collaboratif, cette expérience a été à la fois formatrice et enrichissante.

Enfin, je remercie ma famille et mes proches pour leur soutien indéfectible et leurs encouragements, sans lesquels cette aventure n'aurait pas été possible.

Introduction

Cette alternance m'a offert l'occasion de mettre en pratique les enseignements de mon Master à l'Institut PMN dans un environnement exigeant et concret. Réalisée au sein de GE HealthCare, acteur majeur des technologies médicales, elle a porté sur l'industrialisation et l'optimisation d'une « usine logicielle » orientée cloud. L'objectif est simple à énoncer, mais ambitieux à réaliser: livrer plus rapidement des logiciels de qualité, fiables et sûrs, tout en respectant les exigences élevées du secteur de la santé en matière de sécurité, de conformité et de protection des données.

Le contexte dans lequel s'inscrit ce travail est marqué par des attentes fortes des établissements de santé et des professionnels: disposer d'outils numériques qui améliorent la prise en charge des

patients, fluidifient les parcours de soins et soutiennent les décisions cliniques. Parallèlement, le cadre réglementaire se renforce, la sensibilité des données augmente et les organisations doivent faire preuve d'une grande rigueur dans la manière de concevoir, de tester, de déployer et d'exploiter leurs applications. Dans ce paysage, la capacité à industrialiser la production logicielle devient un levier stratégique: elle permet d'accélérer l'innovation sans compromettre la qualité ni la confiance.

Par « usine logicielle », on entend un ensemble cohérent de pratiques, d'outils et de règles de fonctionnement qui structurent tout le cycle de vie des applications: de la conception au déploiement, de l'exploitation au retour d'expérience. Une usine efficace repose sur des méthodes de travail standardisées, une automatisation poussée des étapes répétitives, des contrôles qualité présents à chaque étape, et des mécanismes de retour en arrière rapides en cas d'imprévu. L'approche orientée cloud renforce ces objectifs: elle favorise l'élasticité (adapter les ressources à la demande), la réactivité (déployer des évolutions plus fréquemment) et la résilience (tolérer les pannes et y remédier rapidement), tout en permettant une maîtrise plus fine des coûts.

Au cœur de la démarche se trouve la recherche d'un équilibre entre plusieurs exigences souvent perçues comme contradictoires. Il s'agit de livrer vite, sans sacrifier la qualité; d'innover, sans prendre de risques disproportionnés; de standardiser, tout en gardant la souplesse nécessaire pour répondre à des besoins spécifiques. Concrètement, cela signifie simplifier et fiabiliser les enchaînements d'étapes qui vont de l'idée jusqu'à la mise à disposition d'une fonctionnalité; rendre plus visible et plus compréhensible l'état réel des applications à chaque instant; instaurer des points de contrôle réguliers et transparents; et documenter des modes opératoires clairs, partagés par tous.

Mon alternance s'est concentrée sur l'amélioration de ces fondations. J'ai contribué à la clarification des étapes clés et à leur enchaînement, afin de réduire les zones d'ombre et d'éviter les retards. J'ai participé au renforcement des contrôles qualité tout au long du cycle, pour détecter plus tôt les anomalies, limiter les reprises tardives et sécuriser la mise en service. J'ai travaillé à rendre le suivi des applications plus lisible, ce qui permet aux équipes d'identifier rapidement les points d'attention et d'agir avec efficacité. Enfin, j'ai accompagné l'optimisation de l'usage des ressources du cloud, avec une attention particulière portée à la sobriété, à la maîtrise budgétaire et à la capacité à absorber des pics d'activité.

Au-delà des aspects techniques, la réussite d'une usine logicielle est d'abord une question d'organisation et de culture. Les équipes de développement, de qualité, de sécurité, d'exploitation et les métiers doivent partager des objectifs, des standards et un langage commun. Cela suppose des rituels réguliers, des revues croisées, des responsabilités clairement définies et une gouvernance qui tranche rapidement quand c'est nécessaire. L'adhésion collective à des règles simples et compréhensibles est un facteur déterminant: elle garantit la répétabilité des bonnes pratiques et facilite l'intégration de nouveaux projets ou de nouvelles équipes.

La dimension humaine est également centrale. Une usine logicielle performante ne se décrète pas: elle s'apprend et se vit au quotidien. La formation, l'accompagnement au changement et la diffusion des retours d'expérience sont essentiels pour ancrer durablement les nouvelles façons de travailler. Dans le cadre de mon alternance, j'ai contribué à la mise à disposition de guides pratiques, de modèles prêts à l'emploi et de supports de communication interne, afin d'aider les équipes à adopter des gestes communs et à gagner en autonomie.

Un autre enjeu majeur réside dans la capacité à mesurer les progrès. Pour rester à un niveau d'exigence élevé, il faut des indicateurs simples, lisibles et partagés. Le temps nécessaire pour livrer une évolution, le taux d'incidents après une mise en service, la fréquence des retours en arrière, la satisfaction des utilisateurs internes et externes: autant de repères concrets qui permettent de piloter l'amélioration continue. Ces mesures ne doivent pas être perçues comme des contraintes supplémentaires, mais comme des outils d'alignement et de pilotage: elles aident à décider où concentrer l'effort, à arbitrer entre rapidité et prudence, et à objectiver les résultats.

Dans le secteur de la santé, la question de la confiance est omniprésente. Elle repose sur la qualité des produits, la protection des données et la transparence des processus. Industrialiser et optimiser une usine logicielle, c'est aussi rendre cette confiance visible: décrire comment une fonctionnalité est conçue, testée, et mise en service; expliquer comment on s'assure qu'elle ne dégrade pas l'existant; montrer comment on surveille son comportement dans le temps; et préciser comment on réagit lorsqu'un imprévu survient. Cette transparence, tournée vers l'interne autant que vers l'externe, contribue à créer un cadre de travail serein et responsable.

Le périmètre de ce rapport est volontairement centré sur les aspects structurants et les résultats observés, sans entrer dans des détails trop techniques. Il présentera d'abord le contexte et les missions menées chez GE HealthCare, en mettant en lumière les besoins identifiés et les objectifs fixés. Il décrira ensuite les choix d'organisation et de fonctionnement de l'usine logicielle orientée

cloud: principes de standardisation, automatisation des étapes clés, règles de contrôle qualité, modes de déploiement progressifs et dispositifs de suivi. Il abordera également la question de la maîtrise des coûts et de la performance, ainsi que l'attention portée à la sobriété des ressources et à la capacité d'adaptation face aux variations d'activité.

Le rapport s'attachera enfin à rendre compte des effets concrets de cette démarche: réduction des délais de mise à disposition, meilleure stabilité lors des mises en service, diminution des interventions d'urgence, clarté accrue dans le suivi des applications, appropriation des bonnes pratiques par un plus grand nombre d'équipes. Il évoquera les limites rencontrées — qu'il s'agisse de contraintes de calendrier, de dépendances externes ou de la nécessaire montée en compétences — et proposera des pistes d'amélioration pour prolonger l'effort engagé: consolidation des standards, élargissement de la couverture des pratiques communes, approfondissement de la mesure, et poursuite de la sensibilisation des équipes.

En définitive, l'industrialisation et l'optimisation d'une usine logicielle orientée cloud chez GE HealthCare visent à concilier deux ambitions qui se renforcent mutuellement: gagner en vitesse et en fiabilité dans la livraison des logiciels, et élever le niveau de confiance, de sécurité et de conformité attendu dans le domaine de la santé. Cette alternance m'a permis d'apporter une contribution concrète à cette trajectoire, d'en observer les bénéfices et d'en mesurer les exigences. Les constats et enseignements présentés dans ce rapport ont pour vocation d'éclairer la suite du chemin: consolider ce qui fonctionne, corriger ce qui doit l'être, et poursuivre une amélioration continue au service des patients, des professionnels et des équipes qui conçoivent les solutions de demain.

1. Contexte de l’aternance

1.1. L' entreprise, le service, le poste L' entreprise

GE HealthCare est une entreprise mondiale de technologie médicale, issue de l’héritage de General Electric et officiellement organisée comme entité dédiée à la santé depuis le début du 19e siècle. D’abord reconnue pour ses équipements d’imagerie médicale, notamment les premiers systèmes de radiographie, elle s’est imposée au fil des décennies comme un pionnier de l’innovation au service des patients et des professionnels de santé.

Aujourd’hui, GE HealthCare est présente dans plus de 160 pays et propose des solutions de pointe en imagerie, diagnostic, outils d’aide à la décision, intelligence artificielle et systèmes connectés pour les établissements de santé. Son ambition est de faciliter l’accès aux soins, d’améliorer la précision des diagnostics et d’optimiser les parcours de santé grâce à la technologie.

1.1.1. Structure d’accueil

GE HealthCare opère à l'échelle internationale, avec une histoire de plus d'un siècle au service des patients et des soignants. L'entreprise compte plus de 51 000 employés à travers le monde et s'affirme comme un acteur majeur du secteur, grâce à un portefeuille couvrant l'imagerie, les solutions numériques et les plateformes cloud dédiées à la santé.

En 2024, GE HealthCare a réalisé un chiffre d'affaires d'environ 18 milliards de dollars, témoignant de sa solidité économique et de sa capacité d'investissement dans des domaines clés tels que l'intelligence artificielle, le cloud médical et l'imagerie avancée.

Figure 1 - Carte des sieges Ge healthCare en France

1.1.2. Quelques chiffres et repères

- Présence dans plus de 160 pays.
- Plus de 51 000 collaborateurs.
- Environ 18 milliards USD de revenus annuels (2024).
- Partenariats technologiques avec de nombreux hôpitaux publics et privés à travers le monde.

- Investissements stratégiques dans l'IA, le cloud et les solutions logicielles pour la santé.

1.1.3. Le service et le poste

- Département dédié au développement d'applications médicales innovantes.
- Une équipe de plus de 200 développeurs impliqués dans la conception, la réalisation et l'évolution des solutions.
- Contribution à la création de logiciels permettant de visualiser et d'analyser des données médicales complexes afin de soutenir la prise de décision clinique.
- Intégration au sein de deux équipes complémentaires, avec des missions couvrant l'amélioration continue des produits, la qualité et la fiabilité des livraisons, ainsi que la collaboration avec les parties prenantes (produit, qualité, sécurité et opérations).

1.2. L'equipe associé – GE Healthcare

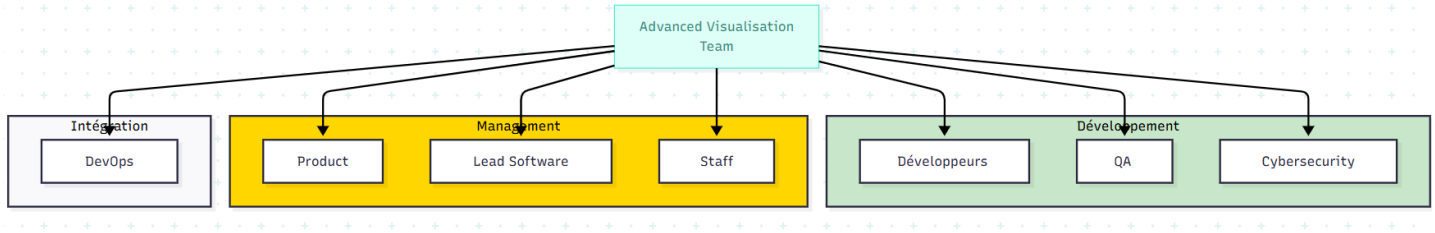

Figure 2 - Organigramme de l'equipe AW

J'ai été intégré à l'Advanced Visualisation Team, une équipe structurée autour de trois pôles très complémentaires: Integration (axé sur DevOps), Management (Product, Lead Software, Staff) et Development (Développeurs, QA, Cybersecurity). Cette organisation claire m'a immédiatement

donné des repères: qui porte la vision, qui arbitre les priorités, qui réalise, qui vérifie, et qui veille à la sécurité. Dès mes premières semaines, j'ai pu circuler entre ces pôles, comprendre leur rôle, et surtout identifier la façon dont ils s'articulent au quotidien pour faire avancer nos produits.
Côté Integration, j'ai trouvé un point d'appui essentiel pour tout ce qui touche à la préparation et à la fluidité des livraisons. L'activité DevOps y joue un rôle de "courroie de transmission" entre l'idée et la mise à disposition: ils installent les fondations, automatisent ce qui peut l'être, et cherchent à rendre les passages d'une étape à l'autre plus simples et plus fiables. Concrètement, c'est avec eux que j'ai réglé des questions très pratiques: comment organiser les enchaînements de construction et de livraison, comment réduire les points de friction, comment rendre nos enchaînements plus prédictibles. Leur approche m'a appris à regarder le flux de bout en bout: ce n'est pas seulement "faire" une fonctionnalité, c'est penser dès le départ à son trajet complet jusqu'aux utilisateurs, puis à sa vie une fois en service.

Le pôle Management réunit trois fonctions qui ont été mes repères pour cadrer les sujets. Avec le Product, j'ai travaillé sur la clarification des besoins et la priorisation. Ces échanges m'ont aidé à distinguer ce qui était "indispensable maintenant" de ce qui pouvait attendre, et à replacer chaque demande dans une logique de valeur pour les utilisateurs finaux. Le Lead Software m'a apporté un regard de cohérence: s'assurer que les orientations techniques restent alignées avec les choix structurants de l'équipe, éviter les impasses, garder de la simplicité lorsqu'elle est possible. Enfin, le Staff a été le point d'orgue de l'organisation quotidienne: préparation des jalons, coordination entre équipes, gestion des dépendances. Grâce à ce trio, je savais à qui m'adresser selon la nature du sujet: la finalité avec le Product, la cohérence avec le Lead Software, le rythme et la coordination avec le Staff.

Le pôle Development rassemble les Développeurs, la QA et la Cybersecurity. Avec les Développeurs, les interactions étaient continues: relectures, découpages de sujets, clarification des hypothèses, gestion des retours. L'ambiance était tournée vers la résolution concrète des problèmes: chacun cherche à simplifier, à rendre le travail du suivant plus facile, à réduire le nombre d'allers-retours inutiles. La QA a joué un rôle décisif pour garder la qualité au centre: vérifications régulières, retours factuels, recherche d'une couverture de tests suffisante pour éviter les surprises lors des mises à disposition. De son côté, la Cybersecurity intervenait comme un

garde-fou permanent: rappel des exigences du secteur santé, sensibilisation aux risques, validation des points sensibles. J'ai appris à intégrer leurs remarques tôt, pour éviter les corrections tardives qui coûtent plus cher et décalent les livraisons.

Au quotidien, cette organisation à trois pôles crée un rythme de travail lisible. Les sujets naissent et se priorisent avec le Product, s'alignent avec le Lead Software, s'organisent avec le Staff; puis ils transitent vers les Développeurs, passent par la QA, et sont accompagnés par l'équipe DevOps pour aller jusqu'en production; tout au long, la Cybersecurity garde un œil sur les zones sensibles. Dans ce circuit, mon rôle a été transversal: je me suis placé aux jonctions. J'ai clarifié des attentes entre Product et Développement, j'ai anticipé des points de coordination avec l'Integration, j'ai consolidé des retours de QA pour qu'ils soient actionnables par les Développeurs, et j'ai pris en compte les recommandations de Cybersecurity dès la conception des sujets, pour éviter les retouches de dernière minute.

Ce fonctionnement m'a aussi appris l'importance des "boucles courtes". Plutôt que d'attendre la fin d'un cycle pour découvrir les problèmes, nous cherchions à obtenir des retours précoces et réguliers: des démonstrations intermédiaires, des points d'avancement factuels, des essais en environnement contrôlé avant d'élargir la diffusion. Cette approche réduit la part d'incertitude et évite les effets tunnel. Elle oblige également à travailler de manière plus modulaire: découper, livrer par étapes, apprendre de chaque incrément, corriger la trajectoire si nécessaire. Dans un contexte de santé, où la fiabilité et la clarté sont essentielles, cette discipline de petites étapes maîtrisées m'a semblé particulièrement précieuse.

La collaboration avec DevOps a été un fil conducteur de ma mission. Nous avons cherché ensemble à rendre les passages de relais plus fluides: mieux documenter ce qui est attendu à chaque étape, réduire les opérations manuelles, et rendre les enchaînements plus stables. Je retiens notamment l'intérêt de rendre visibles les "engorgements": lorsqu'un point bloque, le mettre au jour rapidement pour que chacun puisse s'ajuster. Cette transparence évite les tensions et permet de traiter les causes plutôt que les symptômes. J'ai aussi vu combien l'anticipation gagnait du temps: préparer tôt les environnements et les paramétrages, valider en avance les chemins de mise à disposition, vérifier les conditions de réussite avant d'engager une étape.

Avec le Product, j'ai appris à formuler des objectifs concrets et mesurables sans tomber dans la sur-spécification. L'idée est de savoir précisément ce que l'on cherche à améliorer (par exemple, la rapidité d'une mise à disposition ou la clarté d'un parcours utilisateur), tout en laissant aux équipes la liberté de trouver la meilleure manière d'y parvenir. Cette posture favorise l'initiative et évite de figer trop tôt des solutions qui pourraient se révéler inadaptées. Les discussions avec le Lead Software m'ont, de leur côté, permis d'adopter un regard plus global: comment une décision locale impacte le reste du système, quels compromis accepter maintenant pour garder des options ouvertes plus tard, où mettre l'effort pour obtenir le meilleur effet de levier.

La relation avec la QA m'a sensibilisé à une vérité simple: la qualité ne se "rajoute" pas à la fin, elle se construit dès le début. En intégrant leurs critères tôt, nous avons réduit le nombre d'allers-retours et gagné en sérénité sur la fin des cycles. Cela passe par des cas de vérification clairs, des critères d'acceptation bien compris, et un langage partagé entre ceux qui demandent, ceux qui réalisent et ceux qui vérifient. C'est aussi un état d'esprit: accepter la contradiction, remercier le signalement d'un problème, et traiter les retours comme une opportunité d'apprendre.

Les échanges avec la Cybersecurity m'ont rappelé l'exigence propre au domaine de la santé. Il ne s'agit pas seulement de faire fonctionner un logiciel; il faut le faire fonctionner de manière responsable, en protégeant les données, en évitant les failles, et en respectant les cadres en vigueur. Leur présence dès la conception est rassurante: elle évite des oublis et des prises de risque involontaires. J'ai pris l'habitude de les consulter à chaque fois que nous touchions à une zone sensible, ou que nous introduisions un nouveau composant susceptible d'ouvrir une surface d'exposition.

Au fil des semaines, j'ai vu cette organisation produire des effets très concrets. Les sujets avançaient de manière plus régulière, les mises à disposition devenaient plus prévisibles, et les retours étaient mieux canalisés. La coordination entre les pôles nous a permis de réduire les urgences de dernière minute et de gagner en confiance lors des jalons importants. Surtout, chacun savait quel était son rôle et sa responsabilité: le Product sur le "quoi" et le "pourquoi", le Lead Software sur le "comment" global, le Staff sur le "quand" et "avec qui", les Développeurs sur la réalisation, la QA sur la validation, la Cybersecurity sur la protection, et DevOps sur le passage à

l'échelle et la continuité.

Enfin, je retiens la dimension humaine de cette équipe. Au-delà des schémas, ce qui fait la différence, ce sont les attitudes: l'écoute, la capacité à expliquer simplement, le respect des contraintes des autres, et la volonté de faire ensemble. J'ai bénéficié d'un accueil qui m'a permis d'être opérationnel rapidement, tout en ayant la liberté de proposer des améliorations. Cette confiance m'a encouragé à prendre des initiatives: clarifier un flux, simplifier une étape, ou structurer un retour pour le rendre plus utile. En résumé, l'Advanced Visualisation Team m'a offert un cadre solide et bienveillant pour apprendre, contribuer et progresser. Grâce à l'articulation entre Integration, Management et Development, j'ai pu mesurer concrètement ce que signifie "industrialiser" une démarche de livraison logicielle: donner de la cadence sans sacrifier la qualité, sécuriser les passages clés, et maintenir un cap commun au service des utilisateurs finaux. Cette expérience a renforcé ma conviction qu'une organisation claire, des rôles bien tenus et des boucles de feedback régulières sont les meilleurs atouts pour délivrer des logiciels fiables dans un environnement exigeant.

1.3. Vue d'Ensemble de l'Alternance

1.3.1. Contexte d'arrivée

Je viens d'intégrer GE HealthCare et je reçois mes premières missions. Je ne suis pas encore en mesure de tirer des conclusions: je découvre l'organisation, les priorités, les contraintes du secteur médical, et les modes de collaboration en place. Mon enjeu immédiat est d'entrer dans la culture de l'entreprise et d'adopter une posture DevOps utile dès le premier jour, sans brusquer les routines qui fonctionnent déjà.

Posture que j'adopte

- Humilité active: je commence par écouter et observer. Je cherche à comprendre pourquoi les choses sont faites d'une certaine manière avant de proposer des ajustements.
- Orientation flux: j'essaie de regarder le chemin complet d'une demande, depuis

l'expression du besoin jusqu'à l'usage réel, pour repérer où je peux aider à réduire les frictions.

- Petits pas visibles: je privilégie des améliorations modestes, concrètes et réversibles, qui apportent vite de la valeur sans créer d'effets de bord.
- Responsabilité partagée: ma mission n'est pas de "passer" un livrable à une autre équipe, mais de contribuer à ce qu'il aboutisse sereinement, avec les bons interlocuteurs.
- Transparence: je rends mon avancement lisible et je partage tôt mes questions et mes hypothèses, pour éviter les malentendus.

Ce que je cherche d'abord à comprendre

- Le "pourquoi" métier: à qui servent nos solutions, quels risques on veut réduire, quelles attentes non techniques comptent le plus (sécurité, conformité, qualité clinique).
- Les jalons réels: qui valide quoi, à quel moment, et avec quels critères explicites.
- Les points de friction: où ça attend, où ça revient en arrière, où l'on rediscute souvent les mêmes sujets.
- Les responsabilités: qui décide, qui exécute, qui opère, et comment on s'alerte mutuellement.
- Les rythmes: quels sont les rituels d'équipe (revues, planifications, points de suivi) et comment je m'y insère sans les alourdir.

1.3.2. Mon plan d'installation – 30/60/90 jours

Jours 1–30: comprendre et cartographier

- Observer un cycle complet de "demande → mise à disposition → retour".
- Identifier 2 à 3 irritants concrets vécus par les équipes (ex: manque de clarté sur une étape, validations tardives, rôles flous).
- Tenir une "page personnelle" d'apprentissage: décisions importantes, critères implicites, acronymes, contacts clés.

- Produire une note courte “ce que j’ai compris / ce que je suppose / ce que je propose de tester” et la faire relire par un référent.

Jours 31–60: tester des améliorations légères

- Proposer une simplification de processus “sans regret” (par exemple, clarifier les critères d’entrée/sortie d’un jalon, ou une checklist partagée pour un passage sensible).
- Mettre en place un point régulier de synchronisation avec les interlocuteurs de mes missions pour anticiper blocages et dépendances.
- Aider à rendre plus visibles les informations utiles à tous (état d’avancement, points ouverts, décisions prises), au bon endroit, sans multiplier les supports.

Jours 61–90: ancrer et élargir

- Évaluer, avec les équipes, ce qui a réellement aidé: garder ce qui simplifie, retirer ce qui ne sert pas.
- Étendre prudemment une pratique qui a fait ses preuves (même cadre, autre équipe ou autre périmètre).
- Formaliser des repères de suivi peu coûteux: ce que l’on observe pour savoir si on progresse (délais, reworks, sérénité des jalons).

1.3.3. Mes engagements personnels

- Clarté: dire ce que je fais, pourquoi je le fais, et quand je livrerai un premier résultat visible.
- Réversibilité: proposer des changements faciles à annuler si l’impact n’est pas celui attendu.
- Respect du cadre: intégrer d’emblée les exigences de sécurité, de confidentialité et de conformité propres au médical.
- Apprentissage continu: après chaque étape importante ou incident, capitaliser ce que j’ai appris dans un format court et réutilisable.

Ce que j’attends des équipes pour réussir

- Un parrain ou point de contact: pour valider mes compréhensions et orienter mes priorités.
- Des retours francs et rapides: pour ajuster le tir sans perdre de temps.
- La possibilité d’expérimenter à petite échelle: un périmètre “bac à sable” où tester une amélioration sans risque pour la production.
- Des critères explicites: savoir ce qui est “assez bon” pour franchir un jalon, afin d’éviter les débats de dernière minute.

Repères simples pour piloter ma progression

- Prévisibilité: moins d’écarts entre ce que l’on pense livrer et ce qui est réellement livré.
- Débit régulier: des petits lots qui avancent sans à-coups plutôt que des pics puis des creux.
- Moins de rework: une meilleure compréhension en amont réduit les retours en arrière.
- Sérénité aux jalons: rôles et critères clairs, moins d’urgence imprévue.
- Partage utile: ce que je produis (notes, checklists, synthèses) est court, trouvé facilement, et réutilisé par d’autres.

Risques identifiés et garde-fous

- Trop vouloir aller vite: je m’impose de valider mes hypothèses avec un référent avant d’impacter un processus partagé.
- Multiplier les documents: je centralise l’information au même endroit et je supprime les doublons.
- Sous-estimer les contraintes du médical: je consulte systématiquement les interlocuteurs qualité et sécurité avant tout changement de routine.
- Changement non accepté: je privilégie la démonstration sur un petit périmètre et je recueille des retours utilisateurs avant d’élargir.

Ce que signifie “adopter la vision DevOps” ici et maintenant

- Penser système: ce n’est pas “mon” bout de chaîne, c’est l’ensemble du parcours jusqu’à l’usage réel.
- Raccourcir les boucles de feedback: voir plus tôt, corriger plus tôt, avec des impacts

maîtrisés.

- Rendre le travail visible: l'information circule, on décide sur des faits et l'on sait quoi faire en cas d'imprévu.
- Protéger la qualité: mieux préparer les passages sensibles, prévoir le repli, et garder la traçabilité des décisions.

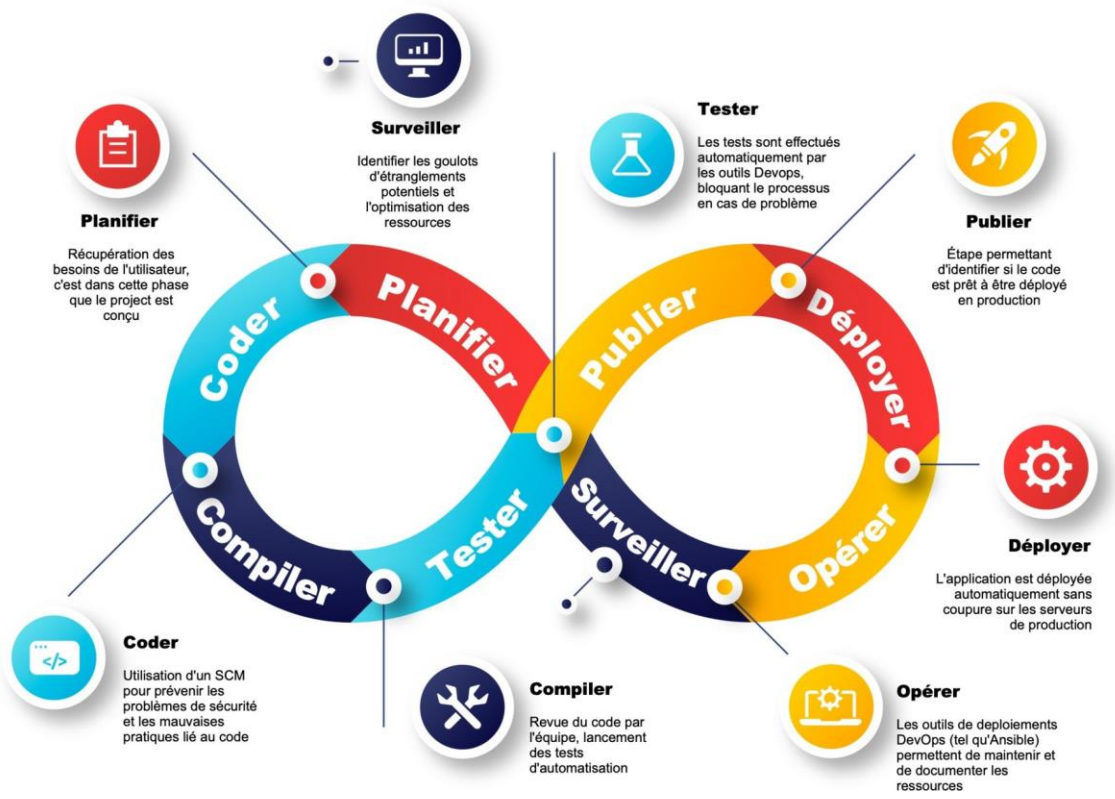

Figure 3 - Cycle de Vie DevOps

1.4. Portée du travail et parties prenantes

Le travail réalisé au cours de cette alternance s'inscrit dans un programme d'innovation chez GE HealthCare, centré sur l'imagerie médicale assistée par l'intelligence artificielle. L'objectif est d'améliorer la qualité et la rapidité des parcours diagnostiques tout en renforçant la fiabilité, la traçabilité et la sécurité des services logiciels en contexte clinique. Ma contribution se situe dans une posture DevOps pragmatique: faire circuler la valeur de façon régulière, visible et sûre, en alignant les équipes et en protégeant les passages sensibles.

Le périmètre couvre les activités qui permettent à des fonctionnalités priorisées d'arriver jusqu'aux utilisateurs finaux sans à-coups: clarification du besoin et des critères d'acceptation; découpage en petits lots; synchronisation des jalons; intégration des points de contrôle qualité "au fil de l'eau"; prise en compte précoce des exigences de sécurité et de confidentialité; préparation des mises à disposition avec critères de décision explicites, plan de repli et traçabilité; recueil structuré des retours terrain pour nourrir l'amélioration continue. Restent hors périmètre direct: la recherche fondamentale en IA, la négociation commerciale avec les établissements et la gestion opérationnelle des infrastructures hospitalières (prises en compte comme contraintes, sans en avoir la responsabilité).

Les parties prenantes sont nombreuses et complémentaires. Côté établissements de santé: radiologues, manipulateurs en électroradiologie et cadres de santé expriment les besoins cliniques, valident l'adéquation au flux de soins et signalent les irritants; les DSI/RSSI veillent à l'intégration, à la sécurité des données et à la continuité de service; les comités qualité/éthique s'assurent de la pertinence clinique et de la conformité. Côté GE HealthCare: le Product Management porte la vision et arbitre les priorités; le développement met en œuvre les fonctionnalités et corrige les anomalies; la fonction DevOps coordonne le flux de bout en bout, anticipe les passages sensibles

et garantit la réversibilité; la cybersécurité prescrit les règles de protection et valide les mesures; la qualité/réglementaire aligne les pratiques sur les référentiels applicables et maintient l'évidence documentaire; le support/service capte la voix du terrain et boucle les retours. La R&D explore de nouvelles approches et transfère les connaissances vers les équipes produit et opérationnelles.

Pour éviter les zones grises, la répartition des responsabilités est clarifiée: le produit répond au "quoi/pourquoi", le développement au "comment fonctionnel", le DevOps au "comment on y va", la qualité au "ce qui est acceptable", la cybersécurité au "ce qui est sûr", le support au "ce qui est vécu", et les utilisateurs référents à "ce qui aide en clinique". La coordination s'appuie sur des rituels courts et utiles: points d'alignement hebdomadaires focalisés sur décisions et blocages; revues de jalons avec critères d'entrée/sortie; retours d'expérience sans blâme après incident avec quelques actions correctives suivies; une source unique et à jour pour décisions, responsabilités et états d'avancement.

L'environnement clinique impose des exigences non négociables: protection des données de santé (confidentialité, minimisation, droits d'accès, journalisation), sécurité des patients et qualité clinique, traçabilité et auditabilité des décisions, continuité de service et éthique (vigilance sur les biais, transparence sur les limites d'usage). Pour accélérer la valeur et réduire les retours en arrière, l'information utile à décider est rendue visible tôt (impacts, risques, dépendances), les critères d'acceptation sont co-construits avec produit/qualité et, lorsque pertinent, des utilisateurs référents; chaque passage sensible comporte une fenêtre de mise à disposition, un plan de repli et une communication claire.

Indicateurs recherchés: prévisibilité (écart réduit entre annoncé et livré), débit régulier (petits lots), baisse du rework (meilleurs critères d'entrée/sortie), sérénité aux jalons (rôles clairs, moins d'urgences) et adoption clinique mesurable via les retours utilisateurs et le support.

Les risques principaux et leurs garde-fous sont identifiés: complexification inutile (standards légers, réversibles, testés à petite échelle avant extension); sous-estimation des contraintes cliniques (implication régulière d'utilisateurs référents et de la qualité/cybersécurité); retour aux silos (canaux unifiés, responsabilités explicites); documentation "pour l'audit" mais inutilisable (formats courts, à jour, orientés action et réemploi). Les hypothèses de réussite incluent la disponibilité des interlocuteurs clés pour des validations rapides, l'accès à un périmètre de test sans risque pour la production, une stabilité minimale des priorités sur des cycles courts et un engagement partagé à rendre visibles les arbitrages.

Dans ce cadre, mon rôle d'alternant DevOps est d'observer le système de bout en bout, cartographier les jalons et rendre visibles les frictions; proposer des améliorations modestes, concrètes et réversibles; aider à clarifier critères d'entrée/sortie et responsabilités; contribuer à une documentation utile et vivante; tenir un journal d'apprentissage et de décisions pour capitaliser. En synthèse, la portée de l'alternance est d'installer des habitudes qui augmentent clarté, prévisibilité et sécurité, pour transformer l'innovation en bénéfices concrets et sûrs pour les équipes et, in fine, pour les patients.

2. ENVIRONNEMENT TECHNIQUE ET ACTIVITÉS

2.1. Écosystème DevOps

Au cours de mon alternance, j'ai évolué dans un écosystème exigeant où la valeur n'est réellement créée que lorsque les changements traversent, sans à-coups, l'ensemble de la chaîne d'environnements : DEV → INT → MET (Mise en Test/UAT) → PRÉ-PROD → PROD. Mon objectif a donc été de rendre ces "MET" robustes, représentatifs et sûrs, afin que les équipes puissent décider et livrer en confiance. La posture DevOps s'est traduite ici par des pratiques qui privilégient la lisibilité des flux, la traçabilité des décisions et la réversibilité des mises à disposition.

L'environnement technique s'appuie sur des briques ouvertes et industrielles. Pour l'orchestration des livraisons, Jenkins et GitLab CI sont utilisés de façon complémentaire, mais le cœur de mon travail a porté sur la qualité et la gouvernance des environnements non-prod (en particulier les MET), bien plus que sur les spécificités d'une plateforme CI donnée. Les applications sont conteneurisées (Docker) et déployées sur Kubernetes avec Helm; les artefacts (images, charts, paquets) sont gérés via Artifactory pour garantir traçabilité et reproductibilité. La qualité de code et la sécurité applicative sont suivies par Sonar (règles, debt, couverture), tandis que les tests end-to-end (Playwright) et la consolidation des résultats (ReportPortal) assurent une vision claire de la stabilité fonctionnelle. L'infrastructure de tests et les serveurs de build tournent principalement sur Linux SUSE et sont gérés en IaC avec Ansible pour standardiser les configurations, durcir les accès (SSH) et assurer des déploiements répétables. Le pilotage opérationnel se fait via des tableaux de bord simples (métriques de pipelines, temps de cycle, taux

de succès par environnement), adossés à des logs d'audit permettant de remonter la chaîne des responsabilités et des décisions.

Ce dispositif n'a de valeur que si les environnements MET restent fiables, proches de la production et cependant "sûrs" pour expérimenter. Concrètement, cela implique:

- Une stratégie claire de données de test: anonymisation ou jeux synthétiques, référentiels communs, procédures de rafraîchissement contrôlées, et séparation stricte des privilèges.
- Des critères d'entrée/sortie par environnement: ce qui est nécessaire pour promouvoir un change (qualité, sécurité, validation clinique), et ce qui permet de revenir en arrière sans friction.
- Des fenêtres de mise à disposition convenues et des plans de repli testés: les "go/no-go" sont argumentés, documentés et réversibles.
- Des configurations "comme en prod, mais pas en prod": parité des variables, secrets gérés de manière sûre, observabilité alignée (journalisation, traces, métriques) pour éviter les surprises tardives.
- Des tests qui suivent la progression: tests unitaires et de contrat tôt, end-to-end stables et rapides en MET, scénarios de non-régression et smoke tests à chaque promotion.
- Une documentation utile et vivante: topologies d'environnements, modes opératoires, critères de validation et journal des décisions, afin que chacun sache "où on en est" et "ce qui bloque".

Dans ce cadre, mon rôle a été d'industrialiser les passages sensibles, de simplifier ce qui peut l'être et de rendre visible ce qui compte pour décider. Les MET ont servi de point d'appui: c'est là que l'on gagne en confiance (ou que l'on perd du temps). J'ai donc concentré mes efforts sur la robustesse des MET, leur proximité fonctionnelle avec la prod, et la discipline des promotions.

2.2. Tâches réalisées

- Architecture et standardisation des parcours de livraison sur plusieurs régions (UE/US/EMEA), en veillant surtout à l'alignement des environnements MET et PRÉ-PROD: exigences de parité, secrets, variables, fenêtres de mise à disposition et critères de promotion; orchestration via Jenkins et, lorsque nécessaire, GitLab CI, avec exécutions containerisées (runners Docker) pour garantir l'isolation.

- Conception et maintenance de charts Helm et conteneurisation d'applications avec publication des images Docker; traçabilité des artefacts (Artifactory) afin de reproduire à l'identique en MET ce qui sera mis en PRÉ-PROD/PROD.
- Industrialisation du cycle de release: génération d'ISO et gestion de Release Candidates, versioning sémantique, feature flags pour activer/désactiver en MET sans risque; publication automatisée de release notes avec marquages ATD/RC/BETA pour une lecture opérationnelle.
- Optimisation des temps de préparation des tests en MET: mutualisation des étapes de préparation (checkout, npm build), réduction des duplications, cohérence de l'état des dépôts entre jobs, afin d'augmenter le débit des validations fonctionnelles.
- Fiabilisation de la QA end-to-end: développement/correction de tests Playwright (install/uninstall, multi-navigateurs), stabilisation des timeouts, exécutions nightly paramétrables en MET pour détecter tôt les régressions et sécuriser les promotions.
- Normalisation qualité/sécurité du code: intégration Sonar avec seuils partagés (quality gates), mutualisation des vérifications de conformité IP (en-têtes copyright), politiques de branches/projets adaptées au flux de promotion vers MET puis PRÉ-PROD.
- Automatisation et IaC: optimisation Ansible (rôles/playbooks) pour Linux SUSE, durcissement SSH, déploiements Build & Release reproductibles; alignement des configurations MET avec PROD pour limiter les écarts de comportement.
- Observabilité et traçabilité des campagnes: standardisation ReportPortal (mergeAllLaunches) pour regrouper les résultats par version et environnement; métriques de pipelines, journaux d'audit et suivi des SLA centrés sur la stabilité MET et le temps de cycle jusqu'à PRÉ-PROD.
- Gouvernance des workflows et productivité équipes: génération de backlog opérationnel, planification Scrum, création de jobs de déclenchement (TRIGGER_TESTS) pour lancer des batteries ciblées en MET; nettoyage automatisé des artefacts (.iso) sur serveurs nightly pour éviter l'entropie.
- Gestion de la configuration DevOps transverse: maintien d'un template d'usine logicielle (ISV-TEMPLATE) et diffusion de standards (nomenclatures, critères d'entrée/sortie, conventions de tagging) afin d'harmoniser les pratiques entre équipes et de sécuriser les promotions d'un environnement à l'autre.

L’écosystème DevOps mis en place chez GE HealthCare combine des outils éprouvés et des pratiques communes qui sécurisent l’ensemble du cycle de vie applicatif et IA, depuis l’idéation jusqu’à la mise en production. L’objectif est double: livrer vite et bien, tout en préservant les exigences réglementaires, la traçabilité et la sécurité propres au médical.

2.3. Organisation et cadence de travail

- Les projets suivent une méthodologie Agile Scrum avec des sprints courts, des revues et des rétrospectives systématiques. Cette cadence favorise la collaboration inter-équipes, l’adaptabilité aux retours du terrain et une livraison continue de valeur.
- Les environnements sont structurés en chaîne DEV → INT → MET (Mise en Test/UAT) → PRÉ-PROD → PROD, avec des critères d’entrée/sortie explicites, de la parité de configuration et des plans de repli testés. Les MET jouent un rôle clé: représentatifs de la production, ils permettent de valider fonctionnellement et cliniquement les changements avant promotion.

2.4. Automatisation et pipelines

- L’automatisation s’appuie principalement sur Jenkins et GitLab CI/CD, utilisés de manière complémentaire. Les pipelines couvrent build, tests multi-niveaux, packaging et déploiement; ils exploitent l’API GitLab pour les intégrations avancées (tagging, gestion de versions, déclenchements ciblés).
- Les exécutions s’effectuent dans des environnements containerisés afin d’assurer isolation, reproductibilité et portabilité entre régions (UE/US/EMEA) et entre environnements (notamment MET et PRÉ-PROD).
- Les cycles de release sont industrialisés: versioning sémantique, Release Candidates, feature flags, génération automatisée de notes de version et gestion rigoureuse des artefacts.

Plateforme applicative et gestion des artefacts

- Les applications sont conteneurisées avec Docker et orchestrées par Kubernetes. Helm standardise les déploiements et garantit la cohérence des configurations entre MET et

PROD.

- Artifactory centralise la gestion des images, packages et charts; chaque artefact est traçable, signé si nécessaire et associé à sa provenance (commit, pipeline, environnement cible).

Infrastructure as Code et systèmes

- Les serveurs (majoritairement Linux, notamment SUSE) sont administrés en Infrastructure as Code via Ansible: rôles/playbooks pour installations reproductibles, durcissement SSH, politique de patching et cohérence inter-environnements.
- Les secrets et variables sensibles sont gérés de manière sécurisée et injectés à l'exécution, évitant toute exposition dans le code ou les scripts.

2.5. Qualité, sécurité et tests

- La qualité du code est évaluée en continu par SonarQube (quality gates partagés, dette technique, couverture). Les échecs bloquent les promotions jusqu'à correction.
- Les tests sont structurés par niveaux: unitaires et de contrat tôt; end-to-end et smoke tests en MET pour valider les parcours critiques. Playwright automatise les scénarios fonctionnels (multi-navigateurs, installation/désinstallation, timeouts stabilisés).
- La conformité IP et les en-têtes copyright sont vérifiés automatiquement; des politiques de branches et de revue garantissent la qualité des contributions.

2.6. Observabilité, supervision et traçabilité

- La supervision combine métriques, logs et traces afin de détecter précocement les anomalies et de surveiller la performance et la disponibilité. Les seuils d'alerte sont calibrés pour limiter le bruit tout en couvrant les risques cliniques.
- ReportPortal consolide les résultats de test (mergeAllLaunches), offrant une vision unifiée par version et par environnement. Des tableaux de bord suivent temps de cycle, taux de succès par étape, flakiness des tests et stabilité des promotions.
- Les décisions de "go/no-go", les fenêtres de mise à disposition et les plans de repli sont documentés et audités, assurant la re-jouabilité et la conformité réglementaire.

2.7. Sécurité des données et conformité

- Les données de test en MET sont anonymisées ou synthétiques, avec procédures de rafraîchissement contrôlées et séparation des privilèges. La journalisation garantit l'auditabilité des accès et des changements.
- Les pratiques intègrent dès l'amont les exigences de cybersécurité, de confidentialité et de sûreté de fonctionnement, afin de protéger patients et établissements.

3. Industrialisation et Optimisation d'une usine logicielle cloud-native chez GE HealthCare

3.1. Contexte et objectifs

Contexte entreprise et produit GE HealthCare opère des plateformes logicielles et des modèles d'intelligence artificielle déployés à l'échelle de plusieurs régions (UE/US/EMEA). Les solutions adressent des usages cliniques critiques — imagerie, workflow diagnostique, supervision et support aux décisions — qui imposent une disponibilité élevée, une conformité stricte et une résilience éprouvée. L'environnement est multi-entités (équipes produit, qualité, sécurité, opérations, cliniques), multi-réglementaire (RGPD/UE, HIPAA/US, exigences pays EMEA) et multi-environnements (DEV, INT, MET/UAT, PRÉ-PROD, PROD).

Dans ce contexte, chaque évolution logicielle doit franchir une chaîne d'intégration et de déploiement sécurisée, traçable et répétable. La promesse vis-à-vis des établissements de santé est double: délivrer de la valeur rapidement et garantir la sûreté de fonctionnement. Cela suppose:

- des déploiements prévisibles et réversibles;
- des environnements de test (MET) proches de la production, avec des données anonymisées/synthétiques et des contrôles d'accès stricts;
- des mécanismes d'observabilité qui détectent tôt les dégradations et éclairent les décisions de "go/no-go";
- une gouvernance qualité/sécurité intégrée à la chaîne de livraison, pour satisfaire audits et exigences cliniques.

3.2. Périmètre géographique et réglementaire

- Protection des données (RGPD), exigences de traçabilité et minimisation des
- Impacts techniques: partitions régionales des artefacts, parité de configuration, latence réseau maîtrisée, observabilité corrélée entre régions, stratégies.

3.2.1. Enjeux de disponibilité, conformité et résilience

- Disponibilité: maintenir un service ≥ 99,9% sur les composants critiques. Concrètement, cela autorise au maximum environ 43 minutes d'indisponibilité par mois; chaque minute compte et doit être expliquée, tracée et évitable.
- Conformité: intégrer automatiquement les contrôles (qualité code, sécurité, conformité IP, chiffrement en transit/au repos, gestion des secrets) dans les pipelines; garantir la re-jouabilité des preuves pour audits.
- Résilience: déploiements blue/green ou canary, feature flags pour désactiver à chaud une capability, plans de repli testés, sauvegardes/restores validés, tests de chaos en environnement non-prod pour valider les hypothèses de récupération.

3.2.2. Objectifs de l'alternance L'alternance vise à consolider l'industrialisation logicielle sur quatre axes prioritaires:

- Fiabiliser la production
  - Renforcer la parité → PRÉ-PROD → PROD pour réduire les écarts de comportement.
  - Mettre en place des stratégies de mise à disposition avec fenêtres, critères d'entrée/sortie, et plans de repli testés.
  - Diminuer le taux d'échec des changements et réduire l'impact utilisateur lors des mises à jour.
- Standardiser la CI/CD
  - Disposer d'un socle de pipelines réutilisable (templates, conventions, quality gates communs).
  - Harmoniser les nomenclatures de versions, tags, artefacts et environnements entre équipes et régions.
  - Automatiser la génération et la publication des notes de version et des artefacts signés.
- Automatiser (IaC)
  - Décrire serveurs et middlewares en Infrastructure as Code pour des déploiements reproductibles.
  - Durcir les configurations (SSH, packages, politiques de patching) et tracer les changements.
  - Réduire les "neiges techniques" en éliminant les écarts de configuration non documentés.
- Améliorer observabilité et qualité
  - Étendre la couverture des tests (unitaires, contrat, end-to-end) et stabiliser les suites.
  - Standardiser la collecte des métriques, logs et traces; créer des tableaux de bord décisions-prêts.

- Intégrer des seuils d'alerte pertinents, limiter le bruit et privilégier les signaux utiles au diagnostic.

3.2.3. Indicateurs cibles et méthodes de mesure Pour piloter ces objectifs, un jeu d'indicateurs cibles est défini, avec méthodes de collecte et seuils d'alerte.

- Disponibilité ≥ 99,9%
  - Définition: uptime agrégé des services critiques, hors maintenances planifiées; mesure par région.
  - Mesure: sondes de disponibilité externes + métriques internes (SLO/SLA), corrélées.
  - Seuils: alerte à 99,95% glissant hebdo; plan d'action si < 99,9% mensuel.
  - Actions: élimination des SPOF, bascule automatisée, validation régulière des runbooks d'incident.
- Temps de build
  - Objectif: réduire de x% le temps médian de build (ex: de 25 min à ≤ 12 min).
  - Mesure: métriques pipelines (commit → artefact prêt), histogrammes par projet.
  - Leviers: mutualisation des étapes, caches gérés, parallélisation des jobs, épuration des dépendances.
- MTTR (Mean Time To Repair) ↓
  - Objectif: passer d'un MTTR moyen de quelques heures à ≤ 60 min sur incidents P1/P2.
  - Mesure: corrélation des événements (alertes → diagnostic → restauration), journaux d'incident.
  - Leviers: playbooks de remédiation, "one-click rollback", feature flags, alertes explicites avec contexte.
- Vulnérabilités critiques
  - Objectif: 0 vulnérabilité critique ouverte > 7 jours; 0 haute > 30 jours.
  - Mesure: scans intégrés aux pipelines; tableaux de bord sécurité.
  - Leviers: quality gates bloquants, priorisation en backlog, mise à jour proactive des dépendances.

3.2.4. Indicateurs complémentaires

- Change Failure Rate < 10% (déploiements entraînant rollback ou patch urgent).
- Lead Time for Changes: ≤ 1 jour ouvré pour un changement faible risque promu jusqu'en MET.
- Flakiness des tests end-to-end: < 2% sur 30 jours glissants.
- Couverture de tests: cibles par type de projet (ex: unitaires ≥ 70%, contrat ≥ 90% sur APIs critiques).Gouvernance et parties prenantes
- Parties prenantes: équipes produit/dev, QA, sécurité (RSSI), ops/SRE,

qualité/réglementaire, représentants cliniques, support.
- Rituels: revues de sprint, comités qualité/sécurité, points d'observabilité, revue d'incidents et de dette technique.
- Artefacts: tableaux de bord partagés, runbooks, templates de pipelines, inventaire des environnements et dépendances, journal des décisions.

3.2.5. Périmètre et hors-périmètre

Dans le périmètre: automatisation des pipelines, IaC serveurs/tests, standardisation des releases, observabilité, critères d'entrée/sortie, plans de repli, sécurité intégrée.
Hors périmètre direct: recherche fondamentale en IA, négociation commerciale, administration des infrastructures hospitalières (considérées comme contraintes), certification réglementaire finale des dispositifs (appuyée, mais non pilotée).

3.2.6. Risques et mesures de mitigation

- Sous-estimation des contraintes régionales: mettre en place des "profiles" par région et des revues de conformité localisées.
- Effet tunnel de l'industrialisation: livrer par petits incréments avec valeur visible; mesurer avant/après.
- Flakiness des tests: stabilisation ciblée, isolation des causes, timeouts et données de test maîtrisées.
- Rejet des standards: co-construction avec les équipes, docs courtes et vivantes, accompagnement et feedback loops.
- Entropie des environnements: nettoyage automatisé, inventaire des artefacts, revues régulières de configuration.

3.2.7. Résultats attendus À l'issue de l'alternance, l'organisation dispose:

- d'un socle CI/CD standard réutilisable entre équipes et régions, avec conventions et quality gates partagés;
- d'environnements MET/PRÉ-PROD représentatifs, observables et sûrs, accélérant les décisions de mise en production;
- d'une IaC permettant des déploiements reproductibles, audités et durcis;
- de tableaux de bord utiles aux décisions, réduisant le temps de diagnostic et le MTTR;
- d'une trajectoire chiffrée sur la disponibilité, le temps de build et la réduction des vulnérabilités critiques.

3.3. les principaux outils

3.3.1. Jenkins

Rôle : Orchestrateur CI/CD polyvalent pour construire, tester et déployer sur plusieurs environnements (DEV → INT → MET → PRÉ-PROD → PROD).
Pourquoi important : Forte extensibilité (plugins), programmabilité (pipelines déclaratifs/scripted), gestion fine des files/agents pour absorber la charge multi-projets.
Usages typiques : Pipelines multi-branches, fan-out des tests, promotion d'artefacts, fenêtres de mise à disposition, déclenchements conditionnels par étiquettes/flags.
Indicateurs : Durée médiane de build, taux de succès par étape, temps d'attente en queue, change failure rate.
Bonnes pratiques : Pipelines "as code", isolation par agents conteneurisés, secrets via credentials store, steps idempotents.

3.3.2. GitLab CI/CD et Runners

Rôle: Chaîne CI/CD intégrée au SCM pour valider commits/merge requests et automatiser les contrôles qualité/sécurité en amont.
Pourquoi important : Feedback rapide au plus près du code, intégrations natives SAST/SCA, règles de protection de branches et approbations.
Usages typiques : Pipelines sur merge request, gating qualité (lint/tests/sonar), publication d'artefacts, déclenchement de jobs de promotion vers MET.
Indicateurs : Lead time for changes, temps de pipeline, couverture de tests, vulnérabilités bloquantes.
Bonnes pratiques : Templates réutilisables, caches partagés, runners éphémères, policy "no secrets in repo".

3.3.3. Docker

Rôle: Conteneurisation des applications et des jobs CI pour garantir portabilité, isolation et reproductibilité.
Pourquoi important: Réduction des "works on my machine", parité MET/PRÉ-PROD/PROD, accélération des tests.
Usages typiques: Images de build standardisées, images applicatives minimales, exécutions de tests end-to-end en environnement éphémère.
Indicateurs: Taille d'image, temps de pull, taux de réutilisation du cache, CVEs par image.
Bonnes pratiques: Multi-stage builds, images de base durcies, SBOM, scans de vulnérabilités

intégrés.

3.3.4. Kubernetes

Rôle: Orchestrateur de conteneurs pour déployer et scaler les workloads; support des déploiements progressifs.
Pourquoi important: Haute disponibilité, auto-rétablissement, policy réseau et secrets gérés, déploiements canary/blue-green.
Usages typiques: Espaces de noms par environnement (dont MET), stratégies de rollout/rollback, autoscaling, NetworkPolicies pour cloisonner.
Indicateurs: SLO par service, taux d'échec de pods, temps de rollout, saturation CPU/RAM.
Bonnes pratiques: Manifests versionnés, probes de liveness/readiness, ressources/quotas, RBAC minimaliste.

3.3.5. Helm

Rôle: Gestionnaire de charts pour packager et paramétrer les déploiements Kubernetes.
Pourquoi important: Standardisation des déploiements multi-régions/environnements, valeurs spécifiques MET vs PROD.
Usages typiques: Charts applicatifs avec valeurs UE/US/EMEA, promotion de versions via "values" contrôlés, packaging reproductible.
Indicateurs: Versions déployées par environnement, drift de valeurs, taux de rollback post-upgrade.
Bonnes pratiques: Charts testés (helm lint/test), values sécurisées, schémas de valeurs documentés, SemVer.

3.3.6. Ansible

Rôle: Infrastructure as Code pour configurer serveurs, middlewares, et opérations répétables.
Pourquoi important: Conformité et traçabilité des changements, durcissement cohérent entre environnements.
Usages typiques: Provisioning d'agents/runners, durcissement SSH, patching, semences de données anonymisées en MET.
Indicateurs: Taux de réussite des runs, drift de configuration, temps moyen d'exécution des playbooks.
Bonnes pratiques: Rôles modulaires, inventaires par région, idempotence, vault pour secrets.

3.3.7. SonarQube (Sonar)

Rôle: Analyse de qualité et sécurité du code (bugs, vulnérabilités, dette, duplications, couverture).
Pourquoi important: Barrières de qualité avant promotion, preuves d'audit intégrables au pipeline.
Usages typiques: Quality gates bloquants sur MR, rapports par version, suivi de la dette technique.
Indicateurs: % couverture, nombre de vulnérabilités critiques/hautes, code smells, tendance dette.
Bonnes pratiques: Règles adaptées par langage, seuils stricts sur projets critiques, correction "new code first".

3.3.8. ReportPortal

Rôle: Plateforme centralisée de reporting de tests (automatisation, flakiness, tendances) multi-projets.
Pourquoi important: Visibilité transverse sur la stabilité, décision "go/no-go" éclairée avant promotion en MET/PRÉ-PROD.
Usages typiques: Agrégation des résultats e2e/smoke/perf, merge des campagnes par release, classification auto des échecs récurrents.
Indicateurs: Taux de réussite par suite, flakiness, temps moyen d'exécution, régressions par version.
Bonnes pratiques: Nommage stable des suites, tags par environnement, intégration pipeline → ReportPortal, analyse régulière des tests instables.