MSE | MASTER OF SCIENCE IN ENGINEERING

Déploiement IT en « vibe-coding »

Chaos ou automatisation : l'IA générative face aux défis du déploiement IT

Management des Projets IT
MSE

Eva Ray, Samuel Roland, Massimo Stefani

5 Janvier 2026

Hes·so

Table des matières

Résumé

Ce rapport, réalisé dans le cadre du Master of Science in Engineering (MSE) de la HES-SO, explore le potentiel de l'intelligence artificielle générative (IAG) pour automatiser la phase de déploiement du cycle de vie de développement logiciel (SDLC). L'objectif central est d'évaluer l'efficacité de l'approche « vibe-coding », qui est un mode de développement assisté par des agents autonomes, pour automatiser le provisionnement d'infrastructures via Terraform sur AWS et l'orchestration de conteneurs avec Kubernetes. L'expérimentation repose sur le déploiement de l'application web open-source OpenDidac, développée par l'HEIG-VD. L'outil principal utilisé est Windsurf, un environnement de développement intégré (IDE) doté de capacités d'IA générative en mode agent. Le modèle d'IA sélectionné est GPT-5 (medium reasoning), reconnu pour ses performances en programmation (décembre 2025).

Les résultats mettent en évidence une disparité de performance selon les technologies : l'IAG a généré avec succès 95 % des manifestes Kubernetes en appliquant rigoureusement les standards de l'écosystème (Kustomize, Secrets, sondes de santé). En revanche, l'automatisation de Terraform s'est révélée plus laborieuse, plafonnant à 80 % de code généré après plus de dix itérations. L'IA a montré des limites structurelles notables, produisant un code redondant et monolithique nécessitant une correction humaine significative. Au global, environ 87,5 % du code de déploiement a été produit par l'IA, ce qui est un résultat prometteur.

En conclusion, si l'IAG agit comme un catalyseur puissant pour le prototypage rapide et la réduction du code répétitif (« boilerplate »), elle ne peut se substituer à une expertise d'ingénierie solide. Il faut souligner que, sans une supervision humaine critique, les risques de dettes techniques et d'erreurs architecturales sont élevés. Enfin, sur le plan éducatif, une délégation excessive à l'IA chez les débutants peut entraver l'acquisition des compétences fondamentales, rendant l'apprentissage manuel préalable indispensable.

Mots-clés : Intelligence Artificielle (IA) Générative (IAG), Vibe-coding, SDLC, Déploiement IT, Infrastructure as Code (IaC), Terraform, Kubernetes, AWS, Windsurf, Automatisation, Cloud Computing, GPT-5.

1. Introduction

1.1. Contexte

La gestion de projets informatiques nécessite des méthodologies structurées pour organiser le développement et le déploiement de systèmes d'information. Le Systems Development Life Cycle (SDLC) représente le cycle de vie produit le plus utilisé dans le domaine des technologies de l'information. Celui-ci est situé dans la phase « Execute and Control Project » du Project Life-Cyle (PLC) et a pour objectif de mener à bien un projet IT de A à Z.

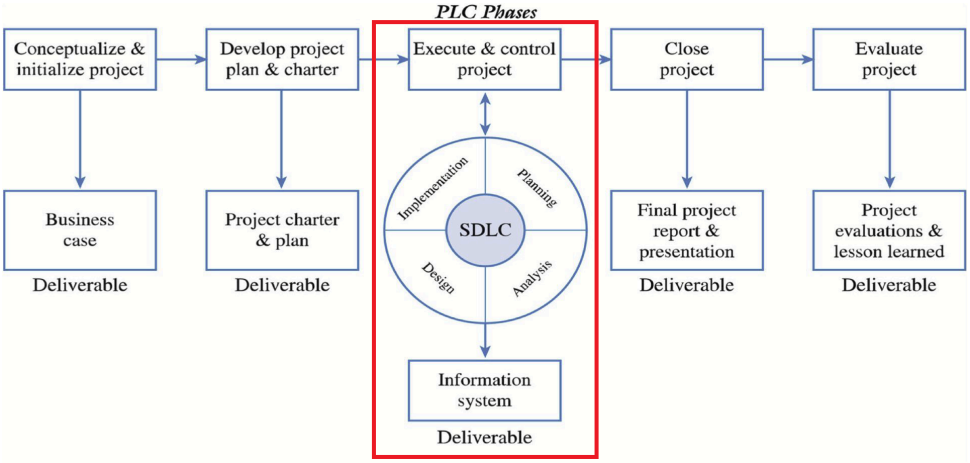

Fig. 1. – Position du SDLC dans le PLC

Le cycle SDLC établit une séquence logique d'activités de développement organisées en phases distinctes. Plusieurs variantes du SDLC existent, avec un nombre variable de phases, selon la granularité recherchée. Puisque ce rapport porte sur l'intégration de l'IA générative dans la phase de déploiement du SDLC, nous adoptons le modèle en 7 phases, afin de pouvoir mettre en évidence celle-ci. Les 7 phases du SDLC sont les suivantes : la planification (planning), l'analyse (analysis), la conception (design), l'implémentation (implementation), le test (testing), le déploiement (deployment) et la maintenance (maintenance) [1].

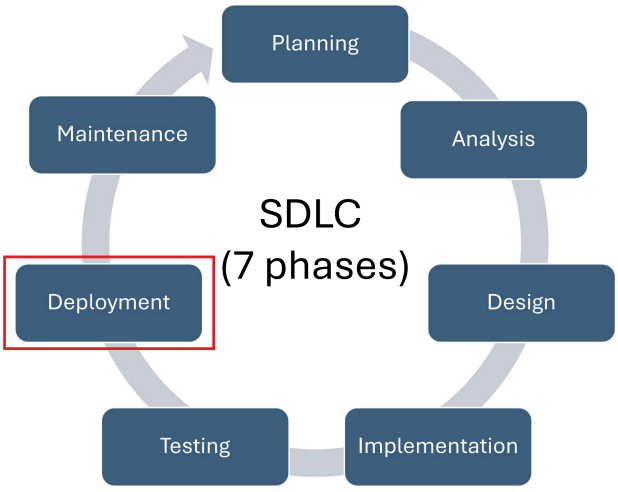

Fig. 2. – SDLC en 7 phases

Chaque phase du SDLC a un but bien précis. [1], [2]

- **Planification**: Cette phase dépend fortement des livrables précédents du PLC et peut se décliner selon trois situations : une planification complète et imposée par le client (date, budget et fonctionnalités), une absence de planification préalable nécessitant une proposition complète du prestataire IT, ou une situation intermédiaire où seul le périmètre fonctionnel est défini. Cette

phase distingue les projets feature-driven (flexibilité temporelle) des projets date-driven (flexibilité fonctionnelle).

- **Analyse**: L'analyse englobe l'analyse du business, des utilisateurs, des technologies, des travaux antérieurs et de la faisabilité du projet. Cette phase produit la Software Requirements Specification (SRS), document décrivant précisément les exigences fonctionnelles et non-fonctionnelles..
- **Conception**: La conception définit l'architecture du système par décomposition en composants, précise interfaces, données et choix technologiques, et fournit une spécification détaillée du système.
- **Implémentation**: L'implémentation concrétise cette architecture par le développement, l'intégration, les tests, l'installation du système, ainsi que la formation et la documentation nécessaires à son utilisation.
- **Test**: La phase de test a pour but de valider que le système développé a la qualité attendue pour la production. Elle inclut des tests unitaires, d'intégration, de système et d'acceptation utilisateur.
- **Déploiement**: C'est pendant la phase de déploiement que le logiciel développé est mis à disposition des utilisateurs finaux. Cela marque le passage du développement et des tests à l'exploitation réelle du système comme un produit opérationnel. Le déploiement comprend plusieurs étapes clés, telles que le choix de la stratégie de déploiement, la préparation de l'environnement de production, la migration des données, la configuration du système, la formation des utilisateurs et le lancement officiel du logiciel. [3] Un déploiement réussi nécessite une planification minutieuse pour minimiser les interruptions de service et garantir une transition fluide vers le nouvel environnement. Il est aussi important que des procédures de retour en arrière soient en place au cas où des problèmes imprévus surviendraient après le lancement. Dans ce projet, nous nous concentrons sur cette phase de déploiement.
- **Maintenance**: La phase de maintenance assure la pérennité du système en production, gérant les corrections d'erreurs, les améliorations fonctionnelles et l'adaptation aux évolutions de l'environnement organisationnel.

La maîtrise des différentes phases et de leurs interdépendances constitue un facteur déterminant pour la gestion de projet IT, permettant d'assurer une progression ordonnée depuis l'identification d'un besoin jusqu'à la mise en production et le maintien opérationnel d'un système d'information.

1.2. Objectif du rapport

L'objectif principal de ce rapport est d'explorer comment l'intégration de l'IA générative dans le processus de déploiement peut offrir plusieurs avantages, comme par exemple:

- **Automatisation de la création des fichiers de déploiement**: Beaucoup d'outils permettant de déployer des applications nécessitent la création de fichiers de configuration spécifiques (Docker Compose, Kubernetes, Terraform etc.). L'IA générative peut automatiser la création de ces fichiers en fonction des spécifications du projet, réduisant ainsi le temps et les erreurs humaines.
- **Analyse automatique des environnements**: Le déploiement varie en fonction des providers cloud, des configurations réseau et des contraintes de sécurité. L'IA générative peut analyser automatiquement l'environnement cible et adapter les fichiers ou scripts de déploiement en conséquence.
- **Assistance à la résolution des problèmes**: Lors du déploiement, des problèmes techniques surviennent souvent. L'IA générative peut fournir une assistance en temps réel pour diagnostiquer et résoudre ces problèmes, en suggérant des solutions concrètes.
- **Automatisation de la documentation**: La documentation du code et des marches à suivre est essentielle pour l'exploitation et la maintenance des systèmes. L'IA générative peut automatiser la création de cette documentation technique, permettant de gagner du temps et d'assurer que tous les membres de l'équipe disposent d'informations à jour.
- **Optimisation des processus de déploiement**: L'IA peut analyser les processus de déploiement existants et suggérer des améliorations pour les rendre plus efficaces, en identifiant les goulots

d'étranglement et en proposant des stratégies d'optimisation. L'IA peut aussi aider à concevoir des stratégies de déploiement qui minimisent le temps d'indisponibilité.

Dans ce projet, nous avons choisi de nous concentrer sur le déploiement cloud sur AWS d'une application web déjà existante en utilisant Kubernetes et Terraform. Ainsi, nous avons surtout exploré comment l'IA générative peut assister dans la création des fichiers de déploiement, dans l'adaptation aux environnements cloud, ainsi que dans l'aide à la résolution des problèmes. Ce rapport présente donc les outils utilisés, les résultats de cette exploration, les bénéfices observés, les défis rencontrés, ainsi que des recommandations pour l'intégration future de l'IA générative dans les processus de déploiement IT.

1.3. Problématique

Le déploiement est une étape du SDLC qui peut être complexe et où plusieurs défis peuvent compromettre la mise en production. En voici certains:

- **Complexité des environnements hétérogènes**: Les systèmes sont souvent déployés sur des infrastructures variées (différents fournisseurs ou types) avec des configurations et contraintes différentes, augmentant le risque d'incompatibilités et d'erreurs.
- **Gestion des fichiers de configuration**: La création et la gestion des fichiers de configuration, qui est une étape cruciale, peut être fastidieuse, chronophages car contenant beaucoup de boilerplate code, et sujettes aux erreurs.
- **Gestion des dépendances et des versions**: Les systèmes modernes comportent de nombreuses dépendances (bibliothèques, frameworks, services externes) dont les incompatibilités de versions peuvent provoquer des échecs de déploiement difficiles à diagnostiquer.
- **Gestion des erreurs et diagnostics**: Le déploiement peut échouer pour diverses raisons, et la détection rapide des erreurs ainsi que leur résolution sont essentielles.
- **Documentation chronophage**: L'écriture et la mise à jour de la documentation technique liée au déploiement sont souvent négligées car chronophages et peu engageantes. Cependant, une documentation inadéquate ou obsolète complique la compréhension du processus de déploiement, rendant difficile la maintenance et les mises à jour ultérieures.
- **Interruption de service**: Le déploiement nécessite souvent des interruptions de service impactant la disponibilité du système, particulièrement critique pour les systèmes en production 24/7.

Ces problématiques semblent être des cas de figure où l'IA générative peut potentiellementapporter des solutions efficaces. Le tableau ci-dessous résume les défis du déploiement et les solutions potentielles offertes par l'IA générative.

Défis du déploiement	**Solutions potentielles avec l'IA générative**
Complexité des environnements hétérogènes	Analyse automatique des environnements et génération de fichiers ou scripts adaptés.
Gestion des fichiers de configuration	Automatisation de la création et mise à jour des fichiers de configuration.
Gestion des erreurs et diagnostics	Analyse des logs, détection des erreurs et suggestion de corrections.
Documentation chronophage	Génération automatique de documentation technique pour faciliter la maintenance et l'exploitation du système.
Interruption de service	Conception de stratégies de déploiement pour minimiser les temps d'indisponibilité.

2. Méthodologie

2.1. Approche pédagogique

Dans le cadre de ce projet, l'approche pédagogique adoptée est celle de la classe inversée. De manière générale, le concept est le suivant: les élèves prennent connaissance du sujet du cours en dehors des heures de classe, afin de consacrer le temps en classe à des activités pratiques, des discussions et des travaux collaboratifs engageants. [4] Dans ce projet, la classe inversée est surtout mise en oeuvre à travers des présentations faites par les étudiants eux-mêmes, inversant ainsi les rôles traditionnels entre enseignants et étudiants. En effet, les étudiants se voient attribuer un thème lié au SDLC ou au PLC qu'ils doivent étudier afin de le présenter à leurs pairs et aux enseignants lors de sessions dédiées. En particulier, pour ce projet, les étudiants sont chargés d'explorer l'intégration de l'IA générative dans le thème lié aux projets IT qui leur a été assigné. Le but est de tester et évaluer comment l'IA générative permet d'améliorer l'efficacité, la qualité et la gestion des projets IT à chaque étape de leur cycle de vie. Le côté pratique est mis en avant à travers des démonstrations des résultats obtenus, des analyses critiques et des discussions sur les bénéfices et défis rencontrés. Le contenu des présentations de tous les étudiants fait partie intégrante du cours, puisqu'il peut apparaître dans l'examen final.

2.2. Outils et technologies utilisés

2.2.1. OpenDidac

Notre but étant d'étudier la phase de déploiement, nous avons tout d'abord sélectionné une application web existante à déployer. Nous avons choisi l'application web `OpenDidac`, qui est une plateforme open-source de gestion d'évaluation en ligne développée par l'HEIG-VD [5]. En particulier, cette application permet aux enseignants de concevoir des questionnaires, aux étudiants de se connecter et de répondre aux évaluations, et enfin de récupérer les réponses aux questions pour analyse. Les questionnaires permettent plusieurs types de questions: vrai/faux, QCM, questions ouvertes, questions avec exécution de code ou encore des requêtes de bases de données. L'architecture de l'application est la suivante :

- **Frontend**: Développé en Typescript avec le framework Next.js 14 pour React.
- **Backend**: Développé en Typescript avec le framework Next.js API Routes.
- **Base de données**: PostgreSQL avec modélisation gérée par l'ORM Prisma.
- **Authentification**: Gestion des utilisateurs et des sessions avec NextAuth.js et keycloak.
- **Conteneurisation**: Utilisation de Docker pour la création d'images et la gestion des conteneurs.

Le but de ce projet est donc de déployer cette application web en utilisant différentes technologies d'infrastructure, en s'appuyant sur l'IA générative pour automatiser le processus de déploiement. Le déploiement doit couvrir trois composants distincts: la base de données PostgreSQL, le serveur d'identité Keycloak, et l'application web OpenDidac (frontend et backend) elle-même.

2.2.2. Terraform

Le déploiement d'applications implique nécessairement la gestion d'une infrastructure sous-jacente capable de les héberger et de les exécuter. L'infrastructure peut prendre différentes formes selon les besoins : serveurs physiques, machines virtuelles, conteneurs ou services cloud. Cette infrastructure doit être provisionnée, configurée et maintenue de manière cohérente et reproductible, ce qui représente un défi majeur dans les environnements de production modernes.

Terraform [6] est un outil open-source d'Infrastructure as Code (IaC) développé par HashiCorp qui permet de définir et de provisionner l'infrastructure informatique de manière déclarative. Contrairement aux approches impératives traditionnelles où l'on spécifie les étapes à suivre pour créer l'infrastructure, Terraform adopte une approche déclarative où l'on décrit l'état désiré de l'infrastructure, et l'outil se charge de déterminer les actions nécessaires pour atteindre cet état. Grâce à cette approche, utiliser une IA générative pour créer les fichiers de configuration Terraform

permet d'automatiser le processus de provisionnement de l'infrastructure. Dans ce projet, Terraform est utilisé pour provisionner un cluster Kubernetes sur AWS (Amazon Web Services), qui servira de plateforme d'hébergement pour l'application OpenDidac.

2.2.3. Kubernetes

Une fois l'infrastructure provisionnée, il est nécessaire de disposer d'une plateforme capable d'orchestrer et de gérer le déploiement des applications. Kubernetes [7] répond à ce besoin en fournissant un système d'orchestration de conteneurs puissant et flexible.

Kubernetes est une plateforme open-source d'orchestration de conteneurs développée initialement par Google et maintenue par la Cloud Native Computing Foundation (CNCF). Kubernetes automatise le déploiement, la mise à l'échelle et la gestion des applications conteneurisées, permettant d'exécuter des systèmes distribués de manière résiliente et efficace. La plateforme a été choisie car elle permet de gérer des applications complexes composées avec des simples fichiers de configuration YAML, que l'IA générative peut créer automatiquement.

L'intégration de Terraform et Kubernetes constitue ainsi une chaîne complète d'Infrastructure as Code. Cette approche permet d'automatiser entièrement le processus de déploiement comme cela se ferait dans un environnement professionnel.

2.2.4. Windsurf

Pour intégrer l'IA générative dans notre processus de déploiement, nous avons utilisé Windsurf [8]. Windsurf est un environnement de développement intégré (IDE) de nouvelle génération, intégrant nativement des capacités d'IA générative pour assister les développeurs tout au long du cycle de développement logiciel. En particulier, la fonctionnalité distinctive de Windsurf réside dans son mode Agent (appelé Code), qui permet à l'IA d'agir de manière autonome sur le code et le projet grâce à un accès complet au contexte. En mode Agent, l'IA ne se limite pas à suggérer du code ou répondre à des questions, elle peut analyser l'architecture du projet, proposer des modifications dans les fichiers, créer des fichiers, refactoriser du code existant, générer des tests, créer de la documentation ou encore exécuter des commandes dans le terminal intégré à l'IDE. Pour atteindre le résultat souhaité, l'utilisateur n'a qu'à formuler une demande en langage naturel dans le panneau de conversation intégré, et l'IA se charge de déterminer les actions nécessaires pour accomplir la tâche.

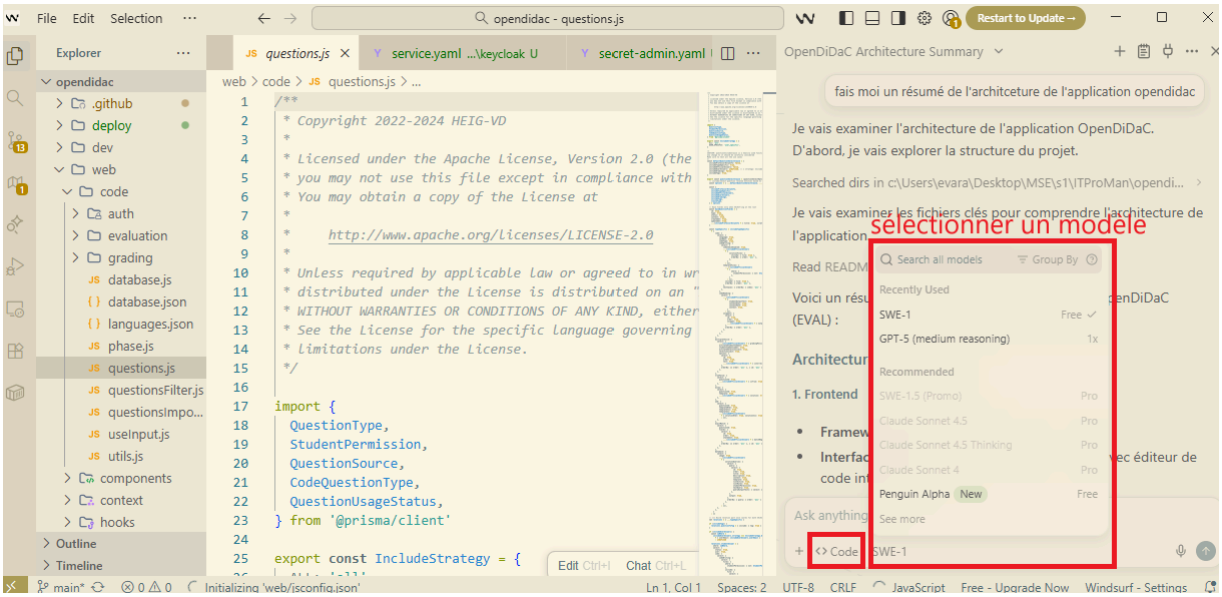

Fig. 3. – Aperçu de l'interface de Windsurf avec les fichiers et le code à gauche, et la conversation et les configurations à droite

Windsurf offre l'accès à plusieurs modèles d'IA avec des capacités et des coûts variables :

- **Modèle de base (SWE-1)**: Un modèle gratuit et illimité adapté aux tâches courantes de développement, permettant une utilisation quotidienne sans consommation de crédits.
- **Modèles premium**: Ces modèles avancés offrent des performances supérieures en termes de compréhension contextuelle, de génération de code complexe et de raisonnement. Chaque requête utilisant un modèle premium consomme 1 crédit.

Un plan gratuit incluant 25 crédits par mois est proposé, permettant aux développeurs de tester les modèles premium pour des tâches nécessitant des capacités avancées ou pour comparer les performances entre les différents modèles. Des plans payants sont également disponibles pour les utilisateurs ayant des besoins plus importants en crédits. Le tarif d'entrée pour 500 crédits par mois pour un utilisateur est de 15 USD. [9]

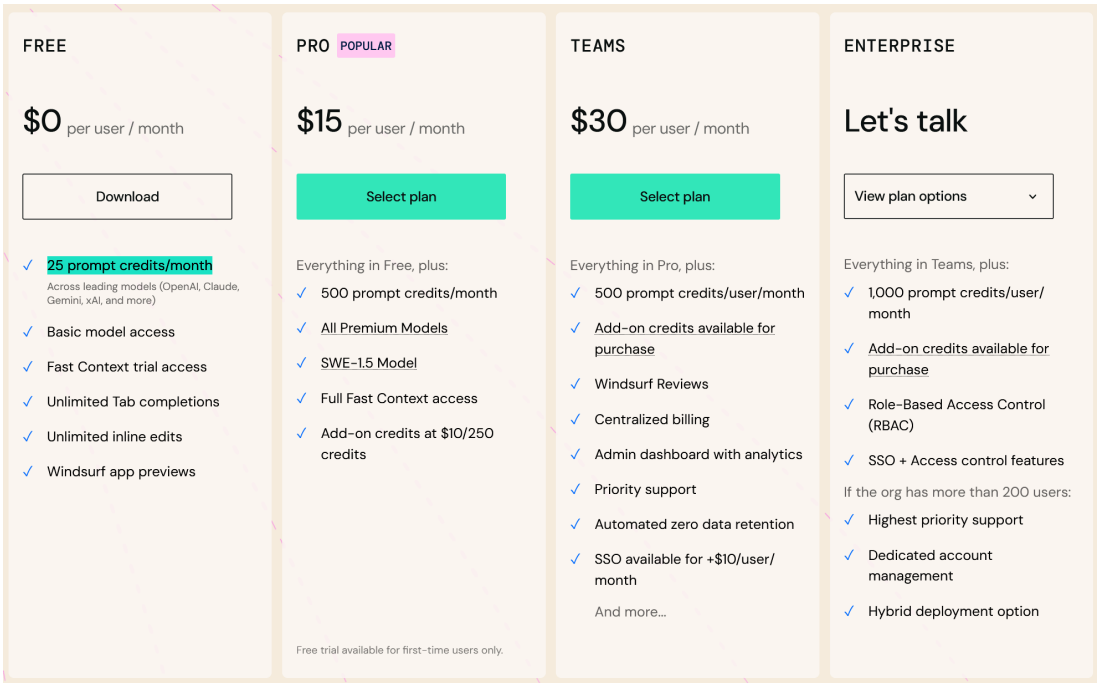

Fig. 4. – Plans de paiements de Windsurf, en décembre 2025

L'utilisation de WindSurd nécessite quelques précautions. En effet, sans contraintes, l'IA peut produire du code non conforme aux standards de l'entreprise, introduire des vulnérabilités de sécurité ou utiliser des patterns incompatibles avec l'architecture existante. Face aux défis de qualité et de cohérence du code généré par l'IA, Windsurf propose deux mécanismes de contrôle : les rules et les workflows [10]. Les rules garantissent que le code généré respecte les conventions établies, tandis que les workflows assurent la reproductibilité et la fiabilité des processus critiques comme les déploiements. Ces outils permettent ainsi de maintenir un contrôle qualité tout en bénéficiant de la productivité apportée par l'IA.

Les rules permettent de définir des directives que l'IA doit suivre lors de la génération de code. Ces règles peuvent être configurées à deux niveaux [10] :

- **Global Rules**: Applicables à tous les projets de l'utilisateur, elles définissent des préférences personnelles comme le style de code ou les conventions de nommage.
- **Project Rules**: Spécifiques à un projet, elles permettent d'établir des guidelines organisationnelles, comme des standards d'architecture, des patterns de sécurité ou des contraintes techniques propres à l'entreprise. Idéalement, chaque entreprise utilisant windsurf devrait définir un ensemble de Project Rules pour garantir la cohérence et la qualité du code généré par l'IA au sein de ses projets.

Windsurf plusieurs modes d'application des rules, dont [10]:

- **Mode Always On**: Les rules sont systématiquement appliquées à chaque interaction avec l'IA, garantissant une conformité constante.
- **Mode Model Decision**: L'IA décide de manière autonome quand appliquer les rules en fonction du contexte de la requête, offrant plus de flexibilité.

Les workflows, quant à eux, permettent d'automatiser des tâches répétitives en créant des séquences d'actions guidées. Par exemple, un workflow de déploiement peut enchaîner automatiquement la vérification des tests, la construction de l'application, la génération de documentation et le déploiement sur l'environnement cible. L'utilisateur invoque le workflow comme une commande dans la conversation, et Windsurf guide ensuite le processus étape par étape, en demandant les paramètres nécessaires et en exécutant les actions définies.

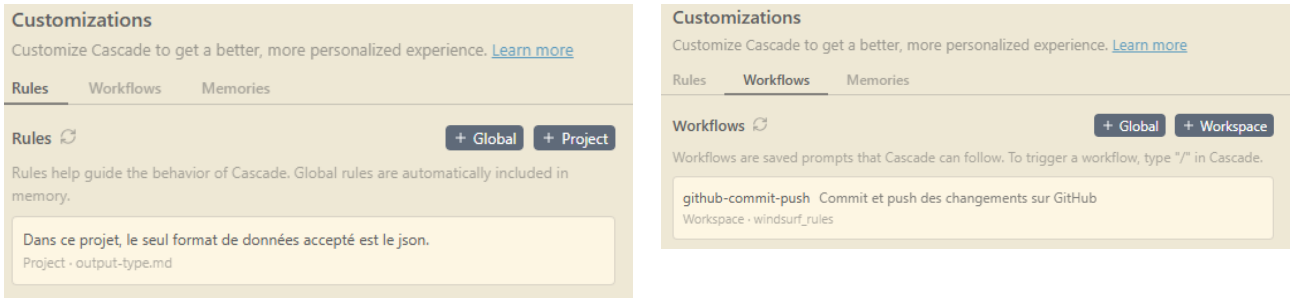

Fig. 5. – Onglets des rules (à gauche) et des workflows (à droite) dans Windsurf

2.2.5. Modèle GPT-5 (medium reasoning)

Pour ce projet, le modèle d'IA générative utilisé est GPT-5 (medium reasoning), lancé par OpenAI le 7 août 2025. Le choix de ce modèle a été appuyé par plusieurs benchmarks démontrant de ses bonnes performances dans des tâches de programmation et de raisonnement complexe, le rendant parfaitement adapté au développement en mode agentique. En particulier, les benchmarks suivant ont été consultés :

- **Aider Polyglot**: Un benchmark open-source évaluant les capacités de différents modèles LLM à suivre des instructions et à modifier du code correctement sans intervention humaine. GPT-5 (medium reasoning) est classé comme second meilleur modèle dans ce benchmark à la date du 12 décembre 2025. [11]
- **SWE (Software Engineering Evaluation)**: Un benchmark spécialisé évaluant la capacité de différents modèles LLM à résoudre des issues GitHub liées à la programmation. C'est un benchmark qui mesure le raisonnement agentique. GPT-5 (medium reasoning) a surpassé plusieurs autres modèles, arrivant en cinquième position à la date du 12 décembre 2025. [12]

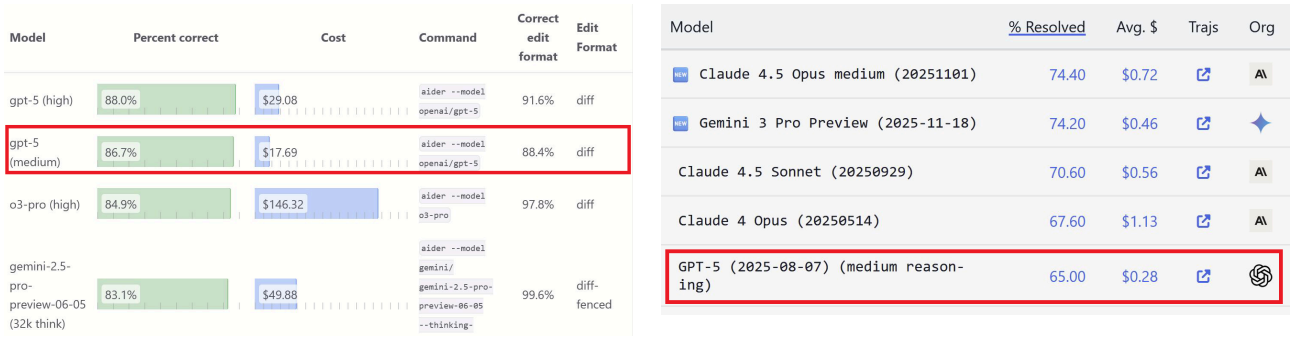

Fig. 6. – Résultats de benchmarks pour GPT-5 medium reasoning dans Aider Polyglot (à gauche) et SWE (à droite)

Plusieurs modèles ont de meilleurs résultats dans ces benchmarks, mais ils ne sont pas disponibles dans le tiers gratuit de Windsurf, ou alors ils sont trop coûteux en crédits pour être utilisés dans le cadre de ce projet. Il est cependant intéressant de noter qu'entre le moment où le projet a été réalisé

(novembre-décembre 2025) et la rédaction de ce rapport (janvier 2026), OpenAI a lancé un nouveau modèle, GPT-5.2, qui surpasse largement les performances de GPT-5 (medium reasoning) dans le benchmark SWE, les résultats pour Aider Polyglot n'étant pas encore disponibles. Cela met en avant la rapidité d'évolution des modèles d'IA générative et souligne l'importance de rester à jour avec les dernières avancées pour maximiser les bénéfices dans les projets futurs.

2.3. Participants

Ce projet a été réalisé par trois étudiants de l'orientation Computer Science du Master of Science in Engineering (MSE) de la HES-SO. Les membres sont tous des ingénieurs détenteurs d'un Bachelor of Science en informatique et systèmes de communication avec une orientation en informatique logicielle, délivré par la HEIG-VD. Les participants sont Eva Ray, Samuel Roland et Massimo Stefani.

Les trois membres ont suivi un cours de Cloud Computing durant leur Bachelor et suivent actuellement une version avancée de ce cours dans le cadre de leur Master. Ainsi, ils possèdent tous des connaissances de base en déploiement d'applications Cloud. Cependant, Samuel et Eva sont débutants en Kubernetes et Terraform, tandis que Massimo a une solide expérience de ces outils qu'il a utilisés dans des projets professionnels antérieurs, ainsi que dans le cadre de son travail de Bachelor.

Pour ce projet, le travail a été divisé en trois parties, de la manière suivante :
- Eva Ray : Déploiement Kubernetes de la base de données PostgreSQL et de keycloak
- Samuel Roland : Déploiement Kubernetes de l'application web d'OpenDidac
- Massimo Stefani : Déploiement Terraform de l'infrastructure Kubernetes sur AWS

2.4. Collecte de données

Pour nous informer sur l'utilisation de l'IA générative dans le déploiement IT, nous avons parcouru plusieurs articles académiques disponibles sur internet, en utilisant des plateformes comme arXiv.org pour accéder à des publications récentes (voir Chapitre 3). En ce qui concerne l'utilisation de Windsurf, nous avons en partie consulté des blogs spécialisés et en partie exploré directement l'outil par nous-mêmes. Nous avons également consulté des benchmarks disponibles sur internet comparant les performances des modèles d'IA générative dans des tâches de programmation, afin de choisir le modèle le plus adapté pour notre projet (voir Chapitre 2.2.5).

En ce qui concerne la rédaction des prompts, nous avons surtout expérimenté de manière itérative, en testant différentes formulations et en ajustant les instructions données à l'IA générative pour obtenir les résultats souhaités, si besoin. Certaines recommandations entendues dans des podcasts spécialisés sur l'IA générative ont aussi été testées.

Au niveau des données collectées à analyser, les prompts utilisés, les réponses obtenues par l'IA générative, des captures d'écran ainsi que les fichiers de configuration générés ont été sauvegardés tout au long du projet. Notre protocole est simple, chaque discussion avec l'IA générative est extraite de WindSurf au format Markdown et publiée dans un repository GitHub dédié au projet.

2.5. Critères d'évaluation

Pour évaluer la pertinence des résultats obtenus avec l'IA générative, les critères suivants ont été utilisés:

- **Fonctionnalité**: Est-ce que l'infrastructure déployée est fonctionnelle ? Est-ce que l'application peut être pleinement utilisée ?
- **Respect des bonnes pratiques**: Est-ce que les bonnes pratiques et recommandations associées aux technologies utilisées sont respectées ?
- **Productivité**: utilisation de l'IA a-t-elle entraîné un gain ou une perte de temps globale ?

- **Sécurité**: Est-ce que l'infrastructure générée respecte les exigences minimales de sécurité (gestion des secrets, configuration réseau, permissions, etc.) L'évaluation ne porte pas sur la sécurité de l'usage de Windsurf lui-même.
- **Coût**: Est-ce que l'utilisation des outils et services d'IA générative a engendré un coût élevé ?

3. Références

Cette section présente les différentes sources consultées pour nous informer sur l'utilisation de l'IA générative dans le déploiement IT, ainsi que les outils cloud utilisés pour le déploiement de l'application OpenDidac. Un état de l'art léger est aussi présenté, regroupant des articles académiques récents et des benchmarks comparant les performances des modèles d'IA générative dans des tâches de programmation.

3.1. Exploration des outils d'IA pour le déploiement IT

Plusieurs outils d'IA générative ont été envisagés:

- **Windsurf** [8]: Outil choisi pour ce projet, car il permet un mode agent avec accès complet au contexte du projet dans un IDE.
- **Cursor** [13]: Outil similaire à Windsurf, mais avec moins de fonctionnalités avancées pour le mode agent, n'a pas été retenu.
- **GitHub Copilot** [14]: Assitant de code intégré dans plusieurs IDE, pas retenu car pas facilement accès au contexte complet du projet.

3.2. Recherches académiques sur le déploiement IT avec l'IA générative

Quelques articles académiques récents explorant l'utilisation de l'IA générative pour le déploiement IT nous ont permis d'avoir un aperçu des avancées dans ce domaine:

- **Multi-Agent Code-Orchestrated Generation for Reliable Infrastructure-as-Code** [15]: Ce travail présente MACOG, une architecture multi-agents destinée à générer du code IaC fiable à partir de descriptions en langage naturel. Les approches classiques en génération unique produisent fréquemment erreurs syntaxiques, violations de politiques et conceptions peu robustes. MACOG répartit la tâche entre des agents spécialisés (architecture, harmonisation fournisseur, ingénierie, revue, sécurité, coûts/capacité, DevOps, mémoire), coordonnés via un tableau partagé et un orchestrateur à états finis, afin de produire des configurations Terraform cohérentes, valides et conformes aux règles. Le système intègre Terraform Plan pour vérifier la validité d'exécution et OPA pour appliquer des politiques personnalisées. Testé sur IaC-Eval, MACOG apporte les meilleures améliorations : par exemple, GPT-5 passe de 54,90 (avec RAG) à 74,02, et Gemini-2.5 Pro de 43,56 à 60,13, avec des hausses simultanées sur BLEU, CodeBERTScore et une évaluation par LLM. Les ablations montrent que le décodage contraint et le retour de déploiement sont essentiels : leur suppression fait chuter les scores à 64,89 et 56,93.
- **Deployability-Centric Infrastructure-as-Code Generation: An LLM-based Iterative Framework** [16]: Cette étude présente un cadre fondé sur des LLM pour générer automatiquement des templates IaC réellement déployables à partir de descriptions en langage naturel. Les approches existantes se limitent surtout à la vérification syntaxique, alors que la déployabilité est l'enjeu principal. Les auteurs introduisent deux apports : IaCGen, un système itératif qui améliore progressivement les templates jusqu'à ce qu'ils se déploient, et DPIaC-Eval, un benchmark de 153 scénarios mesurant syntaxe, déploiement, intention utilisateur et sécurité. Les modèles actuels (Claude-3.5, Claude-3.7) ne dépassent pas 30 % de déploiements réussis au premier essai. Avec IaCGen, ils franchissent 90 % après 25 itérations et atteignent jusqu'à 98 %. Malgré ces progrès, l'alignement avec l'intention (25,2 %) et la conformité sécurité (8,4 %) restent très faibles, montrant des limites majeures à résoudre.

Ces deux articles nous mènent à la conclusion que l'IA générative a un potentiel significatif pour automatiser la création de code d'infrastructure mais qu'il faut privilégier des approches itératives pour atteindre des résultats déployables.

3.3. Benchmarks de comparaison des modèles d'IA générative

Quelques benchmarks récents comparent les performances des modèles d'IA générative pour des tâches de programmation et nous ont aidé à choisir le modèle le plus adapté pour notre projet (voir Chapitre 2.2.5):

- **Aider Polyglot** [11]
- **SWE (Software Engineering Evaluation)** [12]
- **Vellum** [17]: Leaderboard regroupant plusieurs benchmarks pour comparer les modèles d'IA générative dans divers tâches et domaines.

3.4. Outils cloud

Nous avons utilisés plusieurs outils cloud pour le déploiement de l'application OpenDidac:

- **AWS (Amazon Web Services)** [18]: Fournisseur cloud choisi pour héberger l'infrastructure Kubernetes.
- **Kubernetes** [7]: Système d'orchestration de conteneurs utilisé pour déployer et gérer l'application.
- **Terraform** [6]: Outil d'infrastructure as code utilisé pour provisionner l'infrastructure cloud sur AWS.

4. Résultats et analyses

4.1. Phase de déploiement

La phase de déploiement a été réalisée en deux étapes distinctes mais complémentaires : d'abord le déploiement de l'application OpenDidac et de ses dépendances sur un cluster Kubernetes, puis le provisionnement de l'infrastructure cloud via Terraform. Cette approche progressive a permis de valider le fonctionnement de l'application dans un environnement local avant de la déployer sur une infrastructure cloud.

4.1.1. Prompts utilisés

Des prompts ont été utilisés pour créer les fichiers de déploiement Kubernetes et Terraform. Voici les prompts de départ utilisés pour chaque étape:

- **Déploiement PostgreSQL sur Kubernetes**: *Je veux déployer la db postgres de l'application (pas celle de keycloak) dans un cluster kurbernetes. Crée les fichiers yaml nécessaires.*
- **Déploiement Keycloak sur Kubernetes**: *Je veux déployer l'identification avec keycloak dans un cluster kubernetes. Crée les fichiers yaml nécessaires.*
- **Déploiement OpenDidac sur Kubernetes**: *Tu es un expert en déploiement Kubernetes. J'aimerai que tu me déploies cette application Opendidac, sur la partie frontend/backend (NextJS). J'ai déjà une partie du déploiement réalisé pour la base de donnée de l'application et pour keycloak, déjà présent dans le dossier kube. J'aimerais que tu continues dans le dossier app pour générer tous les fichiers nécessaires à lancer l'application au complet. Définis moi un Pod pour l'app NextJS et un load balancer devant.*
- **Déploiement Terraform sur AWS**: *okay tu peux me créer un code Terraform capable de déployer une infrastructure kubernetes j'aimerais un master et 3 enfants sur aws. Pour ce faire part du principe que j'ai déjà les credentials AWS dans ma machine. En ce qui concerne la clef SSH pour les machine on va aussi la créer à la volée pour automatiser au plus la chose. On va installer k3s car c'est plus facile.*

On constate des styles différents entre les prompts, reflétant les préférences individuelles des membres de l'équipe. Certains prompts sont plus détaillés, tandis que d'autres sont plus concis. Cette diversité

a finalement permis d'explorer différentes approches pour guider l'IA générative dans la création des fichiers de déploiement. Pour les déploiements Kubernetes, ces prompts se sont avérés suffisants pour obtenir des résultats fonctionnels immédiatement. En revanche, pour le déploiement Terraform, des ajustements et des itérations supplémentaires ont été nécessaires pour atteindre un résultat pleinement opérationnel. Les discussions complètes avec l'IA générative, incluant les prompts, les réponses et les fichiers générés, sont disponibles dans le repository GitHub du projet mis en annexe.

4.1.2. Architecture et connectivité

L'architecture applicative dans Kubernetes finale se compose de trois composants:

- **PostgreSQL**: Base de données relationnelle hébergeant les données de l'application OpenDidac.
- **Keycloak**: Serveur d'identité gérant l'authentification et l'autorisation des utilisateurs.
- **Application web**: Backend et frontend de l'application OpenDidac.

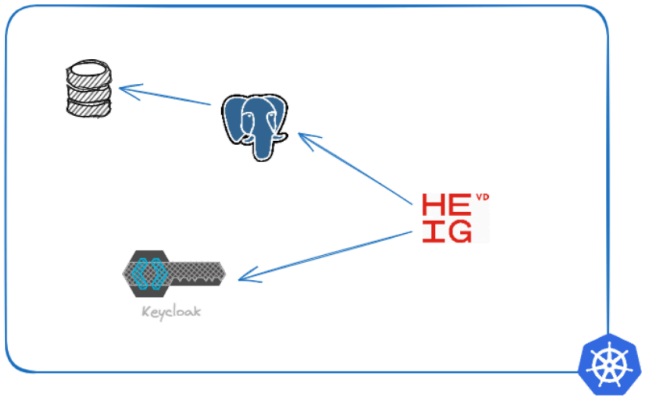

Fig. 7. – Architecture applicative dans Kubernetes.

4.1.3. Déploiement Kubernetes

Le déploiement sur Kubernetes a été réalisé en trois phases correspondant aux composants principaux de l'application : la base de données PostgreSQL, le serveur d'identité Keycloak, et l'application web OpenDidac. Ci-dessous, les résultats obtenus pas l'IA générative pour chaque composant sont présentés.

4.1.3.1. Bonnes pratiques et organisation

Le déploiement Kubernetes a été structuré en respectant les bonnes pratiques de l'écosystème, notamment :

- **Kustomize** : Utilisation de `kustomization.yaml` dans chaque dossier de composant pour faciliter la gestion des manifestes et permettre des déploiements déclaratifs via `kubectl apply -k`.
- **Namespaces** : Isolation logique de tous les composants dans un namespace dédié `opendidac`, séparant les ressources de l'application du reste du cluster.
- **Secrets** : Stockage sécurisé des informations sensibles (credentials de base de données, clés d'authentification) dans des objets Kubernetes Secret, encodés en base64 et injectés comme variables d'environnement.
- **Services** : Exposition cohérente des composants via des Services ClusterIP pour la communication interne et LoadBalancer pour l'accès externe.
- **Health checks** : Configuration systématique de readiness et liveness probes pour garantir la santé des applications.

4.1.3.2. Base de données PostgreSQL

Le déploiement de PostgreSQL a constitué la première étape, car il s'agit de la dépendance critique dont dépendent les autres composants. L'IA générative a créé les manifestes suivants :

- **StatefulSet** : Contrairement à un Deployment classique, un StatefulSet a été utilisé pour garantir une identité stable et un ordre de démarrage prévisible, essentiel pour une base de données.
- **PersistentVolume** : Configuration d'un PVC (PersistentVolumeClaim) de 10Gi via `volumeClaimTemplates`, assurant la persistance des données même en cas de redémarrage du pod.
- **Variables d'environnement** : Les credentials par défaut ont été définis dans un Secret (`POSTGRES_USER`, `POSTGRES_PASSWORD`, `POSTGRES_DB`), en s'inspirant des valeurs utilisées dans l'environnement de développement Docker existant.
- **Health checks** : Implémentation de readiness et liveness probes utilisant la commande `pg_isready` pour vérifier la disponibilité de la base avant d'accepter des connexions.

Vérification pour valider le déploiement :

```
massimostefani@fedora:~/Master/ProjMang/ItProMan_project/keyclock$ kubectl get all -n opendidac
NAME                             READY   STATUS    RESTARTS   AGE
pod/keycloak-7ffc78f65c-89766    1/1     Running   0          23m
pod/opendidac-web                1/1     Running   0          16m
pod/postgres-0                   1/1     Running   0          61m

NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/keycloak        NodePort    10.43.69.179    <none>        8080:30880/TCP   60m
service/opendidac-web   NodePort    10.43.172.142   <none>        80:30080/TCP     59m
service/postgres        ClusterIP   10.43.171.95    <none>        5432/TCP         61m

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/keycloak   1/1     1            1           60m

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/keycloak-55775c8b5f   0         0         0       43m
replicaset.apps/keycloak-66759c7bbc   0         0         0       60m
replicaset.apps/keycloak-78f484bb78   0         0         0       37m
replicaset.apps/keycloak-7ffc78f65c   1         1         1       23m
replicaset.apps/keycloak-b5745ffd     0         0         0       31m
replicaset.apps/keycloak-bf449d548    0         0         0       30m

NAME                          READY   AGE
statefulset.apps/postgres     1/1     61m
```

Fig. 8. – Validation du déploiement Kubernetes.

4.1.3.3. Keycloak (Identity Provider)

Le déploiement de Keycloak a présenté plus de complexité, nécessitant une configuration post-déploiement et la résolution d'erreurs liées aux variables d'environnement, qui a finalement dû être faite à la main. L'IA générative a généré :

- **Deployment** : Configuration d'un Deployment Keycloak avec l'image officielle `quay.io/keycloak/keycloak`.
- **Variables d'environnement** : Définition des credentials administrateur via un Secret (`KEYCLOAK_ADMIN`, `KEYCLOAK_ADMIN_PASSWORD`) et configuration de la connexion à la base de données PostgreSQL.
- **Service et Ingress** : Exposition de Keycloak via un Service ClusterIP et un Ingress pour permettre l'accès externe avec un nom de domaine personnalisé.

Après le déploiement, une configuration manuelle dans l'interface Keycloak a été nécessaire pour :

1. Créer un realm dédié pour l'application OpenDidac.
2. Configurer un client OIDC avec les URLs de callback appropriées.
3. Créer des utilisateurs de test avec les rôles nécessaires.

L'IA générative a fourni une marche à suivre fonctionnelle pour mener à bien cette configuration.

Résolution d'erreurs : L'IA générative a aidé à diagnostiquer et corriger plusieurs erreurs :

- **Erreur de connexion à la base** : L'IA a identifié que la variable `KC_DB_URL` devait pointer vers le Service PostgreSQL (`postgres.opendidac.svc.cluster.local`) plutôt qu'une adresse externe.

4.1.3.4. Application web OpenDidac

Le déploiement de l'application web a nécessité une approche plus sophistiquée pour gérer les migrations de base de données et l'intégration avec Keycloak :

- **Pod avec Init Container** : Utilisation d'un init container pour exécuter les migrations Prisma (`npx prisma migrate deploy`) avant le démarrage de l'application principale. Cet init container attend que la base de données soit prête et réessaye jusqu'à 30 fois avec un délai de 5 secondes.
- **Variables d'environnement** : Configuration complète via un Secret incluant `DATABASE_URL`, `NEXTAUTH_URL`, `NEXTAUTH_SECRET`, et les paramètres de connexion à Keycloak.
- **Health checks** : Configuration de readiness probe (délai initial de 15s) et liveness probe (délai de 30s) sur l'endpoint racine pour garantir que Next.js est complètement initialisé avant d'accepter du trafic.
- **Service et Ingress** : Exposition de l'application via un Service ClusterIP et un Ingress pour l'accès externe.

4.1.4. Déploiement Terraform

Le provisionnement de l'infrastructure pour héberger le cluster Kubernetes sur AWS a constitué un défi significatif, révélant les limites de l'IA générative face à des problématiques techniques complexes.

4.1.4.1. Bonnes pratiques non respectées

Contrairement au déploiement Kubernetes, le code Terraform généré ne respectait pas toujours les bonnes pratiques :

- **Pas de modules** : Le code était monolithique, sans réutilisation via des modules Terraform.
- **Commentaires excessifs ou manquants** : Alternance entre des commentaires trop verbeux et des sections sans documentation.
- **Pas de validation** : Aucune validation des variables.

Une refactorisation majeure a donc été demandée pour encapsuler toute la logique dans des structures de données complexes. Ces manquements illustrent que l'IA générative excelle dans la génération rapide de code fonctionnel, mais nécessite une supervision humaine pour les aspects de qualité et de maintenabilité.

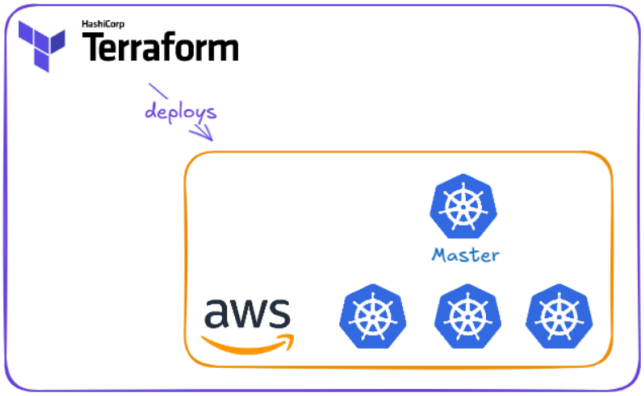

Fig. 9. – Architecture AWS.

4.1.4.2. Succès final

Après de nombreuses itérations (environ 10-15 échanges avec l'IA, consultables en annexe) et quelques ajustements manuels, l'infrastructure a finalement été provisionnée avec succès :

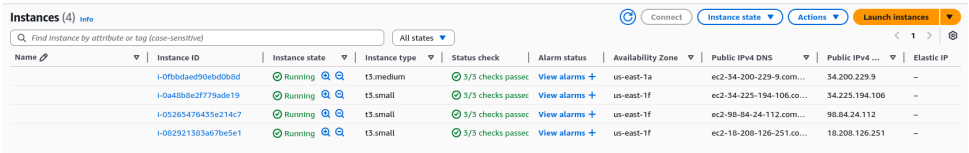

Fig. 10. – Infrastructure AWS provisionnée avec succès.

Le cluster K3s était opérationnel, accessible via le kubeconfig généré, et prêt à recevoir les déploiements Kubernetes créés précédemment :

```
massimostefani@fedora:~/Master/ProjMang/ItProMan_project/infra$ export KUBECONFIG=./k3s.yaml
massimostefani@fedora:~/Master/ProjMang/ItProMan_project/infra$ kubectl get nodes
NAME               STATUS   ROLES           AGE     VERSION
ip-172-31-15-104   Ready    control-plane   4m36s   v1.34.3+k3s1
ip-172-31-75-26    Ready    <none>          3m44s   v1.34.3+k3s1
ip-172-31-78-180   Ready    <none>          3m45s   v1.34.3+k3s1
ip-172-31-78-69    Ready    <none>          3m44s   v1.34.3+k3s1
```

Fig. 11. – Accès au cluster K3s via kubeconfig.

4.1.5. Déploiement complet

L'intégration des deux phases a permis de réaliser un déploiement complet de l'application OpenDidac sur une infrastructure cloud AWS. Quelques changements mineurs ont été nécessaires pour adapter les configurations Kubernetes à l'environnement cloud, notamment la gestion des nouveaux endpoints et des adresses IP.

Ces changements ont également été effectués à l'aide de l'IA. Bien que toutes les bonnes pratiques ne soient pas respectées et que la puissance de tous les nœuds ne soit pas pleinement utilisée, l'application reste fonctionnelle et accessible.

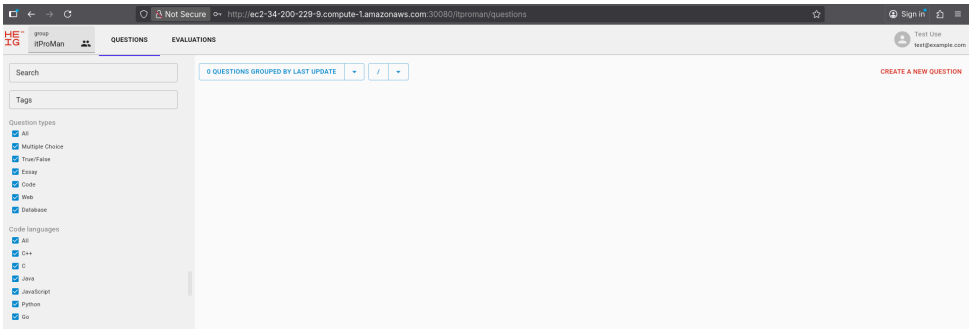

Fig. 12. – Déploiement complet d'OpenDidac sur AWS.

Pour tout essai, l'application reste disponible à l'adresse suivante : http://ec2-34-200-229-9.compute-1.amazonaws.com:30080

Utilisateur professeur :

- Username : `testuse`
- Password : `change-me`

4.2. Synthèse des resultats

Nous estimons que 95% des fichiers de déploiement Kubernetes finaux ont été générés par l'IA, contre 80% pour les fichiers Terraform. Le reste du code a été écrit manuellement pour corriger des erreurs ou ajuster des configurations spécifiques lorsque nous n'avons pas réussi à obtenir le résultat souhaité avec l'IA générative après plusieurs tentatives. Ainsi, nous estimons qu'environ 87.5% du code total de déploiement a été généré avec l'aide de l'IA.

Type de fichier	Pourcentage du code généré par l'IA dans le déploiement final
Fichiers Kubernetes	95%

Fichiers Terraform	80%
Total	87.5%

4.2.1. Comparaison avec une approche traditionnelle

La comparaison entre l'approche par IA générative et une approche traditionnelle révèle des différences dans la structure du code produit.

- **Kubernetes**: L'IA générative a produit une architecture modulaire avec génération de multiples fichiers séparés respectant les conventions de la communauté. Sans connaissance préalable de Kubernetes, une approche manuelle aurait abouti à une structure plus monolithique avec moins de séparation des responsabilités et un respect moins rigoureux des bonnes pratiques établies.
- **Terraform**: L'IA générative a généré du code présentant des répétitions notables pour des ressources similaires où seuls les noms diffèrent. Une approche manuelle traditionnelle aurait privilégié la modularisation avec abstraction et paramètres pour éviter cette duplication.

4.2.2. Évaluation selon les critères définis

Cette section évalue les résultats obtenus selon les cinq critères définis dans la méthodologie (Chapitre 2.5).

Fonctionnalité : L'infrastructure déployée est pleinement fonctionnelle. L'application OpenDidac est accessible et utilisable avec toutes ses fonctionnalités. Le déploiement Kubernetes sur AWS permet une exploitation complète du système. Une intervention manuelle mineure a été nécessaire.

Respect des bonnes pratiques :

- **Kubernetes**: l'IA a systématiquement respecté les bonnes pratiques de la communauté (namespaces, Kustomize, Secrets, health checks, services).
- **Terraform**: Le code généré présentait des lacunes significatives : absence de modularisation, duplication excessive, commentaires inadéquats, et manque de validation des variables. Une refactorisation manuelle a été nécessaire.

Productivité :

- **Kubernetes**: Gain de temps net, avec du code généré rapidement et de bonne qualité.
- **Terraform**: Processus chronophage, avec plus de 10-15 itérations nécessaires et corrections manuelles, aboutissant à une perte de temps globale par rapport à une approche manuelle directe.

Sécurité : L'infrastructure respecte les exigences de base : gestion des secrets via Kubernetes Secrets, isolation des composants dans un namespace dédié, et configuration réseau appropriée. Cependant, aucune analyse de sécurité approfondie n'a été proposée (scan de vulnérabilités, audit des permissions IAM AWS, …).

Coût : Le coût a été de 15 USD car nous avons finalement dépassé le quota gratuit de 25 crédits mensuels en configurant Terraform. Ce coût est raisonnable mais exploserait pour des projets plus complexes nécessitant beaucoup de crédits et plusieurs comptes.

5. Discussion

5.1. Bénéfices observés

L'utilisation de l'IA générative dans le processus de déploiement IT a apporté plusieurs bénéfices notables, tant au niveau général qu'au niveau spécifique des technologies utilisées, à savoir Kubernetes et Terraform.

Au niveau général :

- Le temps d'amorçage du projet a été considérablement réduit grâce à l'IA générative, qui a permis de générer rapidement tous les fichiers de configuration de base.

- L'utilisation de l'IA générative permet une approche d'expérimentation rapide. En effet, il est très facile d'obtenir rapidement un déploiement fonctionnel, même sans maîtrise approfondie des technologies utilisées. Cela permet de tester différentes configurations et d'itérer rapidement mais est évidemment à éviter en production sans une revue humaine approfondie.
- L'IA générative permet la decouvertes de nouvelles approches que le développeur n'aurait pas envisagées. Par exemple, dans notre cas, l'IA a suggéré l'utilisation de certaines organisations de fichiers et de ressources Kubernetes que nous n'aurions pas pensé à utiliser, à cause de notre manque d'expérience avec cet outil.
- La résolution d'erreurs classiques est grandement facilitée par l'IA générative, qui peut analyser les messages d'erreur et proposer des solutions adaptées. Cela a permis de surmonter rapidement des obstacles techniques, sans devoir lire en détail la documentation officielle, comme illustré dans l'exemple ci-dessous.

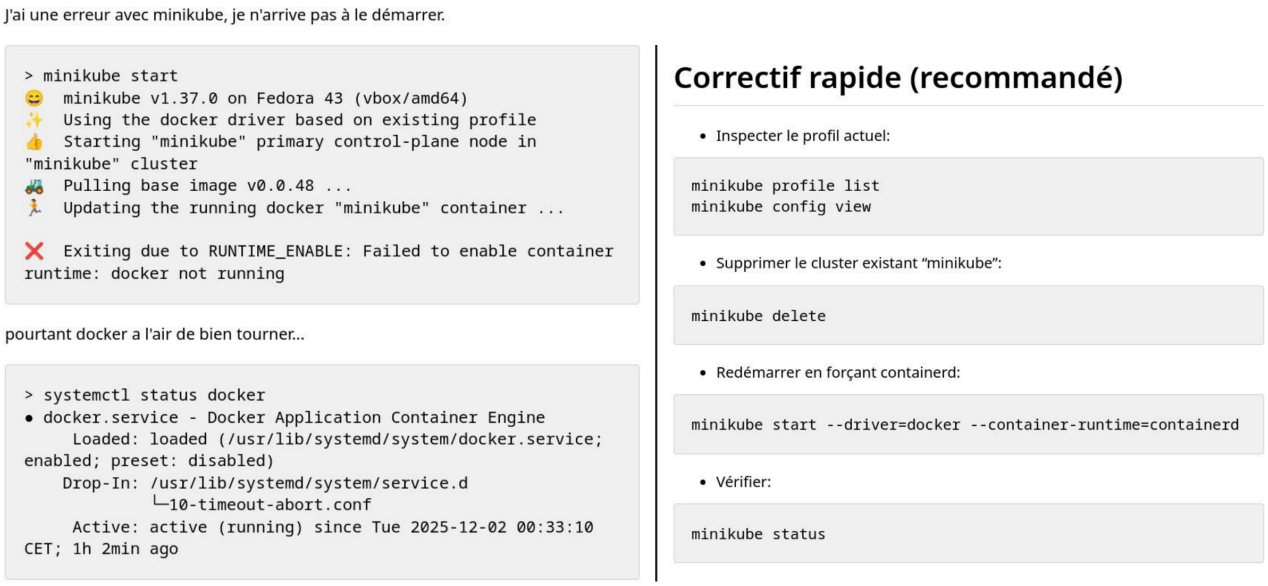

Fig. 13. – Exemple d'aide à la résolution d'erreur par l'IA générative

Pour Kubernetes :

- Grâce à l'IA générative, il n'y a pas besoin de connaître Kubernetes en profondeur pour réussir un déploiement. L'IA peut générer les fichiers YAML nécessaires et guider l'utilisateur à travers le processus de déploiement parfaitement.
- Le gain de temps sur le boilerplate code des fichiers de déploiement est significatif en générant les fichiers de configuration par IA générative.
- Lorsqu'on est débutant, l'IA générative permet un énorme gain de qualité car elle suit les bonnes pratiques recommandées par la communauté Kubernetes diligemment.

Pour Terraform :

- Le seul bénéfice observé est le gain de temps initial pour la création des fichiers de configuration Terraform.

Synthèse :

Catégorie	Principaux bénéfices
Général	• Réduction du temps d'amorçage du projet • Expérimentation et itération rapides • Découverte de nouvelles approches • Résolution facilitée des erreurs classiques

Kubernetes	• Déploiement sans expertise approfondie • Gain de temps sur le boilerplate • Gain en qualité (bonnes pratiques respectées)
Terraform	• Gain de temps pour la création initiale des fichiers

5.2. Défis rencontrés

Pour Kubernetes :

- **Fusion des 3 parties difficile**: L'IA a eu du mal à fusionner les trois parties du déploiement (PostgreSQL, Keycloak, OpenDidac) en un seul déploiement cohérent. Chaque partie fonctionnait bien individuellement, mais lorsque nous avons demandé à l'IA de les combiner, elle a eu du mal à gérer les dépendances entre les composants, notamment la connexion d'OpenDidac à Keycloak.
- **Compréhension et direction générale**: Quand nous étions bloqué sur cette fusion, nous étions obligés de comprendre l'architecture de communication entre conteneurs, pour comprendre les erreurs liée à Keycloak. Comme le LLM tournait en rond, nous avons du choisir une autre direction, c'est à dire de revenir sur une ancienne version de l'application qui utilisait une instance Keycloak en local (commit 732487f98226c37825cbe600be6cdf59f7289e18). Pour choisir cette direction, nous avons du lire le README, qui mentionnait clairement que Keycloak en local ne pouvait plus fonctionner. Nous aurions gagné du temps si le LLM avait également lu le README et s'il nous avait tout de suite demandé de faire une choix parmi une liste d'options possibles.
- **Verbosité des réponses**: Durant les phrases de réflexions et de résultats final, le LLM génère beaucoup de texte, souvent très verbeux. Une partie du texte est complètement inutile pour les humains. Par exemple, les « pensées » du LLM peuvent être très longues et inutiles: « Je vais mettre à jour le fichier terraform.tfvars.json pour adopter la nouvelle structure d'objet cluster, puis je mettrai à jour la todo list pour marquer la revue des tfvars comme terminée. ». Au milieu des dizaines de lignes de texte, certaines informations ou choix importants sont facilement loupés par une lecture humaine. On est très loin d'un expert coach qui peut se concentrer sur les décisions à prendre, à rendre l'information concise et aider à avancer à débloquer les problèmes un par un.
- **Direction vague**: Quand il y a des problèmes et que l'humain n'a pas pris de décision claire sur la direction à prendre, le LLM navigue dans différentes directions à la fois et ne reste pas concentré sur une seule.

Pour Terraform :

- **Prérequis**: l'IA ne mentionne pas les prérequis d'accès ou de permissions nécessaires au déploiement sur AWS, il faut connaître un minimum le fonctionnement des services et avoir configuré sa machine pour être connecté à AWS.
- **Qualité du code Terraform moyenne/mauvaise**: Massimo a été confronté à une qualité de code moyenne voir mauvaise, notamment à cause une quantité importante de duplications. Au lieu de créer des abstractions, l'IA qui ne peut pas se fatiguer des copier-coller, duplique sans problème des dizaines d'éléments et avec le même paramètre légèrement adapté. Le LLM part régulièrement dans des directions trop complexes, alors que des alternatives plus simples existent. Massimo a du lui demander plusieurs fois d'améliorer la structure pour améliorer la lisibilité. L'hypothèse est que contrairement au YAML pour Kubernetes, il existe plusieurs approches valides dans le language de Terraform (HCL). La meilleure approche n'est pas toujours choisie par l'IA.
- **Guidage important**: Pour corriger la qualité du code moyenne, de nombreuses itérations ont été nécessaires. Le guidage régulier est requis et il doit contenir des pistes précises.
- **Perte de temps**: Cette expérience mitigée, avec la toute fin qui a du être terminée à la main, a globalement été une perte de temps. Plusieurs étapes auraient été plus directement droit au but dans les bonnes pratiques, si toute ou partie avait été rédigée à la main.

- **Les fichiers rules.md non efficace**: Pour tenter que notre LLM utilise directement les bonnes pratiques, nous avons défini un fichier de règles. A nombreuses reprises les directives, pourtant courtes, objectives et claires, n'ont pas été respectées. Il est difficile de déterminer si l'existence de ce fichier ait eu un impact, au vu des mauvais résultats obtenus.
- **Le LLM fonce dans le mur si on le lui demande**: A une occasion, l'approche de correction que Massimo a proposé n'était pas possible techniquement. Pourtant l'IA a généré un résultat qui correspondait à cette approche, contrairement à un expert humain qui aurait su que c'était impossible ou qui aurait pu le découvrir en le testant.

Au niveau général :

- **Tendance à minimiser les interactions**: L'usage des LLM est financé en fonction de l'usage, en comptant chaque message un certain nombre de crédit. Cette approche permet de payer un coût faible à l'essai, mais a tendance à minimiser les prompts.
- **Evaluation difficile de l'impact du prompt engineering**: Il est difficile de juger de la qualité de nos prompts en fonction des résultats, comme les difficultés des tâches ne sont pas comparables. Parfois, une tâche simple peut être très bien faite avec peu d'information. A d'autres occasions, donner un rôle « d'expert Kubernetes » ne suffit pas à le faire devenir expert dans la technologie et la compréhension d'une
- **Générations lentes et ennuyantes**: Cela dépend bien sûr du service et du modèle, mais attendre 5 minutes pour qu'une correction de 5 caractères soit faite est bien frustrante. Le temps de « réflexion » ne varie pas en fonction de la complexité de la tâche. Pour éviter cette limite, nous devrions changer de LLM pour des questions faciles.
- **Léger retard technologique**: Les versions des plugins Terraform utilisés avaient un léger retard, certaines API ont changées depuis. Cette limite est basée sur la date d'entrainement du LLM ou des capacités limitées d'accès aux dernières documentations en ligne. Des hallucinations pourraient aussi en être la cause.
- **Temps de relecture**: Le temps de rédaction étant grandement réduit, le temps de relecture des conversations et des fichiers modifiés devient important.

Synthèse :

Catégorie	**Principaux défis**
Général	• Tendance à minimiser les interactions (coût en crédits) • Évaluation difficile de l'impact du prompt engineering • Générations lentes et parfois ennuyantes • Verbosité excessive des réponses • Léger retard technologique des versions • Temps de relecture important
Kubernetes	• Difficulté à fusionner les composants en un déploiement cohérent • Manque de compréhension de l'architecture globale • Direction vague quand l'humain n'a pas pris de décision claire • L'IA ne lit pas systématiquement la documentation projet (README)
Terraform	• Prérequis AWS non mentionnés • Qualité de code moyenne avec beaucoup de duplication • Guidage important et itérations nombreuses nécessaires • Perte de temps globale comparé à une approche manuelle • Fichiers rules.md peu efficaces

	• L'IA peut foncer dans une impasse technique sans s'en rendre compte

5.3. Retour d'expérience du groupe

Notre retour d'expérience découle directement des bénéfices et défis observés.

- Nous avons constaté que l'IA générative permet de gagner un temps significatif sur les tâches répétitives et automatisables, mais qu'elle reste limitée dès qu'une réflexion ou de l'imagination sont nécessaires.
- Dans le cadre du déploiement Kubernetes, nous avons pu obtenir des configurations fiables et entièrement exploitables, ce qui nous encourage à réutiliser cette approche pour de futurs projets.
- Pour Terraform, l'IA est utile pour générer des fichiers de base, mais nous n'avons pas confiance dans la génération d'un déploiement complet.
- Nous avons trouvé l'IA très pratique pour résoudre rapidement des erreurs simples. Nous continuerons à l'utiliser comme un assistant de dépannage.
- Se cantonner au "vibe coding", c'est-à-dire laisser l'IA générer le code de manière répétitive avec des échanges constants, s'est révélé frustrant, long et peu stimulant. Il est mieux de privilégier une utilisation mixte, combinant intervention humaine et assistance ciblée de l'IA.

5.4. Comparaison avec d'autres approches ou pratiques

Comme le montre la Fig. 14, nous avons défini 5 niveaux d'adoption de l'IA, pour cette comparaison. Tout à gauche, le niveau d'adoption zéro, sans aucune aide d'intelligence artificielle. Ensuite, nous avons les chatbot en ligne qui regroupent ChatGPT, Copilat Chat, et beaucoup d'autres. Ils sont accessibles via des sites web et permettent souvent de choisir entre différents LLM mises à disposition. Ces chatbots en ligne n'ont comme contexte que l'historique de conversation des messages fournis, et peuvent parfois faire également des recherches sur le web. Leur réponse ne peut être donnée que dans l'interface de chat, les changements doivent donc être intégrés à la main. Dans le contexte d'une base de code existante comme Opendidac, il aurait fallu lui donner le contenu de certains fichiers qui nous semblent pertinents, pour qu'il puisse comprendre la structure de l'application, ce qui est fastidieux.

Pour donner par défaut accès à suffisament de contexte, nous avons choisi l'option des IDE dédié, comme Cursor et Windsurf. En effet, le mode agent de Windsurf peut de lui-même décider d'aller lire les fichiers du projets qui lui sont utiles, afin de comprendre le projet ouvert dans l'IDE. Une fois l'architecture identifiée, il peut modifier des fichiers dans n'importe quelle partie du projet. Un panneau de conversation est intégré et permet de demander des modifications.

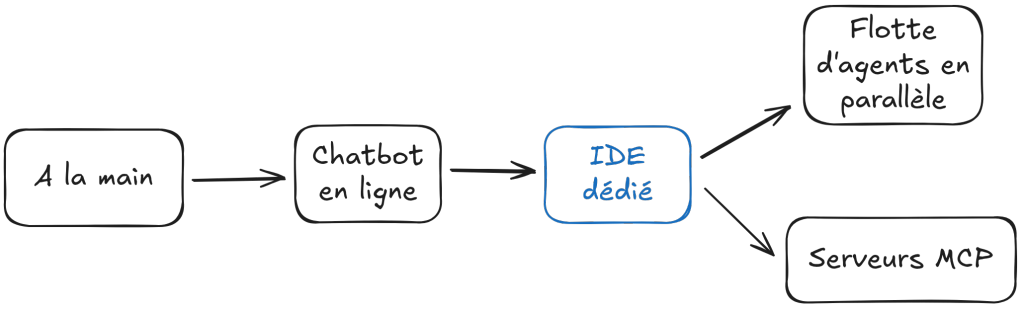

Fig. 14. – Niveaux d'adoption de l'IA, avec notre approche en bleu

Les dernières approches qui poussent encore plus loin l'intégration sont les flottes d'agents en parallèle et les serveurs MCP. Nous n'avons pas exploré ces outils, pour limiter le temps de configuration et d'apprentissage, mais ils pourraient fournir une meilleure expérience. Au lieu d'avoir un seul agent, pourquoi ne pas en avoir plusieurs, avec différents rôles, pour gérer les étapes, découper en tâches, coder, revoir le résultat, tester le déploiement… Ces systèmes de flottes permettent de simuler les rôles

d'une équipe de développement, pour résoudre des tâches plus complexes et une meilleure qualité finale.[1]

L'autre option qui n'est pas incompatible avec la précédente, consiste à donner accès à des outils ou ressources avancées via des serveurs MCP (Model Context Protocol) [19]. Nous aurions pu utiliser un serveur MCP pour Kubernetes, permettant à notre LLM d'accéder à l'état du cluster et de créer des ressources, sans avoir de CLIs installés sur nos machines. Au lieu de nous donner des commandes `kubectl` pour lancer les services, pods et autres fichiers définis, une intégration d'un serveur MCP aurait pu permettre de lancer ces actions tout seul (après certaines approbations). [20]

6. Conclusion

6.1. Points clés

Nous avons réussi à déployer une application web complète (OpenDidac) sur un cluster Kubernetes hébergé sur AWS, en utilisant principalement de l'IA générative pour créer les fichiers de configuration nécessaires. Le déploiement Kubernetes a été particulièrement réussi, avec 95% du code généré par l'IA, respectant les bonnes pratiques de la communauté. Le déploiement Terraform a été plus complexe, nécessitant environ 80% de code généré par l'IA, mais avec des ajustements manuels pour améliorer la qualité et la maintenabilité du code. Ce déploiement a été réalisé dans l'IDE Windsurf, qui permet une intégration fluide avec l'IA générative.

Nous avons identifié plusieurs bénéfices, notamment le gain de temps significatif sur les tâches répétitives, la possibilité d'expérimentation rapide, et l'aide précieuse pour résoudre des erreurs courantes. Cependant, des défis subsistent, tels que la nécessité d'une supervision humaine pour garantir la qualité du code, les limitations dans la compréhension globale du projet par l'IA, et la gestion des directions techniques complexes. Globalement, cette expérience démontre le potentiel de l'IA générative pour accélérer les déploiements IT, tout en soulignant l'importance d'une collaboration étroite entre l'humain et la machine.

6.2. Bilan général

Le déploiement par IA fonctionne correctement jusqu'à un certain niveau de complexité. Tout le code « boilerplate », une fois la direction choisie, peut facilement être rédigé par IA. Une fois la base définie, nous avons pu mieux cerner les limites de l'IA et la nécessité de cadrer son utilisation, en prenant certaines décisions stratégiques et techniques. Au final, nous pensons qu'un déploiement Cloud accéléré par l'IA permet de tester plus rapidement et plus souvent une infrastructure. La possibilité d'avoir rapidement un prototype fonctionnel, facilite l'accès aux déploiements staging durant le développement, ce qui peut s'apparenter à l'architecture exécutable définie par RUP.

6.3. Contributions du travail

Ce projet démontre concrètement comment l'IA générative peut être intégrée dans le processus de déploiement IT, en automatisant la création de fichiers de configuration pour Kubernetes et Terraform. Il met en lumière les avantages et les limites de cette approche, fournissant ainsi une base pour d'autres ingénieurs souhaitant exploiter l'IA dans leurs projets. En documentant les bénéfices, défis et retours d'expérience, ce travail contribue à une meilleure compréhension de l'utilisation pratique de l'IA générative dans le domaine du déploiement cloud, ouvrant la voie à des pratiques plus efficaces et innovantes dans la gestion de projets IT. Il est à noter que les outils liés à l'IA générative sont en plein essor et évolutent très rapidement. Ainsi, ce projet sert de référence actuelle (décembre 2025), mais devra être complété par des études futures pour suivre les avancées technologiques.

[1] Voir par exemple CrewAI: https://www.crewai.com/

6.4. Perspectives

Comme indiqué précédemment, le domaine de l'IA est vaste et en évolution rapide, rendant impossible un inventaire exhaustif de toutes les pistes d'exploration. Si le projet devait se poursuivre, voici les sujets que nous étudierions en premier.

- **Explorer une intégration avancée des LLM avec des serveurs MCP et une flotte d'agents**. Cette option nous permettrait d'offrir une boucle de feedback sur le fonctionnement de l'infrastructure au LLM, lui permettre d'inspecter l'état d'un cluster Kubernetes par lui-même ou encore de bénéficier de revues de code d'autres LLM en parallèle. Nous pourrions ainsi, avec un travail de configuration plus important en amont, avoir une orchestration plus fine du travail des LLM.
- **Approfondir l'usage des rules** pour obtenir des résultats plus précis et mieux contrôlés. La longueur, le style et le type de règles à inclure ont probablement un impact sur leur efficacité.
- **Travailler le prompt engineering** pour réduire le nombre d'itérations et améliorer la qualité du code généré. Tout comme les rules, la manière d'écrire nos prompts a probablement été étudié et comparé. Nous pourrions continuer les recherches existantes en approfondissant et adaptant les astuces pour le domaine du déploiement Cloud.

7. Recommandations

Au terme de ce projet, voici ce que nous souhaiterions recommander à des collègues ingénieurs ou des étudiants qui souhaiteraient utiliser l'IA générative pour les aider dans leurs déploiements IT.

Pour l'industrie: L'IA peut accélérer significativement un déploiement, surtout sur les premières étapes de mise en place de premières versions. Sans relecture des résultats, il est possible d'avoir des mauvaises surprises lors de problèmes en production, qui peuvent coûter très cher s'ils concernent la sécurité ou des erreurs de stratégie. Des erreurs de déploiement peuvent causer des pannes qui impactent fortement tout le business d'une entreprise, s'il concerne des services critiques. l'IA générative ne doit *jamais* être utilisée sans validation humaine, puisque seuls les humains pourront prendre la responsabilité finale de ce qui est déployé.

Pour l'éducation: Le contexte est à notre avis bien différent, comme le but principal est d'acquérir des nouvelles compétences sur des technologies spécifiques. L'expérience d'Eva et Samuel, en tant que débutants sur Kubernetes, s'est révelée peu bénéfique en terme de connaissances et compétences acquises. A part avoir appris quelques commandes `kubectl` et découvert de nouvelles ressources et bonnes pratiques de gestion des fichiers inconnues auparavant, la contribution éducative à Kubernetes en déleguant le travail à une IA, est faible. Nous recommandons ainsi vivement d'encourager la découverte manuelle des outils avant de s'appuyer sur l'IA. La première phrase de découverte, recherche personnelle et de familiarisation aux concepts, bonnes pratiques et à la syntaxe s'ancre beaucoup mieux par l'effort et la réflexion. Une fois des bases acquises et qu'il devient possible de juger de la qualité d'une implémentation, l'IA peut aider à générer du code fonctionnel, mais il reste essentiel de comprendre ce qui est produit.

8. Références

Bibliographie

[1] H. Team, « The Seven Phases of the Software Development Life Cycle ». Consulté le: 9 décembre 2025. [En ligne]. Disponible sur: https://www.harness.io/blog/software-development-life-cycle-phases

[2] J. T. Marchewka, *Information Technology Project Management: Providing Measurable Organizational Value, 5th Edition*. Wiley, 2015.

[3] A. Talreja, « Deployment Phase in SDLC: Strategies, CI/CD & Best Practices (2026) ». Consulté le: 6 décembre 2025. [En ligne]. Disponible sur: https://teachingagile.com/sdlc/deployment

[4] Beedeez, « Qu'est-ce que la classe inversée ? ». Consulté le: 3 janvier 2026. [En ligne]. Disponible sur: https://www.beedeez.com/fr/ressources/blog/quest-ce-que-la-classe-inversee

[5] HEIG-VD, « GitHub - opendidac/opendidac: Opendidac is an educational platform for managing training exercises and exams, featuring specialized question types for software engineering education and training. ». Consulté le: 2 janvier 2026. [En ligne]. Disponible sur: https://github.com/opendidac/opendidac

[6] HashiCorp, « Automate Infrastructure on Any Cloud ». Consulté le: 2 janvier 2026. [En ligne]. Disponible sur: https://www.terraform.io/

[7] « The Linux Foundation ». Consulté le: 2 janvier 2026. [En ligne]. Disponible sur: https://kubernetes.io/

[8] Windsurf, « Windsurf Homepage ». Consulté le: 9 décembre 2025. [En ligne]. Disponible sur: https://windsurf.com/

[9] Windsurf, « Windsurf Pricing ». Consulté le: 9 décembre 2025. [En ligne]. Disponible sur: https://windsurf.com/pricing

[10] P. Duvall, « Windsurf Rules & Workflows: AI-Driven Software Delivery Best Practices ». Consulté le: 6 décembre 2025. [En ligne]. Disponible sur: https://www.paulmduvall.com/using-windsurf-rules-workflows-and-memories/

[11] P. Gauthier, « Aider LLM Leaderboards | aider ». Consulté le: 6 décembre 2025. [En ligne]. Disponible sur: https://aider.chat/docs/leaderboards/

[12] S.-b. Team, « SWE-bench Leaderboards ». Consulté le: 6 décembre 2025. [En ligne]. Disponible sur: https://www.swebench.com/

[13] Cursor, « Cursor ». Consulté le: 2 janvier 2026. [En ligne]. Disponible sur: https://cursor.com/

[14] GitHub, « GitHub Copilot · Your AI pair programmer · GitHub ». Consulté le: 2 janvier 2026. [En ligne]. Disponible sur: https://github.com/features/copilot

[15] J.-H. C. A. B. Rana Nameer Hussain Khan Dawood Wasif, « Multi-Agent Code-Orchestrated Generation for Reliable Infrastructure-as-Code ». Consulté le: 9 décembre 2025. [En ligne]. Disponible sur: https://arxiv.org/abs/2510.03902

[16] Z. Z. Z. X. X. S. Tianyi Zhang Shidong Pan, « Deployability-Centric Infrastructure-as-Code Generation: An LLM-based Iterative Framework ». Consulté le: 9 décembre 2025. [En ligne]. Disponible sur: https://arxiv.org/abs/2506.05623

[17] Vellum, « Best LLM for Coding ». Consulté le: 6 décembre 2025. [En ligne]. Disponible sur: https://www.vellum.ai/best-llm-for-coding

[18] Amazon, « Cloud Computing Services - Amazon Web Services (AWS) ». Consulté le: 2 janvier 2026. [En ligne]. Disponible sur: https://aws.amazon.com/

[19] « Understanding MCP servers - Model Context Protocol ». Consulté le: 2 janvier 2026. [En ligne]. Disponible sur: https://modelcontextprotocol.io/docs/learn/server-concepts

[20] « GitHub - containers/kubernetes-mcp-server: Model Context Protocol (MCP) server for Kubernetes and OpenShift ». Consulté le: 2 janvier 2026. [En ligne]. Disponible sur: https://github.com/containers/kubernetes-mcp-server/

9. Annexes

Les annexes se trouvent dans un repository publique GitHub https://github.com/evarayHEIG/ItProMan_project. Le contenu du repository est le suivant:

- Export des conversations complètes de WindSurf
- Fichiers finaux de déploiement Kubernetes
- Fichiers finaux de déploiement Terraform