PETELET Virgil
BTS SIO SLAM – Arès Formation

VEILLE TECHNOLOGIQUE – DEVOPS

Table des matières

1 - Introduction

J'ai choisi de centrer ma veille technologique sur **DevOps**, car cette approche correspond parfaitement aux enjeux actuels des projets informatiques, notamment pour une application métier comme GSB. DevOps ne se limite pas à un simple système de management : c'est une véritable culture qui favorise l'automatisation, la collaboration et l'amélioration continue entre développement et exploitation. C'est une réelle philosophie de travail.

Dans le cadre de GSB, qui gère les notes de frais des visiteurs médicaux, DevOps présente plusieurs atouts qui ont motivé ce choix. D'abord, l'automatisation des tests et des déploiements permettrait de réduire fortement les erreurs humaines lors des mises à jour, ce qui est essentiel pour une application utilisée au quotidien par de nombreux utilisateurs. Ensuite, les pipelines d'intégration et de livraison continues offriraient la possibilité de livrer plus rapidement de nouvelles fonctionnalités ou correctifs, sans bloquer l'activité ni mobiliser lourdement les équipes.

DevOps répond aussi à un autre problème classique dans les organisations : le manque de communication entre les équipes de développement et les équipes d'exploitation. En instaurant des pratiques communes (supervision partagée, retours rapides en production, outils collaboratifs), GSB gagnerait en réactivité et en stabilité. Enfin, cette approche permettrait de mieux suivre les performances de l'application, d'anticiper les incidents, et de simplifier la maintenance grâce à une meilleure traçabilité des changements.

En résumé, DevOps a été choisi comme sujet de veille car il apporte une réponse concrète aux besoins de GSB : améliorer la qualité et la fiabilité de l'application, accélérer les déploiements, renforcer la collaboration technique et, au final, offrir un outil plus performant et plus facile à maintenir.

2 - Contexte

GSB est une application web et mobile qui permet aux visiteurs médicaux de gérer facilement leurs notes de frais.

Ils peuvent :

- **Saisir** leurs dépenses (repas, transport, hébergement).
- **Soumettre** ces notes à leur administration pour validation.

L'application **génère** ensuite des rapports financiers pour le service comptable.

Schèma à faire

Utilisation de l'application de note de frais (Relation Visiteur → Administrateur → Comptable)

Pour l'instant, GSB repose sur une architecture classique PHP 8.4 contenant le framework Symfony 7.4, avec des mises à jour faites manuellement et une supervision limitée. Cela engendre quelques difficultés : les déploiements sont lents et susceptibles d'erreurs, les tests ne sont pas toujours faits systématiquement, ce qui peut laisser passer des bugs en production, et la collaboration entre les développeurs et les administrateurs système est assez réduite.

Ces limites montrent clairement l'intérêt de DevOps, non seulement comme méthode de gestion, mais avant tout comme une philosophie qui favorise l'automatisation, la collaboration et la rapidité tout en assurant la qualité et la fiabilité de l'application. Ce choix correspond donc bien aux besoins actuels de GSB.

3 - Problématique

Constat :

⇒ Les mises à jour prennent trop de temps et sont peu fiables.

⇒ L'application manque de surveillance en temps réel.

⇒ Les équipes travaillent de leur côté, sans échanges ni coordination entre elles, ce qui complique le travail commun.

Ligne directrice (enjeu) :

La gestion de projet GSB n'est pas optimale. Comment l'optimiser ?

4 - Historique de DevOps

Sur le projet GSB, je réalise mes tests sur une machine virtuelle (VM) individuelle, ce qui reflète une pratique assez traditionnelle. Cette méthode illustre bien le contexte d'avant DevOps, une époque où les développeurs déployaient leur code chacun sur leur propre VM, tandis que les équipes opérationnelles s'occupaient manuellement du déploiement et de la maintenance des serveurs. Cette séparation entre développement et exploitation créait inévitablement des problèmes de communication, des erreurs fréquentes et des retards dans la mise en production. C'est d'ailleurs ce mode opératoire qui est encore utilisé aujourd'hui dans le projet GSB.

Au début des années 2000, avec l'émergence des méthodes Agiles, certains de ces dysfonctionnements ont commencé à être corrigés, mais l'automatisation complète des déploiements faisait encore défaut. C'est dans ce contexte que DevOps a vu le jour, pour combler ce fossé et promouvoir une collaboration étroite entre développement et exploitation, alliée à l'automatisation.

Avant 2010, les déploiements manuels étaient la norme, avec des cycles longs et souvent sujets à erreurs. Entre 2010 et 2015, on a vu l'introduction de scripts automatisant certains tests et déploiements, améliorant partiellement la situation. La période 2015-2020 a été marquée par la généralisation des pipelines CI/CD (intégration et déploiement continus) et l'essor de la conteneurisation avec Docker, permettant des livraisons plus rapides et plus cohérentes.

Depuis 2020, l'orchestration des conteneurs via Kubernetes a pris une place centrale, complétée par l'intégration de la sécurité dès la conception des applications avec DevSecOps, ainsi que par le déploiement d'outils avancés de monitoring et d'observabilité. Cette évolution permet aujourd'hui de livrer des logiciels de haute qualité à un rythme soutenu, tout en réduisant les coûts.

Pour mon projet GSB, passer d'une gestion traditionnelle basée sur des VM isolées à une démarche DevOps intégrée serait un réel bénéfice pour accélérer les déploiements, limiter les erreurs et faciliter la maintenance.

Ainsi, DevOps est passé d'une simple amélioration de processus à une nouvelle culture de travail qui fusionne développement, exploitation, sécurité et supervision pour offrir des applications fiables, évolutives et rapides à déployer

5 - Avantages

Gestion de projet – Agile :

Pour le projet GSB, la méthode Agile offre un vrai cadre pour avancer pas à pas, en découpant le travail en petits morceaux appelés sprints. Chaque sprint correspond à un objectif précis, ce qui facilite la gestion du projet et permet de voir rapidement des résultats concrets. Par exemple, on pourrait commencer avec la fonction de saisie des notes de frais, puis passer à la validation par les managers, avant de finir par le suivi et la génération des rapports.

Ce découpage présente plusieurs avantages. D'abord, il rend l'équipe très réactive : si une fonctionnalité pose problème ou si les besoins changent, on peut ajuster le tir dès le sprint suivant, sans attendre la fin du projet. Ensuite, les livraisons fréquentes permettent aux utilisateurs de tester régulièrement le produit et de donner leur avis, ce qui limite les surprises désagréables. Enfin, ce fonctionnement favorise une amélioration continue, grâce à des moments réguliers de retour sur ce qui a marché ou non.

Au-delà de la technique, Agile encourage surtout la communication et la collaboration entre tous les acteurs du projet — développeurs, administrateurs, responsables financiers — ce qui est essentiel pour un outil aussi vital que la gestion des notes de frais. En adoptant cette méthode, GSB gagne en souplesse, en qualité et surtout en capacité à s'adapter rapidement aux besoins réels de ses utilisateurs.

CI/CD (Intégration et Livraison Continues) :

Dans le cadre du projet GSB, **l'intégration continue (CI)** et la **livraison continue (CD)** sont des pratiques essentielles pour faciliter le développement et le déploiement du logiciel de note frais.

Imaginez que chaque fois qu'un développeur modifie le code par exemple pour corriger le calcul des indemnités kilométriques ou ajouter une nouvelle fonctionnalité de saisie des notes de frais, ces changements sont envoyés automatiquement sur un dépôt centralisé. Sur GSB nous utiliserons Git. Dès cette action, plusieurs étapes se mettent en place sans intervention humaine :

- ⇒ Premièrement, le code est automatiquement compilé pour vérifier qu'il s'intègre bien avec l'existant.
- ⇒ Ensuite, des tests unitaires et fonctionnels s'exécutent pour s'assurer que rien ne casse.
- ⇒ Puis, une analyse qualité évalue la robustesse et la maintenabilité du code.

Si toutes ces vérifications sont validées, on passe à la **livraison continue**. Le code est alors déployé automatiquement sur un environnement de test, où il peut être validé dans des conditions proches de la production. Une fois cette validation réussie, le déploiement se fait en production, en toute sécurité et avec une grande rapidité.

Ce processus automatisé réduit les risques d'erreurs humaines, raccourcit considérablement les délais de mise à jour et garantit une qualité constante du logiciel.

Voici un schéma illustrant ce pipeline CI/CD appliqué à mon projet GSB :

Schéma à faire

Développement → Dépôt Git (versionning) → Intégration Continue (compilation, tests, contrôle qualité)
↓
Livraison Continue (déploiement automatique)
↓
Environnement de test → Validation → Production

Dans ce schéma, chaque étape est automatisée et intégrée. Par exemple, une correction dans le calcul des indemnités kilométriques est immédiatement testée et, si validée, déployée en production, sans délai ni intervention manuelle.

Supervision et feedback – Monitoring et alertes :

La supervision continue et les alertes en temps réel jouent un rôle crucial pour garantir la performance et la réactivité de l'application face aux besoins des utilisateurs. Un suivi constant permet de détecter rapidement toute anomalie afin que le système fonctionne toujours de manière fluide, sans coupure.

Surveillance permanente de l'application et des serveurs

Le monitoring dans GSB consiste à collecter en temps réel des données précises sur les performances du système, en portant attention à plusieurs aspects essentiels :

- **Temps de réponse des requêtes** : mesurer la rapidité avec laquelle un utilisateur peut saisir, valider ou consulter ses notes de frais.

- **Disponibilité des services** : s’assurer que les fonctionnalités clés, comme l’interface utilisateur ou la génération de rapports, sont opérationnelles en continu.

État des serveurs : surveiller l’utilisation des ressources (CPU, mémoire, espace disque) pour anticiper les risques de surcharge.

Des outils tels que Prometheus permettent de collecter ces indicateurs sous forme de métriques, tandis que Grafana les rend visibles via des tableaux de bord clairs et dynamiques. Ces outils aident les équipes à surveiller activement l’état du système et à détecter immédiatement toute anomalie.

Alertes en temps réel

Un des grands atouts du monitoring est la capacité à lancer des alertes automatiques. Ces alertes, paramétrées par des seuils précis, se déclenchent instantanément lorsqu’un problème survient. Par exemple, si une requête liée aux notes de frais devient trop lente ou si un serveur s’approche de sa capacité maximale, une notification est envoyée sans délai.

Ces alertes permettent de réagir rapidement pour empêcher qu’un petit incident ne devienne un problème majeur. Elles sont envoyées aux administrateurs ou à l’équipe technique en charge, qui peut alors intervenir efficacement. Parfois, ces alertes peuvent même déclencher des actions automatiques, comme le redémarrage d’un service pour rétablir rapidement les performances.

Outils de supervision recommandés

Pour assurer un suivi fiable et des alertes pertinentes, Prometheus et Grafana sont des solutions complémentaires idéales :

- **Prometheus** collecte régulièrement les données de performance en scrutant l’ensemble des composants de l’application. Il rassemble des métriques comme la durée des requêtes, l’utilisation mémoire, ou le taux d’erreur.
- **Grafana** sert à visualiser ces données grâce à des graphiques et des tableaux de bord personnalisables. Cela facilite la lecture et la compréhension des tendances, pour une gestion proactive et réactive du système GSB.

Cette approche garantit que GSB reste performant, stable et capable de s’adapter rapidement en cas de problème, offrant ainsi une qualité de service optimale aux utilisateurs.

Revue de code (Code Review) :

Dans le cadre du développement de GSB, la revue de code est une étape incontournable avant chaque mise à jour importante du système de gestion des notes de frais. Lorsqu’un développeur termine une modification, que ce soit pour améliorer la saisie des notes ou pour renforcer la validation des frais, un autre développeur examine attentivement le code. Cette vérification vise à s’assurer qu’aucune erreur n’a été introduite et qu’aucune faille de sécurité ne compromettra les données des utilisateurs.

Ce contrôle mutuel est fondamental pour prévenir les bugs qui pourraient impacter des fonctionnalités critiques comme la soumission ou l’approbation des notes de frais. En évitant ces dysfonctionnements, la revue de code garantit la continuité du processus de remboursement, assurant ainsi la confiance des utilisateurs.

Pour mon projet GSB, les avantages sont multiples : la stabilité de l’application est renforcée, les erreurs sont réduites, et les futures mises à jour s’intègrent plus facilement au système existant.

Mise en production :

Le déploiement de nouvelles versions d'une application représente une étape clé pour assurer une évolution continue du système tout en préservant une expérience utilisateur sans faille. Pour GSB, cela veut dire pouvoir intégrer rapidement des corrections ou des fonctionnalités inédites par exemple, un export PDF des notes de frais de manière automatisée et transparente, sans jamais interrompre le service.

Déploiement automatisé

Grâce à des outils de déploiement automatisé, chaque nouvelle version de GSB peut être mise en ligne avec rapidité et précision. Ce processus réduit drastiquement les risques d'erreurs humaines et garantit que les mises à jour sont déployées uniformément sur tous les environnements, que ce soit en test ou en production.

Conteneurisation avec Docker

Une des pierres angulaires de ce déploiement efficace est la conteneurisation de l'application via Docker. Grâce à Docker, on crée un environnement isolé et identique entre la phase de test et celle de la production. Cela signifie que la version testée est strictement la même que celle déployée sur les serveurs en production, ce qui minimise les soucis liés aux différences d'environnement.

Gestion de la scalabilité avec Kubernetes

Pour assurer que l'application reste performante même en cas de forte affluence par exemple lors du rush des soumissions de notes de frais en fin de mois, Kubernetes entre en jeu. Cet orchestrateur gère plusieurs instances de conteneurs Docker simultanément, garantissant la redondance et l'équilibrage de charge. En cas de défaillance d'une instance, Kubernetes redirige automatiquement le trafic vers une autre instance fonctionnelle, assurant une disponibilité maximale du service.

Voici un schéma simple pour visualiser ce processus de mise en production :

Schéma à faire

1. Développement : mise à jour du code (fonctionnalité ou correction)
↓
2. Conteneurisation Docker : création d'un environnement identique pour test et production
↓
3. Déploiement automatisé : mise en ligne dans l'environnement de production
↓
4. Kubernetes : gestion des instances, redondance et haute disponibilité
↓
5. Application disponible et fonctionnelle en continu, sans interruption pour l'utilisateur

Cette approche assure non seulement un déploiement fiable et rapide, mais aussi une gestion optimale des ressources, permettant au système GSB de monter en charge facilement et de fonctionner sans interruption, même sous de fortes sollicitations.

PAS FINI

6 - Inconvénients

7 - Évaluation et études d’impact

8 - Études financières

9 - Comment mettre en place DevOps pour GSB