TP – Administrateur système DevOps

2023-2024

Jérémy OBJOIS

Jérémy OBJOIS

Table des matières

Blocs de compétence

CCP1 : Automatiser le déploiement d'un infra-cloud

- Automatiser la création de serveurs à l'aide de scripts
- Automatiser le déploiement d'une infrastructure
- Sécuriser l'infrastructure

CCP2 : Déployer en continu une application

- Préparer un environnement de test
- Gérer le stockage des données
- Gérer les containers
- Automatiser la mise en production d'une application avec plateforme

CCP3 : Superviser les services déployés

- Définir et mettre en place des statistiques services
- Exploiter une solution de supervision
- Échanger sur des réseaux professionnels éventuellement en anglais

Introduction

Section 1. Contexte

Dans un contexte où la responsabilité environnementale est devenue un enjeu majeur, les entreprises industrielles sont de plus en plus incitées à intégrer des pratiques durables dans leurs processus.

Le projet présenté dans ce mémoire a été réalisé au sein d'une entreprise industrielle de taille internationale, spécialisée dans les matériaux de construction. Cette organisation souhaite améliorer la mesure et le suivi de l'empreinte carbone de ses produits.

Afin de répondre à ce besoin, une approche DevOps a été retenue. Elle vise à concevoir et déployer une application cloud permettant de calculer l'empreinte carbone des produits en fonction de leur fabrication et de leur transport, tout en garantissant automatisation, sécurité et supervision des services.

Section 2. Objectifs du Mémoire

Ce mémoire vise à explorer l'utilisation du DevOps dans le cadre du développement d'une application de calcul d'empreinte carbone, avec les objectifs suivants :

- Optimiser le développement de l'application tout en réduisant les délais et en améliorant la qualité et la performance de celle-ci.
- Surmonter les obstacles liés à l'intégration de DevOps au sein du contexte défini.
- Évaluer les bénéfices environnementaux potentiels liés à l'utilisation de l'application.

Ce mémoire abordera ces différents points à travers un examen précis des étapes de planification, de mise en œuvre et d'évaluation. Il documentera chaque phase du processus et de son impact sur l'efficacité environnementale de l'entreprise.

Partie 1. Cahier des charges

Chapitre 1. Le contexte du projet

Comme évoqué précédemment, l'entreprise d'accueil est fortement engagée dans une démarche de transition environnementale. Elle s'est notamment fixée pour objectif de réduire significativement l'empreinte carbone de ses produits industriels à l'horizon 2030, ainsi que d'atteindre la neutralité carbone à plus long terme.

Dans ce contexte, l'entreprise souhaite disposer d'une application dédiée au calcul de l'empreinte carbone de ses produits, intégrant à la fois les paramètres de fabrication et de transport.

Problématique : *Comment l'intégration des pratiques DevOps peut-elle faciliter le développement et l'opérationnalisation d'une application de calcul d'empreinte carbone, tout en contribuant aux objectifs de développement durable de l'entreprise ?*

Chapitre 2. L'identification des besoins

Pour réaliser la satisfaction de cette demande, nous devons analyser les besoins :

- **Concernant la partie Opérations**, la création de l'application nécessitera la mise en place d'une infrastructure cloud adéquate, permettant d'héberger l'application et d'offrir un accès facile aux utilisateurs.
  L'utilisation des outils du DevOps (CI/CD) sera également nécessaire afin d'automatiser les processus et de garantir une intégration et une surveillance fluide et continue.

- **Concernant la partie Développement**, l'application devra être capable de calculer l'empreinte carbone de la fabrication du produit, ainsi que son transport. Elle devra être agréable à visualiser et rapide.

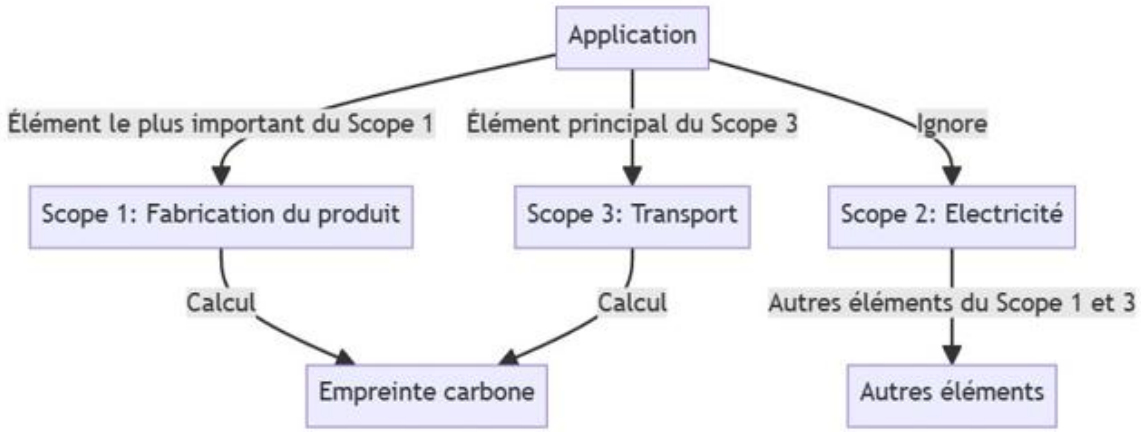
Chapitre 3. Les limites

Concernant la partie Développement, j'ai choisi de me focaliser sur l'empreinte carbone émise par la fabrication du produit (scope 1)[1], ainsi que le celle du transport (scope 3). En effet, lors du calcul de l'empreinte carbone, la fabrication du produit (scope 1) est la partie la plus

[1] Les scopes sont les différents périmètres qui forment l'empreinte carbone.

importante. Pour ce qui est du scope 3, c'est-à-dire le transport, il s'agit de l'élément le plus pertinent parmi les autres éléments qui y figurent.

Ainsi, je ne calculerai pas l'empreinte carbone émise par l'électricité lors de la fabrication des produits, qui correspond au scope 2. J'ôterai aussi tous les autres éléments appartenant aux scopes 1, 2 et 3. Le schéma ci-dessous expose mes choix.



SCHEMA DES SCOPES CHOISIS ET IGNORES POUR LE DEVELOPPEMENT DE L'APPLICATION

Chapitre 4. L'enveloppe budgétaire

Dans le cadre de cette réalisation, un budget dédié a été alloué par l'entreprise afin de mettre en place l'infrastructure nécessaire au projet, incluant la location de ressources cloud adaptées aux besoins de l'application.

Chapitre 5. Les délais

Concernant le délai, je n'ai pas eu de délai prévu. Pour autant, je me suis fixé comme objectif de terminer le projet avant le 1er mai 2024, en concordance avec mon alternance.

Finalement, nous avons défini le projet. Il est temps d'aborder sa planification et sa conception.

Partie 2. Planification et conception du projet

Pour traiter la planification et la conception du projet, nous aborderons d'abord la partie du DevOps, avant celle de l'infrastructure cloud. Nous terminerons par l'application.

Chapitre 1. La partie DevOps

Section 1. Les fondements du DevOps

Nous avons beaucoup évoqué le terme de DevOps. Mais que signifie-t-il vraiment ?

Fusion de « Développement » (Dev) et d'« Opérations » (Ops), le DevOps se présente comme une philosophie et une pratique qui vise à renforcer la collaboration entre les équipes du développement logiciel et les équipes responsables des opérations informatiques. Cette philosophie a pour objectif d'accélérer le processus de livraison du logiciel, tout en rehaussant la qualité et la stabilité de ce même logiciel.

Le DevOps repose sur plusieurs principes clés que nous allons désormais présenter.

§1. Une collaboration et une communication ouverte

Le DevOps repose d'abord sur une culture de collaboration et de communication ouverte. En effet, les équipes de développement et d'opérations travaillent ensemble dans un objectif commun. Cela permet d'optimiser la cohérence du travail et de réduire les délais de déploiement.

Dans ce modèle, la communication transparente est cruciale. Les équipes doivent échanger des informations, partager leurs objectifs et collaborer de manière étroite afin de répondre aux besoins métier. L'utilisation d'outils de communication modernes et de tableaux de bord partagés facilite la compréhension et réduit les risques de conflits.

§2. Des automatisations

L'automatisation constitue un autre axe majeur du DevOps. Elle permet de fluidifier les processus, d'améliorer la fiabilité et d'accélérer le cycle de livraison. D'une part, les équipes peuvent se concentrer sur des tâches générant davantage de valeur, ce qui améliore l'utilisation des ressources techniques, d'autre part, l'automatisation renforce la fiabilité en éliminant les erreurs liées aux processus manuels.

Les pipelines[2] CI/CD (Intégration Continue/Déploiement Continu) automatisent les étapes du cycle de développement et garantissent des déploiements fréquents, sécurisés et uniformes.

L'intégration continue (CI)[3] permet une validation systématique du code et de sa compatibilité après chaque modification, tandis que le déploiement continu (CD)[4] assure une mise en production fluide et automatique des versions qui ont été validées par l'intégration continue.

§3. Un feedback continu

Le feedback continu[5], recueilli auprès des utilisateurs et des équipes internes, est indispensable à l'amélioration continue des services et produits. Il permet d'adapter rapidement les solutions aux besoins, en intégrant les retours dans le cycle de développement.

§4. Une surveillance et une amélioration continue

Enfin, la surveillance proactive est essentielle pour gérer efficacement les systèmes informatiques. Au lieu de réagir aux problèmes une fois que ceux-ci sont survenus, le DevOps favorise une veille continue sur les performances, l'utilisation des ressources et la santé globale des systèmes. Cette stratégie préventive vise à anticiper les dysfonctionnements avant même qu'ils n'affectent l'expérience de l'utilisateur.

Les métriques collectées (santé des systèmes, disponibilité des services, temps de réponse des applications…) offrent un aperçu fiable sur les points forts et les axes d'amélioration. En d'autres termes, elles permettent d'ajuster les processus et de prévenir les problèmes futurs.

Maintenant que nous venons de définir le DevOps et sa philosophie, abordons les outils du DevOps que j'ai sélectionné pour répondre à la demande de Terreal.

Section 2. La sélection et la justification des outils DevOps

[2] L'ensemble des outils et des processus de développement d'un logiciel.
[3] Etapes de planification, de compilation et de tests automatisés.
[4] Déploiement du code.
[5] Feedback signifie « retour » en français.

Pour réaliser notre objectif de mise en place d'une application dans le cloud calculant l'empreinte carbone des produits de l'entreprise, j'ai établi une liste d'outils DevOps que je vais présenter et justifier. Commençons par les outils d'intégration et de déploiement continus.

§1. Les outils d'intégration et de déploiement continus (CI/CD)

Les outils de CI/CD sont ceux qui permettent aux équipes de développer, tester, et déployer des applications rapidement, de manière automatique. Parmi les outils qui existent, j'ai sélectionné GitLab et Jenkins.

A) GitLab

GitLab est une plateforme en ligne qui a pour objectif de gérer les cycles de vie du développement d'un logiciel. Elle utilise le système de contrôle de version « Git », et elle permet d'introduire du code et de le déployer à l'aide de pipelines.

Mais ce qui distingue GitLab, et c'est la raison pour laquelle je l'ai sélectionné, est son intégration de fonctionnalités supplémentaires. En effet, GitLab offre une intégration native de CI/CD, ce qui signifie que les équipes de développement peuvent configurer des pipelines automatisés pour exécuter des tests, et déployer du code directement depuis l'interface de GitLab. Autrement dit, il n'y a pas besoin de solutions tierces.

B) Jenkins

Jenkins est quant à lui est un outil open-source d'automatisation des pipelines.

La raison pour laquelle j'ai choisi Jenkins réside dans le fait qu'il s'agit d'une plateforme qui comporte une flexibilité dans ses pipelines et de nombreux plugins. Ainsi, Jenkins me permettra de déployer mon application à l'aide de pipelines, tandis que les plugins m'offriront la possibilité d'intégrer des outils externes nécessaires pour la réalisation du projet (tels que SonarQubes, OWASP, ou encore Ocean-Blue).

C) Ocean-Blue plugin

J'ai décidé d'ajouter à Jenkis le plug-in Ocean-Blue, afin de permettre à nos équipes d'avoir une visibilité graphique et simplifiée des pipelines CI/CD que nous allons effectuer.

Passons maintenant à la conteneurisation.

Section 2. La conteneurisation

Pour ce qui est de la conteneurisation, j'ai choisi d'associer Docker et Docker Compose.

A) Docker

Docker est une plateforme qui permet de lancer des applications dans des conteneurs logiciels.

Selon la firme de recherche sur l'industrie 451 Research[6], « *Docker est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur* ». Autrement dit, avec Docker, il est possible de packager une application avec toutes ses dépendances dans un conteneur virtuel et d'exécuter cette application sur n'importe quel système. Nous pourrons donc exécuter notre application de calcul d'empreinte carbone à l'aide de Docker sans se poser de question quant au système.

Mais si Docker est utilisé pour packager une application dans un conteneur, Docker Compose permet, lui, de packager plusieurs conteneurs Docker en même temps. Et c'est ce qui nous intéresse.

B) Docker Compose

Docker Compose est un outil open-source conçu pour gérer des applications conteneurisées ou multiconteneurs. Concrètement, il permet aux développeurs de gérer des environnements avec plusieurs conteneurs Docker à l'aide de fichiers de configuration YAML.

Nous utiliserons donc Docker Compose afin de gérer les conteneurs Docker de notre application, et créerons un fichier YAML qui comprendra les instructions pour gérer les conteneurs d'Apache (de l'application web) et de MYSQL (du serveur de base de données) de notre application.

Après la conteneurisation, passons à la gestion de configuration d'infrastructure as Code avec Terraform.

[6] Une société de recherche technologique qui fournit des analyses approfondies sur les tendances du marché.

Section 3. L'environnement infrastructure as Code (IaC) : Terraform

Terraform est un environnement logiciel d'« infrastructure as code »[7] qui permet de créer des topologies cloud et de les appliquer sur AWS.

J'ai sélectionné Terraform pour plusieurs raisons : d'une part, pour sa pleine compatibilité avec le fournisseur de cloud que j'ai choisi, ainsi qu'avec GitLab. Nos équipes pourront ainsi définir l'infrastructure cloud de notre projet sans avoir à se soucier d'éventuels problèmes de compatibilité. D'autre part, Terraform offre une approche de gestion de l'infrastructure en tant que code, ce qui facilitera l'automatisation des processus de déploiement et de gestion.

Terraform sera donc utilisé pour appliquer l'infrastructure cloud.

Section 4. L'infrastructure cloud

Un fournisseur de cloud public propose des services d'infrastructure permettant le déploiement d'environnements applicatifs flexibles, évolutifs et hautement disponibles.

Les services de calcul et de stockage mis à disposition constituent une base adaptée à la mise en œuvre d'architectures cloud orientées automatisation, scalabilité et résilience, en cohérence avec une démarche DevOps.

Le choix de cette solution cloud repose sur plusieurs critères techniques, notamment la fiabilité de l'infrastructure, la capacité d'adaptation aux variations de charge, ainsi que la possibilité de supporter un environnement CI/CD dynamique, répondant aux besoins du projet.

Le fournisseur de cloud et son architecture ayant été définis, la section suivante aborde les mécanismes de surveillance et de supervision mis en place.

Section 3. La surveillance et l'analyse

Pour surveiller notre infrastructure cloud, j'ai décidé de combiner Prometheus et Grafana.

A) Prometheus

[7] Il s'agit d'une approche où l'infrastructure informatique est gérée et provisionnée à l'aide de code et de scripts plutôt que de processus manuels.

Prometheus est un logiciel libre de surveillance informatique qui peut générer des alertes. Sa compétence est d'enregistrer des métriques[8] en temps réel dans une base de données. Ces métriques peuvent ensuite être interrogées à l'aide d'un langage de requête et analysées. Elles donnent des indications sur leur cible.

Prometheus nous permettra de cibler nos machines EC2 et de les analyser grâce à l'enregistrement de leur métrique. Prometheus sera combiné à Grafana.

B) Grafana

Grafana est un logiciel libre qui permet de réaliser des tableaux de bord et des graphiques depuis des bases de données générées par Prometheus.

En résumé, Prometheus collectera et stockera des métriques, tandis que Grafana permettra de visualiser ces données à travers des tableaux de bord interactifs.

La combinaison de Prometheus pour la collecte de métriques en temps réel, et de Grafana pour la visualisation des données, créera un système de surveillance complet qui permettra à nos équipes de surveiller facilement la performance des applications et de notre infrastructure cloud.

La surveillance de nos systèmes ayant été définie, passons à la sécurité et à la qualité de notre code.

Section 5. La sécurité et la qualité du code

En ce qui concerne la sécurité et la qualité du code, j'ai sélectionné OWASP et SonarQube[9].

A) OWASP

OWASP est un outil qui permet d'identifier les vulnérabilités de sécurité des applications web et de publier des recommandations pour les sécuriser.

J'utiliserai OWASP pour m'assurer que l'application ne disposera d'aucune vulnérabilité de sécurité.

[8] Une mesure quantitative utilisée pour évaluer, surveiller ou quantifier différents aspects d'un système.
[9] Trivy, qui analysera l'application conteneurisée, sera abordée plus loin.

Comme je l'ai mentionné, OWASP sera couplé à SonarQube.

B) SonarQube

SonarQube est un logiciel libre qui aide à détecter et à résoudre des problèmes de sécurité dans le code source d'une application. Pour ce faire, SonarQube exécute des analyses dans le code afin de détecter les bugs, vulnérabilités, et mauvaises pratiques de codage.

SonaQube va nous permettre de sécuriser notre application et de fournir une fenêtre contenant les mises à jour à effectuer et les problèmes potentiels. Associé à OWASP, SonarQube ajoutera une couche « cyber » à notre pipeline.

La couche « cyber » ayant été définie, il ne reste plus qu'à choisir nos outils de développement et notre environnement de test pour notre application.

Section 6. L'environnement de développement et l'outil de programmation

Pour créer mon environnement de test, j'ai choisi d'associer Visual Studio Code avec XAMPP.

A) Visual Studio Code (et Git)

Visual Studio Code est un éditeur de code source populaire développé par Microsoft. Il offre des débogages intégrés, dispose d'un support natif pour Git et surtout d'une interface agréable à utiliser.

Couplé à Git, Visual Studio Code me permettra de coder l'application de manière confortable grâce à ces nombreuses fonctionnalités pratiques. De plus, en associant cet éditeur à XAMPP, il me sera facile de tester les développements en local avant de les déployer.

B) XAMPP

XAMPP est une distribution de logiciel qui permet de configurer un environnement de développement local complet, notamment avec PHP, MySQL, et Apache.

XAMPP est particulièrement utile pour tester et déployer des applications web en local avant de les mettre en production. En résumé, je combinerai Visual Studio Code pour créer l'application de l'entreprise et XAMPP pour tester cette application.

La partie qui concernait le DevOps et sa sélection d'outils étant terminée, voyons à présent comment tous ces outils vont s'articuler au sein de la partie Opérations.

Chapitre 2. La partie Opérations

L'architecture de l'infrastructure cloud repose sur une séparation claire des responsabilités entre le code applicatif et l'infrastructure.

Le développeur DevOps travaille à partir d'un environnement local et utilise un système de gestion de versions pour maintenir deux ensembles distincts :

– la définition de l'infrastructure sous forme d'Infrastructure as Code,

– le code source de l'application.

L'infrastructure est déployée de manière automatisée via des pipelines CI/CD, permettant le provisionnement dynamique des ressources cloud nécessaires à l'exécution des services.

Le cycle de livraison applicative intègre des étapes d'analyse de qualité, de contrôles de sécurité et de déploiement automatisé, garantissant une mise en production fiable et reproductible.

Une solution de supervision centralisée permet de surveiller l'état des services, les performances et la disponibilité de l'infrastructure, afin d'assurer un suivi continu et d'anticiper les incidents.

Cette architecture illustre une approche DevOps complète, combinant automatisation, supervision et qualité logicielle, tout en restant indépendante des détails d'implémentation spécifiques.

Abordons à présent la partie Développement.

Chapitre 3. La partie Développement

Comme défini précédemment, l'application devra être capable de calculer l'empreinte carbone émise par la fabrication du produit et par son trajet. Pour ce faire, j'ai décidé de disposer l'application en deux pages principales :

- Une première page dans laquelle l'utilisateur sélectionnera à l'aide de champs le produit et son lieu de fabrication, ainsi que le mode de transport. Un bouton « Calculer » sera mis en évidence.

- Une seconde page affichant le résultat du calcul.

L'application fonctionnera à l'aide de Docker Compose.

Les scripts d'infrastructure, pipelines CI/CD et fichiers de configuration ont été volontairement simplifiés et non publiés dans ce portfolio.

Pour mettre en œuvre toutes ces opérations, j'ai eu recours à trois méthodologies.

Partie 3. Mise en œuvre : les méthodologies utilisées

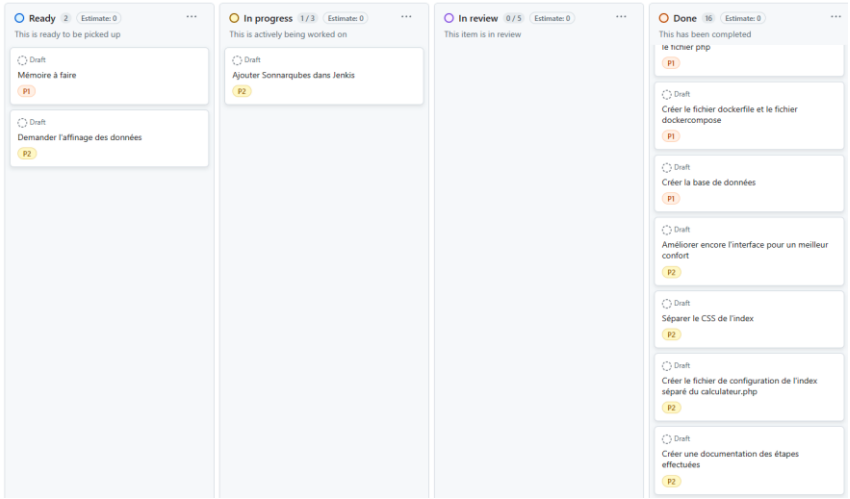
Durant l'intégralité de mon projet, j'ai effectivement associé trois méthodologies : la méthode AGILE, la méthode KANBAN, et le diagramme de GANTT.

Section 1. La méthode AGILE

La première méthode que j'ai utilisée a été la méthode AGILE. L'idée était de diviser le travail en petites portions (appelées sprints) afin d'ajuster les tâches à faire et d'avoir un feedback sur ce qui avait été fait. Les réunions nous ont permis de travailler de manière fluide et étroite entre les différentes équipes, et de parvenir à terminer le projet dans le temps imparti.

Section 2. La méthode KANBAN

En plus de la méthode AGILE, j'ai utilisé la méthode KANBAN. Cette méthode, basée sur les tâches à faire, celles en cours d'exécution et celles réalisées, nous a permis de définir les tâches à planifier pour chaque équipe. En outre, notre tableau permettait d'avoir un visuel de l'état d'avancement du projet et d'en discuter lors des réunions AGILES.



Section 3. Le diagramme de GANTT

Enfin, parallèlement aux méthodes AGILES, et KANBAN, j'ai utilisé le diagramme de GANTT. Cela a permis à l'ensemble de nos équipes d'avoir une visibilité générale des tâches réalisées en fonction du temps. Et de même qu'avec le tableau KANBAN, nous avons pu prendre appui sur le diagramme de GANTT lors des réunions AGILES pour ajuster la planification des tâches et estimer la fin du projet.

La mise en œuvre étant achevée, il ne reste plus qu'à vérifier que l'application est opérationnelle avec les résultats.

Partie 4. Résultats

Lors de notre test, l'accès au site a fonctionné sans problème.

L'application se présente alors sous la forme d'une liste de formulaires (Type de produit, Mode de transport…) à sélectionner ou à remplir, et d'un bouton « Calculer ».

Une fois les opérations effectuées, la deuxième page s'affiche et annonce l'empreinte carbone émise en fonction des critères sélectionnés par l'utilisateur. La barre latérale affiche l'historique des derniers calculs effectués.

Après plusieurs tests, le site semble assez rapide, l'interface répond bien aux manipulations de l'utilisateur et le résultat affiché est cohérent. Les tests réalisés ont permis de valider le bon fonctionnement global de l'application dans le cadre du projet.

Conclusion

Finalement, les résultats obtenus à la suite de l'utilisation de l'application sont très positifs, en particulier sur l'impact environnemental. En effet, grâce à l'utilisation de l'application, l'entreprise a pu optimiser ses processus de production et de transport, et réduire son empreinte carbone globale.

De plus, l'introduction de l'application a sensibilisé les employés à l'importance de la durabilité et les a incités à adopter des pratiques plus respectueuses de l'environnement dans leur travail quotidien.

En conclusion, on peut affirmer que l'intégration des pratiques de DevOps et le développement de l'application ont permis à l'entreprise de réaliser un bénéfice dans sa quête d'une construction plus durable, prouvant que le DevOps pouvait être un puissant levier pour la transition écologique dans le secteur des matériaux de construction.

Ce document constitue une version adaptée pour un usage portfolio. Les informations sensibles ont été volontairement anonymisées ou simplifiées.