# Hackathon IPSSI

## BAD - Équipe 6

BITARELLE Cyprien, DUFLOT Raphaël, GUERDOU Rania, KEMGANG FOMEGNI Marie Williane

greenShop

.ipssi

GRANDE ÉCOLE D'INFORMATIQUE & DU NUMÉRIQUE

# INTRODUCTION

L'ensemble de ce projet et de sa documentation est disponible sur : https://github.com/RaphDuf/HACKATHON-IPSSI-equipe6

Dans un contexte économique où la transformation numérique est devenue un impératif stratégique pour toute entreprise cherchant à maintenir sa compétitivité, la modernisation des infrastructures informatiques représente un défi majeur. Ce mémoire s'intéresse particulièrement à la migration d'une architecture traditionnelle vers le cloud, à travers l'étude du cas concret de GreenShop, un e-commerçant en pleine croissance spécialisé dans les produits biologiques. Cette introduction présente le contexte de l'entreprise, les problématiques rencontrées, les objectifs du projet de migration cloud, ainsi que l'approche méthodologique adoptée.

## 1. Présentation du contexte et de l'entreprise GreenShop

GreenShop est une entreprise de e-commerce spécialisée dans les produits biologiques et écologiques. Grâce à une forte croissance récente et à l'élargissement de son catalogue (huiles artisanales, tisanes, farines sans gluten, etc.), la plateforme enregistre une augmentation importante du trafic et des transactions.

Cependant, l'infrastructure actuelle, basée sur une machine unique hébergeant à la fois le site et la base de données, atteint ses limites. Ce système on-premise, peu évolutif et administré manuellement, ne répond plus aux exigences de performance, de disponibilité et de scalabilité.

Les pics de charge, notamment en période promotionnelle, provoquent des ralentissements critiques. De plus, l'absence d'automatisation, de traçabilité et de monitoring freine la réactivité de l'équipe technique.

Face à ces enjeux, GreenShop a sollicité InnovaTech, une ESN experte en cloud et DevOps, pour piloter une migration vers une infrastructure cloud moderne, résiliente et adaptée à la croissance de l'entreprise.

## 2. Problématique et enjeux de la migration cloud

La migration vers le cloud représente bien plus qu'un simple changement d'infrastructure ; elle constitue une transformation profonde des méthodes de travail et de la culture organisationnelle. Pour GreenShop, cette démarche s'articule autour d'une problématique centrale : **comment moderniser l'infrastructure technique pour soutenir la croissance de l'entreprise tout en garantissant fiabilité, sécurité et agilité opérationnelle ?**

Cette question principale se décline en plusieurs problématiques spécifiques :

- **Évolutivité et performance** : Comment concevoir une architecture capable de s'adapter dynamiquement aux variations de charge, notamment lors des périodes promotionnelles qui génèrent des pics de trafic importants ?
- **Fiabilité et haute disponibilité** : Comment garantir une continuité de service optimale et minimiser les risques d'interruption qui impacteraient directement le chiffre d'affaires et la satisfaction client ?
- **Automatisation et industrialisation** : Comment passer d'un mode de fonctionnement artisanal à des processus standardisés et reproductibles, permettant des déploiements fréquents et sans risque ?
- **Observabilité et proactivité** : Comment mettre en place un système de surveillance permettant d'anticiper les problèmes avant qu'ils n'affectent les utilisateurs finaux ?
- **Sécurité et conformité** : Comment assurer la protection des données sensibles (informations clients, transactions) dans un environnement cloud, tout en respectant les réglementations en vigueur ?

Ces problématiques s'inscrivent dans un contexte où les enjeux sont multiples et stratégiques pour GreenShop :

- **Enjeux business** : Maintenir la croissance de l'activité en offrant une expérience utilisateur fluide et performante, même lors des périodes de forte affluence, et ainsi capitaliser sur les opportunités commerciales.
- **Enjeux financiers** : Optimiser les coûts d'infrastructure en adoptant un modèle de facturation à l'usage, tout en minimisant les risques de perte de revenus liés à des indisponibilités du service.
- **Enjeux organisationnels** : Transformer les méthodes de travail des équipes techniques en introduisant les pratiques DevOps, favorisant ainsi la collaboration, l'automatisation et l'amélioration continue.
- **Enjeux technologiques** : Moderniser la stack technique pour faciliter l'intégration de nouvelles fonctionnalités et l'adoption de technologies innovantes à l'avenir.

La réussite de cette migration représente donc un tournant décisif pour GreenShop, conditionnant sa capacité à poursuivre son développement dans un marché de plus en plus compétitif.

## 3. Objectifs du projet et approche méthodologique

Face aux problématiques identifiées, le projet de migration cloud pour GreenShop poursuit plusieurs objectifs clairement définis :

### 3.1. Objectifs principaux

1. **Migration complète de l'infrastructure vers AWS** : Transférer l'application web et sa base de données vers une architecture cloud moderne, en tirant parti des services managés et de l'élasticité offerte par AWS.
2. **Mise en place d'une infrastructure hautement disponible** : Concevoir une architecture résiliente, répartie sur plusieurs zones de disponibilité, capable de tolérer la défaillance de composants individuels sans impact sur le service global.
3. **Automatisation de l'infrastructure et des déploiements** : Implémenter l'approche "Infrastructure as Code" avec Terraform et automatiser la configuration des environnements avec Ansible, garantissant ainsi reproductibilité et cohérence.
4. **Conteneurisation de l'application** : Encapsuler l'application dans des conteneurs Docker pour assurer la portabilité et standardiser les environnements de développement, test et production.
5. **Implémentation d'un pipeline CI/CD** : Mettre en place avec Jenkins une chaîne d'intégration et de déploiement continus, permettant des livraisons fréquentes, fiables et automatisées.

### 3.2. Approche méthodologique

Pour atteindre ces objectifs, ce projet adopte une approche méthodologique structurée, s'inspirant à la fois des principes Agile et des bonnes pratiques DevOps :

- **Approche itérative et incrémentale** : Le projet est découpé en plusieurs phases, chacune apportant une valeur ajoutée concrète et mesurable. Cette progression par incréments permet de livrer rapidement des fonctionnalités utilisables tout en minimisant les risques.
- **Priorité à l'automatisation** : Conformément aux principes DevOps, une attention particulière est accordée à l'automatisation de tous les aspects du cycle de vie de l'application, de sa construction à son déploiement, en passant par les tests et le monitoring.

## 4. Annonce du plan

Afin d'aborder de manière structurée et exhaustive la migration cloud de GreenShop, ce mémoire s'articule autour de trois grandes parties :

**La première partie** est consacrée à l'analyse de l'existant et au cadrage du projet. Elle présente un état des lieux détaillé de l'infrastructure initiale de GreenShop, expose les limites et problèmes identifiés, et justifie les choix technologiques retenus pour la migration cloud. Cette partie pose également les bases de la planification du projet, en détaillant la méthodologie de travail adoptée et les différentes phases prévues.

**La deuxième partie** se concentre sur la conception et la mise en œuvre de l'infrastructure cloud. Elle détaille le provisionnement de l'infrastructure avec Terraform, incluant la création du VPC, la configuration du Load Balancer. Cette partie aborde également la migration de la base de données et l'automatisation de la configuration des instances avec Ansible.

**La troisième partie** traite de la conteneurisation de l'application et de la mise en place du pipeline CI/CD. Elle explique le processus de création des Dockerfiles, la publication des images vers un registre de conteneurs, et la configuration de Jenkins pour automatiser les déploiements.

Ce plan permet d'aborder méthodiquement l'ensemble des dimensions du projet de migration cloud, depuis l'analyse préliminaire jusqu'à l'évaluation des résultats, en passant par toutes les étapes techniques de mise en œuvre.

# Partie I: Analyse de l'existant et cadrage du projet :

Dans le cadre du projet de migration cloud de GreenShop, une phase d'étude préliminaire approfondie a été nécessaire pour déterminer les solutions technologiques les plus adaptées. Cette étape cruciale a permis d'évaluer différentes options disponibles sur le marché, de comparer leurs avantages et inconvénients, et finalement de sélectionner l'écosystème technique le plus pertinent pour répondre aux besoins spécifiques de l'entreprise. Cette section présente l'analyse comparative des solutions cloud envisagées, la justification du choix d'AWS, la sélection des outils DevOps, et l'architecture cible retenue.

## 1. Choix d'AWS comme plateforme cloud

À l'issue de l'analyse comparative et après consultation des différentes parties prenantes du projet, AWS a été retenu comme plateforme cloud pour la migration de GreenShop. Ce choix s'appuie sur plusieurs facteurs déterminants, particulièrement pertinents dans le contexte spécifique de l'entreprise.

## 2. Services AWS spécifiquement retenus

Parmi la vaste gamme de services proposés par AWS, une sélection ciblée a été effectuée pour répondre aux besoins de GreenShop :

- **Amazon EC2** : Pour héberger les instances applicatives avec une flexibilité totale sur le choix des ressources
- **Amazon RDS** : Pour la gestion de la base de données relationnelle, avec options de haute disponibilité et de sauvegarde automatique
- **Elastic Load Balancing** : Pour la répartition de charge et la garantie de haute disponibilité
- **Amazon VPC** : Pour l'isolation réseau et la sécurisation des communications

Ce choix de services constitue un socle solide et évolutif pour l'infrastructure cible de GreenShop.

## 3. Sélection des outils DevOps (Terraform, Ansible, Docker, Jenkins)

La migration vers le cloud s'accompagne naturellement d'une transformation des pratiques opérationnelles. Pour tirer pleinement parti des avantages du cloud, il est essentiel d'adopter des outils DevOps permettant d'automatiser et d'industrialiser les

processus de déploiement et de gestion de l'infrastructure. Cette section présente les outils sélectionnés et justifie ces choix.

## 3.1. Infrastructure as Code avec Terraform

**Pourquoi Terraform ?**

Terraform a été choisi comme outil principal pour la gestion de l'infrastructure en tant que code (IaC) pour plusieurs raisons :

- **Approche déclarative** : Terraform permet de décrire l'état souhaité de l'infrastructure plutôt que les étapes pour y parvenir, simplifiant ainsi la maintenance et l'évolution.
- **Indépendance vis-à-vis du fournisseur** : Bien que le projet actuel soit centré sur AWS, l'adoption de Terraform offre une flexibilité future si une approche multi-cloud devenait nécessaire.
- **Gestion d'état** : Terraform maintient un état de l'infrastructure, facilitant le suivi des modifications et la détection des dérives de configuration.
- **Modularité** : L'approche modulaire de Terraform permet de créer des composants réutilisables, favorisant les bonnes pratiques et la standardisation.
- **Communauté active** : L'écosystème riche autour de Terraform offre de nombreux modules prêts à l'emploi et une documentation abondante.

## 3.2. Configuration automatisée avec Ansible

**Pourquoi Ansible ?**

Ansible a été sélectionné pour l'automatisation de la configuration pour les raisons suivantes :

- **Simplicité et légèreté** : Ansible fonctionne sans agent, nécessitant uniquement SSH pour la communication avec les hôtes cibles, ce qui réduit la complexité opérationnelle.
- **Syntaxe YAML accessible** : Les playbooks Ansible utilisent un format YAML facile à lire et à maintenir, favorisant la collaboration entre équipes.
- **Idempotence** : Les tâches Ansible sont conçues pour être idempotentes, permettant des exécutions répétées sans effets secondaires inattendus.
- **Large compatibilité** : Ansible s'intègre parfaitement avec les instances EC2 et d'autres composants de l'infrastructure AWS.

- **Gestion de l'inventaire dynamique** : La capacité d'Ansible à découvrir dynamiquement les ressources AWS facilite la gestion d'une infrastructure élastique.

## 3.3. Conteneurisation avec Docker

**Pourquoi Docker ?**

Docker a été choisi comme solution de conteneurisation pour les raisons suivantes :

- **Standardisation des environnements** : Docker permet de créer des environnements identiques entre développement, test et production, réduisant ainsi les problèmes de compatibilité.
- **Isolation des applications** : Les conteneurs Docker offrent une isolation efficace des processus, améliorant la sécurité et la gestion des ressources.
- **Portabilité** : Les images Docker peuvent être déployées de manière cohérente sur différentes infrastructures, facilitant les migrations futures.
- **Déploiements rapides** : La légèreté des conteneurs permet des déploiements et des redémarrages rapides, améliorant l'agilité opérationnelle.
- **Écosystème mature** : Docker bénéficie d'un vaste écosystème d'outils complémentaires et d'une adoption généralisée dans l'industrie.

## 3.4. Intégration et déploiement continus avec Jenkins

**Pourquoi Jenkins ?**

Jenkins a été retenu pour l'implémentation du pipeline CI/CD pour les raisons suivantes :

- **Flexibilité et extensibilité** : L'architecture basée sur des plugins de Jenkins permet une adaptation précise aux besoins spécifiques du projet.
- **Maturité et stabilité** : Avec plus de 15 ans d'existence, Jenkins a fait ses preuves dans de nombreux environnements d'entreprise.
- **Large éventail d'intégrations** : Jenkins s'intègre nativement avec de nombreux outils, notamment GitHub, Docker, et AWS.
- **Pipeline as Code** : La possibilité de définir les pipelines sous forme de code (Jenkinsfile) favorise la traçabilité et la reproductibilité des déploiements.
- **Communauté active** : Une vaste communauté d'utilisateurs et de contributeurs garantit un support continu et des évolutions régulières.

# 4. Présentation de l'architecture cible

À l'issue de cette phase d'étude préliminaire et de sélection des technologies, une architecture cible a été définie pour répondre aux besoins de GreenShop. Cette architecture tire parti des services AWS choisis et intègre les bonnes pratiques DevOps pour offrir une solution robuste, évolutive et maintenable.

## 4.1. Principes architecturaux

L'architecture proposée s'appuie sur plusieurs principes fondamentaux :

- **Haute disponibilité** : Déploiement sur plusieurs zones de disponibilité pour éliminer les points uniques de défaillance.
- **Évolutivité automatique** : Capacité à s'adapter dynamiquement aux variations de charge.
- **Sécurité multicouche** : Protection à tous les niveaux de l'infrastructure (réseau, accès, données).
- **Automatisation maximale** : Réduction des interventions manuelles pour limiter les erreurs et accélérer les déploiements.
- **Observabilité complète** : Visibilité sur tous les composants pour détecter et résoudre rapidement les problèmes.

## 4.2. Composants de l'architecture

**Réseau et sécurité**

- **VPC (Virtual Private Cloud)** : Environnement réseau isolé avec des sous-réseaux publics et privés répartis sur plusieurs zones de disponibilité.
- **Groupes de sécurité** : Contrôle fin des flux réseau entrants et sortants pour chaque composant.
- **Bastion Host** : Point d'entrée sécurisé pour l'administration de l'infrastructure.
- **NAT Gateway** : Permet aux instances des sous-réseaux privés d'accéder à Internet pour les mises à jour et téléchargements.

**Couche applicative**.

- **Application Load Balancer (ALB)** : Répartition du trafic entre les instances applicatives.

**Couche données**

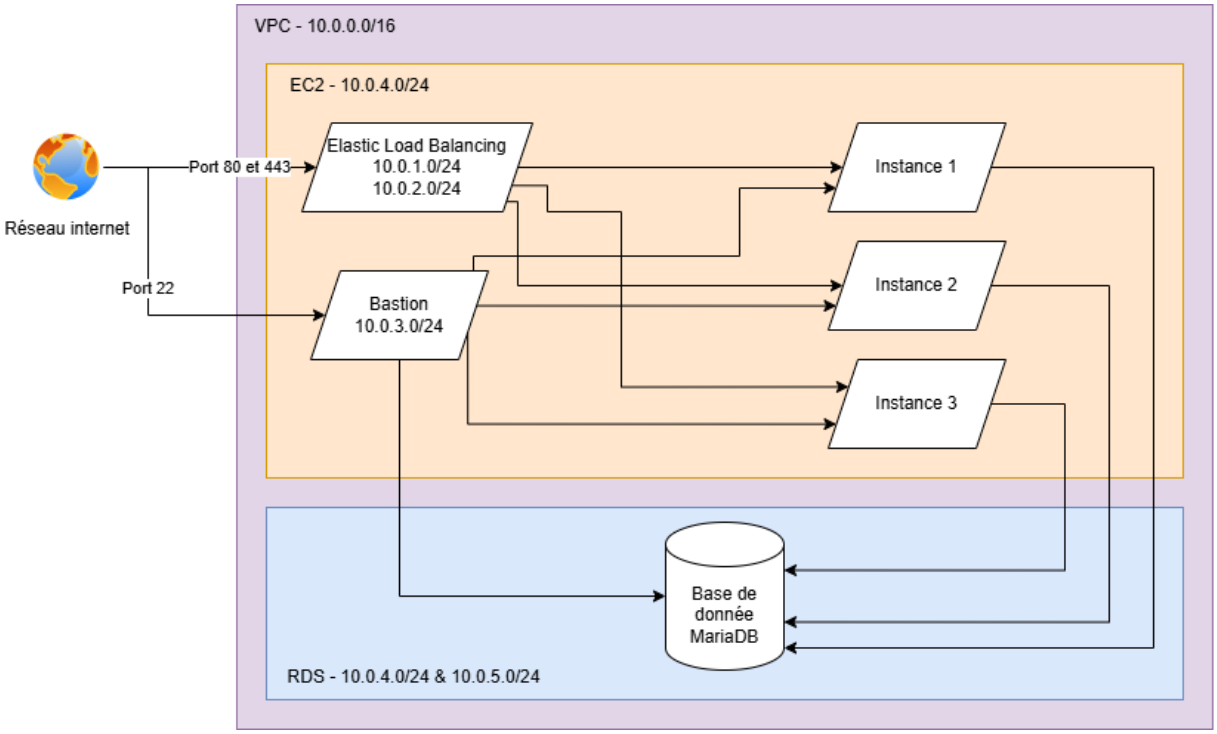
- **RDS**: Base de données relationnelle avec réplication synchrone sur une seconde zone de disponibilité pour la haute disponibilité.

**Pipeline CI/CD**

- **Jenkins** : Serveur d'intégration continue déployé sur une instance EC2 dédiée.
- **GitHub** : Dépôt de code source avec webhooks configurés pour déclencher automatiquement les builds Jenkins.

## 4.3. Schéma d'architecture

Le schéma ci-dessous illustre l'architecture cible complète, mettant en évidence les différentes couches, les composants et leurs interactions :




## 4.4. Flux de déploiement

Le flux de déploiement continu de l'application suivra les étapes suivantes :

1. Les développeurs poussent leurs modifications vers le dépôt GitHub.
2. Un webhook déclenche automatiquement un build dans Jenkins.
3. Jenkins exécute les tests d'intégration sur le code.
4. En cas de succès des tests, Jenkins construit une nouvelle image Docker.

5. L'image Docker est taguée et publiée dans Dockerhub.
6. Terraform provisionne l'infrastructure nécessaire.
7. Ansible est utilisé pour configurer les environnements cibles.

Cette architecture cible représente une transformation significative par rapport à l'infrastructure initiale de GreenShop, offrant des gains majeurs en termes de disponibilité, de performance, d'évolutivité et de sécurité. Sa mise en œuvre progressive, en suivant les principes DevOps, permettra une transition maîtrisée vers le cloud, tout en minimisant les risques pour l'activité de l'entreprise.

# Partie II: Conception et mise en œuvre de l'infrastructure cloud

## 1. Provisionnement de l'infrastructure avec Terraform

**Terraform :**
Terraform est un outil d'infrastructure-as-code (IaC) permettant de définir, déployer et gérer des ressources cloud de manière automatisée et déclarative. Il permet de versionner et de maintenir la configuration des infrastructures de façon reproductible. Il prend en charge l'intégration des ressources AWS comme les VPC, les sous-réseaux, les instances EC2 et d'autres services dans l'infrastructure cloud.

## 2. Conception du réseau VPC et sous-réseaux

**VPC (Virtual Private Cloud) :**
Un VPC est un réseau isolé dans le cloud qui permet de déployer des ressources de manière sécurisée, en définissant les sous-réseaux, les tables de routage et les paramètres de sécurité. Le VPC est le fondement de l'architecture réseau dans le cloud. Il permet de créer des segments de réseau privés et publics et de contrôler l'accès entre les différentes ressources (instances EC2, bases de données, etc.). Un VPC bien conçu garantit la sécurité et la scalabilité de l'infrastructure.

**Sous-réseaux (Subnets) :**
Les sous-réseaux sont des subdivisions du VPC qui permettent de segmenter le réseau en différentes zones de disponibilité (AZ) pour assurer la haute disponibilité des ressources. Chaque sous-réseau peut être configuré pour être privé (pour les ressources internes, comme les bases de données) ou public (pour les ressources accessibles depuis internet, comme les instances web). Ils permettent de mieux organiser l'infrastructure et d'assurer une isolation efficace des ressources sensibles.

## 3. Mise en place du bastion host et des groupes de sécurité

**Bastion Host :**
Un bastion host est une instance située dans un sous-réseau public qui sert de point d'accès sécurisé pour administrer les ressources dans un sous-réseau privé. Il permet d'accéder à des instances privées en SSH tout en étant protégé derrière un pare-feu. Le bastion host facilite l'accès distant aux instances privées sans exposer directement ces dernières à internet. Il joue un rôle clé dans la sécurité, en agissant comme une passerelle pour l'accès administratif.

**Groupes de sécurité (Security Groups) :**
Les groupes de sécurité sont des pares-feux virtuels qui contrôlent le trafic entrant et sortant des instances EC2. Ils sont associés à chaque instance pour définir les règles d'accès. Ils permettent de définir des règles de sécurité pour chaque ressource en fonction de l'adresse IP source, du type de protocole et des ports. Par exemple, le groupe de sécurité du bastion host permet uniquement les connexions SSH (port 22).

## 4. Configuration du Load Balancer

**Elastic Load Balancer (ELB) :**
Un ELB est un service qui répartit automatiquement le trafic entrant entre plusieurs instances EC2 ou autres ressources pour assurer une haute disponibilité et une répartition efficace de la charge. Le Load Balancer gère le trafic réseau en répartissant les requêtes entre plusieurs serveurs afin d'éviter la surcharge d'une seule instance. Cela garantit la résilience et la disponibilité de l'application, même en cas de panne d'une instance.

## 5. Migration et configuration de la base de données

### 5.1. Choix entre base de données managée (Amazon RDS) et hébergement sur EC2

**Deux approches sont possibles pour héberger une base de données dans AWS :** utiliser Amazon RDS (Relational Database Service) ou installer le moteur de base sur une instance EC2.

- Amazon RDS est un service managé qui prend en charge la gestion des sauvegardes, des mises à jour, de la haute disponibilité (via Multi-AZ), du monitoring, et de la sécurité (chiffrement, intégration IAM). Il offre un déploiement rapide et sécurisé, au prix d'une flexibilité limitée.
- L'hébergement sur EC2 permet un contrôle total sur l'environnement, mais implique une gestion manuelle de l'installation, de la sécurité, des sauvegardes et de la tolérance aux pannes.

Dans ce projet, Amazon RDS a été choisi pour sa simplicité d'administration, sa haute disponibilité native, et son intégration avec l'écosystème AWS. Cela permet de se concentrer sur l'automatisation et le déploiement applicatif plutôt que sur la maintenance.

## 5.2. Stratégie de migration des données

**La migration de la base de données vers le cloud s'est déroulée en plusieurs étapes :**

1. **Évaluation de la base existante** : analyse du moteur utilisé, du volume de données, et des contraintes de disponibilité.
2. **Choix de la méthode :**
   - Pour une migration rapide et ponctuelle : utilisation de mysqldump pour exporter les données et du module mysql_db sur RDS.
3. **Migration et vérification :**
   - Transfert des données.
   - Comparaison des enregistrements source/cible.
   - Test de l'application avec la nouvelle base.
4. **Mise en production :**
   - Bascule des connexions vers la base RDS.

# 6.Automatisation de la configuration avec Ansible

## 6.1. Définition et rôle d'Ansible

Ansible est un outil d'automatisation open source utilisé pour le provisionnement, la configuration, le déploiement d'applications et la gestion d'infrastructure. Il repose sur une syntaxe simple basée sur YAML (appelée playbook), ne nécessite aucun agent installé sur les machines cibles (connexion via SSH), et garantit une exécution idempotente, c'est-à-dire reproductible sans effet secondaire.

Dans ce projet, Ansible joue un rôle clé dans l'automatisation de la **configuration des instances EC2. Il permet de :**

- Installer et configurer automatiquement les logiciels requis (serveur web, base de données, monitoring, etc.),
- Appliquer les configurations de manière cohérente sur tous les serveurs,
- Gérer les mises à jour sans intervention manuelle.

- **Organisation des playbooks :** Afin d'assurer une gestion claire, modulaire et évolutive des configurations, le projet s'appuie sur la structure native d'Ansible, basée sur les **playbooks** .
- Un **playbook** est un fichier YAML décrivant une suite de tâches à exécuter sur un ou plusieurs groupes de machines. Il permet d'automatiser l'installation, la configuration et la mise à jour des services.

### 6.2. Configuration dynamique des instances EC2

Dans un environnement cloud, les instances EC2 peuvent changer d'adresse IP à chaque redémarrage si elles n'ont pas d'Elastic IP. Pour s'adapter à cette réalité, Ansible peut être utilisé avec une **configuration dynamique**. Cela signifie que l'inventaire des machines cibles n'est pas statique, mais généré dynamiquement à partir des informations fournies par AWS.

Cette méthode repose sur :

- Un **plugin d'inventaire dynamique AWS** fourni par Ansible,
- L'utilisation de balises (tags) pour regrouper logiquement les instances (ex : env=prod, role=web, etc.),
- L'interrogation automatique de l'API AWS pour obtenir les IPs à jour des machines à chaque exécution.

Cela permet de :

- Éviter les erreurs liées aux adresses IP obsolètes,
- Automatiser le déploiement dans des environnements évolutifs,
- Gérer des architectures auto-scalées.

## 7. Conteneurisation de l'application GreenShop

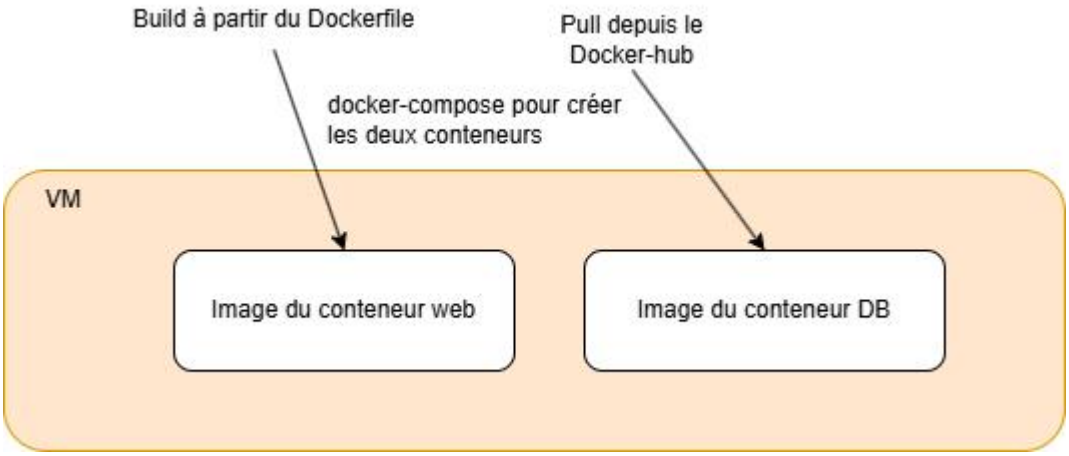
La conteneurisation de l'application GreenShop vise à améliorer la portabilité, la scalabilité et la facilité de déploiement de la solution e-commerce. Elle repose principalement sur l'utilisation de Docker, une technologie de virtualisation légère qui permet d'emballer le code, les dépendances et la configuration dans des conteneurs autonomes**.**

### 7.1. Analyse de l'application et de ses dépendances

Avant de procéder à la création des conteneurs, une analyse de l'architecture applicative a été réalisée pour identifier :

- Les composants à conteneuriser (ex. : API backend, base de données),
- Les ports utilisés,
- Les variables d'environnement nécessaires.

***Schéma de l'architecture :***




## 7.2. Conception des Dockerfiles optimisés

Le composant applicatif a son propre Dockerfile, situé dans le dossier racine correspondant.

- Les Dockerfiles ont été optimisés pour :
- Utiliser des images légères
- Réduire le nombre de couches
- Gérer la configuration via des variables d'environnement

## 7.3. Orchestration avec docker-compose

Pour gérer l'ensemble des conteneurs de manière cohérente et automatisée, un fichier docker-compose.yml a été utilisé à la racine du projet :

- Il définit les différents services (base de données, serveur web)
- Il précise les liens entre les conteneurs
- Il configure les volumes, réseaux, variables d'environnement.

### 7.4. Tests locaux et validation

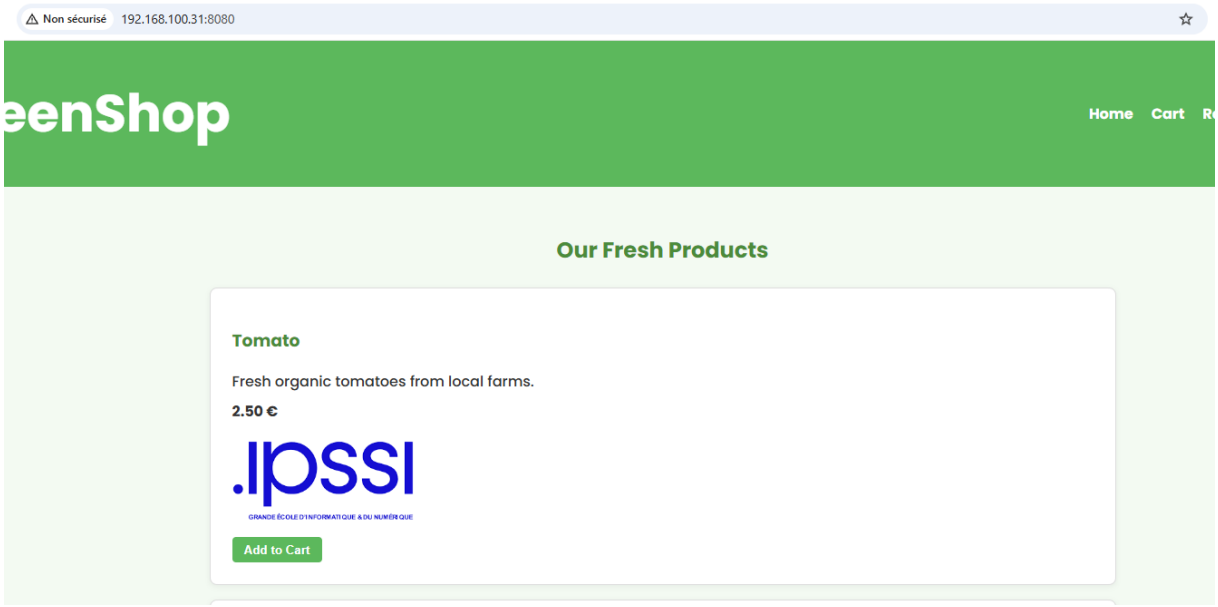
Une fois la configuration terminée, l'environnement complet a été testé localement à l'aide de la commande :

docker-compose up -d

Ce test a permis de vérifier :

- Le bon démarrage des services,
- La communication inter-conteneurs (ex. : API qui interagit avec la base),
- L'accessibilité via navigateur ou outil de test (ex. : Postman).

**Accès au site web :**




## 8. Mise en place du pipeline CI/CD avec Jenkin:

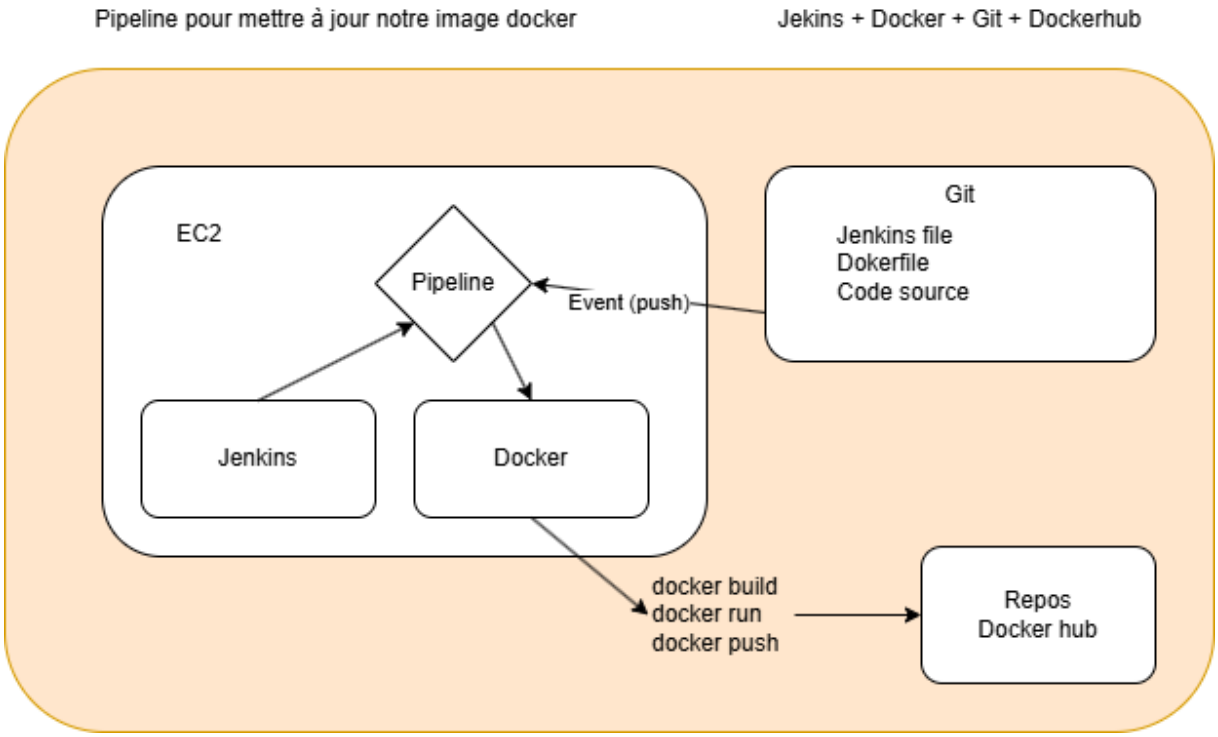
Afin de renforcer la fiabilité, la traçabilité et la rapidité du processus de livraison logicielle chez GreenShop, une chaîne CI/CD (Intégration Continue / Déploiement Continu) a été mise en œuvre en s'appuyant sur **Jenkins**, un outil open source de référence dans l'automatisation DevOps.

## 8.1. Architecture de la chaîne d'intégration continue

La solution s'articule autour des composants suivants :

- **Jenkins Server** (déployé sur une instance EC2)
- **GitHub** comme référentiel de code source
- **Docker Hub** comme registre d'images conteneurisées

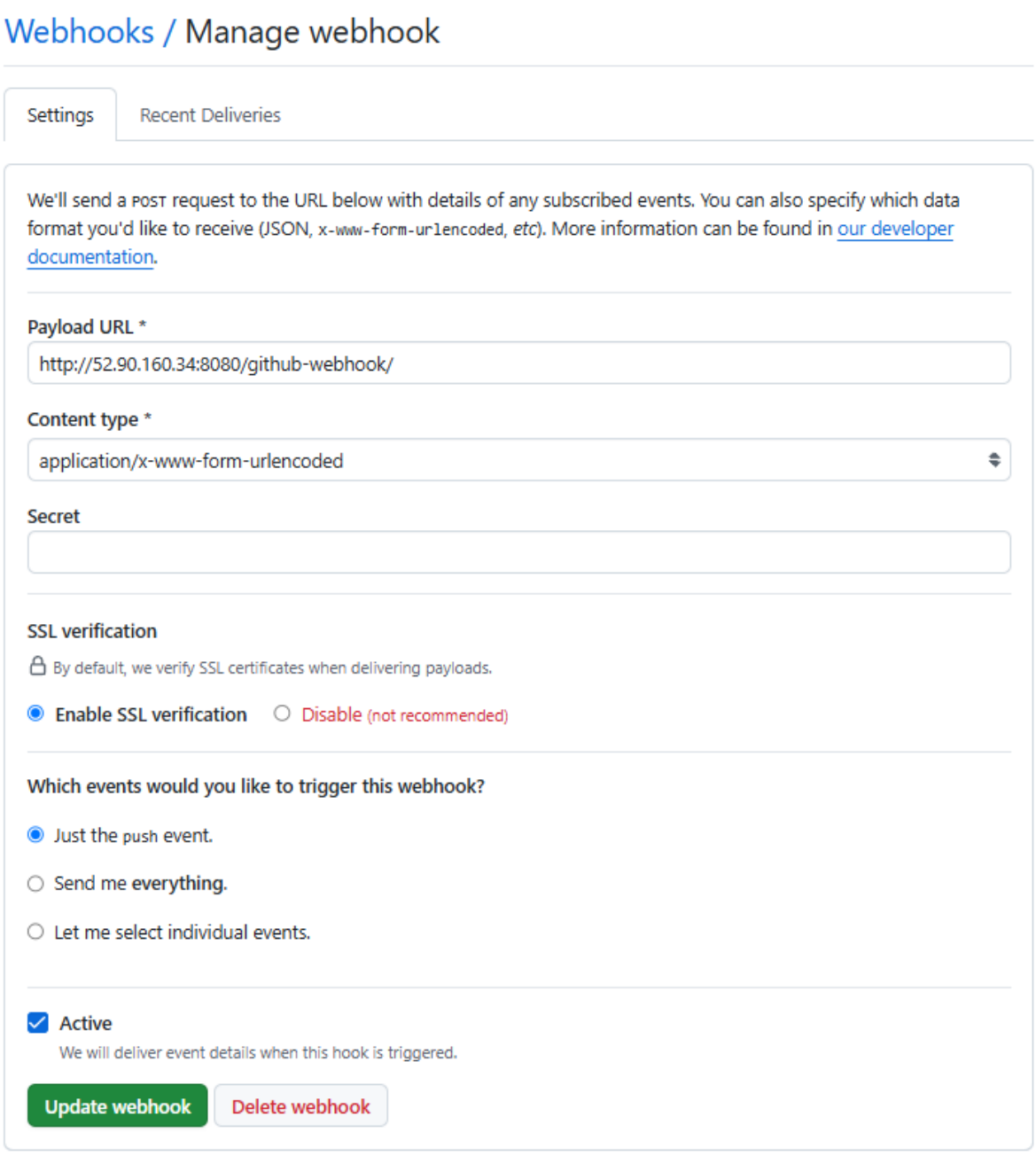
***Schéma simplifié de l'architecture CI/CD montrant les interactions Jenkins ↔ GitHub ↔ Docker ↔ AWS :***




## 8.2. Configuration des Webhooks GitHub

Pour permettre le déclenchement automatique du pipeline lors de chaque push un **webhook** a été configuré dans le dépôt GitHub. Il envoie une requête POST à Jenkins dès qu'un changement est détecté sur la branche surveillée (ex. : pipeline-ci-cd)

**Paramétrage du webhooks sur github:**




## 8.3. Implémentation du Jenkinsfile

Le pipeline CI/CD est défini dans un fichier **Jenkinsfile**, versionné dans le dépôt Git. Ce fichier décrit les étapes de la chaîne, telles que :

- **Checkout** du code source
- **Build** des images Docker
- **Run** du conteneur

- **Push** des images vers le registre

## 8.4. Validation du pipeline de bout en bout

Une fois l'ensemble mis en place, plusieurs scénarios de bout en bout ont été testés :

- Simulation de commit → Build → push.

Ce pipeline permet désormais à GreenShop de **livrer rapidement, fréquemment et de façon fiable**, tout en réduisant les risques liés aux déploiements manuels.

# CONCLUSION

Dans le cadre de ce projet réalisé en groupe, nous avons accompagné GreenShop dans sa transition vers une infrastructure cloud moderne et automatisée. Cette migration, motivée par la croissance rapide de l'entreprise et les limites de son infrastructure on-premise, a été l'occasion de concevoir et de déployer une architecture scalable, résiliente et sécurisée sur AWS.

Nous avons abordé le projet de manière structurée, en combinant les principes DevOps avec des outils adaptés comme **Terraform, Ansible, Docker et Jenkins**. Chaque membre du groupe a contribué à une phase spécifique du projet, assurant ainsi une bonne répartition des responsabilités et une montée en compétences collective.

Les résultats obtenus sont significatifs :

- Une infrastructure **automatisée** et **reproductible** grâce à Terraform ;
- Un système de **déploiement continu** performant via Jenkins ;
- Une **application conteneurisée** pour plus de portabilité.