## **Compte Rendu de Projet – Migration d'une Application vers le Cloud AWS**

### **1. Introduction**

### **Objectif du projet**

Le projet a pour objectif principal de migrer une application web traditionnelle vers une infrastructure cloud, en adoptant les bonnes pratiques de l’approche DevOps. Il s’agit de mettre en œuvre des outils d’automatisation pour orchestrer à la fois l’infrastructure et le déploiement des applications.

### **Objectif globaux**

* Provisionner un réseau sécurisé et des instances éphémères à la demande.
* Conteneuriser l’application GreenShop et un second service compagnon, pour gagner en portabilité.
* Automatiser la configuration de chaque composant, sans jamais poser la main sur un terminal à la va-vite.
* Mettre en place un pipeline CI/CD, garant d’une livraison continue et sans surprise.
* Assurer la haute disponibilité, même lors des plus gros pics de trafic.
* Superviser la santé de la plateforme, et déclencher des alertes avant que les clients ne se plaignent.
* Documenter l’ensemble pour que l’équipe interne puisse prendre le relais en toute confiance.

### **Technologies utilisées**

* **AWS**
* **Terraform**
* **Ansible**
* **Docker**
* **Jenkins**
* **GitHub**
* **VirtualBox**

### **2. Etude de l’application existante en vue de sa migration vers le Cloud**

Afin d’obtenir les informations nécessaires pour effectuer la migration, nous avons lancez la VM Greenshop sur VirtualBox:

* Lancement de la VM à l’aide du .vdi
* Récupération des infos présentes dans le Readme de la VM Greenshop
* Extraction des fichiers utiles à la migration du serveur web et de la base de données (front-end, back-end, base de données)

#### **3. Automatisation de l’infrastructure avec Terraform**

#### **3.1. Déploiement de l'Infrastructure**

### **VPC et Sous-Réseaux**

Nous avons défini un Virtual Private Cloud (VPC) avec plusieurs sous-réseaux :

* **VPC** : Un réseau virtuel isolé pour déployer les ressources.
* **Sous-Réseaux** : Plusieurs sous-réseaux publics et privés pour organiser les ressources.

### **Passerelles et Tables de Routage**

Pour permettre la communication entre les sous-réseaux et Internet, nous avons configuré :

* **Internet Gateway** : Pour permettre aux ressources au sein du VPC d'accéder à Internet.
* **NAT Gateway** : Pour permettre aux instances dans les sous-réseaux privés d'accéder à Internet.
* **Tables de Routage** : Pour diriger le trafic réseau au sein du VPC.

### **Groupes de Sécurité**

Nous avons défini des groupes de sécurité pour contrôler le trafic entrant et sortant des instances EC2. Le Bastion, ainsi que le Jenkins, le Grafana et le Proxy sont placés à l’extérieur afin de pouvoir y accéder à leur ressource web. Le Prometheus, et les différents serveurs webs et le serveur de Base de données ne sont pas à l’extérieur et communique dans un LAN. L’ensemble du management se fait depuis le Bastion accessible depuis l’extérieur avec une whiteliste.

### **Instances EC2**

Plusieurs instances EC2 ont été définies pour différents rôles :

* **Serveurs Web** : Pour héberger les applications web.
* **Proxy/Loadbalancer** : Pour rediriger le traffic des utilisateurs vers les différents serveurs web pour la gestion de charge.
* **Bases de Données** : Pour stocker les données des applications.
* **Outils de Monitoring** : Pour surveiller les performances et la santé des applications.
* **Bastion** : our accéder de manière sécurisée à l’infrastructure.

### **4. Automatisation de la configuration avec Ansible**

### **4.1 Objectifs**

Cette partie vise à automatiser la configuration complète de l’infrastructure. L’objectif est de préparer les instances EC2, déployer l’application GreenShop (front et back), installer la base de données, et mettre en place la stack de supervision (Prometheus et Grafana), le tout sans intervention manuelle.

**4.2 Infrastructure et accès**

L’inventaire Ansible est structuré par groupes de machines (web, db, monitoring, etc.). La connexion SSH se fait :

* soit en public (ex. : Jenkins, monitoring),
* soit via un bastion (adm1) pour les serveurs privés (web1 à web3, db1).

Chaque hôte possède une configuration dédiée (clé SSH, utilisateur, proxy SSH si besoin), ce qui permet une administration centralisée et sécurisée, même pour les machines en réseau privé.

### **4.3 Déploiement de l’application**

Le playbook greenshop.yml automatise :

* La préparation et le transfert des fichiers du site (/var/www) depuis le poste de contrôle
* Le nettoyage de l’ancien contenu pour éviter les conflits
* La mise en place des permissions correctes (utilisateur www-data)
* Le déploiement de la base de données sur db1 avec import automatique d’un dump SQL

Les opérations sont séparées en blocs par rôle (web et db) et tout est réalisé de manière idempotente, donc relançable sans effet secondaire.

### **4.4 Supervision avec Docker**

Un second playbook gère la mise en place de Prometheus et Grafana :

* Utilisation du rôle geerlingguy.docker pour installer Docker et Docker Compose
* Déploiement des conteneurs à partir de fichiers docker-compose.yml
* Installation ciblée sur les machines prometheus1 et grafana1

Cela permet une gestion souple des services de supervision, facilement redéployables en cas de besoin.

### **4.5 Makefile et automatisation**

Un Makefile centralise les principales commandes utilisées :

* install-ansible : installe Ansible sur la machine de contrôle
* requirements : installe les rôles définis dans requirements.yml
* greenshop : lance le playbook principal

Ce fonctionnement rend le déploiement plus rapide, reproductible, et moins sujet aux erreurs humaines.

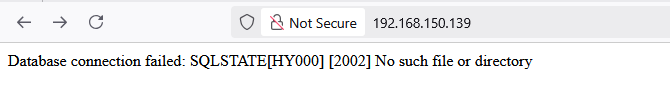
### **5. Conteneurisation de l’application**

#### **5.1. Création des Images Docker**

Pour conteneuriser notre application, nous avons écrit un Dockerfiles pour conteneuriser l’application web et la base de données. L'objectif était de créer un environnement isolé partie des applications.

* **Développement du Dockerfiles** : Nous avons créé un Dockerfiles pour l'application web et la base de données. En amont, nous avions récupéré les différentes sources des applications (dump mysql & code source du site & fichiers de conf Apache)
* **Tests Locaux et Optimisation** : Après avoir construit l’image Docker, nous avons testé localement pour nous assurer qu'elles fonctionneraient correctement.

Cependant, lors de cette phase, nous avons rencontré un problème de connectivité. Bien que nous ayons réussi à conteneuriser l'application web et à accéder à la page, une erreur de connexion est apparue. Cela indique que l'application web n'arrivait pas à se connecter à la base de données conteneurisée.  
  
Exemple de l’erreur :



### **6. Mise en place de la pipeline CI/CD avec Jenkins**

### **6.1 Objectifs de la CI/CD**

L’objectif de cette étape était d'automatiser tout le processus de livraison : de la récupération du code à la mise en production via des étapes de build, test, création d’image Docker, et déploiement sur les serveurs. L’outil retenu pour orchestrer cette chaîne était Jenkins, intégré à GitHub via des webhooks.

### **6.2 Tentative de configuration de Jenkins**

Nous avions prévu d’automatiser l’installation de Jenkins via un script Terraform exécuté lors du provisionnement de l’instance dédiée. Ce script devait :

* Installer Jenkins et ses dépendances
* Lancer le service automatiquement
* Préparer une première configuration (plugins, accès web)

Cependant, **l’exécution du script Terraform a échoué** à l’étape de provisionnement de Jenkins, à cause d’erreurs dans l’initialisation de l’environnement (packages manquants ou configuration réseau incorrecte). En conséquence, Jenkins **n’a pas pu être installé ni lancé correctement**, et aucun job n’a été configuré.

### **6.3 Pipeline envisagée**

Si l’installation avait réussi, la pipeline aurait suivi cette structure :

* Déclenchement automatique à chaque push sur GitHub via webhook
* Build de l’image Docker de l’application GreenShop
* Push de l’image vers un registre Docker (Docker Hub ou ECR)
* Déploiement automatique via Ansible ou un container orchestrator

### **7. Résultats et Bilan**

### **7.1 Résultats obtenus**

Le projet a permis de mettre en place une infrastructure cloud complète sur AWS en suivant les bonnes pratiques DevOps :

* **Infrastructure provisionnée automatiquement** avec Terraform : réseau, instances EC2, règles de sécurité, et architecture bastion/public/privé bien segmentée.
* **Configuration automatisée** grâce à Ansible, permettant le déploiement de l’application GreenShop (code + base de données) ainsi que des outils de monitoring (Prometheus, Grafana).
* **Séparation claire des rôles** entre les serveurs (web, DB, proxy, monitoring, Jenkins), facilitant la maintenance et l'évolutivité.
* **Documentation structurée et automatisation centralisée** via un Makefile pour simplifier les déploiements futurs.

### **7.2 Difficultés rencontrées**

Malgré les réussites, plusieurs problèmes ont limité l’aboutissement de certaines parties :

* **Conteneurisation partiellement fonctionnelle** : nous avons réussi à builder des images Docker pour Apache et MariaDB, mais la communication entre les conteneurs ne fonctionnait pas. Le site n'était pas accessible à cause d'un problème de connexion à la base de données dans l’environnement Docker.
* **Mise en place de Jenkins non aboutie** : bien que le script Terraform prévoyait le lancement automatique d'une instance Jenkins, il n’a pas fonctionné correctement. Par conséquent, la partie CI/CD (pipeline Jenkins, webhooks, build/push Docker) n’a pas pu être finalisée.
* **Temps de debugging** important lié aux permissions réseau, à la configuration de Docker et à la gestion des scripts de démarrage dans Terraform.

### **7.3 Pistes d’amélioration**

* Corriger et tester les **scripts de provisionnement Jenkins** pour permettre une CI/CD fonctionnelle.
* Revoir la **configuration réseau Docker** pour assurer la connectivité entre conteneurs.
* Implémenter des tests automatisés dans la pipeline pour garantir la stabilité à chaque mise à jour.

### **8.0 Annexe**

### **8.1 Schéma de l’infrastructure**