



Ensimag



---

Déploiement et Sécurisation d'une Infrastructure  
Cloud pour une application WEB :  
OWASP Juice Shop.

---

*Réalisé par le Groupe 1 :*

BOUMEHDI Mohammed-Yassine  
ENNOUR Nassim  
QUEHLAOUI Mohamed

*Encadré par :*

Pr. AMAR Paul

Année Universitaire 2024/2025

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Contexte du projet . . . . .	7
1.2	Technologies et outils prévus . . . . .	8
<b>2</b>	<b>Infrastructure en tant que Code</b>	<b>9</b>
2.1	Architecture logicielle : IaC . . . . .	9
2.2	Justification des choix . . . . .	10
<b>3</b>	<b>Politique de sécurité</b>	<b>12</b>
3.1	Architecture logicielle : Sécurité . . . . .	12
3.2	Stratégies de sécurité . . . . .	13
3.3	Configuration détaillée de l'ALB . . . . .	18
3.3.1	Appartenance au VPC . . . . .	19
3.3.2	Règles du Security Group . . . . .	19
3.3.3	Configuration du Listeners de l'ALB . . . . .	20
3.3.4	Mapping réseau de l'ALB . . . . .	21
3.3.5	Sécurité renforcée . . . . .	21
3.4	Configuration détaillée du cluster EKS . . . . .	25
3.4.1	Intégration avec le VPC . . . . .	25
3.4.2	Capacité et mise à l'échelle automatique . . . . .	25
3.4.3	Instances et Sécurité . . . . .	26
3.4.4	Détails supplémentaires de l'auto-scaling group . . . . .	27
3.5	Surveillance et accès sécurisé . . . . .	30
3.5.1	CloudWatch - Surveillance de l'ALB . . . . .	30
3.5.2	Le Jump Host (Bastion) . . . . .	32

3.6	Tests de sécurité effectués sur le WAF . . . . .	32
3.6.1	Blocage des connexions hors de France . . . . .	32
3.6.2	Protection contre les injections SQL . . . . .	35
3.6.3	Conclusion des tests . . . . .	35
3.7	Automatisation du déploiement : Pipeline CI/CD avec GitLab	36
3.7.1	Structure du Pipeline CI/CD . . . . .	37
3.7.2	Variables d'Environnement Essentielles . . . . .	40
3.7.3	Avantages et Résultats du Pipeline CI/CD . . . . .	41
3.7.4	Conclusion . . . . .	42
3.8	Détection des flags via AWS WAF . . . . .	42
3.8.1	Configuration des règles dans AWS WAF . . . . .	42
3.8.2	Logs et détection dans CloudWatch . . . . .	44
3.8.3	Requêtes interceptées . . . . .	44
	<b>Conclusion</b>	<b>47</b>
	<b>Bibliographie</b>	<b>48</b>

# Liste des figures

2.1	Architecture logicielle : IaC	9
3.1	Architecture logicielle : Sécurité	12
3.2	Création et configuration du nom de domaine sur Namecheap.	14
3.3	Gestion du domaine via AWS ACM pour la certification SSL.	14
3.4	Notre Domain est désormais sécurisé.	15
3.5	Configuration des règles WAF sur AWS.	16
3.6	Architecture VPC : Configuration des sous-réseaux publics et privés, des tables de routage, de l'Internet Gateway et du NAT Gateway.	18
3.7	Détails du Load Balancer (ALB) utilisé dans la configuration AWS.	18
3.8	Configuration des listeners dans l'ALB	21
3.9	Mapping réseau de l'ALB avec les sous-réseaux et zones de disponibilité.	21
3.10	Configuration du Security Group pour l'ALB.	22
3.11	Vue des pods du contrôleur de Load Balancer en cours d'exécution dans le cluster.	22
3.12	Configuration du certificat dans AWS Certificate Manager (ACM)	24
3.13	Détails du groupe de cibles pour l'ALB, montrant les cibles saines et leur répartition.	24
3.14	Gestion des auto-scaling groups pour les nœuds du cluster EKS.	26

3.15	Détails du groupe de nœuds et de ses instances dans le cluster EKS. . . . .	27
3.16	Surveillance des ressources CPU et Mémoire pour le nœud dans le cluster EKS. . . . .	28
3.17	Liste des pods en fonctionnement dans le cluster EKS. . . . .	29
3.18	Pod "juice-shop" sur le nœud <code>ip-10-0-0-177.ec2.internal</code> . . . . .	30
3.19	Pod "juice-shop" sur le nœud <code>ip-10-0-2-71.ec2.internal</code> . . . . .	30
3.20	Vue des métriques CloudWatch pour l'Application Load Balancer, montrant notamment les erreurs HTTP, le nombre de requêtes, et les ressources consommées. . . . .	31
3.21	Connexion via un VPN basé aux Pays-Bas. . . . .	33
3.22	accès au site via un VPN basé aux Pays-Bas. . . . .	33
3.23	Statistiques des requêtes bloquées par la règle géographique. . . . .	34
3.24	Évolution des requêtes bloquées et autorisées pour la règle géographique. . . . .	34
3.25	Blocage d'une tentative d'injection SQL par le WAF. . . . .	35
3.26	Pipeline GitLab CI/CD : Succès des différentes étapes du déploiement. . . . .	36
3.27	Détails du pipeline réussi sur GitLab. . . . .	36
3.28	Configuration du stage <code>provision</code> dans GitLab CI/CD. . . . .	38
3.29	Configuration du stage <code>deploy</code> dans GitLab CI/CD. . . . .	39
3.30	Configuration du stage <code>secure</code> dans GitLab CI/CD. . . . .	40
3.31	Configuration des variables d'environnement dans GitLab CI/CD. . . . .	41
3.32	Pipeline exécuté avec succès. . . . .	42
3.33	Détails de la règle <code>detect_flag_body</code> dans AWS WAF. . . . .	43
3.34	Action associée à la règle <code>detect_flag_body</code> (Count). . . . .	43
3.35	Groupe de logs configuré dans CloudWatch pour capturer les événements WAF. . . . .	44
3.36	Démonstration de la détection par la règle <code>detect_flag_body</code> . . . . .	45
3.37	Tableau des requêtes échantillons détectées dans AWS WAF. . . . .	45
3.38	La requête dans laquelle on a détecté l'existence d'un flag dans le Body. . . . .	46



# Liste des abréviations

**OWASP** Open Worldwide Application Security Project

**AWS** Amazon Web Services

**WAF** Web Application Firewall

**EC2** Elastic Compute Cloud

**EKS** Elastic Kubernetes Service

**ALB** Application Load Balancer

**VPC** Virtual Private Cloud

**ACM** AWS Certificate Manager

**SSH** Secure Shell

**IaC** Infrastructure as Code

**DNS** Domain Name System

# Chapitre 1

## Introduction

### 1.1 Contexte du projet

Le projet global vise à concevoir et déployer une architecture logicielle performante et sécurisée, soutenue par une infrastructure moderne et évolutive. Un aspect clé de ce projet est l'étude et l'analyse des vulnérabilités de sécurité potentielles auxquelles les applications web peuvent être confrontées. Dans ce cadre, l'application **OWASP Juice Shop**, qui est reconnue comme étant probablement la plus moderne et sophistiquée des applications web intentionnellement vulnérables, sera utilisée comme référence pour simuler et comprendre les failles de sécurité typiques.

L'objectif spécifique de ce rapport est de présenter la politique de sécurité et les aspects techniques de l'infrastructure envisagée, en détaillant les mesures nécessaires pour garantir la sécurité et la robustesse du système. L'OWASP Juice Shop permet de mettre en évidence des problèmes de sécurité courants et d'étudier comment l'IaC et des pratiques de sécurité avancées peuvent être mises en œuvre pour les atténuer efficacement.

Les enjeux principaux de l'infrastructure envisagée incluent la nécessité de créer des environnements reproductibles, sécurisés et conformes aux normes. L'utilisation de l'IaC permet de réduire les erreurs manuelles,

d'automatiser les déploiements et d'assurer la cohérence entre les différents environnements, tout en garantissant que les meilleures pratiques de sécurité sont intégrées dès le début du développement.

## 1.2 Technologies et outils prévus

Dans le cadre de ce projet, plusieurs outils et technologies seront utilisés pour assurer la mise en place efficace et sécurisée de l'infrastructure et des processus associés. Ces outils sont essentiels pour garantir l'automatisation, la gestion cohérente des environnements et la sécurité de l'application. Voici un aperçu des principaux outils envisagés :

- **Ansible** : Utilisé pour l'automatisation de la configuration et la gestion des déploiements, Ansible simplifie le déploiement d'applications et la gestion des serveurs, réduisant ainsi les risques d'erreurs humaines.
- **Terraform** : Choisi pour la gestion de l'infrastructure en tant que code (IaC), Terraform permet la création, la modification et la version de l'infrastructure en toute sécurité et efficacité.
- **GitLab** : Utilisé pour la gestion du code source et l'intégration continue/déploiement continu (CI/CD), GitLab facilite le versionnement et l'automatisation des pipelines de déploiement.
- **Docker** : Permet la containerisation des applications pour assurer la portabilité, la cohérence entre les environnements et la réduction des dépendances système.
- **AWS** : Choisi comme plateforme cloud pour héberger et gérer les services, AWS offre des solutions évolutives et sécurisées pour déployer des applications et des services web.
- **Draw.io** : Utilisé pour créer des schémas et des diagrammes, Draw.io permet de concevoir et de documenter visuellement l'architecture et les processus techniques.

# Chapitre 2

## Infrastructure en tant que Code

### 2.1 Architecture logicielle : IaC

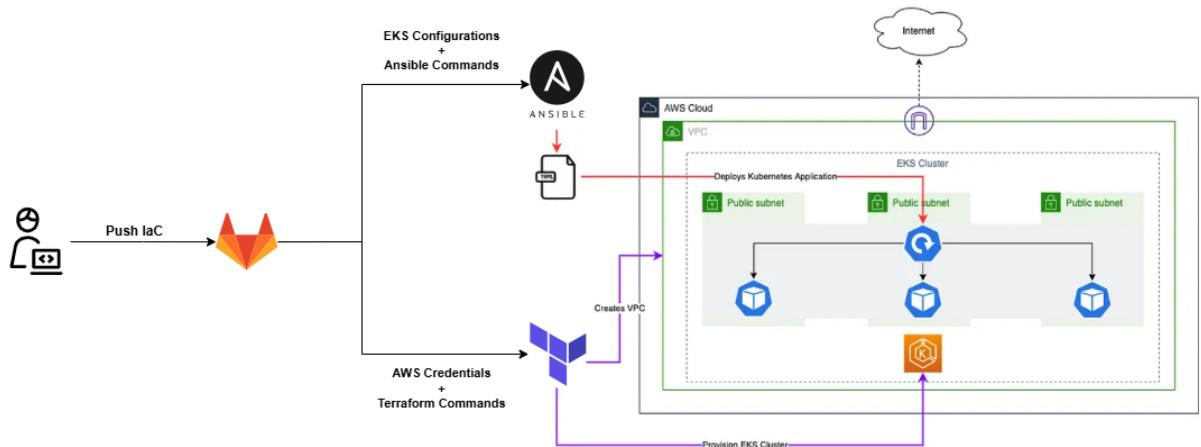


Figure 2.1: Architecture logicielle : IaC

Nous avons mis en place un pipeline d'automatisation solide pour gérer efficacement le provisionnement et le déploiement de notre infrastructure sur AWS. GitLab CI est au cœur du processus : à chaque modification de notre code, il déclenche des commandes Terraform pour gérer notre VPC, notre cluster EKS et d'autres configurations de sécurité. L'objectif est d'assurer que toute l'infrastructure de base soit correctement provisionnée et alignée sur nos besoins.

Une fois cette infrastructure en place, Ansible prend le relais pour déployer nos configurations Kubernetes (Deployments, Services, etc.) sur le cluster EKS. Cela nous permet de contrôler précisément la configuration tout en garantissant un accès optimal et sécurisé.

En somme, cette architecture nous permet de déployer de manière fiable et répétable, tout en intégrant des outils de gestion d'infrastructure et de configuration, ce qui améliore notre agilité et réduit les risques d'erreurs humaines. Nous sommes convaincus que cette approche répond aux exigences de flexibilité et de sécurité, tout en exploitant au mieux les fonctionnalités des outils modernes comme Terraform, Ansible et GitLab.

## 2.2 Justification des choix

- **Choix de Terraform pour le Provisionnement :**

Terraform a été sélectionné pour provisionner l'infrastructure AWS, car il offre une syntaxe déclarative qui facilite la définition des ressources telles que les clusters EKS, les sous-réseaux et le VPC. De plus, grâce à son fichier d'état, Terraform permet un suivi précis des ressources, garantissant que les modifications sont appliquées de manière contrôlée. Cette solution convient parfaitement à notre besoin de créer des infrastructures évolutives et contrôlées.

- **Rôle d'Ansible dans le Déploiement :**

Ansible est préféré pour le déploiement et la gestion des configurations, car il est particulièrement adapté pour appliquer des configurations détaillées et automatiser des tâches sur des ressources déjà provisionnées. Là où Terraform se charge de l'infrastructure, Ansible intervient après pour configurer les applications et les conteneurs, notamment dans un cluster Kubernetes. Ansible peut utiliser les "outputs" générés par Terraform, comme le fichier kubeconfig contenant les informations de connexion au cluster EKS, ce qui lui permet d'automatiser l'accès et le déploiement des

applications directement après le provisionnement. Dans ce flux, Terraform génère les outputs nécessaires, puis Ansible les utilise comme entrée pour ses playbooks de déploiement.

- **Rôle de Gitlab CI :**

GitLab CI automatise le cycle de vie de l'infrastructure et des applications via des pipelines CI/CD. À chaque push de code, il déclenche un pipeline qui exécute Terraform pour provisionner l'infrastructure, suivi d'Ansible pour déployer les configurations. Les identifiants AWS (**AWS\_ACCESS\_KEY\_ID**, **AWS\_SECRET\_ACCESS\_KEY**, **AWS\_SESSION\_TOKEN**) sont stockés comme variables sécurisées, assurant leur protection tout en permettant leur utilisation automatique dans les scripts. Cette approche rend le déploiement rapide, cohérent et réduit les erreurs humaines en automatisant chaque étape du processus.

# Chapitre 3

## Politique de sécurité

### 3.1 Architecture logicielle : Sécurité

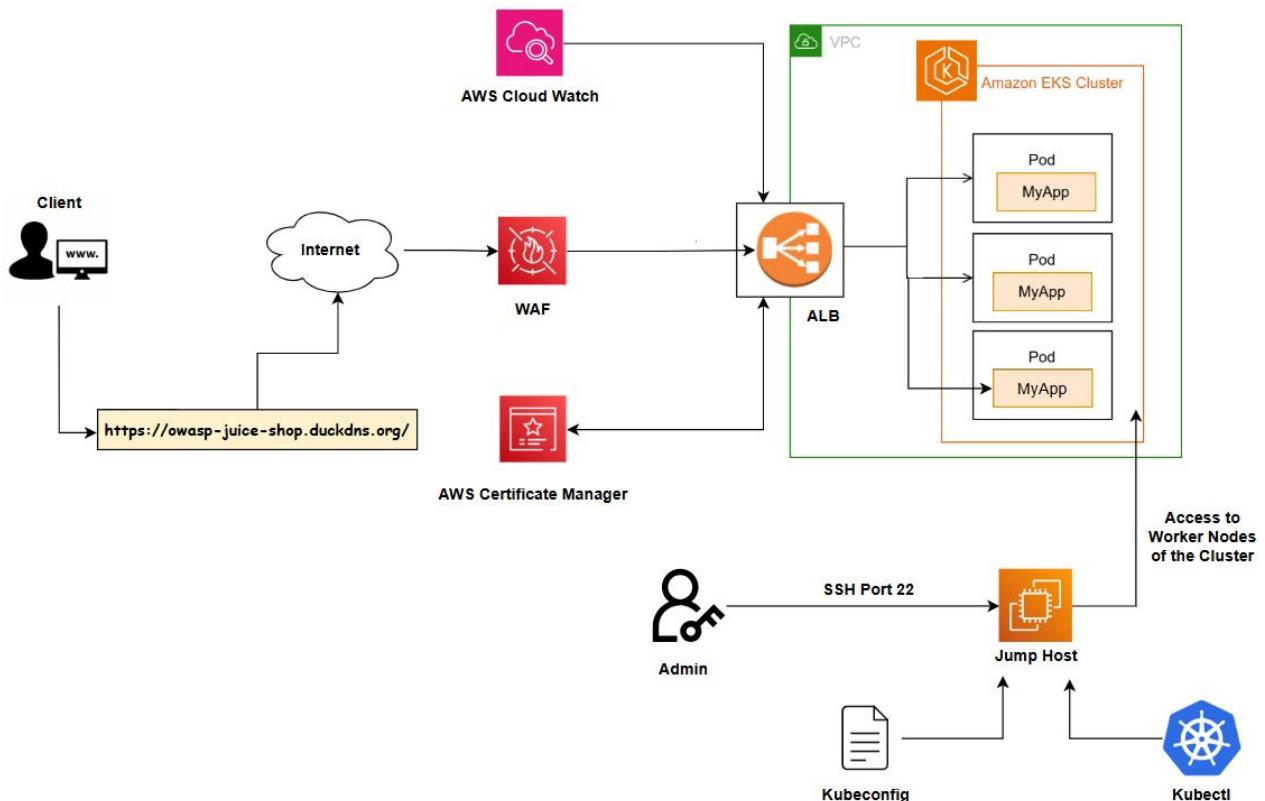


Figure 3.1: Architecture logicielle : Sécurité

La sécurité est un pilier essentiel dans la conception et la gestion de toute architecture logicielle moderne, et l'architecture présentée ne fait pas exception. Cette section détaille les mesures et les pratiques de sécurité à mettre en place pour protéger l'ensemble des composants de l'infrastructure, depuis les points d'entrée jusqu'aux applications hébergées.

## 3.2 Stratégies de sécurité

**DNS et WAF (Première ligne de défense) :**

**\* Pour la partie DNS :**

Initialement, nous avions prévu d'utiliser le service DNS d'AWS pour gérer le nom de domaine de notre application. Cependant, en raison des restrictions liées à notre compte AWS Academy (Learner Lab), nous avons dû opter pour une alternative externe. Cette adaptation nous a permis de conserver une architecture fonctionnelle tout en respectant les exigences du projet.

**\*\* Solution mise en place :**

1. Création d'un nom de domaine externe : Nous avons d'abord utilisé DuckDns comme solution temporaire. Cependant, pour offrir une meilleure gestion et un domaine plus professionnel, nous avons acquis un domaine via Namecheap : <https://owasp-juice-shop.me>.
2. Certification de notre nom de domaine en utilisant **ACM (AWS Certificate Manager)** pour sécuriser les connexions.

The screenshot shows the Namecheap DNS management interface for the domain `owasp-juice-shop.me`. The top navigation bar includes icons for Home, Domain, Products, Sharing & Transfer, and Advanced DNS (which is highlighted in teal). Below the navigation is a dropdown for 'DNS TEMPLATES' and a search bar. The main area displays 'HOST RECORDS' with three entries:

Type	Host	Value	TTL
CNAME Record	@	k8s-juicesho-juicesho-b8e52314c0-1734737359.us-east-1.elb.amazonaws.com	1 min
CNAME Record	_2cbb5efa3c99d2...	_3fb04cb832ab610dc6c75c59f435cfedjqtssxkq.acm-validat...	Automatic
CNAME Record	www	owasp-juice-shop.me	1 min

Figure 3.2: Création et configuration du nom de domaine sur Namecheap.

## Certificate

The screenshot shows the AWS Certificate Manager (ACM) details for the domain `owasp-juice-shop.me`. The certificate is issued by `Amazon RSA 2048 M02` and signed by `Amazon Root CA 1`.

**Subject Name**

Common Name	owasp-juice-shop.me
-------------	---------------------

**Issuer Name**

Country	US
Organization	Amazon
Common Name	<a href="#">Amazon RSA 2048 M02</a>

**Validity**

Not Before	Thu, 28 Nov 2024 00:00:00 GMT
Not After	Sat, 27 Dec 2025 23:59:59 GMT

**Subject Alt Names**

DNS Name	owasp-juice-shop.me
----------	---------------------

Figure 3.3: Gestion du domaine via AWS ACM pour la certification SSL.

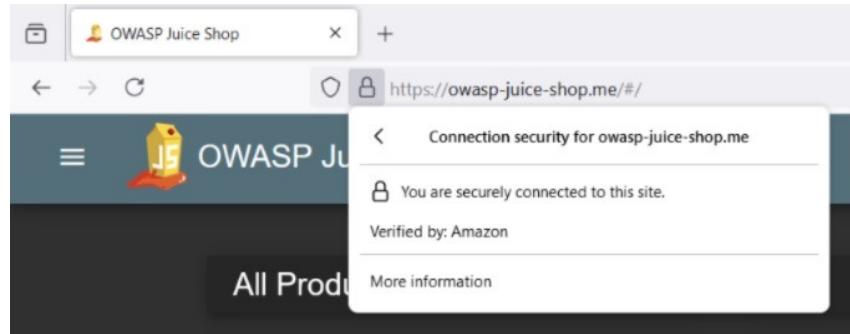


Figure 3.4: Notre Domain est désormais sécurisé.

### \* Partie WAF : Renforcement de la Sécurité

Dans le cadre de la protection de notre application web, nous avons configuré un **Web Application Firewall (WAF)** sur AWS. Cette étape est cruciale pour détecter et prévenir les attaques potentielles sur notre infrastructure.

#### Règles configurées dans le WAF :

Nous avons mis en place une combinaison de règles spécifiques et gérées pour offrir une défense robuste contre les menaces courantes. Voici les détails des règles définies :

- **BlockNonFrenchIPs (Priorité : 0) :**

Cette règle bloque les adresses IP provenant de pays autres que la France, restreignant ainsi l'accès à l'application à une audience ciblée.

- **AWSManagedRulesAnonymousIpList (Priorité : 1) :**

Utilisation des règles gérées par AWS pour bloquer les IP associées à des proxys anonymes et des services VPN, réduisant le risque d'accès frauduleux.

- **AWSManagedRulesCommonRuleSet (Priorité : 2) :**

Implémentation des règles communes fournies par AWS pour détecter les menaces génériques, telles que les injections SQL et les attaques XSS.

- **AWSManagedRulesKnownBadInputsRuleSet (Priorité : 3)** :  
Protection contre les entrées malveillantes connues, basée sur une analyse approfondie des modèles d'attaque.
- **AWSManagedRulesSQLiRuleSet (Priorité : 4) :**  
Spécifiquement conçu pour bloquer les tentatives d'injection SQL, renforçant la sécurité des bases de données connectées à notre application.

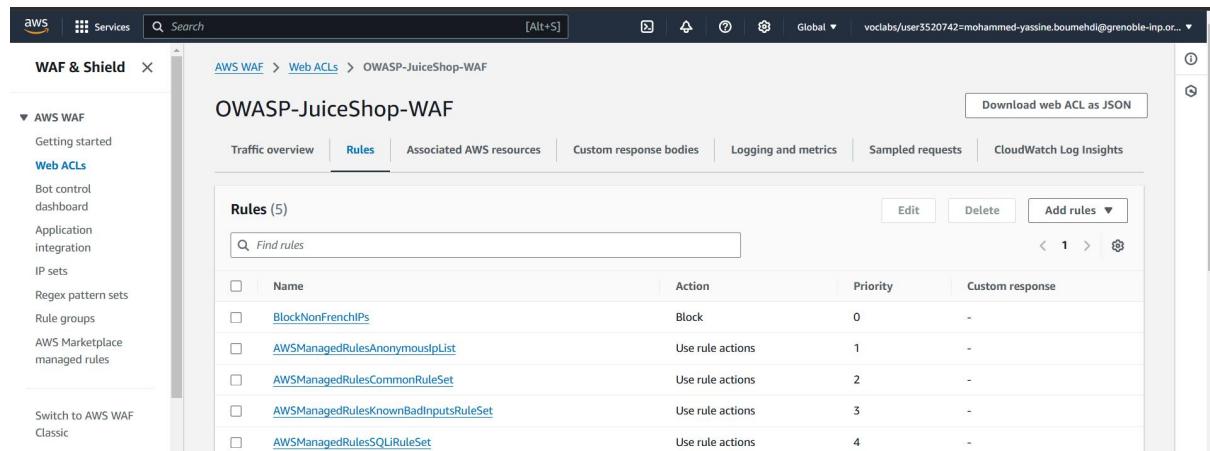


Figure 3.5: Configuration des règles WAF sur AWS.

## Architecture VPC détaillée

Dans le cadre de la configuration réseau de notre application, une architecture Virtual Private Cloud (VPC) a été mise en place sur AWS. Cette architecture exploite six sous-réseaux répartis sur trois zones de disponibilité (*Availability Zones, AZ*) pour garantir une haute disponibilité et une séparation des charges entre sous-réseaux publics et privés.

**Sous-réseaux publics :**

1. **Déploiement multi-AZ** : Trois sous-réseaux publics (`k8s-public-subnet-1`, `k8s-public-subnet-2`, et `k8s-public-subnet-3`) sont configurés, chacun étant situé dans une AZ différente (`us-east-1a`, `us-east-1b`, et `us-east-1c`).
2. **Tables de routage associées** : Une table de routage publique (`k8s-public-rt`) connectée à une Internet Gateway (`k8s-igw`) permet aux ressources des sous-réseaux publics d'avoir un accès direct à Internet.

**Sous-réseaux privés :**

1. **Hébergement sécurisé** : Trois sous-réseaux privés (`k8s-private-subnet-1`, `k8s-private-subnet-2`, et `k8s-private-subnet-3`) sont déployés, avec un dans chaque AZ. Ces sous-réseaux servent à héberger des ressources sensibles, comme les nœuds Elastic Kubernetes Service (EKS), en restant isolés d'Internet.
2. **Tables de routage privées** : Une table de routage (`k8s-private-rt`) redirige le trafic sortant des sous-réseaux privés vers un NAT Gateway (`k8s-nat-gateway`) dans un sous-réseau public, permettant un accès sécurisé à Internet sans exposition directe.

**Connexion réseau :**

- **Internet Gateway (`k8s-igw`)** : Connectée aux sous-réseaux publics pour permettre un accès direct à Internet.
- **NAT Gateway (`k8s-nat-gateway`)** : Située dans les sous-réseaux publics, elle permet aux ressources des sous-réseaux privés de communiquer avec Internet pour les besoins de mise à jour et autres accès sortants.

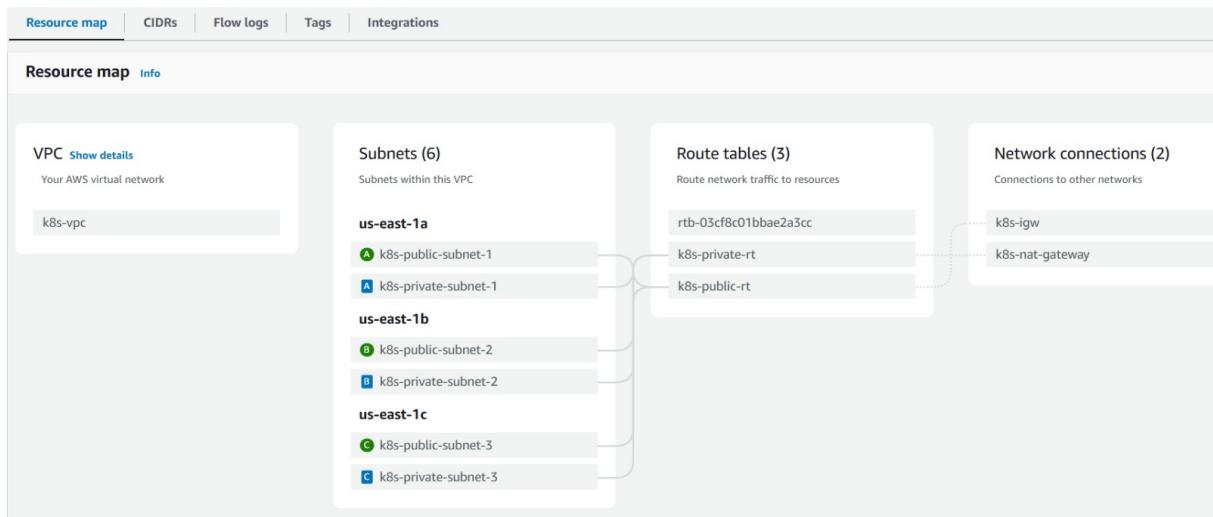


Figure 3.6: Architecture VPC : Configuration des sous-réseaux publics et privés, des tables de routage, de l’Internet Gateway et du NAT Gateway.

### 3.3 Configuration détaillée de l’ALB

L’Application *Load Balancer (ALB)* a été sécurisé en définissant un *Security Group* rigoureux, comme le montre la Figure 3.10. Cette configuration vise à garantir que l’ALB autorise uniquement le trafic nécessaire et filtre les connexions indésirables.

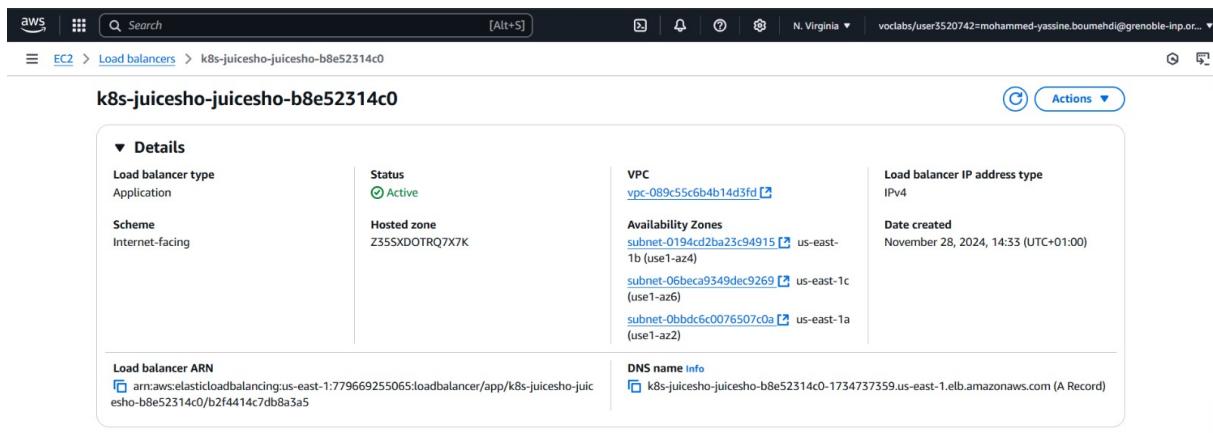


Figure 3.7: Détails du Load Balancer (ALB) utilisé dans la configuration AWS.

## Description des détails du Load Balancer :

- **Type de Load Balancer :** Application Load Balancer (ALB), utilisé pour gérer le trafic HTTP et HTTPS.
- **Schéma :** Internet-facing, signifiant que l'ALB est accessible depuis Internet, ce qui est typique pour les applications web accessibles au public.
- **VPC :** L'ALB est associé au VPC `vpc-089c55c56b4b14d3fd`, assurant un environnement réseau isolé et sécurisé.
- **Zones de disponibilité :** L'ALB est déployé dans les sous-réseaux publics suivants, couvrant plusieurs zones de disponibilité :
  - `subnet-0194cd2ba23c94915` (us-east-1b)
  - `subnet-06bceca9349dec9269` (us-east-1c)
  - `subnet-0bbdc6c0076507c0a` (us-east-1a)
- **Type d'adresse IP :** IPv4, offrant une compatibilité étendue avec les clients.
- **DNS :** `k8s-juicesho-juicesho-b8e52314c0-1734737359.us-east-1.elb.amazonaws.com`, utilisé pour accéder à l'ALB via un nom de domaine.

### 3.3.1 Appartenance au VPC

Le *Security Group* de l'ALB appartient au VPC qui a été créé précédemment. Cela garantit une isolation réseau et un contrôle strict des flux de données au sein de l'infrastructure cloud.

### 3.3.2 Règles du Security Group

Règles entrantes (Inbound)

- **HTTPS (Port 443) :**

- **Protocole :** TCP.
  - **Source :** Tout trafic (0.0.0.0/0) est autorisé pour permettre les connexions HTTPS depuis des navigateurs ou outils compatibles SSL/TLS.
  - **Justification :** Ce port est utilisé pour sécuriser les communications entre les utilisateurs finaux et l’ALB.
- **HTTP (Port 80) :**
    - **Protocole :** TCP.
    - **Source :** Tout trafic (0.0.0.0/0).
    - **Justification :** Port activé pour des raisons de test (**désactivé par la suite**).

#### Règles sortantes (Outbound)

- **Trafic sortant :** Limité uniquement vers les pods hébergeant les applications.
- **Ports :** Définis selon les services applicatifs déployés (spécifiés dans vos configurations Kubernetes).

#### 3.3.3 Configuration du Listeners de l’ALB

La gestion du trafic entrant et sortant au sein de l’ALB est configurée de manière spécifique pour garantir la sécurité de l’infrastructure. Comme montré dans la figure suivante, l’ALB écoute sur les ports HTTP (80) et HTTPS (443), avec des règles définissant les actions appropriées en fonction du type de requête (**Le port HTTP a été désactivé par la suite**).

Protocol:Port	Default action	Rules	ARN	Security policy	Default SSL/TLS
HTTPS:443	Return fixed response • Response code: 404 • Response body • Response content type: text/plain	2 rules	ARN	ELBSecurityPolicy-2016-08	owasp-juice-shop
HTTP:80	Return fixed response • Response code: 404 • Response body • Response content type: text/plain	2 rules	ARN	Not applicable	Not applicable

Figure 3.8: Configuration des listeners dans l’ALB

### 3.3.4 Mapping réseau de l’ALB

Pour mieux comprendre la correspondance entre l’ALB, les sous-réseaux et les zones de disponibilité, la Figure 3.9 illustre le mapping réseau. Cette configuration montre les liens entre les composants réseau (sous-réseaux, VPC, etc.) et le Load Balancer.

Zone	Subnet	IPv4 address	Private IPv4 address	IPv6 address
us-east-1b (use1-az4)	subnet-0194cd2ba23c94915	Assigned by AWS	Assigned from CIDR 10.0.5.0/24	Not applicable
us-east-1c (use1-az6)	subnet-06beca9349dec9269	Assigned by AWS	Assigned from CIDR 10.0.6.0/24	Not applicable
us-east-1a (use1-az2)	subnet-0bbdc6c0076507c0a	Assigned by AWS	Assigned from CIDR 10.0.4.0/24	Not applicable

Figure 3.9: Mapping réseau de l’ALB avec les sous-réseaux et zones de disponibilité.

### 3.3.5 Sécurité renforcée

Les configurations des ports et des sources garantissent que l’ALB accepte uniquement les connexions nécessaires à son fonctionnement.

Pour renforcer cette sécurité, il est recommandé d'ajouter des règles spécifiques limitant les sources autorisées, par exemple en restreignant à l'adresse IP du WAF.

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-036169a82fabdf7af	HTTPS	TCP	443	Cu... 0.0.0.0/0	<a href="#">Delete</a>
sgr-0603cdf823c431046	HTTP	TCP	80	Cu... 0.0.0.0/0	<a href="#">Delete</a>

Figure 3.10: Configuration du Security Group pour l'ALB.

### Déploiement des pods pour le contrôleur de Load Balancer :

Afin de gérer le trafic entrant et de garantir que le Load Balancer distribue correctement le trafic aux cibles appropriées, nous avons utilisé **Helm** pour installer un **AWS Load Balancer Controller**, qui se charge de déployer des **pods** du contrôleur de l'ALB pour chaque Replica de l'application dans notre cluster EKS. Les détails de ces pods sont visibles dans la figure 3.11, montrant deux instances du contrôleur de Load Balancer qui sont en cours d'exécution dans le cluster.

Name	Age
<a href="#">aws-load-balancer-controller-86bb8bf8cc-ftxn5</a>	Created 9 minutes ago
<a href="#">aws-load-balancer-controller-86bb8bf8cc-qsoftw</a>	Created 9 minutes ago

Figure 3.11: Vue des pods du contrôleur de Load Balancer en cours d'exécution dans le cluster.

- **Nom des pods :** Les pods portent des noms tels que ‘aws-load-balancer-controller-86bb8bf8cc-ftxn5’ et ‘aws-load-balancer-controller-86bb8bf8cc-qsfw’, ce qui indique qu’ils font partie du même ReplicaSet et assurent la haute disponibilité du contrôleur de Load Balancer.
- **Statut des pods :** Les pods sont en bon état, comme en témoigne le statut ”Created” affiché, montrant qu’ils sont opérationnels et prêts à gérer les connexions du Load Balancer.
- **Rôle dans la gestion du trafic :** Ces pods facilitent la gestion du trafic HTTP/HTTPS vers les services Kubernetes en interagissant avec l’ALB pour diriger le trafic vers les bonnes cibles en fonction des règles définies.

#### \* Configuration SSL/TLS :

Pour sécuriser les connexions au sein de notre infrastructure, nous avons recours à **AWS Certificate Manager (ACM)** pour la gestion des certificats SSL/TLS. ACM nous permet d’obtenir et de gérer des certificats de manière simplifiée et automatisée.

- **Intégration avec ALB (Application Load Balancer) :**

Le certificat ACM est associé à notre Application Load Balancer (ALB), qui redirige le trafic vers les différents pods de notre application sur Amazon EKS. Grâce à cette intégration, le trafic entre les utilisateurs et le ALB est sécurisé via HTTPS.

- **Utilisation d’un domaine externe sécurisé :**

Nous avons configuré un domaine externe via Namecheap (<https://owasp-juice-shop.me/>) et l’avons validé avec ACM. Cela permet de fournir un accès sécurisé et chiffré à notre application, renforçant la confidentialité des données échangées.

En utilisant ACM, nous assurons une connexion **HTTPS** fiable pour les utilisateurs finaux, tout en simplifiant la gestion des certificats dans notre architecture.

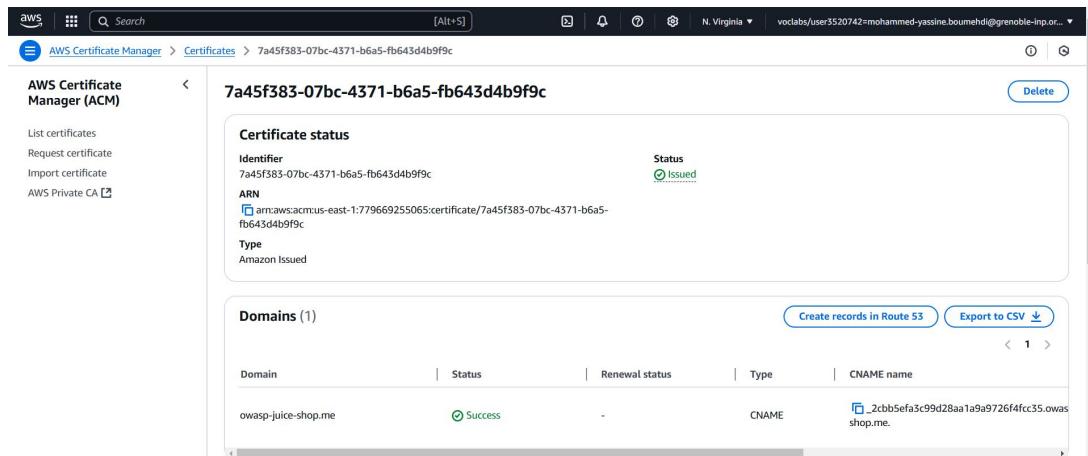


Figure 3.12: Configuration du certificat dans AWS Certificate Manager (ACM)

### Configuration du groupe de cibles (Target Group) :

Pour garantir une gestion optimale du trafic et de la répartition de charge entre les pods de notre application, nous avons configuré un groupe de cibles associé au Load Balancer. Ce groupe de cibles définit les cibles (pods) qui recevront le trafic HTTP redirigé par l'ALB. Les détails du groupe de cibles sont illustrés dans la figure 3.13.

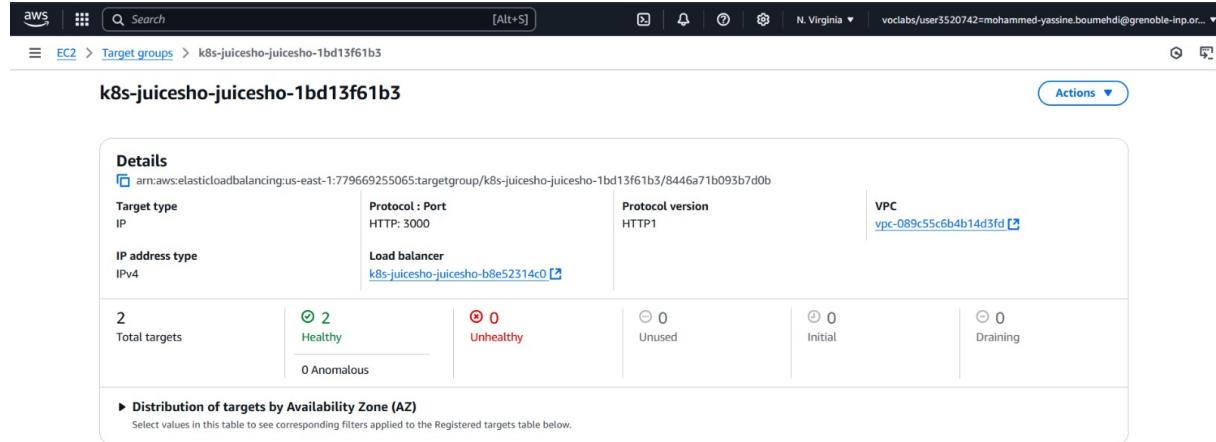


Figure 3.13: Détails du groupe de cibles pour l'ALB, montrant les cibles saines et leur répartition.

- Protocole et Port :** Le groupe de cibles utilise le protocole HTTP sur le port 3000, qui est le port d'écoute des pods associés au service.

- **Cibles en bonne santé** : Comme illustré, les deux cibles dans le groupe sont saines et prêtes à recevoir du trafic.
- **VPC associé** : Le groupe de cibles est rattaché au VPC vpc-089c55c6b4b14d3fd.
- **Rôle dans la gestion du trafic** : Ce groupe de cibles permet à l'ALB de diriger efficacement le trafic vers les pods disponibles, ce qui garantie une haute disponibilité et une répartition équilibrée de la charge entre les instances.

## 3.4 Configuration détaillée du cluster EKS

Pour garantir une gestion sécurisée et optimisée des nœuds de notre cluster Kubernetes, nous avons configuré un groupe de nœuds (*Node Group*) associé à notre cluster EKS. Les principales configurations et paramètres mis en place sont décrits ci-dessous :

### 3.4.1 Intégration avec le VPC

Le groupe de nœuds est rattaché au VPC précédemment créé et utilise les sous-réseaux privés. Cela garantit une isolation réseau et un contrôle précis du trafic.

### 3.4.2 Capacité et mise à l'échelle automatique

- **Nombre minimum de nœuds** : 1 nœud, pour assurer la disponibilité minimale du cluster.
- **Nombre désiré de nœuds** : 2 nœuds, pour répondre aux charges de travail standard.
- **Nombre maximum de nœuds** : 3 nœuds, permettant au cluster de s'adapter aux pics de charge grâce à la mise à l'échelle automatique.

Cette configuration assure une performance optimale, tout en limitant les coûts d'infrastructure.

La mise à l'échelle automatique est renforcée par la gestion des auto-scaling groups, comme montré dans la figure 3.14, illustrant le groupe de nœuds et les instances associées.

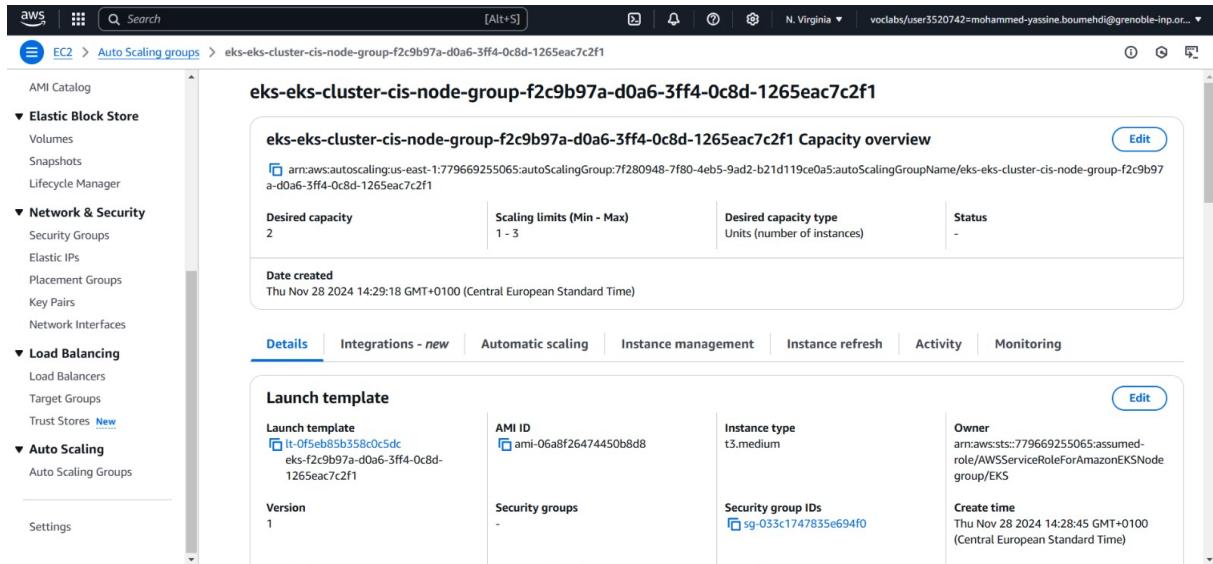


Figure 3.14: Gestion des auto-scaling groups pour les nœuds du cluster EKS.

### 3.4.3 Instances et Sécurité

Chaque nœud du cluster est sécurisé par des *Security Groups* spécifiques et utilise des types d'instances AWS EC2 optimisées pour les charges de travail Kubernetes. Ces groupes permettent également un accès contrôlé aux ressources internes et aux services du cluster, tout en filtrant les connexions indésirables.

Avec cette configuration, notre cluster EKS est prêt à supporter des charges de travail dynamiques, tout en maintenant un haut niveau de sécurité et de disponibilité.

### 3.4.4 Détails supplémentaires de l'auto-scaling group

En complément des configurations principales, les groupes d'auto-scaling offrent les fonctionnalités suivantes :

- **Paramètres d'auto-scaling** : Basés sur des métriques spécifiques comme l'utilisation du CPU ou de la mémoire.
- **Régions couvertes** : L'auto-scaling group est réparti sur plusieurs zones de disponibilité pour améliorer la tolérance aux pannes.
- **Intégration avec le Load Balancer** : Chaque instance EC2 dans le groupe est automatiquement enregistrée avec l'Application Load Balancer pour assurer une distribution efficace du trafic.

Node name	Instance type	Node group	Created	Status
ip-10-0-0-177.ec2.internal	t3.medium	eks-cluster-cis-node-group	Created 15 minutes ago	Ready
ip-10-0-2-71.ec2.internal	t3.medium	eks-cluster-cis-node-group	Created 15 minutes ago	Ready

Figure 3.15: Détails du groupe de nœuds et de ses instances dans le cluster EKS.

**Type d'instances et configuration des nœuds** : Les nœuds utilisent des instances de type **t3.medium**, offrant un équilibre entre performances et coût. Chaque instance est équipée d'un disque de 20 GiB pour le stockage des données nécessaires au fonctionnement du cluster.

**Gestion des rôles et permissions :** Le groupe de nœuds est associé à un rôle IAM spécifique, permettant aux instances d'interagir avec les services AWS nécessaires tout en limitant les accès non requis.

**Configuration réseau :** L'accès à distance est désactivé pour renforcer la sécurité et empêcher les connexions non autorisées. Les noeuds fonctionnent dans des sous-réseaux définis pour un contrôle précis de la connectivité. Ceci est illustré dans la figure 3.14 où les sous-réseaux sont spécifiés.

### Surveillance de l'utilisation des ressources :

- **CPU et Mémoire :** La gestion des ressources pour chaque nœud est surveillée à l'aide de graphiques détaillant l'allocation des ressources. Par exemple, la figure 3.16 montre l'utilisation du processeur et de la mémoire pour un nœud particulier. On observe qu'il y a une utilisation de 79% de la capacité du processeur et 82% de la mémoire disponible, laissant suffisamment de ressources pour le fonctionnement des pods.

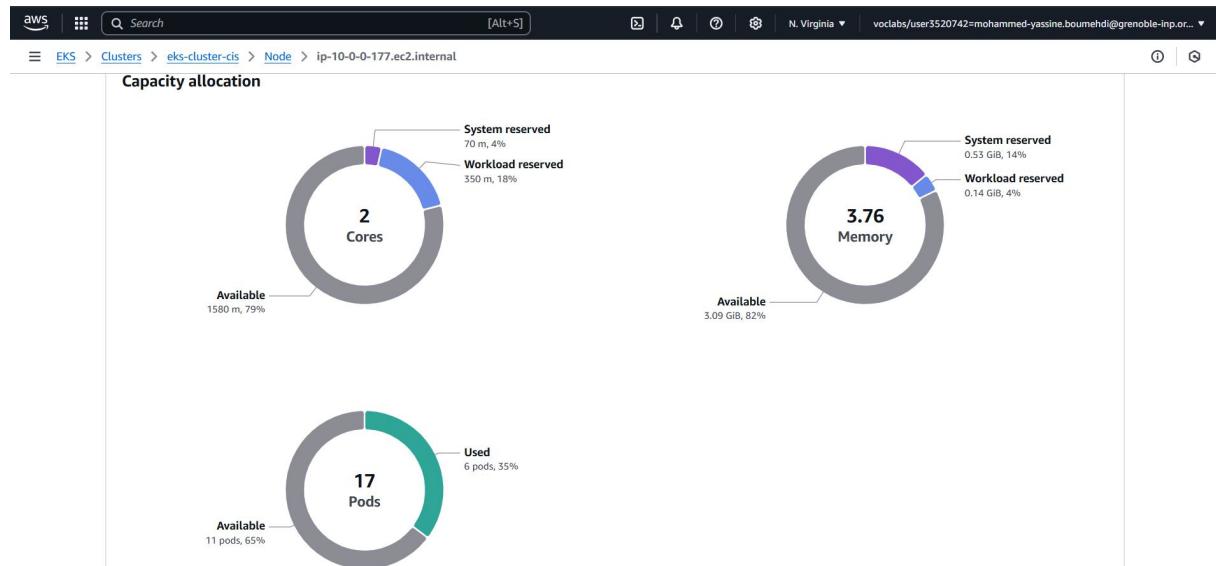


Figure 3.16: Surveillance des ressources CPU et Mémoire pour le nœud dans le cluster EKS.

- **Pods :** Les nœuds exécutent plusieurs **pods**. La Figure 3.17 montre une vue des pods en fonctionnement dans notre cluster, incluant des

pods associés au service ”juice-shop”, un projet test d’application web vulnérable. Ces pods sont déployés sur différents nœuds pour garantir une haute disponibilité.

**Déploiement dans les sous-réseaux privés :** Ces pods ont été déployés spécifiquement dans des sous-réseaux privés, conformément aux meilleures pratiques de sécurité. Les sous-réseaux privés permettent d’isoler les ressources sensibles du trafic public et de limiter les expositions potentielles. Ainsi, les pods fonctionnent dans un environnement sécurisé tout en assurant l’interaction via l’Application Load Balancer (ALB), qui agit comme point d’accès principal pour le trafic externe.

The screenshot shows the AWS EKS console interface. On the left, there's a navigation sidebar with options like 'Clusters' (selected), 'Amazon EKS Anywhere', and 'Related services'. The main area is titled 'Workloads: Pods (10)' and lists ten pods. Each pod entry includes the name, creation time, and age. The pods listed are: aws-load-balancer-controller-86bb8bf8cc-qsfkw, aws-node-qwcmc, aws-node-tfkfl, coredns-5b676c58cf-65648, coredns-5b676c58cf-7h6dv, juice-shop-68c79b9bdb-d6jsr, and juice-shop-68c79b9bdb-sdwgf. All pods were created 25 minutes ago.

Name	Age
aws-load-balancer-controller-86bb8bf8cc-qsfkw	Created 25 minutes ago
aws-node-qwcmc	Created 28 minutes ago
aws-node-tfkfl	Created 28 minutes ago
coredns-5b676c58cf-65648	Created 30 minutes ago
coredns-5b676c58cf-7h6dv	Created 30 minutes ago
juice-shop-68c79b9bdb-d6jsr	Created 25 minutes ago
juice-shop-68c79b9bdb-sdwgf	Created 25 minutes ago

Figure 3.17: Liste des pods en fonctionnement dans le cluster EKS.

- **Configuration des ReplicaSets :** Afin d’assurer la haute disponibilité et la gestion des répliques de l’application déployée, nous avons configuré des **ReplicaSets**. La Figure 3.18 et la Figure 3.19 montrent deux répliques actives pour le pod ”juice-shop”, démontrant l’équilibrage de la charge entre les nœuds ip-10-0-0-177.ec2.internal et ip-10-0-2-71.ec2.internal.

**Répartition dans les sous-réseaux privés :** Les pods gérés par les ReplicaSets sont déployés sur différents nœuds hébergés dans des sous-réseaux privés. Cette approche garantit que les pods sont protégés contre les accès directs depuis l'extérieur, tout en permettant une communication interne fluide et sécurisée grâce aux règles de routage configurées dans notre architecture réseau.

juice-shop-68c79b9bdb-d6jsr		Structured view	Raw view	C
<b>Details</b>				
Status <span style="color: green;">Running</span>	Created <span style="color: green;">10 minutes ago</span>	Namespace <span style="color: green;">juice-shop-namespace</span>		
Controlled by ReplicaSet/juice-shop-68c79b9bdb	Last transition time 10 minutes ago	Node <span style="color: green;">ip-10-0-0-177.ec2.internal</span>		
Pod IP 10.0.0.78				

Figure 3.18: Pod ”juice-shop” sur le nœud ip-10-0-0-177.ec2.internal.

juice-shop-68c79b9bdb-sdwgf		Structured view	Raw view	C
<b>Details</b>				
Status <span style="color: green;">Running</span>	Created <span style="color: green;">11 minutes ago</span>	Namespace <span style="color: green;">juice-shop-namespace</span>		
Controlled by ReplicaSet/juice-shop-68c79b9bdb	Last transition time 11 minutes ago	Node <span style="color: green;">ip-10-0-2-71.ec2.internal</span>		
Pod IP 10.0.2.219				

Figure 3.19: Pod ”juice-shop” sur le nœud ip-10-0-2-71.ec2.internal.

## 3.5 Surveillance et accès sécurisé

### 3.5.1 CloudWatch - Surveillance de l’ALB

Pour assurer une surveillance continue de notre Application Load Balancer (ALB), nous utilisons **Amazon CloudWatch**. Ce service nous permet de collecter des métriques clés, telles que le taux de requêtes, la latence et les erreurs HTTP, afin de surveiller les performances de l’ALB et de détecter

rapidement toute anomalie. Grâce à CloudWatch, nous avons également configuré des alertes pour être informés en temps réel de tout problème affectant la disponibilité ou les performances de notre application.

Les métriques principales surveillées incluent :

- **Requêtes HTTP (2XX, 4XX, 5XX)** : Suivi des réponses réussies et des erreurs côté client ou serveur.
- **Latence** : Temps moyen de réponse des requêtes, un indicateur clé de performance.
- **Utilisation des ressources (LCUs)** : Surveillance des unités de capacité utilisées pour garantir un dimensionnement adéquat.
- **Nombre total de requêtes** : Analyse du trafic global passant par l'ALB.

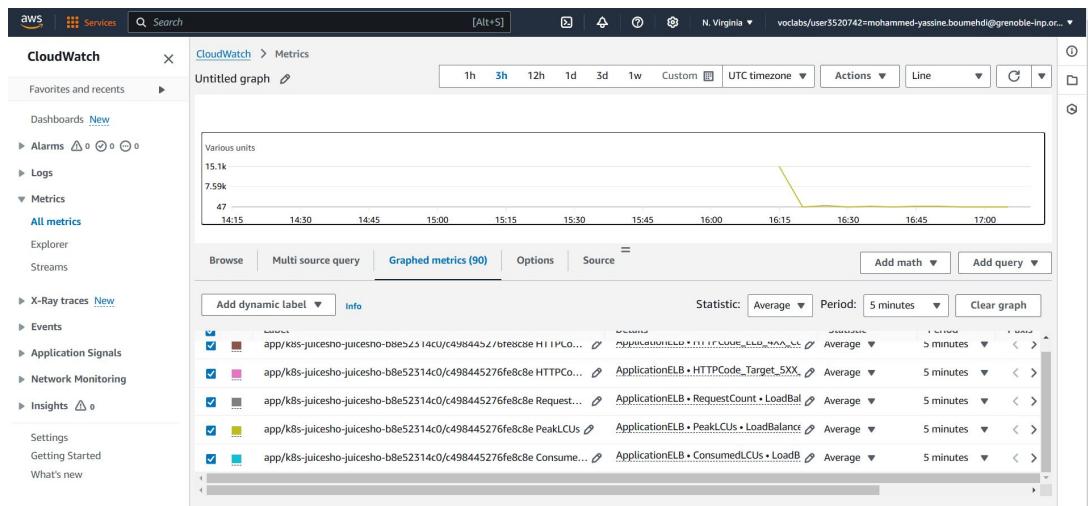


Figure 3.20: Vue des métriques CloudWatch pour l'Application Load Balancer, montrant notamment les erreurs HTTP, le nombre de requêtes, et les ressources consommées.

Le graphique ci-dessus illustre un aperçu des métriques collectées, avec un exemple d'évolution des erreurs HTTP et de la consommation des LCUs sur une période donnée. En cas de dépassement des seuils définis (par exemple, un taux d'erreurs 5XX supérieur à 5% pendant 10 minutes), des

alertes sont déclenchées pour permettre une intervention rapide. Cette configuration garantit une haute disponibilité et optimise les performances de l’application.

### 3.5.2 Le Jump Host (Bastion)

Il centralise l'accès SSH aux nœuds de travail, permettant aux administrateurs autorisés de réaliser des opérations de maintenance et de déploiement sans exposer directement les nœuds. L'accès au bastion est limité par des règles d'IP, assurant que seuls des appareils de confiance peuvent initier des connexions SSH.

## 3.6 Tests de sécurité effectués sur le WAF

Afin de vérifier l'efficacité du *Web Application Firewall (WAF)*, plusieurs tests ont été réalisés pour simuler des scénarios d'attaque ou des restrictions de connexion.

### 3.6.1 Blocage des connexions hors de France

La première expérience consistait à tester la règle de restriction géographique du WAF, qui permet uniquement aux utilisateurs situés en France d'accéder à l'application. Pour cela, nous avons tenté d'accéder à l'application depuis un VPN localisé aux Pays-Bas.



Figure 3.21: Connexion via un VPN basé aux Pays-Bas.

Ainsi, quand on veut accéder à notre site en utilisant le Vpn, on aura le résultat suivant :

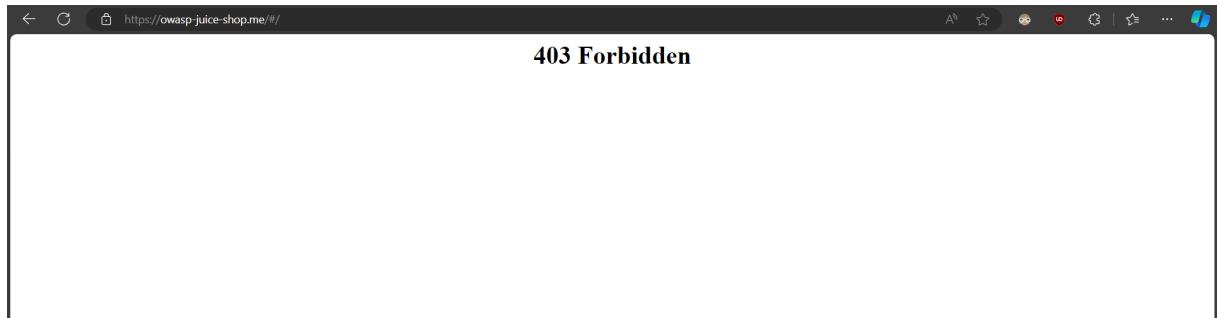


Figure 3.22: accès au site via un VPN basé aux Pays-Bas.

Comme attendu, la requête a été bloquée par la règle `BlockNonFrenchIPs` configurée dans le WAF. Les graphiques ci-dessous montrent les statistiques des requêtes autorisées et bloquées sur une période donnée.

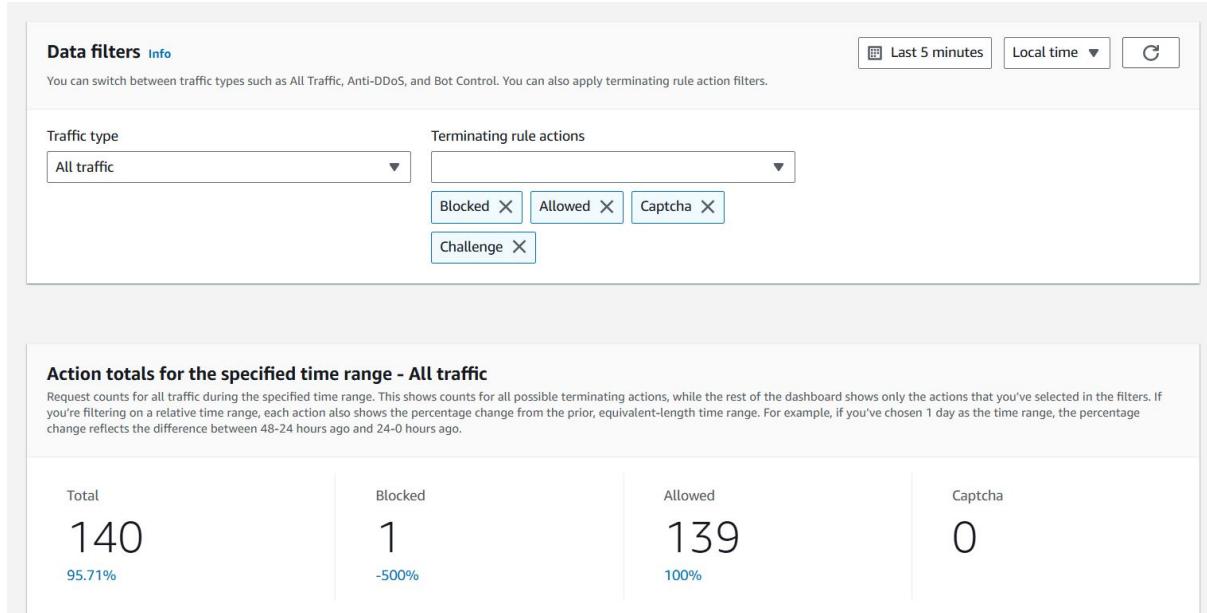


Figure 3.23: Statistiques des requêtes bloquées par la règle géographique.

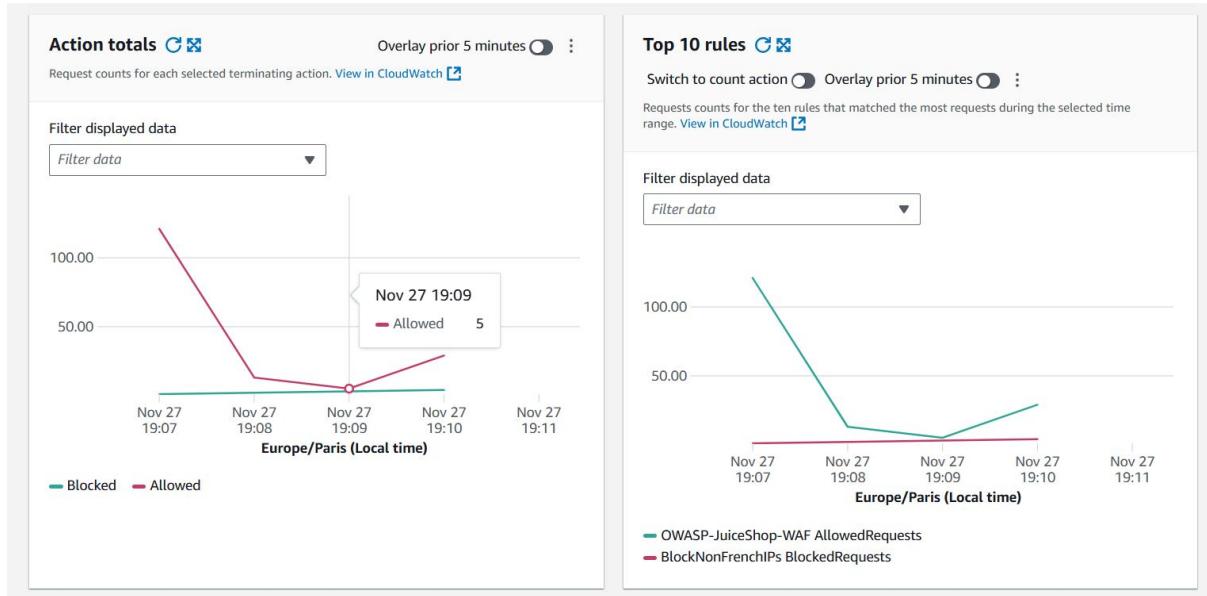


Figure 3.24: Évolution des requêtes bloquées et autorisées pour la règle géographique.

Ces résultats confirment que le WAF est capable de filtrer efficacement les connexions non autorisées en fonction de leur origine géographique.

### 3.6.2 Protection contre les injections SQL

Pour tester la capacité du WAF à protéger contre les attaques par injection SQL, une requête malveillante a été envoyée à l'application. L'objectif était de vérifier si le WAF détecte et bloque automatiquement ce type d'attaque.

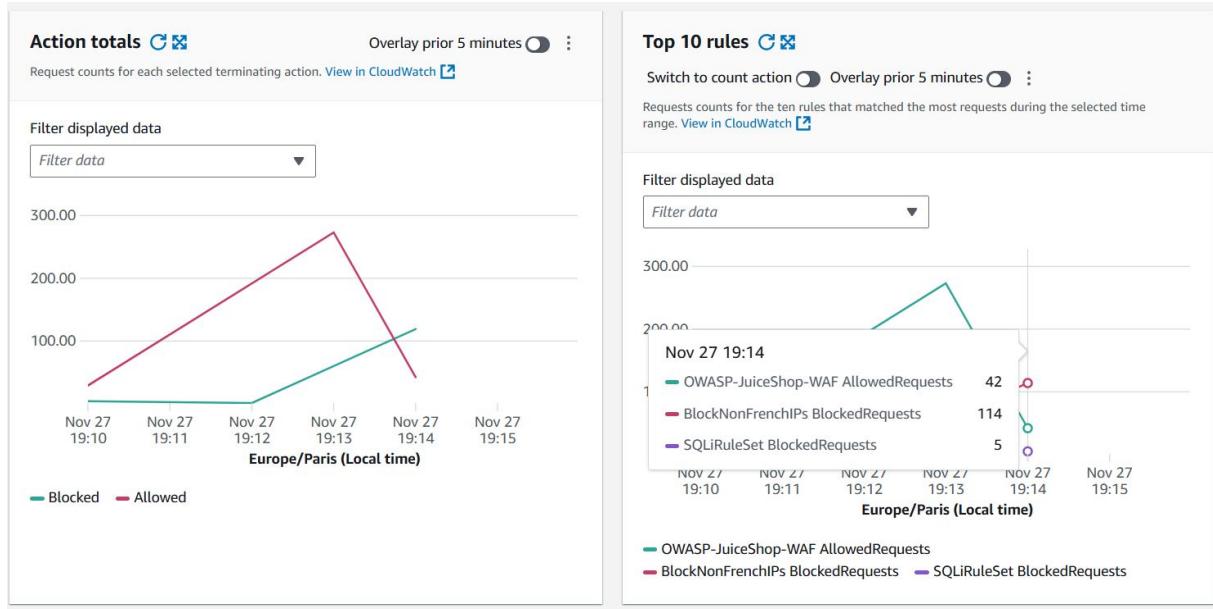


Figure 3.25: Blocage d'une tentative d'injection SQL par le WAF.

Le graphique ci-dessus met en évidence la détection et le blocage de la requête malveillante. La règle correspondante, configurée pour bloquer les requêtes contenant des patterns liés aux injections SQL, a bien fonctionné, empêchant ainsi tout accès non autorisé à la base de données.

### 3.6.3 Conclusion des tests

Les différents tests montrent que le WAF est opérationnel et efficace pour bloquer :

- Les connexions provenant de régions non autorisées.
- Les requêtes contenant des tentatives d'injection SQL.

Ces résultats confirment que le WAF contribue significativement à la sécurité de l'application et protège contre des vecteurs d'attaques courants.

### 3.7 Automatisation du déploiement : Pipeline CI/CD avec GitLab

Dans cette section, nous allons présenter le pipeline CI/CD que nous avons conçu pour automatiser le déploiement de notre application sur AWS. Ce pipeline est structuré en trois étapes principales : **provision**, **deploy**, et **secure**. Il permet de provisionner l'infrastructure, déployer l'application, et sécuriser l'environnement, le tout de manière entièrement automatisée. Cela permet d'éviter le besoin de lancer manuellement chaque commande nécessaire, ce qui garantit la cohérence et la sécurité du processus de déploiement.

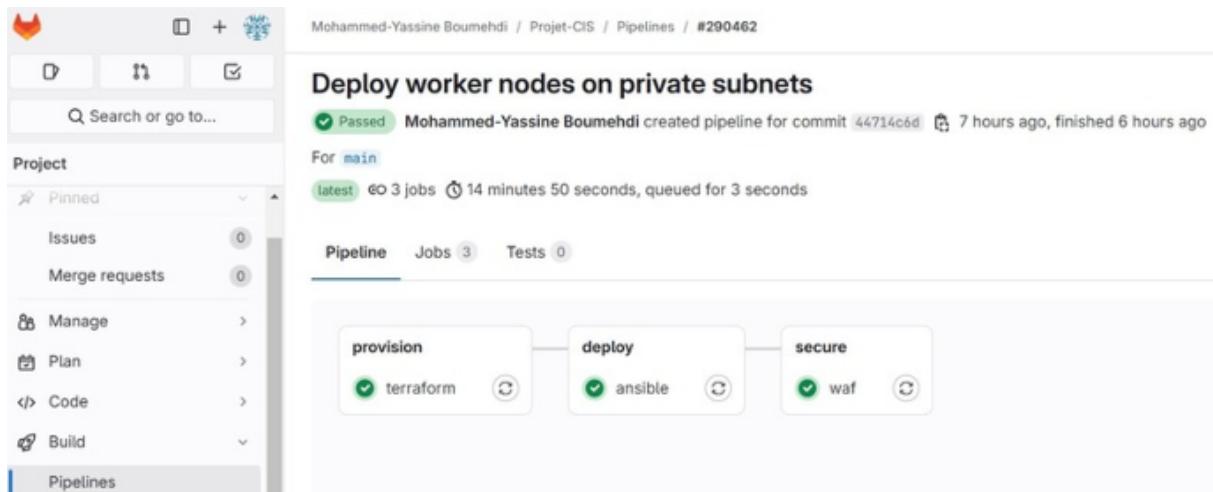


Figure 3.26: Pipeline GitLab CI/CD : Succès des différentes étapes du déploiement.

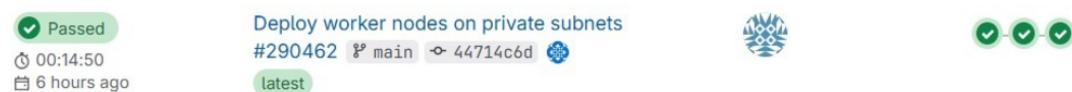


Figure 3.27: Détails du pipeline réussi sur GitLab.

### 3.7.1 Structure du Pipeline CI/CD

Le pipeline est divisé en trois *stages* :

#### Stage 1 : Provision

- **Description** : Cette étape prépare l'infrastructure nécessaire au déploiement à l'aide de **Terraform**.
- **Image utilisée** : `hashicorp/terraform:light` - Terraform est un outil d'infrastructure en tant que code qui nous permet de gérer notre infrastructure de manière déclarative.
- **Commandes spécifiques** :
  - `terraform init` : Initialise le répertoire Terraform en téléchargeant les plugins nécessaires.
  - `terraform apply -auto-approve` : Applique automatiquement les configurations définies, créant ainsi les ressources AWS nécessaires, comme le cluster EKS et le réseau.
- **Variables passées** : `aws_access_key`, `aws_secret_key`, et `aws_session_token` pour authentifier Terraform auprès d'AWS.
- **Résultat attendu** : Les ressources AWS, telles que le cluster EKS et le réseau associé, sont provisionnées automatiquement.

```

PROJECT-CIS
├ ansible-env
└ deploying-ansible
  └ deploy.yml
    └ provisioning-terraform
      └ terraform
        └ .terraform.lock.hcl
      └ main.tf
      └ variables.tf
      └ security-terraform
        └ security.tf
        └ variables.tf
      └ .gitignore
      └ .gitlab-ci.yml M
      └ README.md
      └ requirements.txt
    └ OUTLINE

main.tf
variables.tf
provisioning-terraform
deploy.yml
security.tf
variables.tf
.gitignore
.gitlab-ci.yml M
README.md
requirements.txt

.gitlab-ci.yml
stages:
- provision
- deploy
- secure

terraform:
stage: provision
image:
  name: hashicorp/terraform:light
  entrypoint: []
script:
  - cd provisioning-terraform
  - terraform init
  - terraform apply --auto-approve -var "aws_access_key=$AWS_ACCESS_KEY_ID" -var "aws_secret_key=$AWS_SECRET_ACCESS_KEY" -var "aws_session_token=$AWS_SESSION_TOKEN"
  - curl https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
  - apk add --no-cache curl bash openssh
  - curl -L "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
  - install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
  - curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
  - python3 -m venv myenv
  - source myenv/bin/activate
  - pip install -r requirements.txt
  - aws eks update-kubeconfig --name eks-cluster-cis --region us-east-1
  - helm repo add eks https://aws.github.io/eks-charts
  - helm repo update
  - export CERT_ARN=$(aws acm list-certificates --region us-east-1 --query "CertificateSummaryList[?contains(DomainName, 'owasp-juice-shop')]")
  - export VPC_ID=$(aws eks describe-cluster --name eks-cluster-cis --region us-east-1 --query "cluster.resourcesVpcConfig.vpcId" --output text)
  - helm install aws-load-balancer-controller eks/aws-load-balancer-controller --set clusterName=eks-cluster-cis --set region=us-east-1
  - export AWS_DEFAULT_REGION=us-east-1
script:

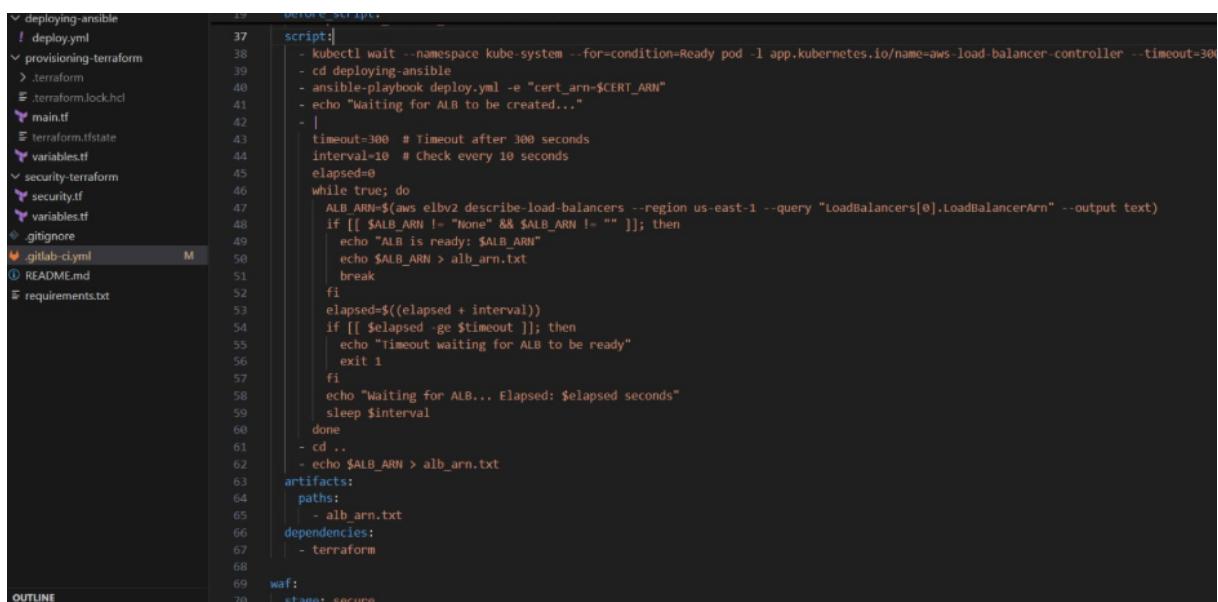
```

Figure 3.28: Configuration du stage provision dans GitLab CI/CD.

## Stage 2 : Deploy

- **Description :** Utilisation d'**Ansible** pour déployer les ressources nécessaires au fonctionnement de l'application.
- **Image utilisée :** `python:3.12-alpine` - Inclut Python pour exécuter Ansible, ainsi que des outils comme `kubectl` et `helm` pour interagir avec Kubernetes.
- **Commandes importantes :**
  - **Préparation de l'environnement :** Téléchargement et configuration de `kubectl` et `helm`, et installation des dépendances Python via un environnement virtuel.
  - **Configuration de Kubernetes :** Mise à jour du fichier `kubeconfig`, déploiement du contrôleur ALB (Application Load Balancer) via Helm.

- **Déploiement avec Ansible** : Utilisation de la commande `ansible-playbook deploy.yml`.
- **Création du Load Balancer** : Une boucle scriptée vérifie la création de l’ALB et stocke l’ARN dans un fichier.
- **Résultat attendu** : L’application est déployée sur Kubernetes, et un ALB est configuré pour rediriger le trafic HTTPS vers les pods de l’application.



```

deploying-ansible
├── deploy.yml
└── provisioning-terraform
    ├── .terraform
    ├── .terraform.lock.hcl
    ├── main.tf
    ├── terraform.tfstate
    └── variables.tf
└── security-terraform
    ├── security.tf
    └── variables.tf
└── .gitignore
└── README.md
└── requirements.txt

script:
  - kubectl wait --namespace kube-system --for=condition=Ready pod -l app.kubernetes.io/name=aws-load-balancer-controller --timeout=300
  - cd deploying-ansible
  - ansible-playbook deploy.yml -e "cert_arn=$CERT_ARN"
  - echo "Waiting for ALB to be created..."
  - |
    timeout=300 # Timeout after 300 seconds
    interval=10 # Check every 10 seconds
    elapsed=0
    while true; do
      ALB_ARN=$(aws elbv2 describe-load-balancers --region us-east-1 --query "LoadBalancers[0].LoadBalancerArn" --output text)
      if [[ $ALB_ARN != "None" && $ALB_ARN != "" ]]; then
        echo "ALB is ready: $ALB_ARN"
        echo $ALB_ARN > alb_arn.txt
        break
      fi
      elapsed=$((elapsed + interval))
      if [[ $elapsed -ge $timeout ]]; then
        echo "timeout waiting for ALB to be ready"
        exit 1
      fi
      echo "Waiting for ALB... Elapsed: $elapsed seconds"
      sleep $interval
    done
  - cd ..
  - echo $ALB_ARN > alb_arn.txt
artifacts:
  paths:
    - alb_arn.txt
dependencies:
  - terraform
waf:
  stage: secure

```

Figure 3.29: Configuration du stage `deploy` dans GitLab CI/CD.

### Stage 3 : Secure

- **Description** : Application des mesures de sécurité via **Terraform** pour protéger l’application grâce à un **Web Application Firewall (WAF)**.
- **Image utilisée** : `hashicorp/terraform:light` - Utilisé de nouveau pour automatiser la configuration du WAF.
- **Commandes importantes** :

- `terraform apply -auto-approve` : Configure les règles du WAF pour sécuriser l’application en utilisant l’ARN de l’ALB comme variable.
- **Résultat attendu** : Le WAF est configuré pour protéger l’application contre les menaces courantes et limiter l’accès aux utilisateurs autorisés.

```

stages:
  - waf

waf:
  stage: secure
  image:
    name: hashicorp/terraform:light
  entrypoint: []
  script:
    - export AWS_ACCESS_KEY_ID=$AWS_ACCESS_KEY_ID
    - export AWS_SECRET_ACCESS_KEY=$AWS_SECRET_ACCESS_KEY
    - export AWS_SESSION_TOKEN=$AWS_SESSION_TOKEN
    - export ALB_ARN=$(cat alb_arn.txt)
    - cd security-terraform
    - terraform init
    - terraform apply -auto-approve -var "aws_access_key=$AWS_ACCESS_KEY_ID" -var "aws_secret_key=$AWS_SECRET_ACCESS_KEY" -var "aws_session_token=$AWS_SESSION_TOKEN"
  dependencies:
    - ansible

```

Figure 3.30: Configuration du stage `secure` dans GitLab CI/CD.

### 3.7.2 Variables d’Environnement Essentielles

L’exécution correcte de ce pipeline a nécessité la configuration préalable de variables d’environnement essentielles, telles que :

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `AWS_SESSION_TOKEN`
- `CERT_ARN` pour la gestion SSL.
- `VPC_ID` pour la gestion du réseau et du load balancer.

Ces variables ont été renseignées pour permettre l’interaction entre les différents outils (Terraform, Ansible, Helm, Kubernetes, etc.) et les ressources AWS.

Key	Value	Environments	Actions
AWS_ACCESS_KEY_ID	*****	All (default)	
AWS_SECRET_ACCESS_KEY	*****	All (default)	
AWS_SESSION_TOKEN	*****	All (default)	

Figure 3.31: Configuration des variables d'environnement dans GitLab CI/CD.

### 3.7.3 Avantages et Résultats du Pipeline CI/CD

- **Automatisation complète :** Le processus de provision de l'infrastructure, déploiement et sécurisation est entièrement automatisé. Il suffit de lancer le pipeline pour que l'application soit opérationnelle et sécurisée.
- **Cohérence et réduction des erreurs :** L'utilisation d'outils comme Terraform et Ansible garantit que les étapes sont répétables, éliminant les erreurs humaines.
- **Sécurité renforcée :** L'ALB et le WAF offrent une protection robuste, tandis que le HTTPS assure des connexions sécurisées.
- **Économie de temps :** Plus besoin de lancer manuellement les commandes telles que `terraform init` ou `kubectl`. Le pipeline s'en charge automatiquement.

### 3.7.4 Conclusion

Tout le pipeline a été exécuté avec succès du début à la fin. Nous avons atteint notre objectif de mettre en place une infrastructure automatisée et sécurisée, avec une application déployée de manière fiable et un environnement de production protégé.

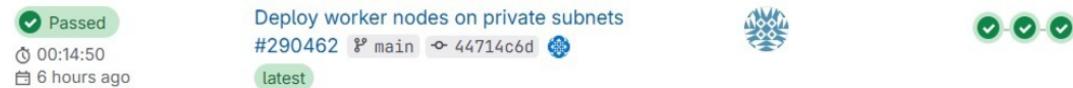


Figure 3.32: Pipeline exécuté avec succès.

## 3.8 Détection des flags via AWS WAF

Dans le cadre du challenge proposé par notre professeur, nous avons configuré **AWS WAF** (**Web Application Firewall**) pour détecter les *flags* cachés envoyés via des requêtes HTTP à notre instance **OWASP Juice Shop**. Ces flags ont une structure standard de la forme `X-Flag-{HEXADECIMAL}`. Voici les étapes que nous avons suivies pour identifier ces flags.

### 3.8.1 Configuration des règles dans AWS WAF

Deux nouvelles règles spécifiques ont été ajoutées pour capturer les flags dans les requêtes HTTP :

- **detect\_flag\_body** : Cette règle inspecte le *corps des requêtes* pour détecter toute occurrence du pattern `X-Flag-`. Une contrainte de position *Contains String* a été utilisée pour cette règle.
- **detect\_flags** : Cette règle permet d'inspecter d'autres parties des requêtes, comme les en-têtes ou les paramètres de requêtes, pour élargir la portée de la détection.

Les actions associées à ces règles sont configurées pour **compter** (*Count*) les occurrences, ce qui permet d'enregistrer et d'examiner les requêtes interceptées sans bloquer le trafic.

Rule		
Rule name detect_flag_body	Type Regular rule	Region US East (N. Virginia)
<b>If a request matches the statement</b>		
Statement 1		
Field to match Body  Positional constraint Contains string  Search string X-Flag-  Oversize handling Continue - Inspect the contents that are within the size limitations according to the rule inspection criteria  Text transformations <ul style="list-style-type: none"> <li>• None (Priority 0)</li> </ul>		

Figure 3.33: Détails de la règle `detect_flag_body` dans AWS WAF.

Then	
Action	The action to take when a web request matches the rule statement.
Action Count  Custom request - Add labels -	

Figure 3.34: Action associée à la règle `detect_flag_body` (Count).

### 3.8.2 Logs et détection dans CloudWatch

Nous avons configuré **Amazon CloudWatch** pour capturer les logs générés par les règles WAF. Cela nous a permis d'examiner en détail les requêtes contenant les flags et de confirmer leur provenance.

- Un groupe de logs dédié (`aws-waf-logs-1`) a été créé pour capturer les événements.
- À l'aide de **Logs Insights**, des requêtes ont été exécutées pour filtrer les logs et identifier les requêtes contenant les flags.

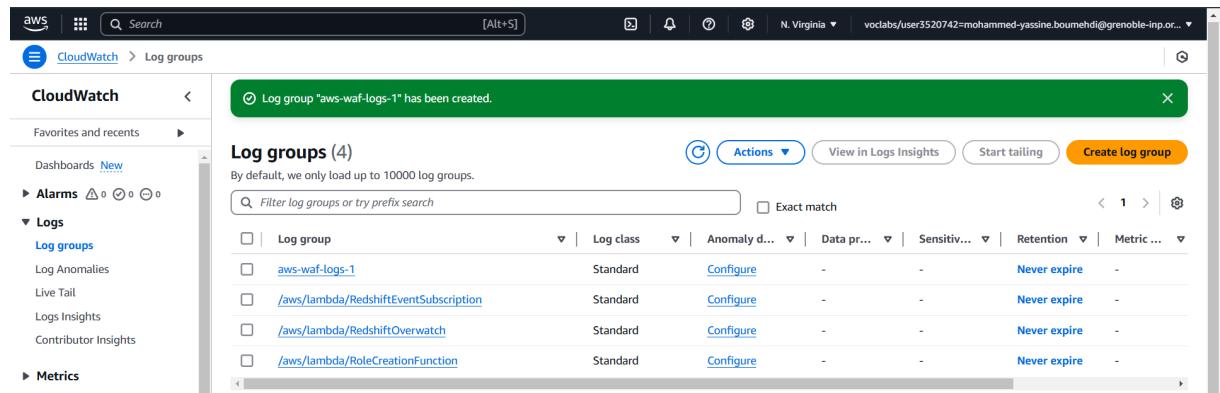


Figure 3.35: Groupe de logs configuré dans CloudWatch pour capturer les événements WAF.

### 3.8.3 Requêtes interceptées

Les règles configurées ont permis de voir directement qu'une requête (parmi les deux envoyées) contient un flag. Les *Sampled Requests* dans AWS WAF montrent des détails comme :

- L'IP source (37.67.122.104 dans ce cas).
- Les en-têtes et la méthode HTTP.
- La date et l'heure des requêtes correspondantes.

Source IP 37.67.122.104	Rule inside rule group -	Action COUNT	Time Fri Nov 29 2024 13:22:02 GMT+0100 (Central European Standard Time)
Country FR	URI /		
Request			
<pre>POST / Host: owasp-juice-shop.me User-Agent: python-requests/2.31.0 Accept-Encoding: gzip, deflate Accept: */ Connection: keep-alive Content-Length: 28 Content-Type: application/x-www-form-urlencoded</pre>			

Figure 3.36: Démonstration de la détection par la règle `detect_flag_body`.

Voici les observations principales concernant ces requêtes :

- Une requête contenant un flag a bien été détectée par la règle `detect_flag_body`. Cependant, le contenu du **Body** était encrypté (HTTPS), ce qui nous a empêchés de l'identifier. Par ailleurs, nous n'avons pas utilisé d'outil externe pour intercepter ou décrypter cette requête.
- Une autre requête, provenant de l'IP de notre professeur (37.67.122.104), a également été interceptée. Toutefois, nos règles configurées dans AWS WAF n'ont pas pu détecter le flag associé en raison de la limite de 5 règles personnalisées imposée par AWS WAF.

Sampled requests (51)					
<input type="text" value="detect_flag_body"/> ▾					
Samples of requests from the past 3 hours.					
<input type="text" value="Find sampled requests"/> <span style="float: right;">◀ 1 2 3 4 5 6 ▶ ⌂</span>					
Metric name	Source IP	URI	Rule inside rule group	Action	Time
<code>detect_flag_body</code>	37.67.122.104 (FR)	/	-	COUNT	Fri Nov 29 2024 13:22:02 GMT+0100 (Central European Standard Time)

Figure 3.37: Tableau des requêtes échantillons détectées dans AWS WAF.

#	httpRequest...	@timestamp	@message
▼ 1	37.67.122.104 2024-11-29T12:22:02.866Z	{ "timestamp": "1732882922866", "formatVersion": 1, "webaclId": "arn:aws:wafv2:us-east-1:779669255065:regional/webacl/OWASP-JuiceShop-WAF/ec5e5085-f0c5-4e3e-8b41-1a7b7b55bafc" }	<p>Field Value</p> <p>@entity.Attributes.AWS.Resource.ARN arn:aws:wafv2:us-east-1:779669255065:regional/webacl/OWASP-JuiceShop-WAF/ec5e5085-f0c5-4e3e-8b41-1a7b7b55bafc</p> <p>@entity.KeyAttributes.Identifier OWASP-JuiceShop-WAF ec5e5085-f0c5-4e3e-8b41-1a7b7b55bafc regional</p> <p>@entity.KeyAttributes.ResourceType AWS::WAFV2::WebACL</p> <p>@entity.KeyAttributes.Type AWS::Resource</p> <p>@ingestionTime 1732882936065</p> <p>@log 779669255065:aws-waf-logs-1</p> <p>@logStream us-east-1_OWASP-JuiceShop-WAF_0</p> <p>@message { "timestamp": "1732882922866", "formatVersion": 1, "webaclId": "arn:aws:wafv2:us-east-1:779669255065:regional/webacl/OWASP-JuiceShop-WAF/ec5e5085-f0c5-4e3e-8b41-1a7b7b55bafc", "terminatingRuleId": "Default_Action", "terminatingRuleType": "REGULAR", "action": "ALLOW", "terminatingRuleMatchDetails": [], "httpSourceName": "ALB", "httpSourceId": "779669255065-app/k8s-juiceshop-b8e52314c0/5a25083bd735caca", "ruleGroupList": [{"ruleGroupId": "AWS#AWSManagedRulesAnonymousIpList", "terminatingRule": null, "nonTerminatingMatchingRules": []}, {"excludedRules": null, "customerConfig": null}, {"ruleGroupId": "AWS#AWSManagedRulesCommonRuleSet", "terminatingRule": null, "nonTerminatingMatchingRules": []}, {"excludedRules": null, "customerConfig": null}, {"ruleGroupId": "AWS#AWSManagedRulesKnownBadInputsRuleSet", "terminatingRule": null, "nonTerminatingMatchingRules": []}, {"excludedRules": null, "customerConfig": null}, {"ruleGroupId": "AWS#AWSManagedRulesSQLRuleSet", "terminatingRule": null, "nonTerminatingMatchingRules": []}, {"excludedRules": null, "customerConfig": null}, {"ruleId": "detect_flag_body", "action": "COUNT", "ruleMatchDetails": [{"ruleId": "detect_flags", "action": "COUNT", "ruleMatchDetails": []}], "rateBasedRuleList": [], "nonTerminatingMatchingRules": [{"ruleId": "detect_flag_body", "action": "COUNT", "ruleMatchDetails": [{"ruleId": "detect_flags", "action": "COUNT", "ruleMatchDetails": []}]}]}, {"requestHeadersInserted": null, "responseCodeSent": null, "httpRequest": {"clientIp": "37.67.122.104", "country": "FR", "headers": [{"name": "Host", "value": "owasp-juice-shop.me"}, {"name": "User-Agent", "value": "python-requests/2.31.0"}, {"name": "Accept-Encoding", "value": "gzip, deflate"}, {"name": "Accept", "value": "*/*"}, {"name": "Connection", "value": "keep-alive"}, {"name": "Content-Length", "value": "28"}, {"name": "Content-Type", "value": "application/x-www-form-urlencoded"}, {"name": "uri", "value": "/"}, {"name": "args", "value": ""}], "httpVersion": "HTTP/1.1", "httpMethod": "POST", "requestId": "1-6749b1e26ecc53754e26a733f0a10"}, "requestBodySize": 28, "requestBodySizeInspectedByWAF": 28, "ja3Fingerprint": "a48c0d5f95b1ef98f560f324fd275dal"}]</p> <p>@timestamp 1732882922866</p>

Figure 3.38: La requête dans laquelle on a détecté l'existence d'un flag dans le Body.

#	httpRequest...	@timestamp	@message
▼ 2	37.67.122.104 2024-11-29T12:22:02.454Z	{ "timestamp": "1732882922454", "formatVersion": 1, "webaclId": "arn:aws:wafv2:us-east-1:779669255065:regional/webacl/OWASP-JuiceShop-WAF/ec5e5085-f0c5-4e3e-8b41-1a7b7b55bafc" }	<p>Field Value</p> <p>@entity.Attributes.AWS.Resource.ARN arn:aws:wafv2:us-east-1:779669255065:regional/webacl/OWASP-JuiceShop-WAF/ec5e5085-f0c5-4e3e-8b41-1a7b7b55bafc</p> <p>@entity.KeyAttributes.Identifier OWASP-JuiceShop-WAF ec5e5085-f0c5-4e3e-8b41-1a7b7b55bafc regional</p> <p>@entity.KeyAttributes.ResourceType AWS::WAFV2::WebACL</p> <p>@entity.KeyAttributes.Type AWS::Resource</p> <p>@ingestionTime 1732882935892</p> <p>@log 779669255065:aws-waf-logs-1</p> <p>@logStream us-east-1_OWASP-JuiceShop-WAF_0</p> <p>@message { "timestamp": "1732882922454", "formatVersion": 1, "webaclId": "arn:aws:wafv2:us-east-1:779669255065:regional/webacl/OWASP-JuiceShop-WAF/ec5e5085-f0c5-4e3e-8b41-1a7b7b55bafc", "terminatingRuleId": "Default_Action", "terminatingRuleType": "REGULAR", "action": "ALLOW", "terminatingRuleMatchDetails": [], "httpSourceName": "ALB", "httpSourceId": "779669255065-app/k8s-juiceshop-b8e52314c0/5a25083bd735caca", "ruleGroupList": [{"ruleGroupId": "AWS#AWSManagedRulesAnonymousIpList", "terminatingRule": null, "nonTerminatingMatchingRules": []}, {"excludedRules": null, "customerConfig": null}, {"ruleGroupId": "AWS#AWSManagedRulesCommonRuleSet", "terminatingRule": null, "nonTerminatingMatchingRules": []}, {"excludedRules": null, "customerConfig": null}, {"ruleGroupId": "AWS#AWSManagedRulesKnownBadInputsRuleSet", "terminatingRule": null, "nonTerminatingMatchingRules": []}, {"excludedRules": null, "customerConfig": null}, {"ruleGroupId": "AWS#AWSManagedRulesSQLRuleSet", "terminatingRule": null, "nonTerminatingMatchingRules": []}, {"excludedRules": null, "customerConfig": null}, {"ruleId": "detect_flag_body", "action": "COUNT", "ruleMatchDetails": [{"ruleId": "detect_flags", "action": "COUNT", "ruleMatchDetails": []}]}], "rateBasedRuleList": [], "nonTerminatingMatchingRules": [{"ruleId": "detect_flag_body", "action": "COUNT", "ruleMatchDetails": [{"ruleId": "detect_flags", "action": "COUNT", "ruleMatchDetails": []}]}]}, {"requestHeadersInserted": null, "responseCodeSent": null, "httpRequest": {"clientIp": "37.67.122.104", "country": "FR", "headers": [{"name": "Host", "value": "owasp-juice-shop.me"}, {"name": "User-Agent", "value": "python-requests/2.31.0"}, {"name": "Accept-Encoding", "value": "gzip, deflate"}, {"name": "Accept", "value": "*/*"}, {"name": "Connection", "value": "keep-alive"}, {"name": "Content-Length", "value": "28"}, {"name": "Content-Type", "value": "application/x-www-form-urlencoded"}, {"name": "uri", "value": "/"}, {"name": "args", "value": ""}], "httpVersion": "HTTP/1.1", "httpMethod": "GET", "requestId": "1-6749b1e574635b92fe87da3ae55c9f"}, "ja3Fingerprint": "a48c0d5f95b1ef98f560f324fd275dal"}]</p> <p>@timestamp 1732882922454</p> <p>action ALLOW</p> <p>formatVersion 1</p>

Figure 3.39: La deuxième requête qui a été interceptée.

# Conclusion

Ce projet nous a permis de concevoir et déployer une infrastructure sécurisée et automatisée pour l'application vulnérable OWASP Juice Shop, en exploitant des outils modernes tels que Terraform, Ansible et GitLab CI/CD. Nous avons provisionné une infrastructure sur AWS, incluant un cluster Kubernetes (EKS), des sous-réseaux privés pour héberger les pods, ainsi qu'un Application Load Balancer (ALB) pour gérer le trafic entrant en HTTPS.

L'automatisation a été au cœur du projet grâce à un pipeline CI/CD structuré en trois étapes : **Provision**, **Deploy** et **Secure**. Terraform a été utilisé pour provisionner l'infrastructure et sécuriser l'application via un Web Application Firewall (WAF), tandis qu'Ansible a permis de déployer les ressources Kubernetes et d'orchestrer l'application. Ces outils ont assuré la reproductibilité et la réduction des erreurs dans le processus de déploiement.

La sécurité a été renforcée par l'intégration du WAF, protégeant l'application contre des menaces courantes comme les injections SQL et les attaques XSS. De plus, des règles personnalisées ont été créées pour détecter des requêtes spécifiques, notamment celles contenant des flags cachés envoyés dans le cadre du challenge.

En résumé, ce projet a démontré notre capacité à concevoir une infrastructure robuste et sécurisée, à intégrer des mécanismes de protection avancés et à automatiser le cycle de vie des déploiements. Ces réalisations fournissent une base solide pour le développement et la gestion d'applications web dans un environnement cloud moderne.

# Bibliographie

- <https://docs.aws.amazon.com/eks/latest/userguide/lbc-helm.html>
- <https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html>
- <https://www.terraform.io/>
- <https://www.ansible.com/>
- <https://docs.gitlab.com/ee/ci/>