



Rapport - Mise en place d'une architecture et d'une infrastructure réseau avec AWS

ÉQUIPE :

Pierre LEOCADIE

Jordan BAUMARD

Charles HURST

Groupe 209



Sommaire

- Introduction
- Schémas
- Étape 1 : Infrastructure provisioning
 - Description de l'infrastructure
 - Code source Terraform de l'infrastructure
 - Déploiement de l'infrastructure avec Terraform
 - Configuration du nom de domaine
- Étape 2 : Configuration de l'infrastructure
 - Description
 - L'utilisation de Docker
 - NGINX Reverse Proxy
 - GoAcces

- Serveur VPN WireGuard
- Interface web d'administration pour WireGuard
- Portainer
- Serveur DNS Pi-hole
- Les deux autres serveurs : `cloudlab_public_app_projects_server` et `cloudlab_internal_server_2`
 - Résumé de l'infrastructure et de sa configuration
- Code source Ansible de la configuration
- Configuration avec Ansible et Portainer
 - Tester la connectivité
 - Exécution des playbooks Ansible
 - Configuration de Portainer
 - Déploiement des autres applications avec Portainer
- Étape 3 : Modification de l'infrastructure provisioning
- Conclusion

▼ Introduction

Dans le cadre de notre projet, nous avons entrepris la conception et la mise en place d'une architecture et d'une infrastructure réseau sur AWS (Amazon Web Services). Ce rapport vise à présenter en détail notre approche et les étapes clés de déploiement de cette infrastructure, en fournissant des explications simples et claires accessibles à tous.

L'objectif principal de notre projet était de créer une infrastructure solide, sécurisée et scalable, permettant de déployer et de gérer efficacement différents services au sein d'un environnement cloud. Pour atteindre cet objectif, nous avons utilisé des technologies telles que Docker, Terraform, Ansible, ainsi que des services AWS tels que VPC (Virtual Private Cloud), EC2 (Elastic Compute Cloud), Route 53 et bien d'autres.

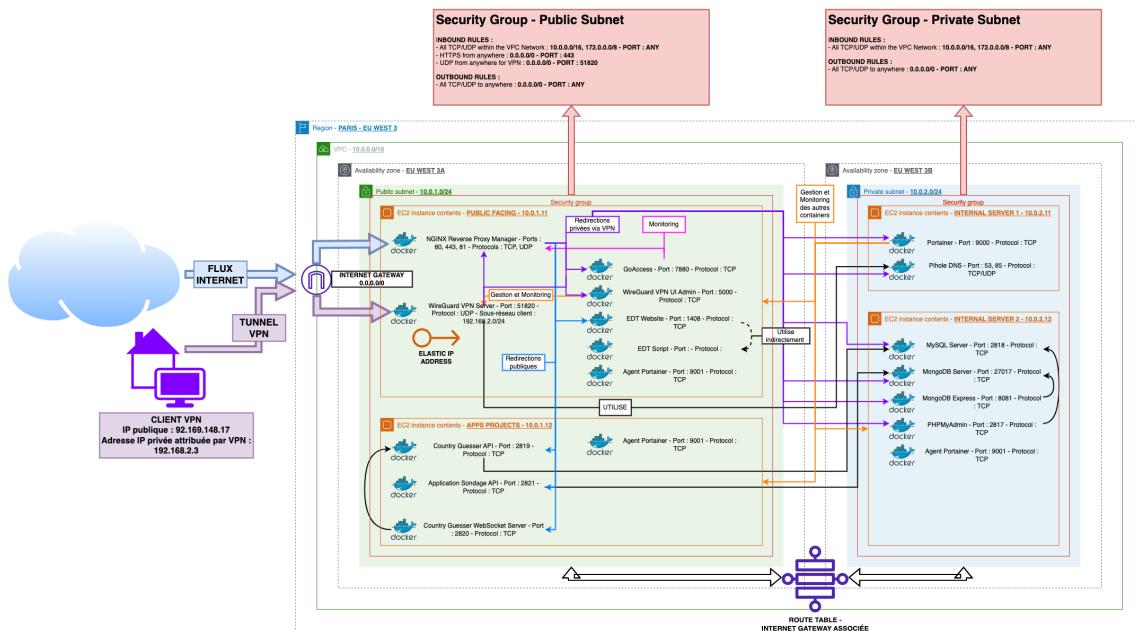
Ce rapport mettra l'accent sur les principales étapes de déploiement de l'architecture et de l'infrastructure réseau, en fournissant des explications détaillées et des instructions simples pour permettre une compréhension complète, même pour les personnes non familiarisées avec ces technologies.

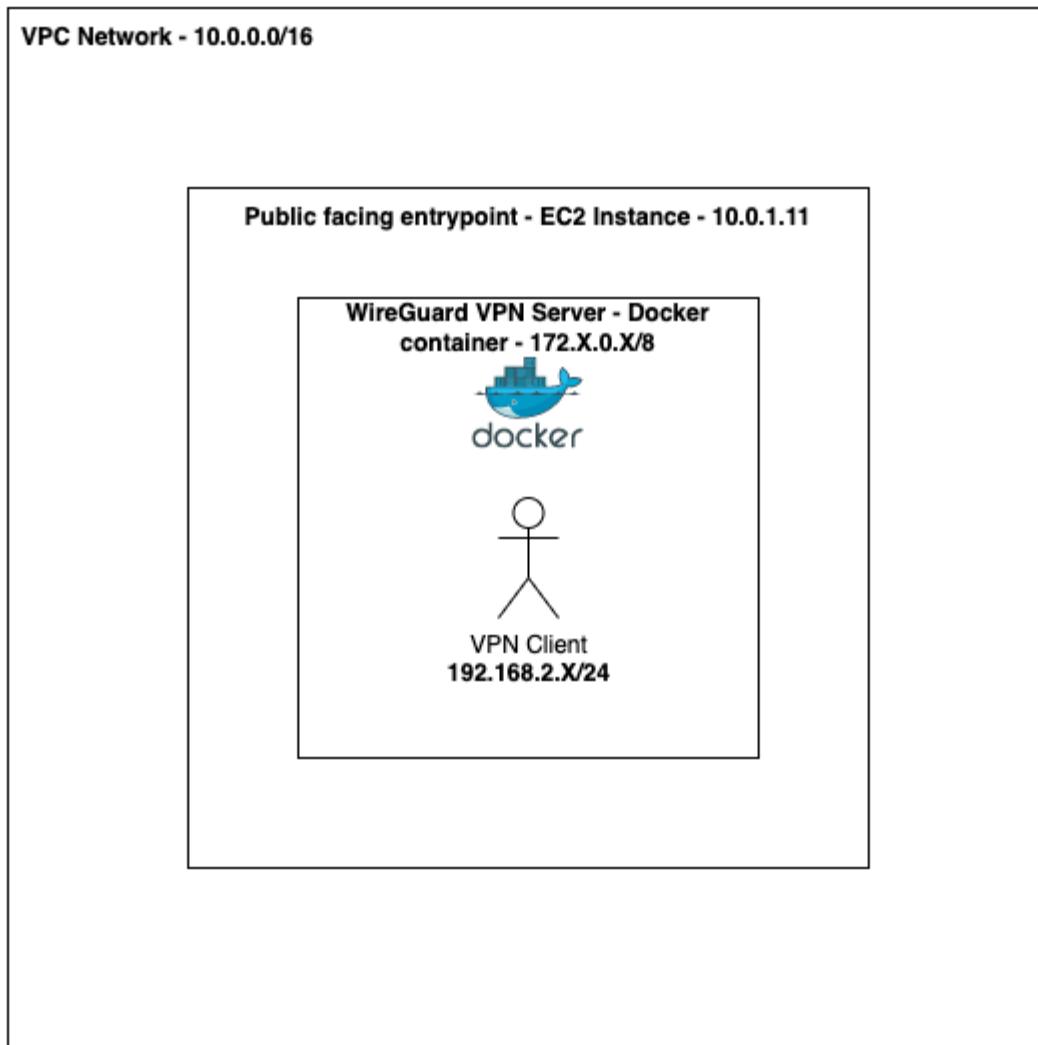
Le rapport commencera par une description générale de l'architecture, en expliquant les différents composants et leur rôle dans notre infrastructure. Nous détaillerons ensuite les étapes de déploiement, en fournissant des instructions claires sur la configuration des différents services tels que NGINX Proxy Manager, WireGuard, PiHole, et GoAccess.

Nous mettrons également l'accent sur les aspects de sécurité, en expliquant les mesures prises pour protéger notre infrastructure, y compris la configuration des groupes de sécurité, l'application d'access lists et la restriction des accès non autorisés.

Ce rapport vise à fournir une vision globale de notre projet, en présentant les étapes clés de déploiement de l'architecture et de l'infrastructure réseau AWS. Nous espérons que les informations présentées ici permettront une compréhension approfondie de notre approche et encourageront d'autres étudiants à explorer les avantages et les possibilités offerts par les services cloud et les technologies modernes de déploiement d'infrastructures.

▼ Schémas





▼ Étape 1 : Infrastructure provisioning

▼ Description de l'infrastructure

- VPC (Virtual Private Cloud) : Nous avons créé un VPC appelé `cloudlab_vpc` avec un bloc CIDR de `10.0.0.0/16`. Le VPC agit comme un réseau isolé dans le cloud d'AWS.
- Internet Gateway : Nous avons attaché un Internet Gateway nommé `cloudlab_vpc_igw` au VPC `cloudlab_vpc`. Cela permet aux ressources situées dans le VPC d'accéder à Internet.
- Table de routage : Nous avons créé une table de routage publique appelée `cloudlab_public_route_table` avec un bloc CIDR source de `0.0.0.0/0` et associée à l'Internet Gateway `cloudlab_vpc_igw`. Cette table

de routage est utilisée pour diriger le trafic Internet des ressources publiques du VPC.

- Sous-réseau 1 : Nous avons créé un sous-réseau public nommé `cloudlab_vpc_public_facing_subnet_3a1` avec un bloc CIDR de `10.0.1.0/24`. Ce sous-réseau est situé dans la zone de disponibilité `eu-west-3a` (PARIS zone A) et est destiné à héberger des instances EC2 qui sont exposées à Internet (directement dans le cas du serveur `cloudlab_public_facing_entrypoint` et indirectement dans le cas du serveur `cloudlab_public_app_projects_server`). Les adresses IP publiques sont attribuées automatiquement aux instances EC2 de ce sous-réseau. Il est associé à la table de routage publique `cloudlab_public_route_table`.
- Groupe de sécurité du sous-réseau 1 : Nous avons créé un groupe de sécurité appelé `cloudlab_public_facing_subnet_3a1_sg` associé au VPC `cloudlab_vpc`. Ce groupe de sécurité autorise les connexions TCP et UDP sur tous les ports (0-65535) à partir des blocs CIDR sources `10.0.0.0/16`, `172.0.0.0/8` et `0.0.0.0/0`. Il est utilisé pour contrôler les accès réseau aux instances EC2 du sous-réseau public.
- Instances EC2 du sous-réseau 1 : Nous avons créé deux instances EC2 dans le sous-réseau `cloudlab_vpc_public_facing_subnet_3a1`. Les instances sont nommées `cloudlab_public_facing_entrypoint` avec l'adresse IP privée `10.0.1.11` et `cloudlab_public_app_projects_server` avec l'adresse IP privée `10.0.1.12`. Les instances utilisent l'AMI Debian 11 architecture ARM64 et le type d'instance `t4g.micro`. Elles sont associées au sous-réseau `cloudlab_vpc_public_facing_subnet_3a1` et au groupe de sécurité `cloudlab_public_facing_subnet_3a1_sg`.
- Elastic IP : Nous avons associé une adresse IP élastique (Elastic IP) nommée `public_facing_entrypoint_eip` à l'instance EC2 `cloudlab_public_facing_entrypoint`. L'Elastic IP permet d'associer une adresse IP statique à l'instance EC2, garantissant ainsi que l'adresse IP ne change pas lors du redémarrage de l'instance.
- Sous-réseau 2 : Nous avons créé un sous-réseau interne nommé `cloudlab_vpc_internal_subnet_3b1` avec un bloc CIDR de `10.0.2.0/24`. Ce sous-réseau est situé dans la zone de disponibilité `eu-west-3b` (PARIS zone B) et est destiné à héberger des instances EC2 qui ne sont accessibles que depuis le réseau interne. Les adresses IP publiques sont

attribuées automatiquement aux instances EC2 de ce sous-réseau. Il est associé à la table de routage publique `cloudbl_public_route_table`.

- Groupe de sécurité du sous-réseau 2 : Nous avons créé un groupe de sécurité appelé `cloudbl_internal_subnet_3b1_sg` associé au VPC `cloudbl_vpc`. Ce groupe de sécurité autorise les connexions TCP et UDP sur tous les ports (0-65535) à partir des blocs CIDR sources `10.0.0.0/16`, `172.0.0.0/8` et `0.0.0.0/0`. Il est utilisé pour contrôler les accès réseau aux instances EC2 du sous-réseau interne.
- Instances EC2 du sous-réseau 2 : Nous avons créé deux instances EC2 dans le sous-réseau `cloudbl_vpc_internal_subnet_3b1`. Les instances sont nommées `cloudbl_internal_server_1` avec l'adresse IP privée `10.0.2.11` et `cloudbl_internal_server_2` avec l'adresse IP privée `10.0.2.12`. Les instances utilisent l'AMI Debian 11 architecture ARM64 et le type d'instance `t4g.micro`. Elles sont associées au sous-réseau `cloudbl_vpc_internal_subnet_3b1` et au groupe de sécurité `cloudbl_internal_subnet_3b1_sg`.
- DNS - Route53 : Nous avons configuré la zone publique `deletesystem32.fr` dans Route53. Les serveurs de noms associés sont `ns-116.awsdns-14.com`, `ns-1352.awsdns-41.org`, `ns-1873.awsdns-42.co.uk` et `ns-958.awsdns-55.net`. Les enregistrements de type A sont utilisés pour associer des adresses IP publiques à des noms de domaine, tels que `deletesystem32.fr`, `proxymanager.deletesystem32.fr`, `goaccess.deletesystem32.fr`, etc. L'adresse IP publique Elastic IP de l'instance `cloudbl_public_facing_entrypoint` est associée aux enregistrements de type A.

▼ Code source Terraform de l'infrastructure

Dans le cadre de notre projet, nous avons utilisé Terraform, une technologie connue sous le nom d'**Infrastructure as Code** (IaC), pour déployer et gérer notre infrastructure AWS. L'Infrastructure as Code consiste à décrire et à gérer l'infrastructure de manière programmable, en utilisant du code source plutôt que des interfaces graphiques ou des actions manuelles.



Étant donné que le code source Terraform de notre infrastructure ne comprend pas la création d'une paire de clés SSH, il est nécessaire d'ajouter vous-même le code Terraform qui le permet, ou de procéder à la création d'une paire de clés SSH via la console AWS. Voici les étapes pour créer une paire de clés SSH via la console AWS :

1. Connectez-vous à votre compte AWS et accédez à la console AWS.
2. Recherchez le service "EC2" dans la barre de navigation à gauche et cliquez dessus. EC2 est le service d'Amazon qui vous permet de créer et gérer des instances de machines virtuelles.
3. Dans la section "Réseau et sécurité" de la barre de navigation à gauche de la console EC2, recherchez l'option "Paires de clés" et cliquez dessus.
4. Sur la page "Paires de clés", cliquez sur le bouton "Créer une paire de clés".
5. Dans la fenêtre qui apparaît, donnez un nom à votre paire de clés pour faciliter son identification. Assurez-vous de sélectionner le format de fichier ".pem" pour votre clé.
6. Cliquez sur le bouton "Créer une paire de clés" pour finaliser la création de la paire. La paire de clés sera alors générée et automatiquement téléchargée sur votre machine.

Après avoir créé la paire de clés SSH, vous pouvez l'utiliser pour accéder à vos instances EC2 dans votre infrastructure AWS. Si vous utilisez Terraform pour le déploiement, vous pouvez ajouter le code Terraform approprié pour spécifier l'utilisation de la paire de clés dans votre configuration.

Il est important de noter que les clés SSH sont utilisées pour sécuriser les connexions à vos instances EC2 et pour vous permettre de vous y connecter de manière sécurisée. Assurez-vous de stocker votre clé privée en lieu sûr et de ne pas la partager avec des personnes non autorisées.

Si vous avez des questions ou des difficultés lors de la création de la paire de clés SSH, n'hésitez pas à consulter la documentation AWS pour obtenir des instructions précises.

Notre code source Terraform est organisé dans le dossier `terraform/` et se compose de cinq fichiers principaux :

- `main.tf` : Ce fichier contient la configuration principale de notre infrastructure. Il permet de définir les ressources et les paramètres nécessaires à la création et à la gestion de notre infrastructure AWS. Dans ce fichier, vous pouvez modifier la variable locale `key_name` pour spécifier le nom de votre propre clé SSH.

```
key_name = "HomelabInfra"
```

- `variables.tf` : Ce fichier contient les variables utilisées dans notre configuration Terraform. Il permet de personnaliser certains paramètres de déploiement, tels que le chemin d'accès à votre clé SSH. Vous pouvez modifier la variable `default` pour spécifier le chemin d'accès à votre propre clé SSH.

```
default      = "../ssh_keys/HomelabInfra.pem"
```

- `3a1_subnet_configuration.tf` et `3b1_subnet_configuration.tf` : Ces fichiers décrivent la configuration des sous-réseaux et de leurs ressources (instances EC2) dans les zones de disponibilité `eu-west-3a` (PARIS zone A) et `eu-west-3b` (PARIS zone B). Ils spécifient le bloc CIDR, la description et les autres paramètres nécessaires à la création des sous-réseaux et leurs ressources.
- `domain.tf` : Ce fichier est utilisé pour configurer notre nom de domaine dans Route53. Vous pouvez le modifier en fonction de votre propre nom de domaine en vous basant sur l'exemple fourni dans le fichier.

En utilisant Terraform, nous avons pu déployer notre infrastructure AWS de manière cohérente, reproductible et évolutive. Le code source Terraform nous permet de versionner notre infrastructure, de la partager et de la gérer de manière efficace en utilisant les principes de l'Infrastructure as Code.

▼ Déploiement de l'infrastructure avec Terraform



Avant de commencer, assurez-vous d'avoir installé AWS CLI et Terraform sur votre machine. De plus, assurez-vous d'être connecté à votre compte avec AWS CLI.

Pour déployer notre infrastructure à l'aide de Terraform, suivez ces étapes simples :

1. Ouvrez votre terminal et rendez-vous dans le dossier `terraform/` de notre projet.
2. Exécutez la commande suivante pour initialiser Terraform et configurer l'environnement de travail :

```
terraform init
```

Cette commande va télécharger les plugins et les dépendances nécessaires pour le déploiement de l'infrastructure.

3. Ensuite, exécutez la commande suivante pour afficher le plan d'exécution de Terraform :

```
terraform plan
```

Cette commande va analyser notre code Terraform et afficher les actions que Terraform va effectuer pour déployer notre infrastructure. Vous pourrez voir les ressources qui seront créées, mises à jour ou supprimées.

4. Après avoir vérifié le plan d'exécution et si tout semble correct, vous pouvez procéder au déploiement de l'infrastructure en exécutant la commande suivante :

```
terraform apply
```

Cette commande va exécuter les actions décrites dans le plan précédent. Terraform va créer et configurer les ressources spécifiées dans notre

code Terraform, telles que les VPC, les sous-réseaux, les groupes de sécurité, etc.

Assurez-vous de vérifier et de confirmer les actions avant de procéder, car cela peut entraîner des modifications dans votre compte AWS.

En suivant ces étapes simples, vous pourrez déployer notre infrastructure réseau AWS à l'aide de Terraform de manière cohérente et reproductible.

▼ Configuration du nom de domaine

Afin de faire fonctionner votre nom de domaine avec AWS, vous devez le configurer auprès de votre fournisseur de nom de domaine. Dans mon cas, j'ai acheté mon nom de domaine chez OVH. Je vais donc accéder à mon espace client OVH pour effectuer les modifications nécessaires.

Dans l'espace client OVH, je vais chercher l'option permettant de modifier les serveurs de noms (name servers) associés à mon nom de domaine. Une fois que j'aurai trouvé cette option, je vais procéder au remplacement des serveurs de noms actuels par ceux fournis par AWS.

En utilisant les informations fournies par AWS, je vais saisir les nouveaux serveurs de noms dans les paramètres de mon nom de domaine chez OVH. Cela permettra de diriger les requêtes DNS vers la configuration DNS que nous avons déployée sur AWS.

Une fois les modifications enregistrées, je vais attendre un certain temps pour que les changements de serveurs de noms se propagent à travers Internet. Ce processus de propagation peut prendre plus ou moins de temps, en fonction des paramètres de votre fournisseur de nom de domaine.

Une fois que les serveurs de noms sont correctement configurés et que la propagation DNS est terminée, votre nom de domaine sera fonctionnel avec votre infrastructure AWS. Vous pourrez utiliser votre nom de domaine pour accéder à vos ressources hébergées sur AWS, telles que des sites web, des applications, des services, etc.

Il est important de noter que la procédure de configuration des serveurs de noms peut varier en fonction de votre fournisseur de nom de domaine. Assurez-vous de consulter la documentation ou de contacter le support de votre fournisseur pour obtenir des instructions précises sur la configuration des serveurs de noms.

Pour les utilisateurs d'OVH, voici les étapes à suivre pour configurer votre nom de domaine avec votre infrastructure réseau AWS :

- Cliquez sur **Web Cloud** dans la barre de navigation,
- Puis cliquez sur **Noms de domaine** dans la barre de navigation à gauche,
- Rendez-vous dans l'onglet **Serveurs DNS**,
- Cliquez sur **Modifier les serveurs DNS**.

1

2

3

4

5

Lorsque vous utilisez Terraform pour déployer votre infrastructure réseau sur AWS, il est important de connaître les noms des serveurs DNS associés à votre configuration. Ces informations vous seront nécessaires pour configurer correctement votre nom de domaine auprès de votre fournisseur de noms de domaine.

Pour obtenir les noms des serveurs DNS à utiliser, vous avez deux options :

1. Outputs de la commande `terraform apply` : Après avoir exécuté la commande `terraform apply` dans votre terminal pour déployer votre infrastructure, vous pouvez consulter les `Outputs` générés par Terraform. Les `Outputs` sont des valeurs que Terraform affiche à la fin du déploiement et qui sont utiles pour récupérer des informations importantes. Parmi ces informations, vous trouverez les noms des serveurs DNS associés à votre infrastructure. Notez ces noms pour les utiliser lors de la configuration de votre nom de domaine.

2. Commande `terraform refresh` : Si vous avez déjà déployé votre infrastructure avec Terraform mais que vous n'avez pas accès aux `Outputs` ou que vous souhaitez simplement actualiser les informations, vous pouvez exécuter la commande `terraform refresh` dans votre terminal. Cette commande permet à Terraform de récupérer les dernières informations de l'état actuel de votre infrastructure sur AWS. Une fois la commande exécutée, vous pouvez rechercher les noms des serveurs DNS dans les résultats affichés par Terraform.

En utilisant ces deux méthodes, vous serez en mesure de trouver les noms des serveurs DNS nécessaires pour configurer correctement votre nom de domaine. Assurez-vous de noter ces informations et de les utiliser lors de la configuration auprès de votre fournisseur de noms de domaine.

Si vous avez des difficultés à obtenir les noms des serveurs DNS, n'hésitez pas à demander de l'aide à votre équipe de développement ou à consulter la documentation de Terraform pour obtenir des instructions précises sur la récupération des `outputs` ou l'utilisation de la commande `terraform refresh`.



```
PORTEURS 7 SORTIE SQL CONSOLE TERMINAL CONSOLE DE DÉBOGAGE zsh - terraform + × ... ^ x
aws_route53_record.mongodb: Creation complete after 43s [id=Z093289020W8KNEXL2KG_mongodb.deleteSystem32.fr_A]
aws_route53_record.wireguard: Still creating... [50s elapsed]
aws_route53_record.api_parisguesser: Creation complete after 38s [id=Z093289020W8KNEXL2KG_api.parisguesser.deleteSystem32.fr_A]
aws_route53_record.wireguard: Creation complete after 52s [id=Z093289020W8KNEXL2KG_wireguard.deleteSystem32.fr_A]
aws_route53_record.traefik: Still creating... [40s elapsed]
aws_route53_record.traefik: Creation complete after 48s [id=Z093289020W8KNEXL2KG_traefik.deleteSystem32.fr_A]

Apply complete! Resources: 19 added, 0 changed, 0 destroyed.

Outputs:
cloudlab/internal_server_1_ip = "10.0.2.11"
cloudlab/internal_server_2_ip = "10.0.2.12"
cloudlab/public/app_projects_server_ip = "10.0.1.12"
cloudlab/public/facing_entrypoint_ip = "10.0.1.11"
deleteSystem32.fr_ns = tolist([
  "ns-1031.awsdns-00.org",
  "ns-1606.awsdns-08.co.uk",
  "ns-300.awsdns-37.com",
  "ns-830.awsdns-39.net",
])
migration/terraform git:main* 1445
```

Vous pouvez également les retrouver avec la console AWS dans **Route53 > Zones hébergées (barre de navigation à gauche) > Cliquez sur la zone de votre nom de domaine > Avec l'enregistrement de type NS.**

The screenshot shows the AWS Route 53 console with the domain 'deletesystem32.fr' selected. The left sidebar has 'Zones hébergées' highlighted. The main pane displays the 'Enregistrements (20)' tab for the zone. A specific NS record for the apex domain is highlighted with a red box. The table lists the following records:

Nom d...	Type	Stratég...	Différ...	Alias	Évaluate/Acheminer le traffi...	Durée...	ID de s...
deletesyst...	A	Simple	-	Non	13.37.148.174	300	-
deletesyst...	NS	Simple	-	Non	ns-830.awsdns-39.net. ns-1031.awsdns-00.org. ns-1606.awsdns-08.co.uk. ns-300.awsdns-37.com.	172800	-
deletesyst...	SOA	Simple	-	Non	ns-830.awsdns-39.net.awsd...	900	-
api.appli...	A	Simple	-	Non	13.37.148.174	300	-

Dans mon cas, les serveurs de noms (name servers) à utiliser pour ma configuration AWS sont les suivants :

- ns-1031.awsdns-00.org
- ns-1606.awsdns-08.co.uk
- ns-300.awsdns-37.com
- ns-830.awsdns-39.net

Modifiez les Serveurs DNS sur votre espace OVH, appliquez les modifications et vous devriez avoir ce qui suit :

Dans votre cas les Serveurs DNS en cours de suppression seront ceux de OVH.



Il est important de noter que les étapes spécifiques pour modifier les serveurs de noms peuvent varier en fonction de l'interface client OVH. Si vous rencontrez des difficultés pour effectuer ces modifications, je vous recommande de consulter la documentation d'OVH ou de contacter leur support technique pour obtenir une assistance supplémentaire.

▼ Étape 2 : Configuration de l'infrastructure

▼ Description

▼ L'utilisation de Docker

Pour commencer, dans notre projet, tous les services seront déployés à l'aide de Docker, qui est une technologie de conteneurisation. Cela signifie que chaque service sera encapsulé dans un conteneur isolé, garantissant ainsi une gestion simplifiée et une portabilité élevée des applications.

L'utilisation de Docker dans notre projet présente plusieurs avantages significatifs pour notre infrastructure basée sur AWS. Voici quelques-uns

des avantages clés d'avoir une infrastructure basée sur Docker :

1. **Isolation et portabilité** : Docker permet d'encapsuler chaque service dans un conteneur isolé, ce qui garantit que chaque application et ses dépendances sont parfaitement séparées les unes des autres. Cette isolation garantit la stabilité et la fiabilité de chaque service, tout en facilitant la gestion des applications. De plus, les conteneurs Docker sont portables, ce qui signifie qu'ils peuvent être exécutés sur n'importe quel environnement compatible Docker, que ce soit sur une machine locale, sur un serveur distant ou dans le cloud.
2. **Gestion simplifiée** : Docker fournit des outils puissants pour créer, déployer, gérer et surveiller les conteneurs. L'utilisation de Docker facilite la gestion de l'infrastructure en automatisant les tâches de déploiement et de configuration, réduisant ainsi la charge de travail administrative. De plus, Docker permet de mettre à jour et de faire évoluer facilement les applications, en minimisant les temps d'arrêt et les interruptions de service.
3. **Scalabilité et flexibilité** : Grâce à Docker, il est facile de mettre en place une infrastructure scalable et flexible. En utilisant des outils tels que Docker Swarm ou Kubernetes, il est possible d'orchestrer la gestion des conteneurs et assurer une distribution équilibrée de la charge de travail sur plusieurs nœuds. Cette capacité de mise à l'échelle horizontale permet d'adapter rapidement l'infrastructure aux besoins changeants, en ajoutant ou en supprimant facilement des instances de conteneurs selon les exigences de charge de travail.
4. **Reproductibilité et facilité de déploiement** : Docker permet de créer des images d'applications reproductibles, qui incluent tous les éléments nécessaires à l'exécution de l'application. Ces images peuvent être partagées et déployées sur n'importe quel environnement compatible Docker, ce qui facilite grandement le déploiement des applications sur différents serveurs ou plateformes. Cela simplifie également le processus de mise en production, en réduisant les problèmes de compatibilité et en garantissant une expérience de déploiement cohérente.

En résumé, l'utilisation de Docker dans notre infrastructure basée sur AWS offre une isolation efficace, une gestion simplifiée, une évolutivité flexible et une reproductibilité élevée. Ces avantages permettent d'optimiser

l'administration, de faciliter le déploiement des applications et de garantir une infrastructure stable et performante.

▼ NGINX Reverse Proxy

Dans la configuration de notre infrastructure, nous allons installer un reverse proxy NGINX avec NGINX Proxy Manager sur l'instance EC2 appelée `cloudlab_public_facing_entrypoint`. Ce service agira comme le routeur et le point d'entrée de notre infrastructure. Sa disponibilité est cruciale, car en cas de panne de ce service, cela peut causer des problèmes majeurs. L'utilisation d'un reverse proxy présente plusieurs avantages :

1. **Gestion du trafic** : Le reverse proxy, ici NGINX, permettra de gérer et de diriger efficacement le trafic entrant vers les services appropriés. Il agira comme une passerelle pour rediriger les requêtes des clients vers les bons services en fonction des règles de routage et de configuration spécifiées.
2. **Sécurité** : Le reverse proxy peut agir comme une barrière de sécurité en filtrant le trafic et en protégeant les services backend des attaques potentielles. Il peut également prendre en charge des fonctionnalités de sécurité supplémentaires, telles que le chiffrement SSL/TLS pour sécuriser les communications avec les clients.
3. **Gestion des certificats SSL** : Le reverse proxy NGINX Proxy Manager facilite la gestion des certificats SSL/TLS pour les domaines personnalisés. Il peut générer automatiquement et renouveler les certificats SSL pour chaque domaine, assurant ainsi une connexion sécurisée entre les clients et les services.

Proxy Hosts					
SOURCE	DESTINATION	SSL	ACCESS	STATUS	
admin.wireguard.deleteystem32.fr Created: 18th June 2023	http://10.0.1.11:5000	Let's Encrypt	Public	● Online	⋮
api.applicationsonnage.deleteystem32.fr Created: 18th June 2023	http://10.0.1.12:2821	Let's Encrypt	Public	● Online	⋮
api.countrygueser.deleteystem32.fr Created: 18th June 2023	http://10.0.1.12:2819	Let's Encrypt	Public	● Online	⋮
deleteystem32.fr Created: 18th June 2023	http://10.0.1.11:80	Let's Encrypt	internal	● Online	⋮
dns.deleteystem32.fr Created: 18th June 2023	http://10.0.2.11:85	Let's Encrypt	internal	● Online	⋮
edt.deleteystem32.fr Created: 18th June 2023	http://10.0.1.11:1408	Let's Encrypt	Public	● Online	⋮
goaccess.deleteystem32.fr Created: 18th June 2023	http://10.0.1.11:7880	Let's Encrypt	internal	● Online	⋮
mongodb.deleteystem32.fr Created: 18th June 2023	http://10.0.2.12:27017	Let's Encrypt	internal	● Online	⋮

Interface web NGINX Proxy Manager - Ici il s'agit des règles de redirections vers les bons services, sur les bons ports, sur les bonnes machines

▼ GoAccess

Pour pouvoir visualiser les logs de notre reverse proxy et obtenir des statistiques sur les requêtes qui passent par celui-ci, nous allons utiliser une application appelée GoAccess.

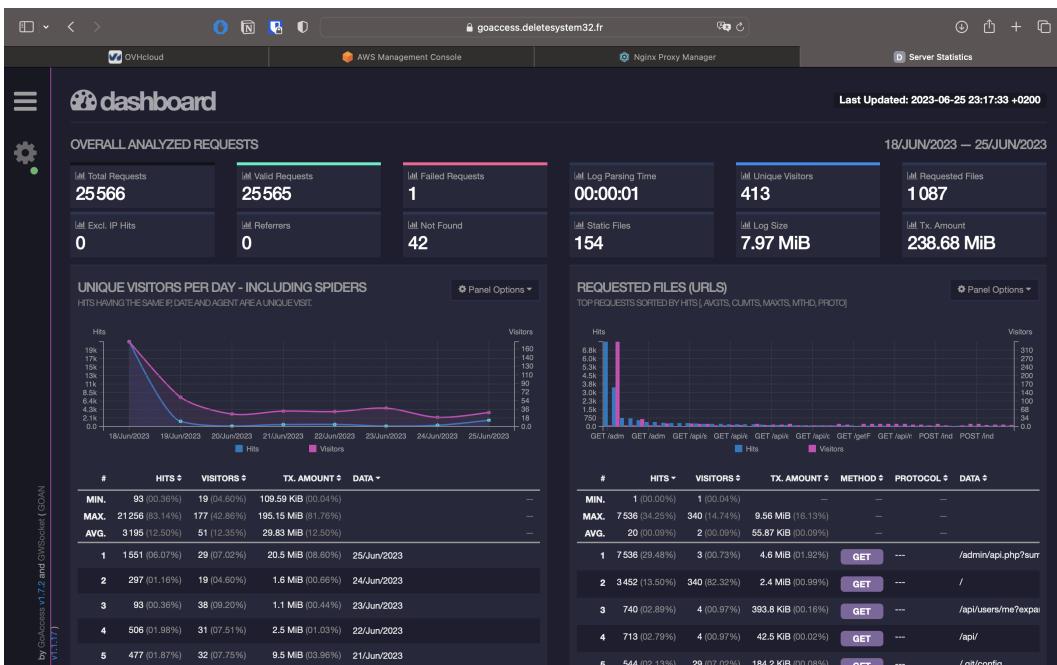
GoAccess est un outil de visualisation de logs qui nous permet d'analyser et de présenter les données de nos logs sous forme de tableaux et de graphiques interactifs. Cela nous offre une meilleure compréhension du trafic à travers notre reverse proxy NGINX.

L'installation et l'utilisation de GoAccess sont relativement simples. Nous allons l'installer (toujours avec Docker bien sûr) sur notre instance EC2 qui héberge le reverse proxy NGINX. Une fois installé, GoAccess pourra traiter les logs générés par NGINX et afficher les informations pertinentes, telles que les adresses IP des clients, les codes de réponse HTTP, les pages les plus consultées etc.

Grâce à cette visualisation des logs, nous pourrons :

- 1. Suivre les performances** : GoAccess nous permettra de surveiller les performances de notre reverse proxy. Nous pourrons ainsi prendre des mesures pour optimiser notre infrastructure et améliorer l'expérience utilisateur.

2. **Identifier les comportements suspects** : En analysant les logs avec GoAccess, nous pourrons repérer les activités anormales ou les tentatives d'accès non autorisées à nos services. Cela nous permettra de réagir rapidement et de renforcer les mesures de sécurité si nécessaire.
3. **Analyser les tendances de trafic** : GoAccess nous offre une vue d'ensemble du trafic, nous permettant d'identifier les périodes de pointe, les tendances de navigation et les pages les plus consultées. Ces informations peuvent être utilisées pour optimiser notre infrastructure, planifier les ressources et améliorer notre stratégie de contenu.



Interface web GoAccess pour visualiser les logs et avoir quelques statistiques sur les requêtes qui sont effectuées au travers du reverse proxy NGINX

▼ Serveur VPN WireGuard

Nous également allons installer un serveur VPN WireGuard sur cette même instance EC2. Le VPN, ou Réseau Privé Virtuel, nous permettra d'accéder aux autres machines et services de notre infrastructure réseau, même si nous sommes à l'extérieur du réseau du VPC et qu'ils ne sont pas directement accessibles depuis Internet.

L'utilisation d'un VPN présente plusieurs avantages :

- 1. Accès sécurisé aux ressources internes** : En utilisant le VPN WireGuard, nous pourrons établir une connexion sécurisée et chiffrée avec notre réseau privé. Cela nous permettra d'accéder aux machines et services internes de manière sécurisée, même depuis des emplacements distants, tels que des bureaux distants ou des connexions Wi-Fi publiques.
- 2. Protection des données** : Le VPN chiffre les données transitant entre notre appareil et le réseau interne, ce qui garantit que nos informations sensibles sont protégées contre les interceptions et les attaques potentielles. Cela est particulièrement important lorsque nous accédons à des ressources sensibles ou confidentielles.
- 3. Contournement des restrictions de réseau** : Si certaines ressources internes ne sont pas directement accessibles depuis Internet, le VPN nous permettra de contourner ces restrictions en créant un tunnel sécurisé qui nous connecte au réseau interne. Cela nous permettra d'accéder aux services et aux données internes comme si nous étions physiquement connectés au réseau local.
- 4. Filtrage basé sur l'adresse IP** : En combinant le VPN avec notre reverse proxy, nous pourrons établir des filtres basés sur l'adresse IP. Cela signifie que nous pouvons contrôler l'accès aux services en fonction de l'adresse IP de l'utilisateur, ajoutant une couche de sécurité supplémentaire à notre infrastructure.

En résumé, l'installation d'un serveur VPN WireGuard sur notre instance EC2, utilisant Docker, nous permettra d'accéder de manière sécurisée aux ressources internes de notre infrastructure réseau AWS, même depuis des emplacements distants. Cela renforce la sécurité de nos communications, nous permet de contourner les restrictions réseau et offre une flexibilité accrue pour l'accès aux services internes.

▼ Interface web d'administration pour WireGuard

Dans le but de simplifier l'administration du serveur VPN WireGuard, nous allons utiliser une interface web appelée WireGuard-UI. Cette interface web conviviale sera installée sur la même instance EC2 que celle utilisée pour notre serveur VPN WireGuard.

WireGuard-UI est une application spécialement conçue pour simplifier la gestion et la configuration des connexions VPN WireGuard. Elle offre une

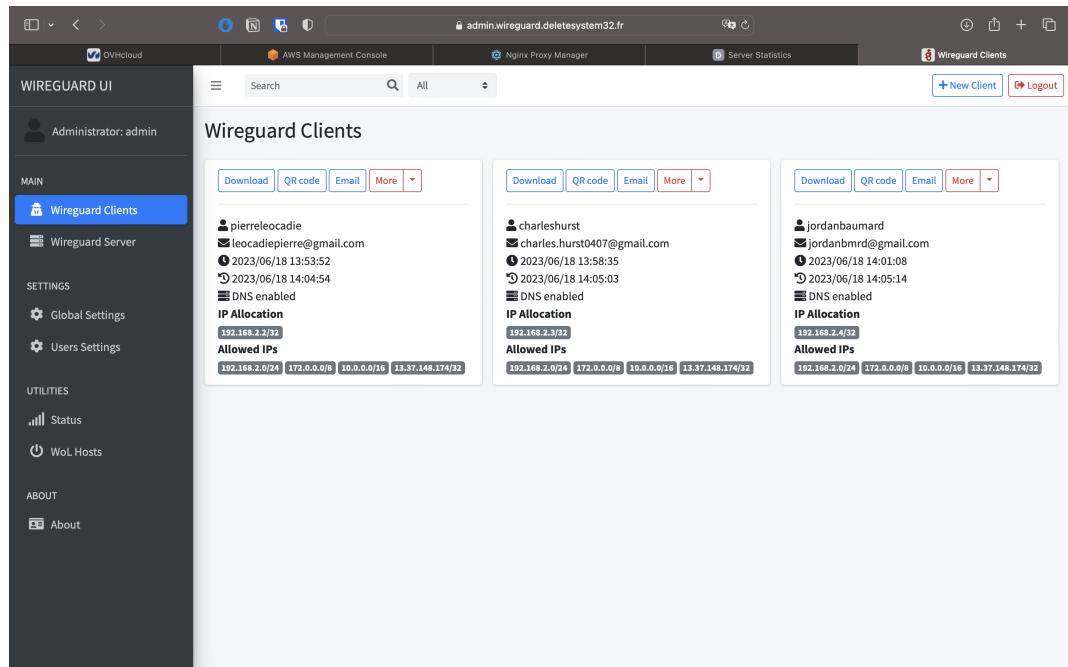
interface utilisateur conviviale qui permet de gérer facilement les clients VPN et les paramètres de configuration.

Pour installer WireGuard-UI, nous utiliserons Docker Compose, une technologie qui facilite le déploiement et la gestion d'applications conteneurisées. Avec Docker Compose, nous pourrons rapidement déployer l'interface web WireGuard-UI en même temps que le serveur VPN WireGuard en quelques étapes simples.

Une fois WireGuard-UI installé, nous pourrons profiter des fonctionnalités suivantes :

1. **Gestion des clients VPN** : WireGuard-UI permettra de créer et de gérer facilement les clients VPN WireGuard. Nous pourrons configurer les paramètres de connexion et accorder ou révoquer les autorisations d'accès.
2. **Configuration simplifiée** : L'interface web de WireGuard-UI simplifie la configuration du serveur VPN WireGuard. Nous pourrons facilement ajouter de nouveaux tunnels VPN et définir les paramètres réseau.
3. **Interface utilisateur intuitive** : WireGuard-UI est conçu pour offrir une expérience utilisateur conviviale et intuitive. Les fonctionnalités principales sont accessibles via une interface simple et bien organisée, permettant aux administrateurs de gérer efficacement le serveur VPN sans avoir à se plonger dans des commandes complexes.

L'installation de WireGuard-UI sur la même instance EC2 que notre serveur VPN WireGuard, en utilisant Docker Compose, nous permettra d'avoir une interface web pratique et facile à utiliser pour administrer notre VPN. Cela facilitera la gestion des clients VPN et la configuration du serveur, contribuant ainsi à une expérience d'administration simplifiée et efficace.



Interface web WireGuard UI - Ici on peut voir les différents clients qui ont été créés

▼ Portainer

Afin de faciliter le déploiement, la gestion et la surveillance de l'ensemble des conteneurs et des piles de conteneurs de notre infrastructure, nous utiliserons le service Portainer. Portainer est une plateforme qui offre une interface graphique conviviale pour administrer les conteneurs Docker.

Pour que le service Portainer puisse fonctionner, nous devrons installer l'agent Portainer sur les serveurs où nos conteneurs sont exécutés :

- `cloudbuild_public_facing_entrypoint`,
- `cloudbuild_public_app_projects_server`,
- `cloudbuild_internal_server_2`.

Cet agent transmettra les informations pertinentes au serveur central Portainer : `cloudbuild_internal_server_1`, permettant ainsi de superviser et de gérer les conteneurs à partir d'une interface utilisateur intuitive.

L'utilisation de Portainer présente plusieurs avantages :

- Déploiement simplifié** : Portainer facilite le déploiement des conteneurs en fournissant une interface graphique conviviale. Nous pourrons créer, démarrer, arrêter et supprimer des conteneurs avec quelques clics, sans avoir à utiliser des commandes complexes.

2. **Gestion centralisée** : Grâce à Portainer, nous pourrons gérer tous nos conteneurs à partir d'une seule interface. Cela simplifie la gestion de l'infrastructure en permettant de surveiller les ressources, de vérifier les journaux, de gérer les images et de mettre à jour les conteneurs, le tout depuis une seule plateforme.
3. **Surveillance des performances** : Portainer nous offre des outils de surveillance qui permettent de suivre les performances de nos conteneurs en temps réel. Nous pourrons visualiser les statistiques clés, telles que l'utilisation des ressources (CPU, mémoire, etc.), les débits réseau, les temps de réponse, et prendre des mesures pour optimiser les performances si nécessaire.

L'installation de l'agent Portainer sur nos serveurs nous permettra de bénéficier de tous les avantages de Portainer pour la gestion et la surveillance simplifiées de notre infrastructure de conteneurs. Nous pourrons déployer, gérer et superviser efficacement nos conteneurs grâce à cette interface graphique conviviale.

En résumé, l'utilisation de Portainer avec l'agent Portainer nous offre une solution pratique et intuitive pour le déploiement, la gestion et la surveillance de nos conteneurs Docker. Cela facilite l'administration de l'infrastructure et permet d'optimiser les performances de nos applications basées sur des conteneurs.

Name	State	Quick Actions	Stack	Image	Created	IP
goaccess	running		goaccess	xavierh/goaccess-for-nginxproxymanager:latest	2023-06-18 19:13:10	17
edt_website	running		-	edt_website	2023-06-18 18:43:24	17
edt_script_ics	running		-	edt_script_ics	2023-06-18 18:38:29	17
wireguard-ui	running		wireguard	ngoduykhanh/wireguard-ui:latest	2023-06-18 13:12:58	-
wireguard	running		wireguard	lscr.io/linuxserver/wireguard:latest	2023-06-18 13:12:58	17
nginx-proxy-manager-app-1	running		nginx-proxy-manager	jc21/nginx-proxy-manager:latest	2023-06-18 12:29:32	17
portainer_agent	running		-	portainer/agent:2.18.1	2023-06-17 18:13:43	17

Interface web portainer - Containers du serveur `cloudlab_public_facing_entrypoint`

▼ Serveur DNS Pi-hole

Dans notre projet, nous avons mis en place un serveur DNS pour notre infrastructure, qui sera également utilisé par notre VPN pour accéder à une configuration DNS personnalisée si nécessaire. Pour cela, nous utilisons Pi-hole, que nous avons installé dans un conteneur Docker sur la même instance EC2 que notre serveur central Portainer, c'est-à-dire sur l'instance EC2 appelée `cloudlab_internal_server_1`.

La raison pour laquelle nous avons choisi de regrouper le serveur central Portainer et le DNS Pi-hole sur une instance EC2 distincte de celle du reverse proxy NGINX et du serveur VPN WireGuard, à savoir `cloudlab_public_facing_entrypoint`, est de réduire les risques d'une défaillance simultanée de tous les services cruciaux de notre infrastructure. En les séparant, nous évitons qu'un problème affecte tous ces services en même temps.

Le DNS Pi-hole est un élément important de notre infrastructure, car il nous permet de configurer des enregistrements DNS internes sans avoir recours aux zones hébergées privées de Route 53 avec AWS. Cela peut être utile pour certains projets qui nécessitent une gestion personnalisée des enregistrements DNS.

En résumé, l'utilisation de Pi-hole en tant que serveur DNS dans un conteneur Docker nous offre une flexibilité et une personnalisation accrues pour la gestion des enregistrements DNS internes. Sa séparation sur une instance EC2 distincte du serveur central Portainer contribue à la résilience de notre infrastructure en cas de problème.

Time	Type	Domain	Client	Status	Reply	Action
2023-06-26 00:17:28	HTTPS	gateway.fe.apple-dns.net	10.0.1.11	OK (answered by dns.google#453)	NODATA (1.6ms)	Block
2023-06-26 00:17:28	A	gateway.fe.apple-dns.net	10.0.1.11	OK (answered by dns.google#453)	IP (12.2ms)	Block
2023-06-26 00:17:26	A	optimizationguide-pa.googleapis.com	10.0.1.11	OK (answered by dns.google#453)	IP (8.6ms)	Block
2023-06-26 00:17:08	HTTPS	googlehosted.googleapisusercontent.com	10.0.1.11	OK (answered by dns.google#453)	NODATA (6.3ms)	Block
2023-06-26 00:17:08	A	caldav.fe.apple-dns.net	10.0.1.11	OK (answered by dns.google#453)	IP (10.3ms)	Block
2023-06-26 00:17:08	AAAA	caldav.fe.apple-dns.net	10.0.1.11	OK (answered by dns.google#453)	IP (11.8ms)	Block
2023-06-26 00:17:01	HTTPS	valid.apple.gapiling.com	10.0.1.11	OK (answered by dns.google#453)	NODATA (5.8ms)	Block
2023-06-26 00:17:01	A	valid.apple.com	10.0.1.11	OK (answered by dns.google#453)	CNAME (17.7ms)	Block
2023-06-26 00:17:01	AAAA	valid.apple.com	10.0.1.11	OK (answered by dns.google#453)	CNAME (29.5ms)	Block
2023-06-26 00:17:01	HTTPS	valid.apple.com	10.0.1.11	OK (answered by dns.google#453)	CNAME (6.0ms)	Block
2023-06-26 00:17:01	A	wpad	10.0.1.11	OK (cache)	NODOMAIN (0.1ms)	Block
2023-06-26 00:17:01	AAAA	wpad	10.0.1.11	OK (cache)	NODOMAIN (0.1ms)	Block
2023-06-26 00:16:40	A	icloud.com	10.0.1.11	OK (answered by dns.google#453)	IP (1.4ms)	Block
2023-06-26 00:16:40	AAAA	icloud.com	10.0.1.11	OK (answered by dns.google#453)	IP (1.4ms)	Block

Interface d'administration du serveur DNS Pi-hole - Ici on peut voir les requêtes qui passent par le serveur DNS lorsqu'une personne est connecté au VPN

▼ Les deux autres serveurs :

`cloudlab_public_app_projects_server` et `cloudlab_internal_server_2`

Dans notre infrastructure, nous utilisons deux serveurs pour héberger nos projets d'applications (`cloudlab_public_app_projects_server`) et nos serveurs de base de données (`cloudlab_internal_server_2`). Cette répartition des tâches nous permet de ne pas surcharger les deux autres serveurs cruciaux de notre infrastructure, qui sont responsables du bon fonctionnement de services essentiels tels que le reverse proxy, le serveur VPN, Portainer et le serveur DNS.

De plus, nos projets d'applications et nos bases de données ne nécessitent pas une utilisation quotidienne. Nous avons donc la flexibilité de les éteindre lorsque nous n'en avons pas besoin, ce qui nous permet d'économiser de l'argent en réduisant les coûts de notre infrastructure. Lorsqu'une instance EC2 est éteinte, nous ne payons que le stockage des données associées à cette instance. En moyenne, notre infrastructure Cloud Lab nous coûte entre 15 et 20€ par mois.

Ces deux serveurs, qui hébergent nos projets d'applications et nos bases de données, n'ont pas de rôle critique dans le fonctionnement global de notre infrastructure, contrairement aux deux autres serveurs. Vous êtes libre de les utiliser pour héberger ce que vous souhaitez, en fonction de vos besoins spécifiques.

▼ Résumé de l'infrastructure et de sa configuration

Pour résumer l'infrastructure et sa configuration, voici les différents rôles et fonctionnalités de chaque serveur :

1. `cloudbl_public_facing_entrypoint` : Ce serveur fait office de point d'entrée vers notre infrastructure depuis Internet. Il permet d'accéder aux applications hébergées sur notre infrastructure, notamment nos projets d'applications qui sont exposés publiquement via le serveur `cloudbl_public_app_projects_server`. De plus, il permet également l'accès aux applications internes via VPN, telles que les interfaces d'administration NGINX Proxy Manager, WireGuard UI, Portainer, etc. Ce serveur est donc crucial pour le bon fonctionnement de notre infrastructure.
2. `cloudbl_public_app_projects_server` : Ce serveur héberge nos projets d'applications. Il est indirectement exposé au public via le serveur `cloudbl_public_facing_entrypoint`. Les applications hébergées sur ce serveur peuvent être accessibles par le public grâce au reverse proxy NGINX configuré sur le serveur public. Cependant, l'accès aux applications internes se fait uniquement via VPN.
3. `cloudbl_internal_server_1` : Ce serveur n'est pas accessible depuis Internet, mais uniquement via VPN. Il héberge une partie critique de nos applications, notamment Portainer et le serveur DNS Pi-hole. Portainer est un outil qui facilite la gestion et le déploiement des conteneurs Docker, tandis que le serveur DNS Pi-hole permet de personnaliser la configuration DNS interne de notre infrastructure. Ce serveur contribue au bon fonctionnement et à l'administration de notre infrastructure.
4. `cloudbl_internal_server_2` : Ce serveur n'est pas accessible depuis Internet, mais uniquement via VPN. Son rôle principal est d'héberger nos serveurs de base de données. Il est responsable du stockage et de la gestion de nos données importantes. Son accès est sécurisé via VPN pour garantir la confidentialité de nos bases de données.

Chaque serveur joue un rôle spécifique dans notre infrastructure et contribue au bon fonctionnement de nos applications et services. L'utilisation de VPN nous permet de renforcer la sécurité en limitant l'accès aux serveurs internes, tandis que l'exposition publique se fait à travers le

serveur `cloudbl_public_facing_entrypoint` pour les applications destinées au grand public.

▼ Code source Ansible de la configuration



Avant de commencer, assurez-vous d'avoir installé Ansible sur votre machine.

Pour configurer notre infrastructure, nous utiliserons Ansible. Rendez-vous dans le dossier `ansible/`.

Ce dossier contient cinq fichiers :

1. `ansible.cfg` : Dans ce fichier, vous devez modifier la variable du chemin de votre clé SSH `private_key_file` avec le chemin vers votre clé SSH. Assurez-vous également de spécifier le nom d'utilisateur dans la variable `remote_user` qui correspond à l'utilisateur pour lequel votre clé SSH est configurée.

```
private_key_file = .../ssh_keys/HomelabInfra.pem
```

2. `hosts.ini` : Dans ce fichier, vous devez spécifier les adresses IP publiques de chaque serveur.
 - La section `[cloudbl_public_facing_entrypoint]` contient l'adresse IP publique du serveur `cloudbl_public_facing_entrypoint`.
 - La section `[cloudbl_public_app_projects_server]` contient l'adresse IP publique du serveur `cloudbl_public_app_projects_server`.
 - La section `[cloudbl_internal_server_1]` contient l'adresse IP publique du serveur `cloudbl_internal_server_1`.
 - La section `[cloudbl_internal_server_2]` contient l'adresse IP publique du serveur `cloudbl_internal_server_2`.



Notez que les adresses IP publiques des serveurs `cloudbl_public_app_projects_server`, `cloudbl_internal_server_1` et `cloudbl_internal_server_2` ne sont pas statiques et changent à chaque redémarrage des serveurs.



Vous pouvez récupérer les adresses IP publiques en accédant à la console AWS, en recherchant EC2, en cliquant sur Instances dans la barre de navigation de gauche, puis en sélectionnant une instance pour afficher son adresse IP publique.

```
[cloudlab_public_facing_entrypoint]
--IP publique du serveur cloudlab_public_facing_entrypoint--

[cloudlab_public_app_projects_server]
--IP publique du serveur cloudlab_public_app_projects_server--

[cloudlab_internal_server_1]
--IP publique du serveur cloudlab_internal_server_1--

[cloudlab_internal_server_2]
--IP publique du serveur cloudlab_internal_server_2--
```

- `install-docker.yml` : playbook ansible pour l'installation de docker sur tous les serveurs.
- `install-portainer-agent.yml` : playbook ansible pour l'installation des agents portainer sur les serveurs `cloudlab_public_facing_entrypoint`, `cloudlab_public_app_projects_server` et `cloudlab_internal_server_2`.
- `install-portainer.yml` : playbook ansible pour l'installation de Portainer sur le serveur `cloudlab_internal_server_1`.

▼ Configuration avec Ansible et Portainer

▼ Tester la connectivité

Pour configurer notre infrastructure avec Ansible, nous allons d'abord tester la connectivité entre Ansible et les serveurs en utilisant la commande suivante :

```
ansible all -m ping
```

Cette commande permet de vérifier si Ansible parvient à se connecter correctement aux serveurs. Si Ansible ne parvient pas à se connecter aux serveurs, assurez-vous d'utiliser la clé SSH appropriée et de spécifier le bon utilisateur pour la connexion. Vérifiez également que les adresses IP des serveurs sont correctement configurées dans le fichier `hosts.ini`

d'Ansible. Veuillez vous assurer d'avoir correctement configuré le CIDR bloc 0.0.0.0/0 dans les sources autorisées des groupes de sécurité des serveurs à l'aide de Terraform.

Dans le contexte d'une infrastructure réseau AWS, les groupes de sécurité jouent un rôle crucial dans la définition des règles de sécurité pour les instances EC2. Lorsque vous configurez les groupes de sécurité à l'aide de Terraform, il est important de spécifier les sources autorisées qui peuvent accéder aux instances EC2.

Le CIDR bloc 0.0.0.0/0 est une notation qui représente toutes les adresses IP possibles, ce qui signifie que toutes les adresses IP sont autorisées à accéder aux instances EC2 concernées. Cela permet un accès ouvert depuis n'importe quelle adresse IP.

Vérifiez donc que vous avez correctement ajouté le CIDR bloc 0.0.0.0/0 dans les sources autorisées des groupes de sécurité de vos serveurs à l'aide de Terraform. Cette configuration garantit que les instances EC2 sont accessibles depuis toutes les adresses IP.

Lors de l'exécution de la commande, vous devriez recevoir une réponse indiquant si la connexion a réussi (SUCCESS) ou échoué (UNREACHABLE). Cette étape est essentielle pour s'assurer que la configuration ultérieure avec Ansible se déroulera correctement.

▼ Exécution des playbooks Ansible

Pour déployer les configurations sur les serveurs de notre infrastructure, nous utiliserons Ansible, qui est une plateforme d'automatisation et de gestion de configuration. Nous avons préparé trois playbooks Ansible pour faciliter l'installation des composants nécessaires sur chaque serveur.

Le premier playbook, `install-docker.yml`, sera utilisé pour installer Docker sur tous les serveurs.

Le deuxième playbook, `install-portainer-agent.yml`, sera exécuté pour installer les agents Portainer sur les serveurs

`cloudlab_public_facing_entrypoint`, `cloudlab_public_app_projects_server` et `cloudlab_internal_server_2`. Les agents Portainer permettent de gérer et de surveiller les conteneurs Docker de manière centralisée.

Enfin, le troisième playbook, `install-portainer.yml`, sera utilisé pour installer Portainer sur le serveur `cloudlab_internal_server_1`.

Pour exécuter un playbook Ansible, ouvrez votre terminal et assurez-vous d'être dans le dossier `ansible/`. Ensuite, exécutez la commande suivante en remplaçant `[nom_du_playbook_à_exécuter]` par le nom du playbook que vous souhaitez exécuter :

```
ansible-playbook [nom_du_playbook_à_exécuter] -vv
```

Cette commande lancera l'exécution du playbook spécifié et vous fournira des informations détaillées sur les tâches en cours d'exécution.

Assurez-vous de suivre cette procédure pour chaque playbook afin d'installer correctement les composants nécessaires sur les serveurs.

▼ Configuration de Portainer



Lien vers la documentation de Portainer si nécessaire :

<https://docs.portainer.io>



Cette infrastructure nécessite la version gratuite de Portainer Business Edition qui permet de gérer jusqu'à 5 noeuds (serveurs). Cela nécessite de se créer un compte pour ce faire : <https://www.portainer.io/take-5>.

Maintenant, nous allons passer à la configuration de Portainer. Pour commencer, ouvrez votre navigateur et saisissez l'adresse IP publique du serveur `cloudlab_internal_server_1`, suivie du port 9000. Par exemple, si l'adresse IP publique est 192.168.0.1, vous devez accéder à l'URL

`http://192.168.0.1:9000`.

Une fois que vous avez configuré Portainer, vous pouvez ajouter les environnements des autres serveurs à votre interface. Pour ce faire, suivez ces étapes :

1. Connectez-vous à Portainer à l'aide de vos informations d'identification.
2. Dans la barre de navigation de gauche, cliquez sur "Settings" (Paramètres).

3. Sélectionnez "Environments" (Environnements).
4. Cliquez sur le bouton "+ Add environment" (Ajouter un environnement).
5. Dans la fenêtre pop-up, sélectionnez "Docker Standalone".
6. Cliquez sur "Start wizard" (Démarrer l'assistant).

Maintenant, vous allez ajouter les informations spécifiques pour chaque serveur :

- Pour le serveur `cloudbl_public_facing_entrypoint`, saisissez son adresse IP privée suivie du port 9001. Par exemple, si l'adresse IP privée est 10.0.1.11, entrez "10.0.1.11:9001".
- Pour le serveur `cloudbl_public_app_projects_server`, utilisez son adresse IP privée suivie du port 9001 de la même manière que précédemment.
- Pour le serveur `cloudbl_internal_server_2`, utilisez également son adresse IP privée suivie du port 9001.

Assurez-vous de remplir le champ "Name" (Nom) avec le nom que vous souhaitez donner à chaque environnement. Par exemple, vous pouvez utiliser "cloudbl_public_facing_entrypoint" pour le premier serveur.

Répétez ces étapes pour tous les serveurs que vous souhaitez ajouter à Portainer.

Une fois que vous avez ajouté tous les environnements, vous pourrez accéder à chacun d'eux depuis l'interface de Portainer. Cela vous permettra de visualiser et de gérer les conteneurs Docker sur chaque serveur de manière centralisée, ce qui facilite l'administration et la surveillance de votre infrastructure.

Portainer portainer.deleteyoursystem32.fr

portainer.io BUSINESS EDITION

Environment management

Environments

Environments

Name	Type	URL	GroupName	Actions
Internal Server 1 - local	Docker	/var/run/docker.sock	Unassigned	Manage access
Internal Server 2	Agent	10.0.2.12:9001	Unassigned	Manage access
Public Facing Entrypoint	Agent	10.0.1.11:9001	Unassigned	Manage access
Public App Projects	Agent	10.0.1.12:9001	Unassigned	Manage access

Items per page: 10

© Portainer Business Edition 2.18.3

Portainer portainer.deleteyoursystem32.fr

portainer.io BUSINESS EDITION

Environment management

Quick Setup

Environment Wizard

Select your environment(s)

You can onboard different types of environments, select all that apply.

Connect to existing environments

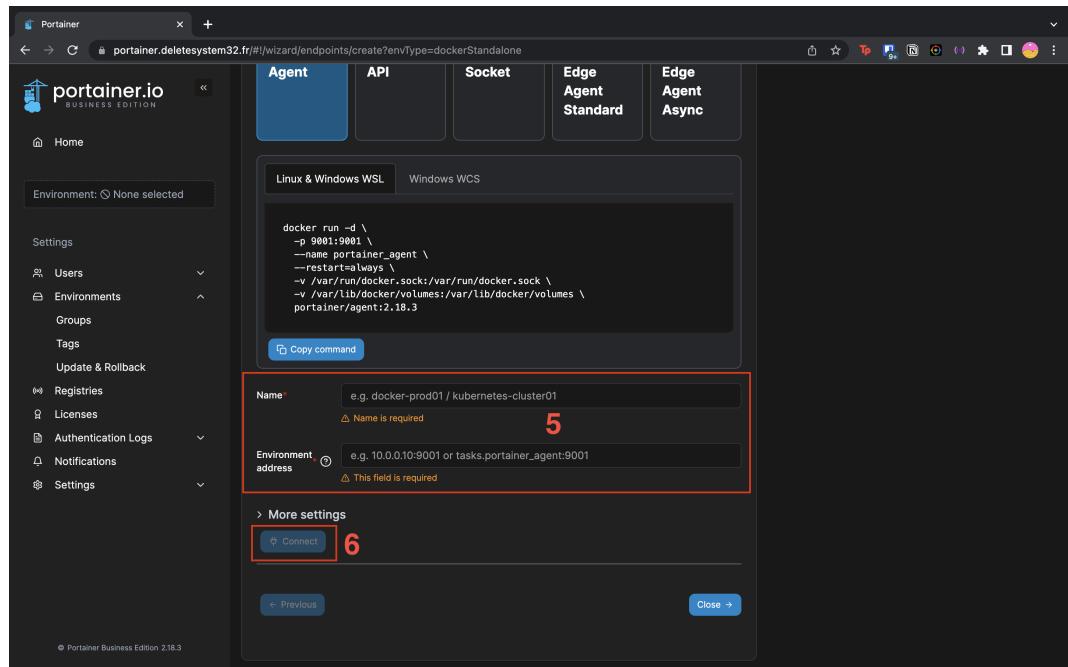
- Docker Standalone** 3 Connect to Docker Standalone via URL/IP, API or Socket
- Docker Swarm** Connect to Docker Swarm via URL/IP, API or Socket
- Kubernetes** Connect to a Kubernetes environment via URL/IP or via kubeconfig import
- ACI** Connect to ACI environment via API
- Nomad** Connect to HashiCorp Nomad environment via API

Set up new environments

- Provision KaaS Cluster** Provision a Kubernetes cluster via a cloud provider's Kubernetes as a Service
- Create a Kubernetes cluster** Create a Kubernetes cluster on existing infrastructure

Start Wizard 4

© Portainer Business Edition 2.18.3



▼ Déploiement des autres applications avec Portainer

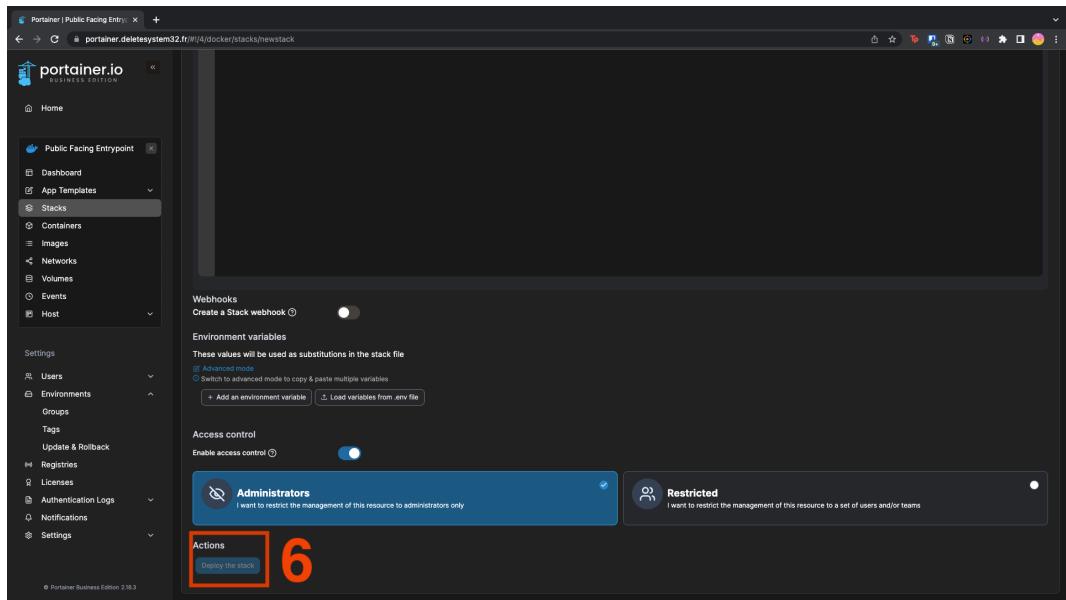
Une fois que tous les serveurs sont connectés à Portainer, nous pouvons procéder au déploiement des services sur nos serveurs en utilisant Portainer. Dans notre cas, nous allons commencer par configurer le serveur `cloudlab_public_facing_entrypoint` en déployant le service NGINX Proxy Manager.

Voici les étapes à suivre :

1. Accédez au dossier `docker/` dans votre système.
2. Ouvrez le fichier `docker-compose-nginx-proxy-manager.yml`.
3. Copiez le contenu du fichier.
4. Connectez-vous à l'interface d'administration de Portainer.
5. Dans la barre de navigation de gauche, sélectionnez "Stacks".
6. Cliquez sur le bouton "+ Add stack" pour créer une nouvelle stack.
7. Donnez un nom à la stack, par exemple `nginx-proxy-manager`.
8. Collez le contenu du fichier `docker-compose-nginx-proxy-manager.yml` dans l'éditeur web de Portainer.
9. Cliquez sur le bouton "Deploy the stack" en bas de la page pour déployer la stack.

The screenshot shows the Portainer Business Edition interface. On the left, a sidebar menu includes Home, Dashboard, App Templates, Stacks (highlighted with a red box and labeled 2), Containers, Images, Networks, Volumes, Events, and Host. The main content area is titled 'Stacks list' and displays a table of stacks. The columns are Name, Type, Images up to date, Control, Created, and Ownership. Three stacks are listed: 'goaccess' (Compose, up to date, Total, 2023-06-18 19:13:04 by admin, administrators), 'nginx-proxy-manager' (Compose, up to date, Total, 2023-06-18 11:08:40 by admin, administrators), and 'wregard' (Compose, up to date, Total, 2023-06-18 12:44:24 by admin, administrators). A red box highlights the '+ Add stack' button in the top right corner of the table header, labeled 3.

The screenshot shows the 'Create stack' page. The left sidebar is identical to the previous screen. The main area has a title 'Create stack'. It features a 'Name' input field containing '4 e.g. mystack' (highlighted with a red box) and a note below it stating 'This stack will be deployed using docker compose'. Below this are four build method options: 'Web editor' (selected, highlighted with a red box and labeled 4), 'Upload' (Upload from your computer), 'Repository' (Use a git repository), and 'Custom template' (Use a custom template). A large red box highlights the 'Web editor' section, which contains a text area for defining or pasting a Docker Compose file. The number '5' is placed inside this red box.



Une fois que vous avez déployé la stack, Portainer se chargera de créer et de configurer les conteneurs nécessaires pour exécuter NGINX Proxy Manager sur le serveur `cloudlab_public_facing_entrypoint`. Ce service agira comme un routeur et un point d'entrée pour notre infrastructure, facilitant la gestion des connexions entrantes et sortantes.

Une fois le déploiement du service NGINX Proxy Manager terminé, vous pouvez accéder à son interface d'administration via votre navigateur. Pour ce faire, entrez l'adresse IP du serveur suivi du port 81 dans la barre d'URL.

Une fois sur la page de connexion de l'interface d'administration de NGINX Proxy Manager, utilisez les identifiants par défaut suivants :

`admin@example.com` comme adresse e-mail et `changeme` comme mot de passe. Après vous être connecté, il est recommandé de modifier ces identifiants par défaut en choisissant une adresse e-mail et un mot de passe personnalisés.

Une fois connecté et après avoir changé vos identifiants, vous pourrez commencer à ajouter vos Proxy Hosts. Les Proxy Hosts vous permettent de configurer les connexions entrantes et de rediriger le trafic vers les applications spécifiques hébergées sur votre infrastructure. Vous pouvez ajouter autant de Proxy Hosts que nécessaire en spécifiant les détails tels que l'adresse IP et le port du serveur cible, ainsi que les paramètres de routage appropriés.

Voici un exemple :

The screenshot displays two views of the Nginx Proxy Manager interface. The top view shows a list of proxy hosts, each with a name, source, destination, SSL status, access level, and status. The bottom view is a detailed edit screen for a specific host, showing fields for domain names, scheme, forward hostname/IP, forward port, and various configuration options like cache assets and websockets support.

Source	Destination	SSL	Access	Status
admin.wireguard.deleteSystem32.fr	http://10.0.1.11:5000	Let's Encrypt	internal	Online
api.applicationusage.deleteSystem32.fr	http://10.0.1.12:2821	Let's Encrypt	Public	Online
api.countryguesser.deleteSystem32.fr	http://10.0.1.12:2819	Let's Encrypt	Public	Online
deleteSystem32.fr	http://10.0.1.11:80	Let's Encrypt	internal	Online
dns.deleteSystem32.fr	http://10.0.2.11:85	Let's Encrypt	internal	Online
etcd.deleteSystem32.fr	http://10.0.1.11:1408	Let's Encrypt	Public	Online
goaccess.deleteSystem32.fr	http://10.0.1.11:7880	Let's Encrypt	internal	Online
mongodb.deleteSystem32.fr	http://10.0.2.12:27017	Let's Encrypt	internal	Online
mongodbsession.deleteSystem32.fr	http://10.0.2.12:8081	Let's Encrypt	internal	Online
phpmyadmin.deleteSystem32.fr	http://10.0.2.12:2817	Let's Encrypt	internal	Online

Une fois que vous avez configuré vos Proxy Hosts dans NGINX Proxy Manager, nous allons maintenant procéder au déploiement du serveur VPN WireGuard et de son interface web d'administration, WireGuard-UI. Nous allons effectuer cette étape sur le même serveur,

`cloudlab_public_facing_entrypoint`.

Pour configurer le déploiement de WireGuard, suivez ces étapes simples :

1. Accédez au dossier `docker/` où se trouvent les fichiers nécessaires pour la configuration.
2. Ouvrez le fichier `docker-compose-wireguard.yml` avec un éditeur de texte.
3. Recherchez la variable d'environnement `SERVERURL` dans la configuration de WireGuard.
4. Remplacez la valeur de la variable `SERVERURL` par l'adresse IP publique du serveur où WireGuard sera déployé ou par le nom de domaine que vous avez spécifiquement dédié à WireGuard. Assurez-vous que cette adresse est accessible depuis Internet.
5. Copiez le contenu du fichier `docker-compose-wireguard.yml`.

Ces étapes permettent de configurer correctement WireGuard en spécifiant l'adresse IP publique ou le nom de domaine pour le serveur. Cette information est essentielle pour que les clients VPN puissent se connecter au bon serveur WireGuard.

Il est important de noter que l'utilisation d'un nom de domaine peut faciliter la gestion et l'accès à votre infrastructure, car vous pouvez associer un nom facilement mémorisable à votre serveur au lieu d'utiliser une adresse IP.

Effectuez la même procédure que pour le déploiement de NGINX Proxy Manager.

Une fois que vous avez déployé WireGuard et WireGuard-UI, connectez-vous en SSH au serveur `cloudlab_public_facing_entrypoint`.

Pour cela, ouvrez votre terminal et exécutez la commande suivante :

```
ssh -i [chemin_vers_votre_clé_ssh] [nom]@[ip]
# Exemple : ssh -i ssh_keys/HomelabInfra.pem admin@13.37.148.174
```

Une fois connecté en SSH, nous allons vérifier la valeur actuelle de la variable de noyau `net.ipv4.ip_forward`. Cette variable permet au système d'envoyer des paquets IP entre les différentes interfaces réseau, ce qui est nécessaire pour le fonctionnement du serveur VPN.

Exécutez la commande suivante :

```
sudo sysctl net.ipv4.ip_forward
```

Si le résultat retourné n'est pas `net.ipv4.ip_forward = 1`, cela signifie que la redirection des paquets IP n'est pas activée. Dans ce cas, suivez les étapes suivantes pour activer la redirection :

1. Exécutez la commande suivante pour ouvrir le fichier de configuration `sysctl.conf` avec l'éditeur de texte Nano :

```
sudo nano /etc/sysctl.conf
```

Ce fichier contient les paramètres de configuration du système. Vous pouvez décommenter la ligne `net.ipv4.ip_forward=1` en supprimant le symbole `#` au début de la ligne.

```

GNU nano 5.4                               /etc/sysctl.conf
#net.ipv4.conf.all.rp_filter=1

# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1

# Uncomment the next line to enable packet forwarding for IPv6
# Enabling this option disables Stateless Address Autoconfiguration
# based on Router Advertisements for this host
#net.ipv6.conf.all.forwarding=1

#####
# Additional settings - these settings can improve the network
# security of the host and prevent against some network attacks
# including spoofing attacks and man in the middle attacks through
# redirection. Some network environments, however, require that these
# settings are disabled so review and enable them as needed.
#
# Do not accept ICMP redirects (prevent MITM attacks)
#net.ipv4.conf.all.accept_redirects = 0
#net.ipv6.conf.all.accept_redirects = 0
# _or_

```

Help Exit Write Out Where Is Cut Execute Location Undo
R Read File Replace Paste Justify Go To Line Redo Set Mark
M-A Copy M-G

- Enregistrez et fermez le fichier `sysctl.conf`. Pour charger les nouvelles configurations, exécutez la commande suivante :

```
sudo sysctl -p
```

Cela permet d'appliquer les modifications du fichier `sysctl.conf` sans redémarrer le système.

- Vérifiez à nouveau la valeur de la variable `net.ipv4.ip_forward` en exécutant la commande suivante :

```
sudo sysctl net.ipv4.ip_forward
```

Assurez-vous que le résultat affiche `net.ipv4.ip_forward = 1`.

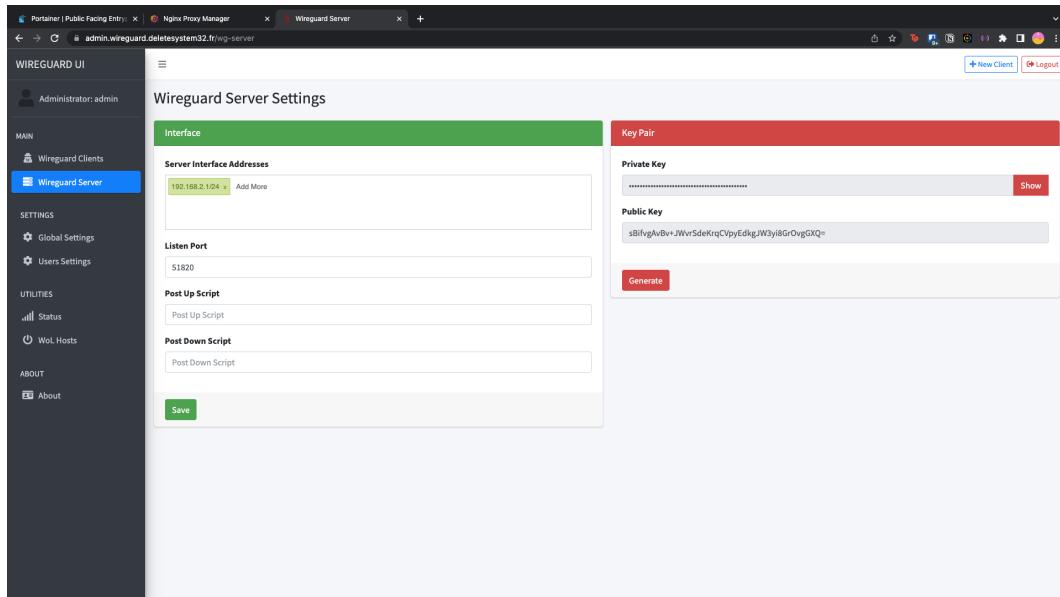
Ces commandes sont importantes pour la configuration d'un serveur VPN parce qu'elles permettent au serveur de relayer correctement le trafic entre le client VPN et le reste d'Internet.

Après avoir configuré les étapes précédentes, vous pouvez accéder à l'interface web d'administration du serveur VPN, WireGuard-UI. Pour cela, ouvrez votre navigateur et entrez l'adresse IP publique du serveur suivi du port 5000.

Une fois sur l'interface, vous serez invité à vous connecter. Les identifiants par défaut sont `admin` pour le nom d'utilisateur et `admin` pour le mot de passe. Il est recommandé de modifier ces identifiants par la suite pour des raisons de sécurité.

Une fois connecté, vous pourrez configurer le serveur WireGuard. Dans la barre de navigation à gauche, cliquez sur “Wireguard Server”. Dans la section “Server Interface Addresses”, vous pouvez spécifier le bloc CIDR du réseau pour vos clients. Par exemple, vous pouvez choisir le bloc CIDR `192.168.2.1/14`.

Après avoir saisi le bloc CIDR, cliquez sur “Save” pour enregistrer les modifications.



Lors de la configuration de la plage d'adresses IP pour les clients du serveur WireGuard, il est important de choisir une plage qui répond à vos besoins. Cependant, il est essentiel de comprendre que cette plage d'adresses IP est uniquement valide à l'intérieur du conteneur Docker de WireGuard.

Lorsque les clients interagissent avec d'autres services sur le même serveur, mais qui se trouvent dans leurs propres conteneurs, tels que NGINX Proxy Manager, le client sera identifié par l'adresse IP du conteneur Docker, par exemple `172.28.0.2`. En revanche, lorsque les clients interagissent avec des services hébergés sur un autre serveur de l'infrastructure, ils seront identifiés par l'adresse IP privée du serveur sur lequel se trouve le serveur VPN, c'est-à-dire

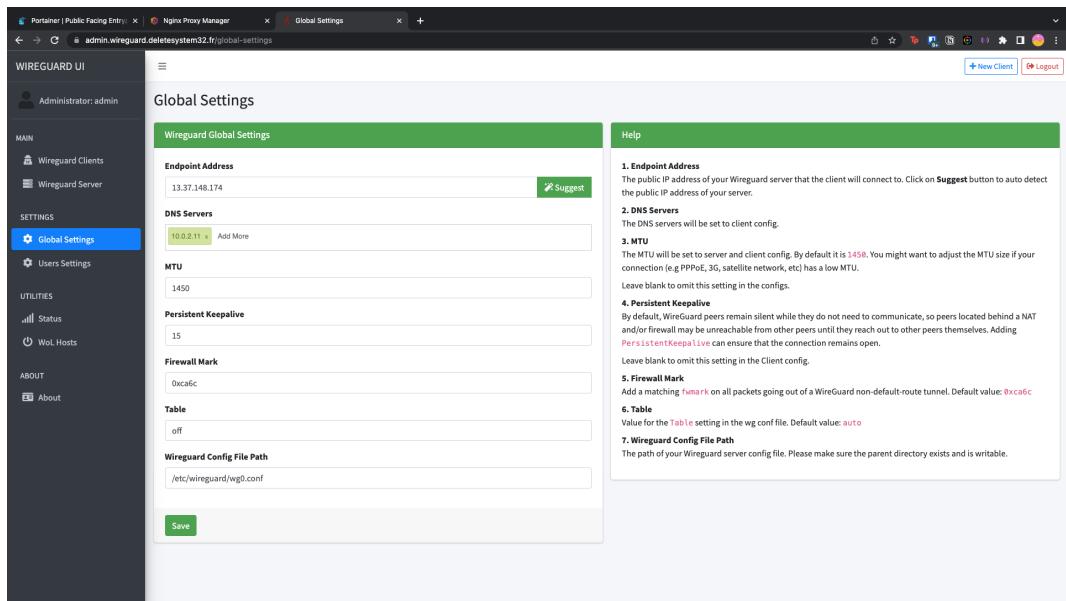
`cloudlab_public_facing_entrypoint` avec l'adresse IP `10.0.1.11`.

Il est important de prendre en compte cette distinction lors de la configuration et de l'utilisation du serveur VPN WireGuard. Les adresses

IP attribuées aux clients peuvent varier en fonction du contexte dans lequel ils interagissent avec les services de l'infrastructure.

Pour continuer la configuration du VPN, rendez-vous dans la barre de navigation à gauche de l'interface d'administration WireGuard-UI et cliquez sur "Global Settings". Dans la section "DNS Servers", saisissez l'adresse IP privée du serveur `cloudlab_internal_server_1`, qui est `10.0.2.11`. Ce serveur héberge notre serveur DNS Pi-hole. Ensuite, cliquez sur "Save" pour enregistrer les modifications.

Cette étape est importante car elle permet au serveur VPN WireGuard de rediriger les requêtes DNS vers le serveur DNS Pi-hole. Ainsi, toutes les requêtes DNS effectuées par les clients du VPN passeront par le serveur Pi-hole.



Maintenant que le serveur VPN WireGuard est configuré, vous pouvez créer des clients pour permettre à d'autres utilisateurs de se connecter au VPN. Pour ce faire, rendez-vous dans la barre de navigation à gauche de l'interface d'administration WireGuard-UI et cliquez sur "Wireguard Clients". Ensuite, cliquez sur le bouton "+ New Client" en haut à droite de la page.

Donnez un nom au client, cela peut être un identifiant ou un nom descriptif. Vous pouvez également renseigner son adresse e-mail si vous le souhaitez. La création d'une adresse IP pour le client est automatique, vous n'avez pas à vous en préoccuper.

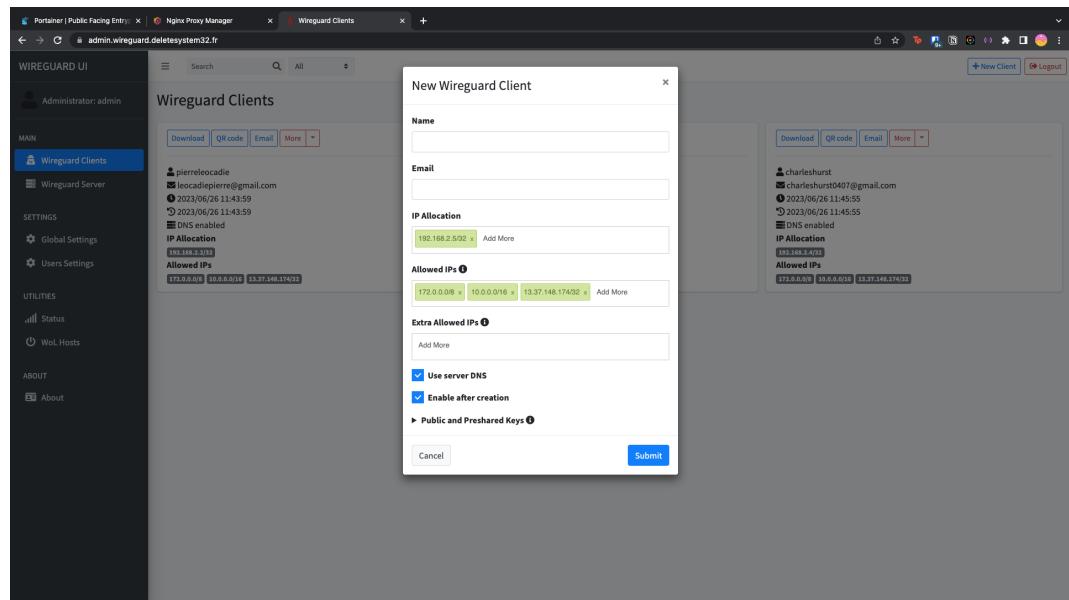
Dans le champ "Allowed IPs", vous devez spécifier les plages d'adresses IP auxquelles le client est autorisé à accéder. Pour cela, vous pouvez utiliser les blocs CIDR suivants : `172.0.0.0/8`, `10.0.0.0/16`. Ces plages d'adresses permettront au client de communiquer avec d'autres services du réseau interne.

Ensuite, ajoutez l'adresse IP publique du serveur

`cloudlab_public_facing_entrypoint` suivie de `/32`. Par exemple, si l'adresse IP publique est `13.37.148.174`, vous pouvez entrer `13.37.148.174/32`. Cela permettra au client d'accéder spécifiquement à ce serveur.

Une fois que vous avez renseigné toutes les informations nécessaires, cliquez sur "Save" pour créer le client.

Une fois que vous avez créé le client et finalisé la configuration du serveur VPN WireGuard, vous devez appliquer les modifications pour les rendre effectives. Pour cela, vous verrez apparaître un bouton "Apply Config" en haut à gauche de la page de l'interface d'administration WireGuard-UI. Cliquez sur ce bouton pour appliquer la configuration.



Maintenant que vous avez créé le client VPN dans WireGuard-UI, vous pouvez télécharger le fichier de configuration correspondant. Cliquez simplement sur le bouton "download" pour récupérer le fichier.



Pour utiliser ce fichier de configuration sur votre machine, vous devez d'abord télécharger et installer le client VPN WireGuard depuis le site officiel (<https://www.wireguard.com/install/>).

Une fois installé, ouvrez le logiciel client WireGuard sur votre machine et importez le fichier de configuration que vous avez téléchargé.

L'importation du fichier de configuration permettra à votre machine d'établir une connexion VPN avec le serveur WireGuard, en utilisant les paramètres spécifiques du client que vous avez créé. Cela vous permettra d'accéder aux ressources de l'infrastructure réseau depuis votre machine de manière sécurisée.

Après avoir configuré votre client VPN WireGuard et établi la connexion avec le serveur, vous pourriez constater que l'accès à Internet ne fonctionne pas. Cela est dû au fait que nous n'avons pas encore déployé le serveur DNS Pi-hole sur le serveur cloudbuild_internal_server_1, qui est utilisé dans notre configuration VPN.

La prochaine étape consistera donc à déployer le serveur DNS Pi-hole. Cela nous permettra d'avoir une résolution DNS personnalisée et efficace pour les requêtes Internet effectuées via le VPN. Grâce à Pi-hole, nous pourrons également mettre en place des filtres et des règles de blocage pour améliorer la sécurité et la confidentialité lors de la navigation.

Le déploiement du serveur DNS Pi-hole sera abordé plus en détail dans les étapes suivantes, afin de compléter la configuration de notre infrastructure réseau et de permettre un accès sécurisé et contrôlé à Internet via le VPN.

Pour le déploiement du serveur DNS Pi-hole, il suffit de suivre la même procédure que pour les autres services sauf que cette fois le déploiement doit se faire sur le serveur `cloudbuild_internal_server_1` veillez donc bien à sélectionner le bon serveur sur l'interface d'administration de Portainer. Rendez-vous dans le dossier `docker/` où vous trouverez les fichiers nécessaires. Ouvrez le fichier `docker-compose-pihole.yml` et copiez son contenu.

Effectuez la même procédure que pour le déploiement de NGINX Proxy Manager, WireGuard et WireGuard-UI.

Une fois que vous avez déployé Pi-hole, rendez-vous sur son interface d'administration web, ouvrez votre navigateur et saisissez l'IP publique du

serveur `cloudbot_internal_server_1` suivi du port 85 avec le chemin `/admin/` le mot de passe par défaut est `admin`. Vous devriez avoir l'interface suivante :



Pour déployer le serveur DNS Pi-hole, suivez la même procédure que pour les autres services en utilisant l'interface d'administration de Portainer. Cependant, cette fois-ci, assurez-vous de sélectionner le serveur `cloudbot_internal_server_1` lors du déploiement.

Pour cela, rendez-vous dans le dossier : `docker/` où vous trouverez les fichiers nécessaires. Ouvrez le fichier `docker-compose-pihole.yml` et copiez son contenu.

Ensuite, suivez la même procédure que pour le déploiement précédent en collant le contenu du fichier dans le web editor de l'interface d'administration de Portainer et en cliquant sur le bouton Deploy the stack pour lancer le déploiement de Pi-hole.

Une fois que Pi-hole est déployé avec succès, vous pouvez accéder à son interface d'administration web. Ouvrez votre navigateur et saisissez l'IP publique du serveur `cloudbot_internal_server_1` suivi du port 85 et ajoutez le chemin `/admin/`. Par exemple, `http://<IP_publique>:85/admin/`.

Vous serez redirigé vers l'interface d'administration de Pi-hole où vous pourrez configurer les paramètres avancés, les filtres et les règles de blocage en fonction de vos besoins spécifiques. Le mot de passe par défaut pour accéder à l'interface est `admin`.

Il reste un dernier service à déployer cette fois sur le serveur `cloudblock_public_facing_entrypoint`, il s'agit de GoAccess.

Pour déployer le service GoAccess sur le serveur `cloudblock_public_facing_entrypoint`, suivez la même procédure que pour les autres déploiements en utilisant l'interface d'administration de Portainer.

Pour configurer correctement le déploiement de GoAccess, suivez ces étapes simples :

1. Accédez au dossier `docker/` où se trouvent les fichiers nécessaires pour la configuration.
2. Ouvrez le fichier `docker-compose-goaccess.yml` avec un éditeur de texte.
3. Recherchez le mapping de volume pour définir le chemin d'accès aux logs de NGINX Proxy Manager.
4. Remplacez `/your/path/to/logs` par le chemin réel sur le serveur où NGINX Proxy Manager est exécuté. Pour trouver ce chemin, accédez à l'interface d'administration de Portainer et sélectionnez le serveur `cloudblock_public_facing_entrypoint`. Accédez aux informations du conteneur NGINX Proxy Manager, puis dans la section "Volumes" en bas de la page, copiez le chemin qui contient le dossier "data". Par exemple, le chemin pourrait être `/data/compose/5/data/logs`.
5. Remplacez `/your/path/to/` dans le mapping de volume du fichier de configuration de GoAccess par le chemin que vous avez copié. Par conséquent, le nouveau chemin sera `/data/compose/5/data/logs`.
6. Copiez l'intégralité du contenu du fichier `docker-compose-goaccess.yml`.

Ensuite, collez le contenu du fichier dans le web editor de l'interface d'administration de Portainer et cliquez sur le bouton Deploy the stack pour lancer le déploiement de GoAccess.

Ce service permettra de générer des rapports et des statistiques sur l'utilisation et l'accès à vos applications hébergées sur le serveur `cloudblock_public_facing_entrypoint`. Vous pourrez ainsi obtenir des informations détaillées sur les visiteurs, les pages consultées, etc.

Une fois que GoAccess est déployé avec succès, vous pouvez accéder à son interface via votre navigateur. Ouvrez votre navigateur et entrez l'IP publique du serveur `cloudblock_public_facing_entrypoint` suivi du port

spécifié dans le fichier de configuration (par exemple, http://<IP_publique>:7880).

Sur la page de connexion de GoAccess, utilisez les identifiants d'accès par défaut : `admin` pour le nom d'utilisateur et `admin` pour le mot de passe. Vous pouvez choisir de les modifier ultérieurement pour des raisons de sécurité.

Si vous souhaitez modifier les identifiants d'accès par défaut, vous pouvez le faire en modifiant la configuration de la stack de GoAccess sur Portainer. Une fois les modifications effectuées, assurez-vous de redéployer la stack pour que les nouvelles configurations prennent effet.

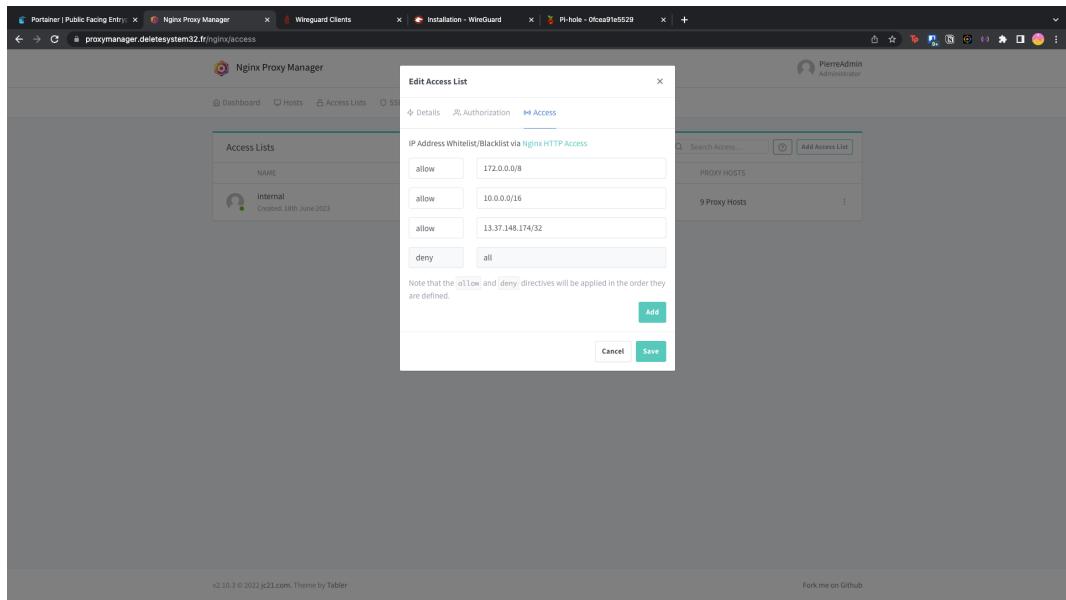
Maintenant que tous les services essentiels de notre infrastructure sont déployés, nous devons terminer la configuration de NGINX Proxy Manager. Pour cela, accédez à son interface d'administration via votre navigateur web.

Dans la barre de navigation de NGINX Proxy Manager, recherchez l'option "Access Lists" et cliquez dessus. Ensuite, cliquez sur le bouton "Add access list" pour créer une nouvelle liste d'accès.

Une fenêtre modale s'ouvrira où vous pourrez donner un nom à votre liste d'accès. Choisissez un nom significatif qui reflète le but de cette liste.

Une fois que vous avez créé la liste d'accès, rendez-vous dans l'onglet "Access" pour configurer les règles. Ajoutez des règles "Allow" pour les plages d'adresses IP suivantes : `172.0.0.0/8`, `10.0.0.0/16` et l'adresse IP publique du serveur cludlab_public_facing_entrypoint suivie de `/32`. Par exemple, si l'adresse IP publique est `13.37.148.174`, vous pouvez entrer `13.37.148.174/32`.

Ces règles permettent d'autoriser l'accès aux adresses IP spécifiées à travers NGINX Proxy Manager. Cela garantit que seules les adresses IP autorisées peuvent accéder aux services exposés par l'infrastructure.



Assurez-vous de sauvegarder les modifications une fois que vous avez configuré les règles d'accès. Cette étape est cruciale pour renforcer la sécurité de votre infrastructure et éviter tout accès non autorisé.

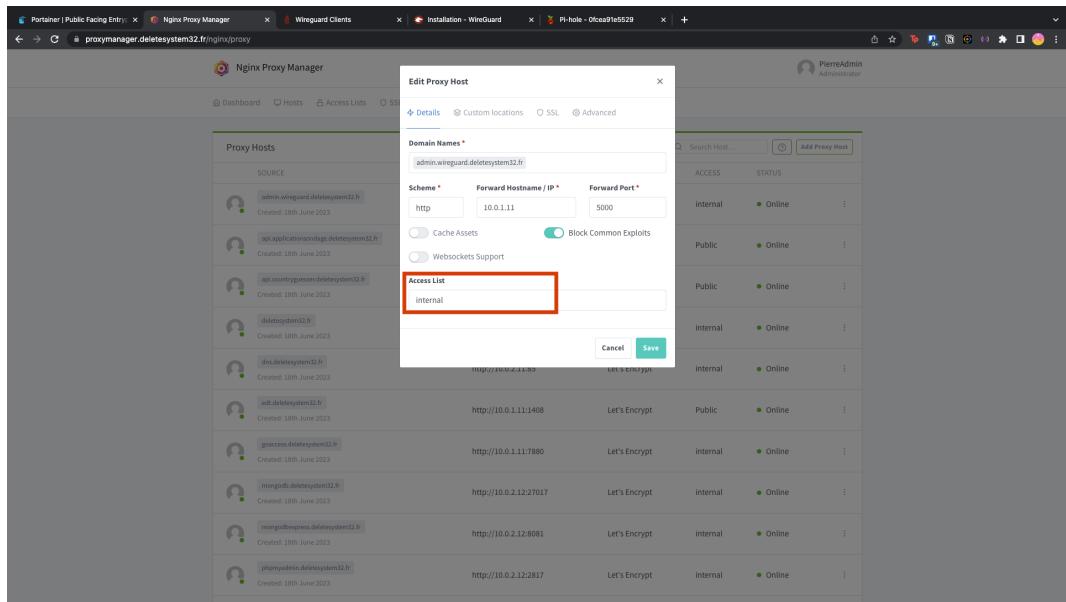
Il est recommandé de régulièrement revoir et mettre à jour les listes d'accès en fonction des besoins de sécurité de votre infrastructure.

Maintenant que vous avez créé votre liste d'accès, vous pouvez l'appliquer à vos Proxy Hosts pour contrôler l'accès à vos applications. Lorsque vous créez ou modifiez un Proxy Host dans NGINX Proxy Manager, vous verrez une option appelée "Access List".

Cliquez sur la liste déroulante et sélectionnez l'access list que vous avez précédemment créée. Cela permettra d'associer cette access list spécifique au Proxy Host en question.

En appliquant l'access list à un Proxy Host, vous définissez les règles d'accès qui seront utilisées pour contrôler qui peut accéder à cette application spécifique. Par exemple, si vous avez défini une règle "Allow" pour une certaine plage d'adresses IP dans votre access list, seules les adresses IP de cette plage seront autorisées à accéder à l'application associée au Proxy Host.

Cette fonctionnalité de contrôle d'accès basée sur les access lists vous permet de restreindre l'accès à vos applications à des utilisateurs spécifiques ou à des plages d'adresses IP approuvées. Cela renforce la sécurité de votre infrastructure en réduisant les risques d'accès non autorisé.



Maintenant que vous avez configuré votre access list, vous pouvez procéder à la création de vos Proxy Hosts pour les différents services de votre infrastructure. Assurez-vous de ne pas oublier cette étape si vous ne l'avez pas encore réalisée.

Lorsque vous créez un Proxy Host pour un service spécifique, vous avez la possibilité d'appliquer l'access list que vous avez précédemment créée. Cela vous permet de contrôler l'accès à ce service en fonction des règles définies dans l'access list.

Cependant, il est important de noter que si vous utilisez un Proxy Host pour le domaine associé à votre VPN WireGuard, il est essentiel de ne pas appliquer l'access list à ce Proxy Host. Si vous le faites, cela risque de bloquer l'accès au VPN et vous ne pourrez plus vous connecter.

La raison en est que le VPN WireGuard agit comme une passerelle d'accès à votre infrastructure depuis des emplacements externes. Si vous appliquez une access list restrictive à ce Proxy Host, cela empêchera les connexions VPN légitimes et rendra impossible l'établissement de connexions sécurisées.

Par conséquent, veillez à faire preuve de prudence lors de la configuration des Proxy Hosts et à ne pas appliquer l'access list au Proxy Host associé à votre VPN WireGuard. Cela garantira que vous pouvez toujours accéder à votre infrastructure de manière sécurisée via le VPN.

Concernant les instances EC2 `cloudlab_public_app_projects_server` et `cloudlab_internal_server_2`, elles ont été ajoutées à notre infrastructure

dans le cadre de nos projets spécifiques, notamment nos projets universitaires. Ces serveurs ne jouent pas de rôles critiques dans le fonctionnement global de l'infrastructure, mais ils offrent une flexibilité supplémentaire pour le déploiement de services supplémentaires.

Ces deux instances offrent donc une flexibilité supplémentaire en nous permettant de déployer des services et d'exécuter des projets spécifiques à notre environnement. Cependant, il est important de noter que ces serveurs ne sont pas essentiels au fonctionnement de base de notre infrastructure et peuvent être adaptés en fonction de nos besoins évolutifs.

▼ Étape 3 : Modification de l'infrastructure provisioning

Au cours de la configuration de notre infrastructure, nous avons utilisé le bloc CIDR `0.0.0.0/0` dans les groupes de sécurité pour faciliter la mise en place de l'architecture. Cependant, maintenant que nous avons terminé la configuration, il est important de prendre des mesures de sécurité supplémentaires en supprimant ce bloc CIDR des autorisations accordées par les groupes de sécurité.

Pour effectuer cette modification, nous allons accéder au dossier `terraform/` et ouvrir les fichiers `3a1_subnet_configuration.tf` et `3b1_subnet_configuration.tf`. Dans ces fichiers, nous trouverons les sections concernant les groupes de sécurité. Nous devrons supprimer la mention du bloc CIDR `0.0.0.0/0` des autorisations de chaque groupe de sécurité.

Une fois que nous avons effectué ces modifications, nous pouvons exécuter les commandes Terraform suivantes :

- `terraform plan` : Cette commande nous permettra de voir les modifications qui seront appliquées à notre infrastructure.
- `terraform apply` : En exécutant cette commande, les modifications seront appliquées aux groupes de sécurité de notre infrastructure.

Ces étapes garantiront que nous avons pris les mesures de sécurité appropriées en restreignant l'accès à notre infrastructure uniquement aux adresses IP spécifiques autorisées. Cela renforce la sécurité de notre environnement et réduit les risques potentiels liés à des accès non autorisés.

▼ Conclusion

En conclusion, ce rapport met en évidence notre travail sur la conception et le déploiement d'une architecture et d'une infrastructure réseau sur AWS. Nous avons réussi à créer un environnement cloud solide, sécurisé et évolutif, permettant le déploiement et la gestion efficace de différents services pour un faible budget.

Au cours de ce projet, nous avons utilisé des technologies telles que Docker, Terraform, Ansible et les services AWS pour créer une infrastructure robuste. Nous avons déployé des services essentiels tels que NGINX Proxy Manager, WireGuard, PiHole et GoAccess, en suivant des procédures détaillées et en veillant à la bonne configuration de chaque composant.

La sécurité a été une préoccupation majeure tout au long du processus de déploiement. Nous avons mis en place des mesures de sécurité telles que la configuration des groupes de sécurité, l'application d'access lists et la restriction des accès non autorisés. Cela garantit que notre infrastructure est protégée contre les menaces potentielles.

Ce projet nous a permis de mieux comprendre les avantages et les défis liés à la mise en place d'une architecture et d'une infrastructure réseau sur AWS. Nous avons acquis une expérience précieuse dans l'utilisation des outils et des services cloud, ainsi que dans la configuration et la gestion des différentes composantes de l'infrastructure.

En conclusion, notre projet a abouti à la création d'une architecture et d'une infrastructure réseau AWS performantes et sécurisées. Nous sommes convaincus que notre approche et les connaissances acquises seront bénéfiques pour d'autres personnes souhaitant développer leur propre infrastructure cloud.