R4-Cyber-11 - Sécurisation de services réseaux

HAproxy

Etudiant

GRONDIN Benjamin

Promotion

BUT 2 TP 1 Réseaux et télécommunications

Professeur

REPUSSEAU Willy

Table des matières

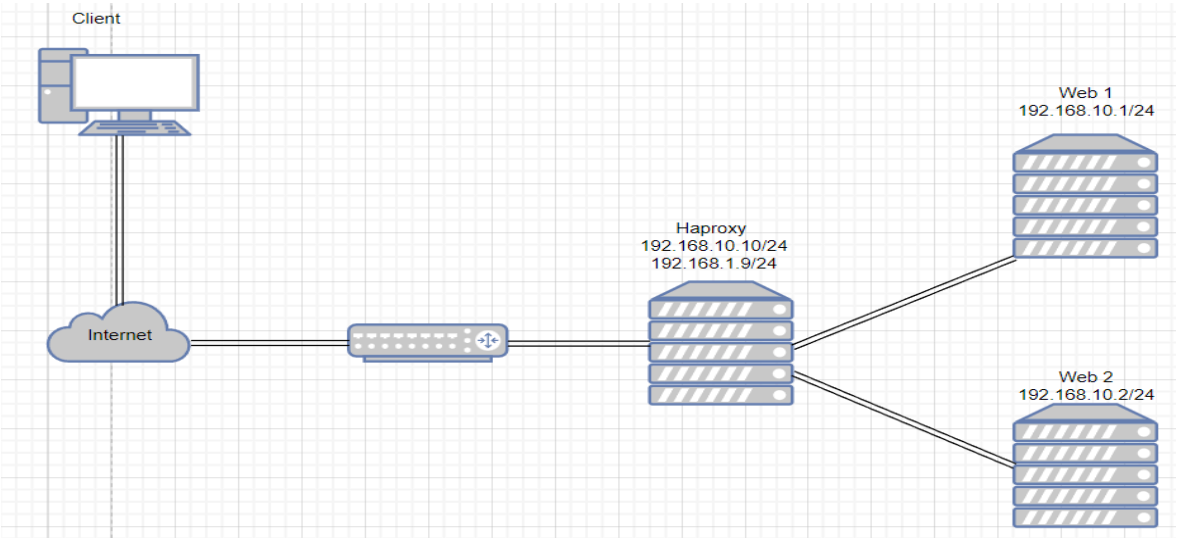
Introduction

L'objectif de ce TP est de mettre en place un équilibrage de charge avec HAProxy pour distribuer les requêtes entre plusieurs serveurs web Apache. L'infrastructure inclut la configuration de HTTPS, la redirection automatique de HTTP vers HTTPS et l'activation des statistiques HAProxy. Une analyse de la communication TLS est également effectuée.

Plan d'adressage

Machines	Adresses IP
Serveur web 1	192.168.10.1/24
Serveur web 2	192.168.10.2/24
Load balancer Haproxy	192.168.1.9/24
	192.168.10.10/24

Schéma réseau



1.Installation d'Apache

1.1 - Choix du serveur

Pour débuter ce TP sur HAProxy, il faut d'abord une distribution Linux pour héberger les sites web. J'ai donc choisi **Ubuntu**, car j'ai déjà installé et configuré des serveurs de répartition de charge (load balancing) sur ce système par le passé.

1.2 - Installation d'Apache

Apache HTTP Server, ou simplement **Apache**, est l'un des serveurs web les plus connus et les plus utilisés au monde. Gratuit et **open-source**, il est à la fois puissant et très flexible. Son rôle principal ? Gérer et diffuser des sites web, des applications et du contenu en ligne via le protocole **HTTP**. Il s'adapte à différents systèmes d'exploitation comme **Linux, Windows et macOS**, ce qui le rend accessible et pratique pour de nombreux utilisateurs.

Pour installer un serveur apache sur une distribution ubuntu, il faut taper les commandes suivantes :

```
#sudo apt update
#sudo apt upgrade
#sudo apt install apache2
```

Avant d'installer Apache lui-même, il est important de s'assurer que le système d'exploitation est à jour en exécutant les deux commandes précédentes. La commande *apt update* permet au système de récupérer la liste des dernières versions des paquets disponibles. Ensuite, la commande *apt upgrade* met à jour les paquets existants en installant les versions les plus récentes détectées grâce à la mise à jour précédente.

1.3 - Vérification du port d'écoute

Le port d'écoute est un point d'entrée pour les requêtes. Ainsi, lorsqu'un utilisateur souhaite accéder à une page web ou à un service web, l'adresse IP sera associée à un port afin de permettre l'échange des requêtes. Dans le cadre d'un serveur Apache, il est important de vérifier que le *port 80* est bien actif en utilisant la commande *netstat -ntl*. Cette vérification permet de s'assurer qu'Apache est correctement configuré et en écoute sur ce port, ce qui est essentiel pour recevoir des requêtes HTTP. Si le *port 80* n'est pas ouvert ou occupé par un autre service, cela peut empêcher l'accès au site web hébergé.

Exemple :

```
tcp        0      0 192.168.1.9:80          0.0.0.0:*               LISTEN
```

Cette capture d'écran a été prise sur mon serveur de répartition de charge (load balancing), car les configurations pour HTTPS ont déjà été effectuées sur mon serveur Apache. Néanmoins, elle peut être utilisée pour illustrer cette partie.

On peut voir qu'après *TCP*, qui est le protocole utilisé par Apache pour les communications, apparaît l'*adresse IP de la machine*, suivie de *:80*,

ce qui indique que le port d'écoute est bien défini. Enfin, la ligne se termine par *LISTEN*, ce qui confirme que la machine est en écoute sur le port *80*, permettant ainsi de recevoir des requêtes *HTTP*.

2.Modification de la page d'accueil du site

À présent, nous allons modifier la page web de notre serveur Apache afin de la rendre identifiable lorsque le load balancing sera opérationnel. Pour ce faire, nous accédons au fichier *000-default.conf*, situé dans /etc/apache2/sites-available/.

Quel(s) fichier(s) ce dossier contient-il ?

Le fichier *000-default.conf*, situé dans */etc/apache2/sites-available/*, définit la configuration par défaut d'Apache.
La directive *DocumentRoot* indique le dossier où sont stockés les fichiers du site web, généralement */var/www/html* sur Ubuntu et Debian.
Ce dossier contient par défaut un fichier *index.html*, qui sert de page de test après l'installation d'Apache.

Page d'accueil du serveur web 1

3.Sécurisation d'un site en https

Pour garantir que les données échangées entre le client et le serveur ne puissent être lues ou altérées par des tiers, il est essentiel d'utiliser HTTPS afin de chiffrer les informations. De plus, bien que l'ANSSI ne rende pas HTTPS obligatoire pour tous les sites, son utilisation est fortement recommandée, notamment pour protéger les données sensibles et se conformer aux bonnes pratiques de sécurité. Par ailleurs, certaines réglementations, comme le RGPD, exigent des mesures de protection adaptées, et HTTPS est une solution standard pour sécuriser les échanges de données.
Dans ce TP, nous aborderons ce principe de chiffrement afin d'acquérir les bonnes pratiques. Cependant, les sites nécessitant un chiffrement des échanges devront obtenir un certificat reconnu délivré par un organisme de confiance. Ici, nous opterons pour un certificat auto-signé.

3.1 - Génération d'un certificat auto-signé

Pour générer un certificat auto-signé il faut taper les commandes suivantes sur le serveur web :

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -out
/etc/apache2/server.pem -keyout /etc/apache2/server.key
```

Le fichier *server.pem* contient le certificat du serveur, tandis que *server.key* stocke la clé privée associée. Ensemble, ils permettent d'établir une connexion sécurisée via HTTPS en assurant le chiffrement et l'authentification des échanges.

3.2 - Création d'un site sécurisé et activation de https

À présent, je vais créer un nouveau dossier *html_ssl*, qui contiendra le fichier *index.html*, accessible par les clients via des requêtes *HTTPS*.

Création du dossier *html_ssl* et du fichier *index.html* :

```
mkdir html_ssl
cd html_ssl
nano index.html
```

Maintenant, nous pouvons activer les connexions *HTTPS*. Pour ce faire, nous utilisons la commande *sudo a2enmod ssl*, qui active le module SSL d'Apache afin de permettre les connexions sécurisées. Ensuite, la commande *sudo a2ensite default-ssl active* la configuration du site sécurisé. Enfin, un redémarrage d'Apache est nécessaire pour appliquer les modifications.

3.3 - Configuration du site sécurisé

Maintenant nous pouvons modifier le fichier *default-ssl.conf* afin de modifier les informations pour que la page affichée soit celle en HTTPS.

Configuration :

```
<VirtualHost *:443>
  DocumentRoot /var/www/html_ssl
  ServerName www.site-de-benjamin.com
  SSLEngine on
  SSLCertificateFile /etc/apache2/server.pem
  SSLCertificateKeyFile /etc/apache2/server.key
</VirtualHost>
```

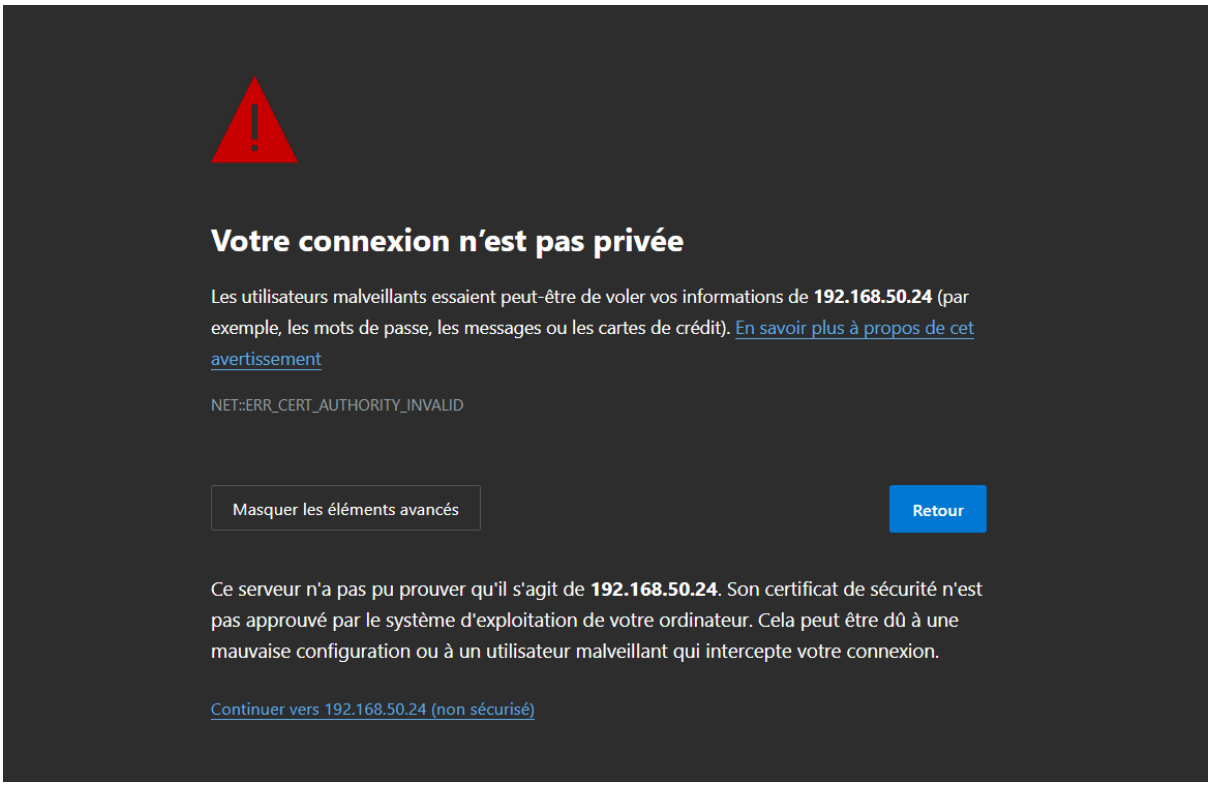
Pourquoi n'est-il pas nécessaire de modifier le fichier ports.conf ?

Il n'est pas nécessaire de modifier le fichier *ports.conf* car ce fichier est déjà configuré pour permettre l'écoute des connexions HTTPS sur le *port 443*, qui est le port standard utilisé pour les communications sécurisées avec *SSL/TLS*.

3.4 - Test

Puisque le site écouté désormais sur le port 443 pour HTTPS, ont peut vérifier que celui-ci est bien actif comme dans partie 1.3.

Page web :



Le message d'erreur affiché par le navigateur indique que le certificat SSL utilisé n'est pas signé par une autorité de certification reconnue (NET::ERR_CERT_AUTHORITY_INVALID), ce qui est courant pour un certificat auto-signé. Pour accéder au site, il faut accepter une exception de sécurité dans le navigateur. Afin de rendre le site digne de confiance, il faudrait utiliser un certificat signé par une autorité de certification reconnue comme Let's Encrypt ou DigiCert.L'algorithme de chiffrement symétrique utilisé pour le Record Protocol est généralement AES ou ChaCha20, avec une clé secrète de longueur variable. L'échange de la clé secrète se fait souvent via des algorithmes comme RSA, Diffie-Hellman ou ECDHE.

4. Mise en place de HD de HTTP

Pour assurer une *haute disponibilité (HA)*, il faut au moins deux serveurs qui hébergent les mêmes services. Pour cela, j'ai cloné la première machine et modifié la page web afin que l'utilisateur sache qu'il est sur le serveur web 2. Grâce à HAProxy, on peut répartir le trafic

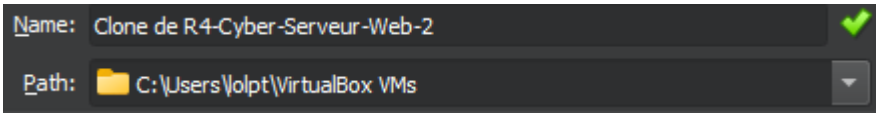
entre ces serveurs, ce qui permet de garantir un service toujours accessible, même en cas de panne d'un serveur. Cela améliore aussi la performance et l'expérience utilisateur en évitant la surcharge d'un seul serveur.

4.1 - Installation d'un second serveur Apache

Pour obtenir un serveur avec les mêmes configurations que le précédent, vous pouvez le cloner sous VirtualBox en suivant les étapes ci-dessous :

Faites un clic droit sur la machine -> puis cliquez sur Cloner.

Ensuite, choisissez le nom de la machine ainsi que le dossier de stockage.



Pour finir, dans le bandeau déroulant, choisissez le paramètre ci-dessous afin d'éviter les conflits d'adressage entre les machines sur le réseau.

4.2 - Installation de Haproxy

Pour installer HAProxy, j'ai utilisé une troisième machine Ubuntu, qui servira de load balancer.

Pour installer HAProxy, exécutez la commande suivante sur votre machine Ubuntu :

```
#sudo apt-get update
#sudo apt-get upgrade
#sudo apt-get install haproxy
```

4.3 - Configuration de Haproxy

Pour que mon haproxy puisse fonctionner voici les paramètre que j'ai ajouté dans le fichier /etc/haproxy/haproxy.cfg :

```
frontend http_front
    bind 192.168.1.9:80
    bind 192.168.10.10:80
```

Dans le fichier haproxy.cfg, les lignes suivantes configurent le frontend nommé http_front. Ces lignes indiquent à HAProxy d'écouter les connexions HTTP sur deux adresses IP différentes (192.168.1.9:80 et 192.168.10.10:80), chacune sur le port 80. Cela permet à HAProxy de recevoir les requêtes HTTP provenant de ces deux interfaces réseau et de les acheminer vers les serveurs backend configurés.

```
backend web_servers
    server web1 192.168.10.1:80 cookie serveur1 check inter 500 rise 2 fall 3
    server web2 192.168.10.2:80 cookie serveur2 check inter 500 rise 2 fall 3
```

Dans la configuration HAProxy, ces lignes définissent un backend nommé web_servers, où deux serveurs web (web1 et web2) sont spécifiés avec leurs adresses IP respectives (192.168.10.1 et 192.168.10.2). Les paramètres cookie, check, inter, rise, et fall définissent des options de suivi de session et de santé pour ces serveurs, assurant une répartition du trafic fiable.

Les paramètres maxconn, timeout (connect, client, server), balance roundrobin et check dans HAProxy permettent de limiter les connexions simultanées, définir les délais d'attente pour la connexion et la communication, répartir le trafic de manière circulaire entre les serveurs, et vérifier la disponibilité des serveurs backend.
Pour finir, il faut aussi configurer les interfaces réseau sur les serveurs web afin d'avoir une adresse IP statique. Pour ce faire, suivez ces étapes :

1.Modifier le fichier de configuration netplan sur les serveur web

Pour commencer, rendez-vous dans */etc/netplan/*. Ensuite, avec nano, modifiez le fichier de configuration *50-cloud-init.yaml* en ajustant les lignes pour qu'elles correspondent à celles-ci :

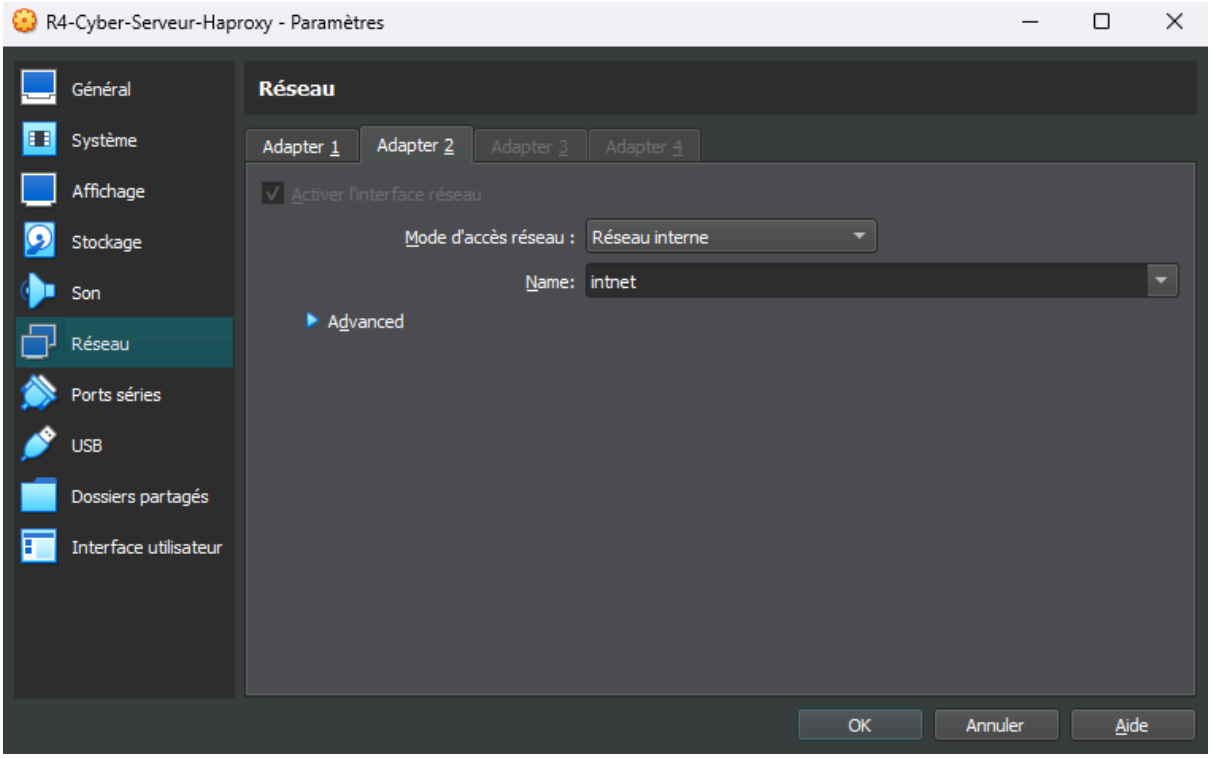
Dans ce fichier, on spécifie l'adresse IP statique du serveur web. Ici, il s'agit du serveur web 1. Il faut appliquer la même configuration pour le serveur web 2 et pour le serveur HAProxy. Une fois les modifications effectuées, quittez le fichier et tapez *netplan try*, puis appuyez sur Entrée si la configuration est correcte.

2.Modifier le fichier de configuration netplan sur les serveur Haproxy

Pour que le serveur HAProxy fonctionne correctement, il faut deux cartes réseau. L'une des cartes sera accessible depuis l'extérieur pour les machines envoyant leurs requêtes, et la deuxième servira d'intermédiaire entre les serveurs web et le réseau internet.

Pour ce faire, il faut ajouter une carte réseau dans virtualbox :

Cliquer sur la machine cible -> Configuration -> Réseau.



Puis, ajoutez une interface réseau en cliquant sur "Activer l'interface réseau" et choisissez le paramètre "Réseau interne" dans le menu déroulant. Une fois cela fait, vous pouvez aller dans *netplan* et modifier l'adresse IP comme précédemment.

4.4 - Vérification de l'accès au sites

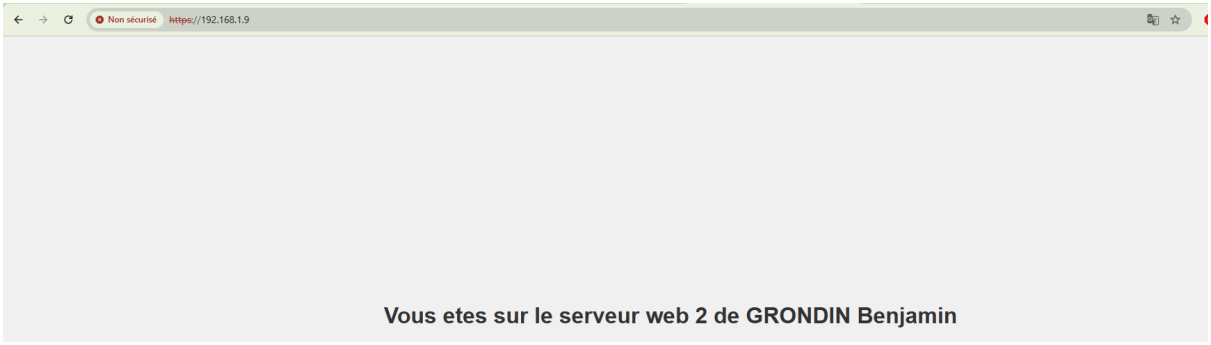
Dès qu'un utilisateur souhaite se connecter au site, il doit maintenant taper l'adresse IP du load balancer, qui se chargera de répartir les requêtes en fonction de la disponibilité des serveurs.

Page web :

Puisque le serveur web 1 est accessible je vais désactiver l'interface réseau sur ce serveur web pour voir si la répartition de charge fonctionne.

Je désactive l'interface enp0s3 sur le serveur 1.

```
root@ubuntu:/# ip link set enp0s3 down
```



Ainsi, nous avons la preuve que le load balancer fonctionne, car roundrobin répartit le trafic de manière équitable entre les serveurs, envoyant chaque nouvelle requête à un serveur différent dans la liste, ce qui permet à chaque serveur de recevoir une part égale du trafic au fil du temps.

Après le redémarrage de l'interface enp0s3 sur le serveur 1, le site 1 sera de nouveau opérationnel. Ici, lorsque je vais redémarrer le site, il restera sur le même serveur car les cookies sont déjà activés.

Cependant, si les cookies ne sont pas encore paramétrés, les pages hébergées sur les deux serveurs devraient s'afficher alternativement lors du rafraîchissement de la page. J'expliquerai comment activer les cookies dans l'étape suivante.

4.5 - Persistance basée sur des cookies

La persistance basée sur des cookies (aussi appelée stickiness ou affinité de session) permet de garantir qu'un utilisateur soit toujours dirigé vers le même serveur backend pendant toute la durée de sa session. Dans cette section, nous allons explorer comment configurer HAProxy pour utiliser les cookies comme mécanisme de persistance, assurant ainsi une expérience utilisateur cohérente, même en présence de plusieurs serveurs backend.

Pour configurer les cookies, allez dans */etc/haproxy/haproxy.cfg* et ajoutez les lignes suivantes :

```
backend web_servers
    cookie SERVERUSED insert indirect nocache
    server web1 192.168.10.1:80 cookie serveur1 check inter 500 rise 2 fall 3
    server web2 192.168.10.2:80 cookie serveur2 check inter 500 rise 2 fall 3
```

Dans ces lignes, le paramètre cookie SERVERUSED permet à HAProxy de définir un cookie qui assure la persistance de la session entre le client et le serveur backend. Le cookie est ajouté dans les réponses des serveurs, avec les options insert, indirect et nocache pour garantir qu'il soit bien envoyé et non mis en cache. Ensuite, pour chaque serveur backend (web1 et web2), l'option cookie serveur1 et cookie serveur2 attribue un identifiant unique, permettant de maintenir la session de l'utilisateur sur le même serveur pendant toute la durée de sa connexion.

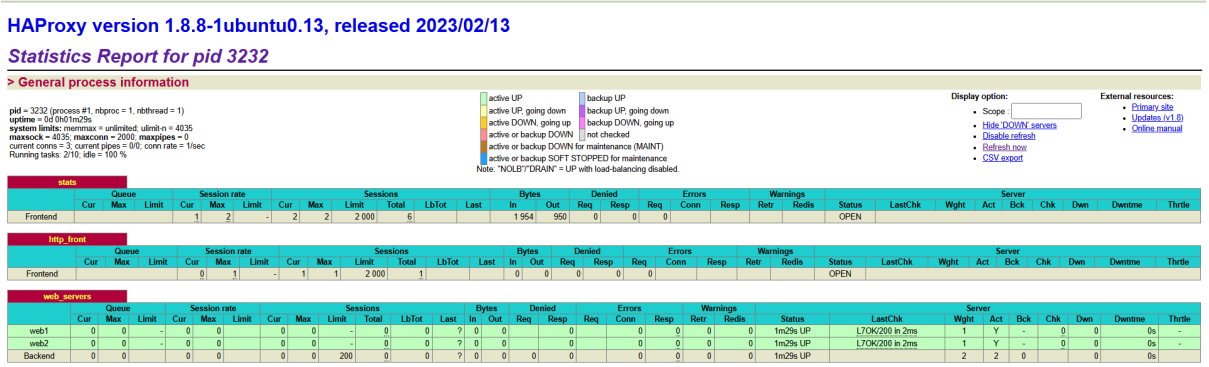
4.6 - Visualisation des statistiques

Pour pouvoir visualiser la statique d'utilisation du load balancer, on peut activer la page stats dans les configurations en ajoutant les lignes suivantes :

```
frontend stats
    bind 192.168.1.9:8080
    stats uri /stats
    stats auth admin:password
```

Ces lignes configurent un frontend stats qui écoute sur l'adresse 192.168.1.9:8080, permet d'accéder aux statistiques via l'URL /stats, et protège l'accès avec une authentification admin:password.

Page web :



5. Mise en place de la HD de HTTPS

Afin de pouvoir échanger des informations en HTTPS entre le serveur HAProxy et les clients, nous devons mettre en place une sécurisation

HTTPS via un certificat auto-signé. Pour ce faire, nous allons installer le certificat en utilisant la commande suivante :

5.1 - Configuration et vérification

```
sudo openssl req -x509 -newkey rsa:2048 -keyout /etc/ssl/certs/haproxy.key
-out /etc/ssl/certs/haproxy.crt -days 365 -nodes
```

Cette commande permet de générer un certificat auto-signé RSA de 2048 bits, avec une clé privée stockée dans /etc/ssl/certs/haproxy.key et un certificat public dans /etc/ssl/certs/haproxy.crt, valable pendant 365 jours. L'option -nodes empêche la création d'un mot de passe pour la clé privée.

Ensuite, il faut copier le certificat et la clé privée dans le même fichier. La commande présentée dans le TP ne fonctionnait pas pour moi, donc j'ai utilisé cette commande :

```
sudo cat /etc/ssl/certs/server.crt /etc/ssl/private/server.key >
/etc/ssl/certs/haproxy.pem
```

Pour finir, ajoutez la ligne de code suivante dans *haproxy.cfg* afin que les utilisateurs puissent se connecter via HTTPS :

```
bind 192.168.1.9:443 ssl crt /etc/ssl/certs/haproxy.pem
```

Cette ligne configure HAProxy pour écouter sur l'adresse 192.168.1.9 et le port 443 en HTTPS, en utilisant le certificat haproxy.pem pour sécuriser les connexions. De plus, on à vu précédemment que le serveur écoute bien sur le port 443 et que l'équilibre de charge est fonctionnel.

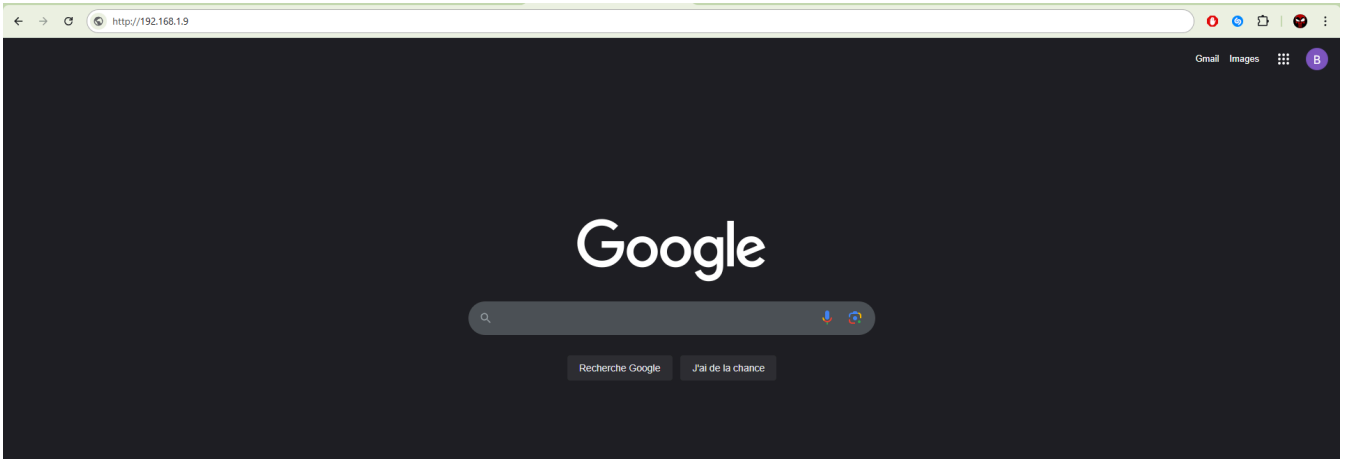
5.2 - Configuration supplémentaire

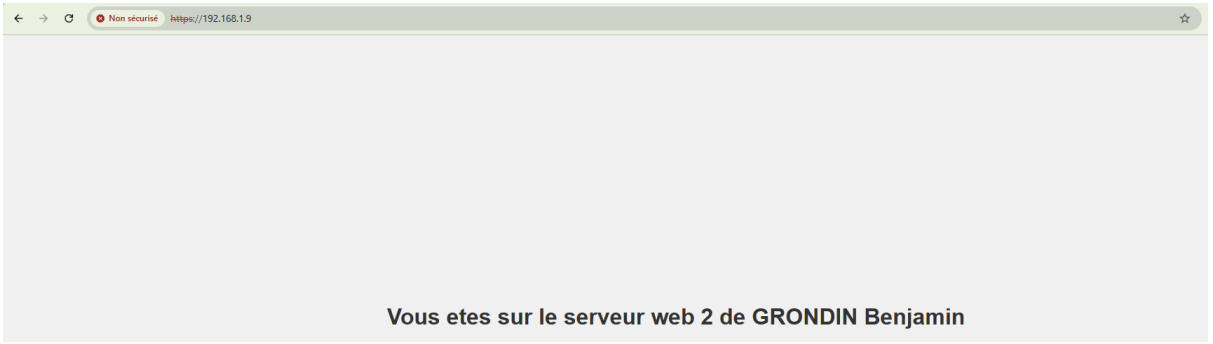
Afin de garantir une sécurité supplémentaire pour l'utilisateur, on peut rediriger toutes les requêtes HTTP vers HTTPS. Pour ce faire, il faut ajouter cette ligne dans *haproxy.cfg* :

```
http-request redirect scheme https if !{ ssl_fc }
```

La ligne ci-dessus redirige toutes les requêtes HTTP vers HTTPS si la connexion n'est pas déjà sécurisée en SSL/TLS.

Exemple :





Conclusion

Ce TP nous a permis de mettre en place un équilibrage de charge avec HAProxy et d'expérimenter différentes techniques pour assurer la disponibilité, la répartition du trafic et la sécurisation des connexions web. Nous avons appris à optimiser la disponibilité d'un site en répartissant la charge entre plusieurs serveurs, à rediriger automatiquement le trafic HTTP vers HTTPS pour garantir une connexion sécurisée, ainsi qu'à gérer des certificats SSL en imposant l'utilisation de TLS 1.2 au minimum.

Nous avons également découvert comment superviser HAProxy à l'aide de son interface de statistiques et analyser les échanges réseau avec Wireshark, notamment en identifiant les différentes étapes du handshake TLS. Ce projet nous a permis de comprendre l'importance de HAProxy dans les architectures modernes et de manipuler des concepts essentiels en réseau et en cybersécurité. Les compétences acquises au cours de ce TP seront précieuses pour la mise en place et la sécurisation de services web en production.