

Sécurité des réseaux avancés

Par Mathis Boultonneau et Ronan Renoux.

Prise en main de l'environnement de tests

Q1. Combien de réseaux sont actuellement créés ? Quels sont les différents types de réseaux listés par docker ?

```
sudo docker network ls
```

Il y a 4 réseaux listés :

Nom du réseau	Type	Détail
bridge	bridge	Réseau par défaut. Réseau isolé de l'hôte mais accessible depuis les conteneurs dans le même réseau.
exo1_net_pub	bridge	Réseau isolé de l'hôte mais accessible depuis les conteneurs dans le même réseau.
host	host	Même réseau que l'hôte sans aucune couche d'isolation
none	null	Réseau spécifique à chaque conteneur complètement isolé de l'hôte et des autres conteneurs

Q2. Quelle est l'adresse IP de la Busybox ?

```
sudo docker inspect --type=network exo1_net_pub
```

L'adresse IP de la Busybox est 192.168.20.2/24.

Le réseau exo1_net_pub est sur l'adresse IP 192.168.20.0/24.

Nom du conteneur	Adresses IP
exo1_web1_1	192.168.20.11/24
exo1_web2_1	192.168.20.12/24
exo1_busy1_1	192.168.20.2/24

```
sudo docker exec -it exo1_busy1_1 sh
ifconfig
```

Avec la commande *ifconfig*, nous avons bien l'ip du conteneur exo1_busy1_1 qui est 192.168.20.2.

Q3. Que constatez-vous ? Quelle fonctionnalité est mise en avant en faisant ce test ?

Toujours depuis la CLI du conteneur Busybox, nous lançons des pings vers les conteneurs *web1* et *web2* avec leurs IP et noms :

```
# Web1
ping web1
ping 192.168.20.11
```

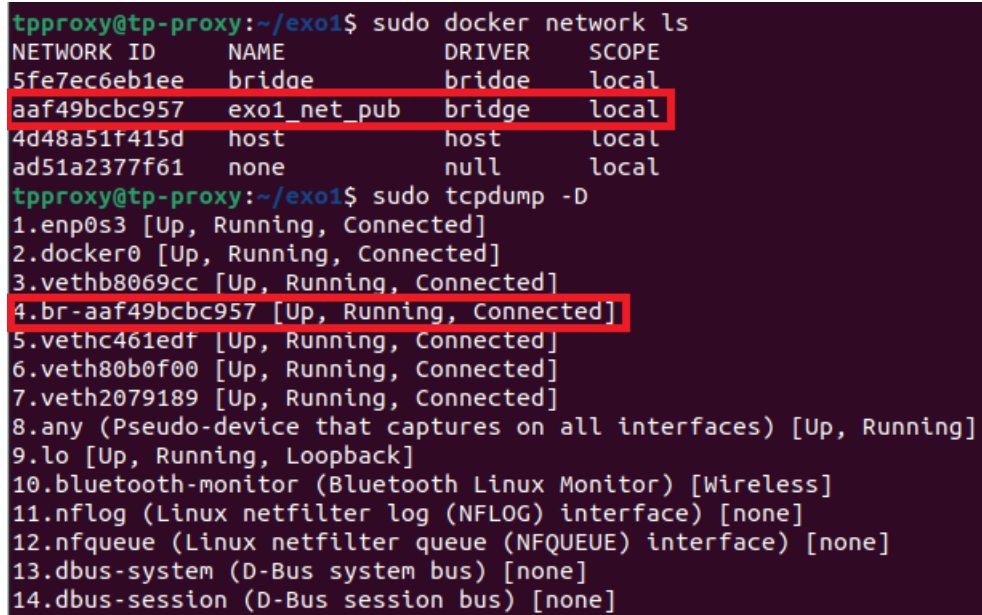
```
# Web2
ping web2
ping 192.168.20.12
```

Nous constatons que le ping fonctionne que ça soit avec les adresses IP ou avec les noms. Docker propose un DNS qui par défaut lie les noms des conteneurs à leur IP (au sein du même réseau).

Q4. A partir de l'inspection des streams TCP/HTTP, quelles sont les informations notables dans les entêtes HTTP échangées avec les deux serveurs web ? Est-ce qu'il y a une différence entre les deux serveurs Web ? Quels sont les codes de retours HTTP et qu'indiquent-ils ? Quelle est la réponse bonus ?

Nous regardons la liste des interfaces réseaux disponibles et la comparons avec la liste des réseaux Docker :

```
sudo docker network ls
sudo tcpdump -D
```



```
tpproxy@tp-proxy:~/exo1$ sudo docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
5fe7ec6eb1ee        bridge             bridge              local
aaf49bcb957         exo1_net_pub       bridge              local
4d48a51f415d        host               host                local
ad51a2377f61        none               null                local
tpproxy@tp-proxy:~/exo1$ sudo tcpdump -D
1.enp0s3 [Up, Running, Connected]
2.docker0 [Up, Running, Connected]
3.vethb8069cc [Up, Running, Connected]
4.br-aaf49bcb957 [Up, Running, Connected]
5.vethc461edf [Up, Running, Connected]
6.veth80b0f00 [Up, Running, Connected]
7.veth2079189 [Up, Running, Connected]
8.any (Pseudo-device that captures on all interfaces) [Up, Running]
9.lo [Up, Running, Loopback]
10.bluetooth-monitor (Bluetooth Linux Monitor) [Wireless]
11.nflog (Linux netfilter log (NFLOG) interface) [none]
12.nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
13.dbus-system (D-Bus system bus) [none]
14.dbus-session (D-Bus session bus) [none]
```

Figure 1: Liste des réseaux Docker et tcpdump

Nous pouvons lancer une capture réseau sur l'interface réseau qui correspond (en comparant les identifiants de réseau) et l'enregistrer dans le fichier `dump.pcap` (disponible dans `getting_started/files/dump.pcap`).

```
sudo tcpdump -w dump.pcap -i br-aaf49bcb957
```

Nous analysons la capture d'écran avec Wireshark :

- Pour chaque site, le client envoie une requête avec la protocole et version (HTTP 1.1), la page demandé (/), la méthode (GET), l'user Agent, etc.
- Dans la réponse, le serveur renvoie la date, la date de modification, le code HTTP (ici 200), le protocole et version utilisé (HTTP 1.1), le type et la version du serveur, etc.

Le serveur *web1* est un serveur Nginx (1.25.3) tandis que le serveur *web2* est un serveur Apache (2.4.58).

Les codes de retours HTTP sont :

- 200 : la ressource est bien trouvé
- 404 : la ressource n'est pas trouvé (le navigateur par défaut demande `favicon.ico` qui n'est pas sur le serveur)

La réponse bonus est un kangouru (à la question `Who is Skippy?`).

3	0.000139	192.168.20.1	192.168.20.11	TCP	66 56968 → 80 [ACK] Seq=1 Ack=1 Wi
4	0.000322	192.168.20.1	192.168.20.11	HTTP	423 GET / HTTP/1.1
5	0.000347	192.168.20.11	192.168.20.1	TCP	66 80 → 56968 [ACK] Seq=1 Ack=358
6	0.000652	192.168.20.11	192.168.20.1	TCP	304 80 → 56968 [PSH, ACK] Seq=1 Ack
7	0.000702	192.168.20.1	192.168.20.11	TCP	66 56968 → 80 [ACK] Seq=358 Ack=23
8	0.000747	192.168.20.11	192.168.20.1	HTTP	724 HTTP/1.1 200 OK (text/html)
9	0.000754	192.168.20.1	192.168.20.11	TCP	66 56968 → 80 [ACK] Seq=358 Ack=89
10	0.040795	192.168.20.1	192.168.20.11	HTTP	376 GET /favicon.ico HTTP/1.1
11	0.040924	192.168.20.11	192.168.20.1	HTTP	374 HTTP/1.1 404 Not Found (text/h
12	0.040934	192.168.20.1	192.168.20.11	TCP	66 56968 → 80 [ACK] Seq=668 Ack=12
13	1.365881	192.168.20.1	192.168.20.12	TCP	74 42582 → 80 [SYN] Seq=0 Win=6424
14	1.365968	192.168.20.12	192.168.20.1	TCP	74 80 → 42582 [SYN, ACK] Seq=0 Ack
15	1.365987	192.168.20.1	192.168.20.12	TCP	66 42582 → 80 [ACK] Seq=1 Ack=1 Wi

Frame 8: 724 bytes on wire (5792 bits), 724 bytes captured (5792 bits)
Ethernet II, Src: 02:42:c0:a8:14:0b (02:42:c0:a8:14:0b), Dst: 02:42:0f:84:e6:3d (02:42:0f:84:e6:3d)
Internet Protocol Version 4, Src: 192.168.20.11, Dst: 192.168.20.1
Transmission Control Protocol, Src Port: 80, Dst Port: 56968, Seq: 239, Ack: 358, Len: 658
[2 Reassembled TCP Segments (896 bytes): #6(238), #8(658)]

Hypertext Transfer Protocol

- HTTP/1.1 200 OK\r\n
 - Server: nginx/1.25.3\r\n
 - Date: Tue, 23 Jan 2024 15:39:37 GMT\r\n
 - Content-Type: text/html\r\n
 - Content-Length: 658\r\n
 - Last-Modified: Wed, 22 Nov 2023 23:12:01 GMT\r\n
 - Connection: keep-alive\r\n
 - ETag: "655e8ac1-292"\r\n
 - Accept-Ranges: bytes\r\n

Figure 2: Capture Wireshark web1

13	1.365881	192.168.20.1	192.168.20.12	TCP	74 42582 → 80 [SYN] Seq=0 Win=642
14	1.365968	192.168.20.12	192.168.20.1	TCP	74 80 → 42582 [SYN, ACK] Seq=0 Ac
15	1.365987	192.168.20.1	192.168.20.12	TCP	66 42582 → 80 [ACK] Seq=1 Ack=1 W
16	1.366079	192.168.20.1	192.168.20.12	HTTP	423 GET / HTTP/1.1
17	1.366130	192.168.20.12	192.168.20.1	TCP	66 80 → 42582 [ACK] Seq=1 Ack=358
18	1.366830	192.168.20.12	192.168.20.1	HTTP	1005 HTTP/1.1 200 OK (text/html)
19	1.367071	192.168.20.1	192.168.20.12	TCP	66 42582 → 80 [ACK] Seq=358 Ack=9
20	1.414836	192.168.20.1	192.168.20.12	HTTP	376 GET /favicon.ico HTTP/1.1
21	1.415017	192.168.20.12	192.168.20.1	HTTP	476 HTTP/1.1 404 Not Found (text/
22	1.415028	192.168.20.1	192.168.20.12	TCP	66 42582 → 80 [ACK] Seq=668 Ack=1

Frame 18: 1005 bytes on wire (8040 bits), 1005 bytes captured (8040 bits)
Ethernet II, Src: 02:42:c0:a8:14:0c (02:42:c0:a8:14:0c), Dst: 02:42:0f:84:e6:3d (02:42:0f:84:e6:3d)
Internet Protocol Version 4, Src: 192.168.20.12, Dst: 192.168.20.1
Transmission Control Protocol, Src Port: 80, Dst Port: 42582, Seq: 1, Ack: 358, Len: 939

Hypertext Transfer Protocol

- HTTP/1.1 200 OK\r\n
 - Date: Tue, 23 Jan 2024 15:39:38 GMT\r\n
 - Server: Apache/2.4.58 (Unix)\r\n
 - Last-Modified: Sat, 25 Nov 2023 22:07:46 GMT\r\n
 - ETag: "290-60b014c76f080"\r\n
 - Accept-Ranges: bytes\r\n
 - Content-Length: 656\r\n
 - Keep-Alive: timeout=5, max=100\r\n
 - Connection: Keep-Alive\r\n
 - Content-Type: text/html\r\n

Figure 3: Capture Wireshark web2

Mise en place d'un reverse proxy simple basé sur NGINX

Q5. Décrivez sous la forme d'un schéma l'environnement de test instancié en distinguant les différents éléments et les différents réseaux/adresses IP impliqués.

Il y a deux réseaux :

Réseau	Subnet
net_pub	192.168.20.0/24
net_priv	192.168.30.0/24

Il y a deux conteneurs :

Conteneur	Description	Réseaux
rp1	Reverse proxy NGINX	net_pub (192.168.20.2), net_priv (192.168.30.10)
web1	Serveur HTTP Apache	net_priv (192.168.30.11)

Si le client veut accéder au serveur web, il doit passer par le reverse proxy qui va intercepter la requête vers le serveur web et renvoyer la réponse au client.

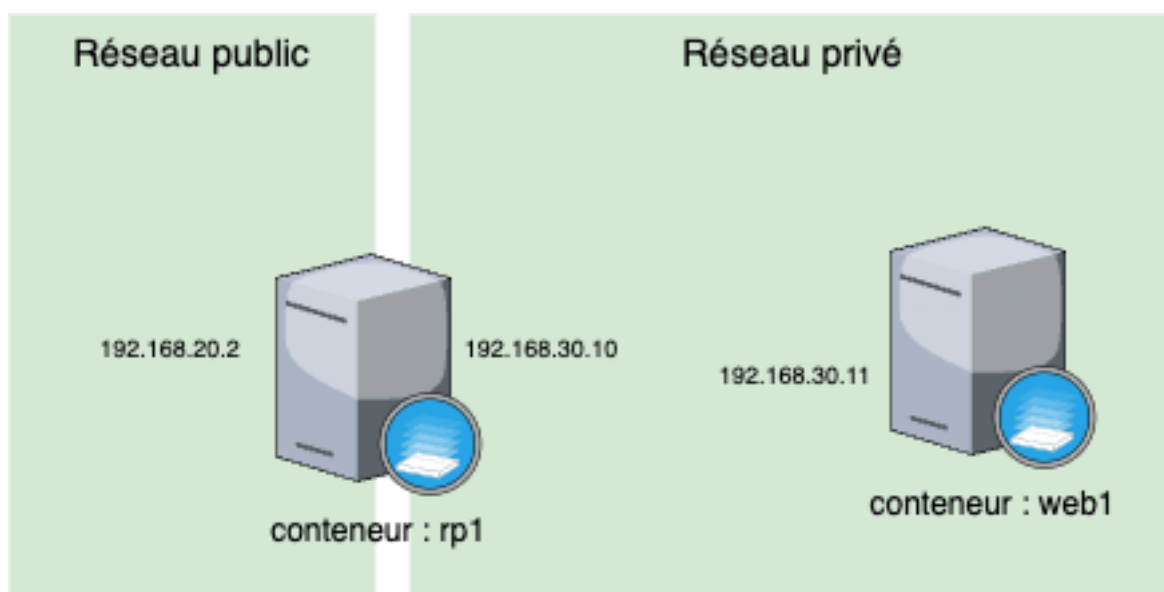


Figure 4: Schéma d'architecture reverse proxy

Q6. Représentez sous la forme d'un graphe de flux le déroulement des échanges (aidez-vous des fonctionnalités de Wireshark). Décrivez et expliquez les différents flux réseaux.

```
sudo docker exec -it exi2_rp1_1 sh
tcpdump -i any -w dump.pcap -s0 port 80
sudo docker cp exo2_rp1_1:/dump.pcap .
```

Pour tcpdump, nous avons utilisé l'option `-i any` pour capturer tous les flux réseaux et filtrer sur le port 80 (HTTP), puis l'option `-s0` pour capturer le contenu des paquets.

Nous pouvons voir les étapes suivantes :

- Le client établit une connexion TCP avec le reverse proxy (sur le port 80).

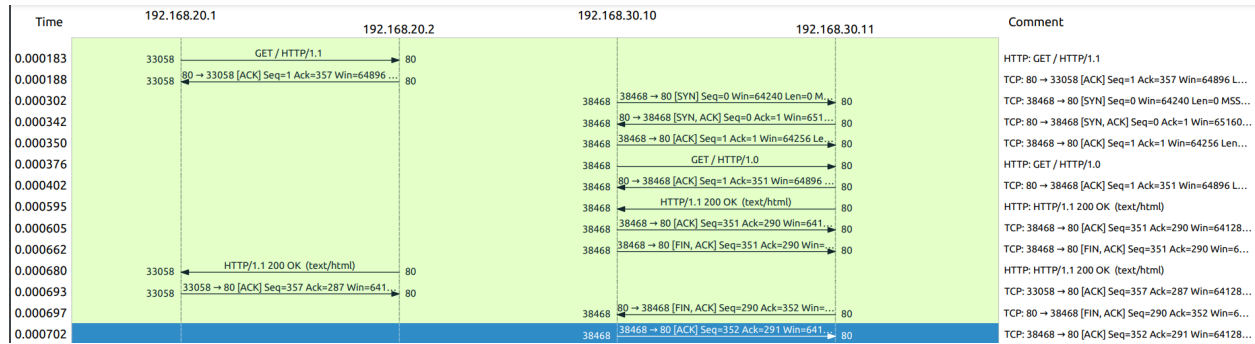


Figure 5: Trame Wireshark

- Le client demande la page / avec une requête GET.
- Le reverse proxy renvoie un ACK pour signaler au client qu'il a bien reçu la requête et qu'il va la traiter.
- Le reverse proxy (sur son port 38468) établit une connexion avec le serveur web (sur son port 80).
- Le reverse proxy demande la page / avec une requête GET.
- Le serveur web renvoie la page au reverse proxy.
- Le reverse proxy confirme réception de la réponse.
- Le reverse proxy renvoie la page web au client.
- Le client confirme la bonne réception de la page.
- Le reverse proxy confirme la bonne réception de la page au serveur web.

Q7. Quelles sont les principales différences entre les logs de rp1 et web1 ? Quels champs sont manquants pour la connexion via Telnet ? Quelles informations pertinentes pourraient être ajoutées côté web1 vis-à-vis du client ?

```
telnet 192.168.20.2 80
```

```
GET /
```

```
sudo docker exec -it exo2_rp1_1 sh
```

```
more /var/log/nginx/rp1.access.log # Réponse : 192.168.20.1 "GET /" 200 45 "-" "-" "
```

```
sudo docker exec -it exo2_web1_1 sh
```

```
more /usr/local/apache2/logs/access_log # Réponse : 192.168.30.10 "GET / HTTP/1.0" 200 45 "-" "-" "
```

Les différences entre les logs de rp1 et de web1 sont :

- l'IP source, le reverse proxy voit l'IP du client (192.168.20.1) tandis que le serveur web1 voit l'IP du reverse proxy (192.168.30.10).
- Le protocole et la version HTTP est ajouté dans les logs du serveur web1 car le reverse proxy l'ajoute si elle est manquante.

Les champs manquants pour la connexion via Telnet sont l'utilisateur agent et le protocole et la version utilisé (HTTP/1.0).

Les informations pertinentes qui pourraient être ajoutées côté web1 vis-à-vis du client sont :

- l'IP du client (via l'entête HTTP X-Forwarded-For)
- le protocole du client (via l'entête X-Forwarded-Proto)

Q8. Comparez les entêtes HTTP côté client et côté serveur web1. Quels sont les avantages d'une telle configuration ?

```
cat /home/tpproxy/exo2/rp1-custom.conf
```

```
cat /home/tpproxy/exo2/rp1-nginx.conf
```

Fichier rp1-custom.conf :

```

server {
    # Ecoute sur le port 80
    listen      80;
    listen  [::]:80;
    server_name localhost;

    access_log  /var/log/nginx/rp1.access.log  main;

    # Reverse proxy du / vers le serveur web1
    location / {
        proxy_pass http://exo2_web1_1/;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #

    # Pages d'erreurs contenu dans le dossier /usr/share/nginx/html
    error_page  500 502 503 504  /50x.html;
    location = /50x.html {
        root    /usr/share/nginx/html;
    }
}

```

Fichier rp1-nginx.conf :

```

user  nginx;
worker_processes  auto;

error_log  /var/log/nginx/error.log notice;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include      /etc/nginx/mime.types;
    default_type application/octet-stream;

    # Format de log
    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile    on;
    #tcp_nopush  on;

    keepalive_timeout  65;
}

```

```
#gzip on;

# Inclusion des autres fichiers de configuration dont rp1-nginx.conf
include /etc/nginx/conf.d/*.conf;
}
```

On a complété la configuration pour permettre l'ajout ou la modification d'entêtes HTTP X-Real-IP, X-Forwarder-For et X-Forwarded-Proto (dans le fichier rp1-custom.conf) :

```
server {
    # ...

    location / {
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-For $remote_addr;

        proxy_pass http://exo2_web1_1/;
    }

    # ...
}
```

Pour vérifier la modification, nous avons fait une capture réseau :

```
tcpdump -i any -w dump_forward.pcap -s0 port 80
sudo docker cp exo2_rp1_1:/dump_forward.pcap .
```

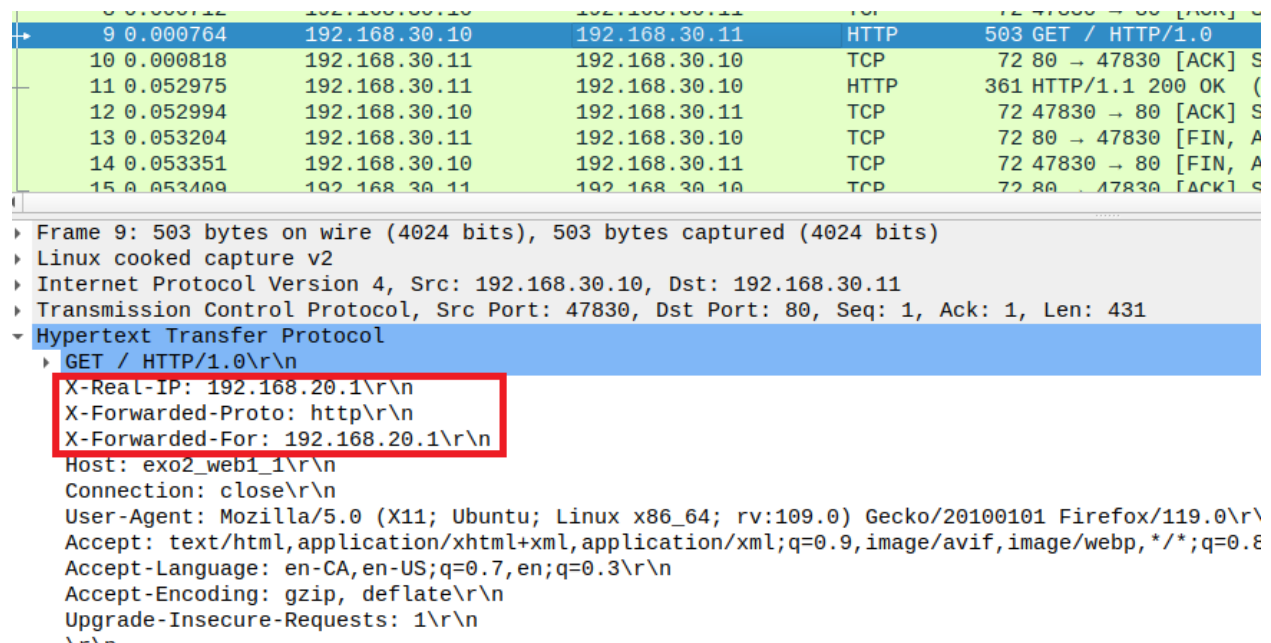


Figure 6: Trame Wireshark

Nous remarquons bien que les 3 entêtes ont été ajoutés.

Les avantages d'une telle configuration sont :

- le serveur web1 peut savoir l'IP du client (via l'entête HTTP X-Forwarded-For et l'entête HTTP

X-Real-IP et par exemple adapter la langue du site selon l'IP)

- le serveur web1 peut savoir le protocole du client (via l'entête X-Forwarded-Proto, http ou https)

9. Quel est l'objectif de cette fonctionnalité ? Expliquez brièvement son fonctionnement

NGINX propose de configurer des buffers. En activant les buffers on a les avantages suivants :

- Mise en cache de la réponse : si 2 clients demandent la même ressource (qui n'est pas différente entre les 2 clients comme des images), le reverse proxy peut la garder en cache pour répondre plus rapidement au client et ne pas solliciter le serveur web.
- Maintient de la session avec keep-alive : réutilise la même connexion TCP pour plusieurs requêtes HTTP en envoyant un "keep-alive" toutes les 5 minutes au client. Cela permet que le client a eu une réponse plus rapidement.
- Requête asynchrone : le reverse proxy stocke la réponse du serveur web jusqu'à ce qu'elle soit complète et la renvoie après au client. Lorsque le client n'a pas une bande passante élevée, le reverse proxy demande l'entièreté de la réponse au serveur et stock la réponse pour la fournir petit à petit au client. Cela peut être un inconvénient pour les clients rapides si il souhaite avoir la réponse le plus vite possible.