Le service MySocket, pour donner un accès Internet à ses développements locaux

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 25 novembre 2020

https://www.bortzmeyer.org/mysocket.html

Le service MySocket <https://www.mysocket.io/> vient d'être lancé. À quoi ça sert ? À plusieurs choses mais le scénario d'utilisation le plus évident est le cas où vous avez un service TCP/IP sur une machine de développement et vous voudriez que des gens dans le vaste monde l'essaient. Mais, pour des raisons diverses, cette machine de développement n'est pas accessible depuis l'Internet. MySocket est un **relais** qui va prendre les connexions des gens extérieurs et les apporter à votre service.

Il y a beaucoup de cas où la machine sur laquelle vous êtes en train de travailler ne peut pas être contactée par des clients situés sur l'Internet. Le plus évident est celui où vous êtes coincé sur un réseau attardé qui n'a pas IPv6 (ou bien que vos clients n'ont pas IPv6) et que le NAT empêche ces connexions entrantes. (Ou, encore pire, le service est dans un conteneur Docker double-NATé…) MySocket va agir comme un mini-Cloudflare, il va fournir une prise réseau à laquelle vos clients se connecteront et, derrière, cela viendra chez vous. C'est très pratique pour des essais, du développement, etc.

MySocket a été présenté dans cet article <https://www.mysocket.io/post/introducing-mysocket>. Mais vous avez une documentation complète en ligne <https://mysocket.readthedocs.io/en/latest/>. Pratiquer est un bon moyen de voir en quoi ce service consiste concrètement, donc allons-y.

MySocket se pilote via son API. Il ne semble pas y avoir d'interface Web pour les utilisateurs en ce moment. Pour parler à l'API, vous pouvez écrire votre propre programme ou, plus simple, utiliser le client `mysocketctl` développé par MySocket. (Son source est disponible <https://github.com/mysocketio/mysocketctl/>.) C'est un script Python donc on l'installe de manière pythonienne :

```
%   pip3 install --user mysocketctl
...
Successfully installed mysocketctl-0.3
```

La connexion entre MySocket et votre service se fera forcément en SSH. Donc, il faut une clé SSH. Ici, je vais en créer une dédiée à ce service MySocket :

```
% ssh-keygen -t ed25519 -f ~/.ssh/id_mysocket
...
Your public key has been saved in /home/stephane/.ssh/id_mysocket.pub.

%  ssh-add ~/.ssh/id_mysocket
```

On peut maintenant se créer un compte sur le service MySocket (je me répète mais il n'y a pas d'interface Web pour cela) :

```
% mysocketctl account create \
    --name "Stéphane Bortzmeyer" \
    --email "foobar@example.com" \
    --password "Plein de caractères" \
    --sshkey "$(cat ~/.ssh/id_mysocket.pub)"
Congratulation! your account has been created. A confirmation email has been sent to foobar@example.com
Please complete the account registration by following the confirmation link in your email.
...
```

Une fois reçu le message de confirmation envoyé par courrier, vous suivez le lien et votre adresse est confirmée. Note amusante : la réponse du serveur Web sera en JSON (oui, oui, avec le type `application/json`), et composée de la chaîne `"You have confirmed your account. Thanks!"`.

Les opérations de gestion des prises réseau nécessiteront un jeton d'authentification que l'on crée avec la commande `login`, et qui est valable quelques heures (après, vous aurez un "*Login failed*") :

```
% mysocketctl login \
    --email "foobar@example.com" \
    --password "Plein de caractères"

Logged in! Token stored in /home/stephane/.mysocketio_token
```

(À l'heure actuelle, ce fichier `.mysocketio_token` est créé avec les permissions par défaut. Vous aurez peut-être besoin de durcir ses permissions.)

Voilà, à ce stade, vous avez un compte, vous avez un jeton d'authentification, on peut créer une prise. `mysocketctl` permet de le faire de manière simple, ou de manière plus compliquée mais permettant davantage de choses. Commençons par la manière simple :

```
% mysocketctl connect \
    --port 8080 \
    --name "Youpi"

+--------------------------------------+---------------------------------+-------+
|              socket_id               |            dns_name             |  name |
+--------------------------------------+---------------------------------+-------+
| 3d66b504-677f-4e68-85a7-bb63da251007 | shy-bush-6765.edge.mysocket.io | Youpi |
+--------------------------------------+---------------------------------+-------+

Connecting to Server: ssh.mysocket.io
...
Youpi - https://shy-bush-6765.edge.mysocket.io
...
```

(Si vous récupérez un "*Permission denied (publickey)*", vérifier que la clé SSH a bien été chargée dans l'agent avec `ssh-add`.) Et voilà, une prise a été créée et elle est accessible du monde extérieur via le nom `shy-bush-6765.edge.mysocket.io`. Par défaut, c'est une prise HTTPS et les clients sont donc censés s'y connecter avec l'URL indiqué `https://shy-bush-6765.edge.mysocket.io`. Notez bien que vous ne choisissez pas le nom de domaine, il est attribué par MySocket et vous êtes donc dépendant d'eux, y compris dans le nom publié. On l'a déjà dit : MySocket est surtout conçu pour du développement, du temporaire, pas pour le site e-commerce de votre entreprise.

Maintenant, un client, par exemple un navigateur Web, va essayer de se connecter à cet URL. Et paf, on a un "*502 Bad Gateway*". Qu'est-ce que cela veut dire ? Tout simplement qu'on a créé la prise, qu'on l'a connectée au port 8080 de notre machine mais que rien n'écoute sur ce port (d'où l'erreur 502, RFC 7231 [1], section 6.6.3). Il nous faut développer un petit service. Ici, on va utiliser le module `http.server` de Python :

```
#!/usr/bin/env python3

import http.server

server_address = ('', 8080)
httpd = http.server.HTTPServer(server_address,
                               http.server.SimpleHTTPRequestHandler)
httpd.serve_forever()
```

Simple, non ? Cela crée un serveur HTTP (rappelez-vous que par défaut `mysocketctl` crée une prise HTTP), écoutant sur le port 8080 et traitant les requêtes avec la classe `SimpleHTTPRequestHandler`. Cette classe contient du code pour les méthodes HTTP comme `GET` et, par défaut, elle crée un petit serveur de fichier qui sert les fichiers locaux. Exécutons ce serveur :

```
% ./test-mysocket.py
...
127.0.0.1 - - [25/Nov/2020 18:13:06] "GET / HTTP/1.0" 200 -
```

Ça marche, le client HTTP reçoit cette fois le code de retour 200 indiquant que tout va bien, et il voit la liste des fichiers locaux. La ligne commençant par `127.0.0.1` est le journal qu'affiche par défaut le module `http.server`. Le premier champ est l'adresse IP du client, ici `127.0.0.1`, l'adresse de la machine locale, car c'est le client MySocket, lancé par `mysocketctl` qui s'est connecté à ce service. Si on veut voir la vraie adresse du client, il faut regarder le journal de `mysocketctl` :

```
192.0.2.106 - - [25/Nov/2020:17:13:06 +0000] "GET / HTTP/1.1" 200 432 "-" "curl/7.64.0" response_time="0.032 sec
```

C'est pratique, on a la vraie adresse IP du client. Elle est également disponible dans les en-têtes de la requête HTTP, champs `X-Real-IP:` et `X-Forwarded-For:` (mais hélas pas avec l'en-tête standard du RFC 7239).

Pendant que tout marche et que le bonheur coule à flot, regardons d'un peu plus près le service que fournit MySocket. Je crée une prise, et on m'obtient le nom `purple-surf-7070.edge.mysocket.io`. Ce nom a une adresse IP (IPv4 seul, hélas) :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc7231.txt`

```
% dig A purple-surf-7070.edge.mysocket.io
...
;; ANSWER SECTION:
purple-surf-7070.edge.mysocket.io. 300 IN A 75.2.104.207
```

Mais cette adresse IP ne correspond pas à une seule machine : MySocket utilise l'"*anycast*" pour que cette adresse IP soit toujours proche de vous. (Évidemment, seul MySocket est "*anycasté*", votre service ne tourne toujours que sur une seule machine.) Pour réaliser cela, MySocket tourne sur un service d'AWS ("*Global Accelerator*") qui n'a hélas pas IPv6. Testons l'adresse IP avec les sondes RIPE Atlas <https://atlas.ripe.net/> :

```
% blaeu-reach -r 200 75.2.104.207
197 probes reported
Test #28277168 done at 2020-11-25T17:23:21Z
Tests: 584 successful tests (98.8 %), 0 errors (0.0 %), 7 timeouts (1.2 %), average RTT: 19 ms
```

On voit un RTT moyen très court, montrant bien l'intérêt de l'"*anycast*".

Je n'ai pas utilisé cette possibilité, mais notez que MySocket permet également de créer des prises protégées (avec l'option `--protected`). Pour HTTP, ce sera avec l'authentification simple du RFC 7235. `mysocketctl connect --help` si vous voulez des détails, je n'ai personnellement pas testé.

J'ai dit qu'il y avait deux manières de gérer les prises, une simple et une plus compliquée. On vient de tester la simple, avec `mysocketctl connect` qui crée la prise, y connecte votre service, et nettoie lorsqu'il se termine. Maintenant, on va procéder en deux temps, d'abord en créant la prise, sans y connecter notre service :

```
% mysocketctl socket create --name "Youpi"
+--------------------------------------+-----------------------------------+---------+------+-------+
|              socket_id               |              dns_name             | port(s) | type |  name |
+--------------------------------------+-----------------------------------+---------+------+-------+
| c5d771fe-1ae8-4254-a8bd-fb8b1f4c293e | lively-rain-9509.edge.mysocket.io |  80 443 | http | Youpi |
+--------------------------------------+-----------------------------------+---------+------+-------+
```

On dispose désormais d'une prise « permanente » (qui ne disparait pas quand on a cessé de se connecter). `mysocketctl socket ls` me donnera la liste de ces prises. Pour utiliser la prise, il faut créer un tunnel :

```
% mysocketctl tunnel create --socket_id c5d771fe-1ae8-4254-a8bd-fb8b1f4c293e
+--------------------------------------+--------------------------------------+---------------+-------------
| socket_id                            | tunnel_id                            | tunnel_server | relay_port
+--------------------------------------+--------------------------------------+---------------+-------------
| c5d771fe-1ae8-4254-a8bd-fb8b1f4c293e | ace4b7bf-78f5-4373-992c-e7854b917b45 |               | 6150
+--------------------------------------+--------------------------------------+---------------+-------------
```

Puis on se connecte à MySocket :

```
% mysocketctl tunnel connect \
    --socket_id c5d771fe-1ae8-4254-a8bd-fb8b1f4c293e \
    --tunnel_id ace4b7bf-78f5-4373-992c-e7854b917b45     --port 8080

Connecting to Server: ssh.mysocket.io
...
Youpi - https://lively-rain-9509.edge.mysocket.io
```

Et on peut s'en servir, à l'URL indiqué.

Quelques petits points divers, avant de montrer deux services actuellements disponibles via MySocket :

— Je n'ai pas creusé la question du modèle d'affaires de MySocket. Pour l'instant, tout est gratuit, fonctionnant au mieux (sans garantie). Je suppose qu'il y aura un service payant avec davantage de possibilités et de garanties, mais je n'en sais rien. Notez que le créateur de MySocket est Andree Tonk, créateur de BGPmon <https://www.bortzmeyer.org/outils-bgp.html>, donc a priori, c'est sérieux.
— Ceci dit, MySocket est une solution "*cloud*", c'est-à-dire tournant sur d'autres ordinateurs que le vôtre. Si la communication entre votre service et MySocket est chiffrée avec SSH, et que celle entre le client et MySocket l'est en général avec TLS, le trafic est en clair sur les serveurs de MySocket (qui sont des machines Amazon). Donc, attention si vous manipulez des données importantes.
— Pensez aussi à la sécurité de **votre** service. MySocket permet à tout l'Internet (sauf si vous utilisez le mode protégé, avec `--protected`) d'appeler un service qui tourne sur votre machine. Un serveur mal écrit et des tas de choses ennuyeuses peuvent se passer. Il est recommandé d'utiliser un langage de programmation sûr (Python plutôt que C...) et de faire tourner le service dans un bac à sable, par exemple un conteneur.
— Un des avantages de MySocket est que votre service apparaitra aux yeux du monde comme sécurisé avec TLS, sans que vous ayiez le moindre effort de programmation ou de configuration à faire, ce qui est très sympa. (Le certificat de MySocket est un Let's Encrypt.)

Et pour finir, voici deux services qui tournent actuellement via MySocket. Tous les deux sont hébergés sur un Raspberry Pi 1 (oui, le premier modèle). Ces deux services sont évidemment sans garantie de fiabilité ou de pérennité, c'est juste pour vous montrer et que vous puissiez tester. Le premier utilise une prise de type HTTPS, comme dans les exemples plus haut. Il indique la température du Raspberry Pi et le nombre de paquets et d'octets passés sur son interface Ethernet. Il est accessible en `https://frosty-butterfly-737.e` Le source du service est disponible (en ligne sur `https://www.bortzmeyer.org/files/mysocket-http-server.py`).

Le second utilise un autre type de prise, TLS, ce qui permet de faire tourner des protocoles non-HTTP. Pour créer une telle prise, on utilise l'option `--type TLS`. À noter qu'un port est alloué pour ce service, pensez à le noter, les clients devront utiliser ce port. Ce service compte simplement le nombre d'octets dans la chaîne de caractères que vous lui envoyer. Il est en `misty-violet-3591.edge.mysocket.io:41106`. Comme c'est du TLS, on ne peut pas utiliser telnet ou netcat comme client, donc on va se servir de `gnutls-cli` :

```
 % gnutls-cli -p 41106 misty-violet-3591.edge.mysocket.io
 ...
 - Status: The certificate is trusted.
- Description: (TLS1.2)-(ECDHE-SECP256R1)-(RSA-SHA256)-(AES-256-GCM)
...
- Simple Client Mode:

toto
4 bytes
```

Et `mysocketctl` va montrer la connexion :

```
192.0.2.106 [25/Nov/2020:19:19:50 +0000] TCP 200 bytes_sent:8 bytes_received:5 session_time: 1.899
```

Le source du service est disponible (en ligne sur `https://www.bortzmeyer.org/files/mysocket-tcp-ser` `py`).

Comme la demande pour un tel service est forte, il y a d'autres solutions possibles (je ne les ai pas testées) :

— Ngrok <`https://ngrok.com/`> (très riche en services <`https://ngrok.com/docs`>, accès gratuit limité, payant ensuite),
— Le service Argo <`https://developers.cloudflare.com/argo-tunnel/learning/how-tunnel-wo` (apparemment désormais "Cloudflare Tunnels") via l'outil `cloudflared` de Cloudflare,
— Expose <`https://beyondco.de/docs/expose/introduction`>.