Communication client / serveur

Dans cet article nous allons parler des différents protocoles de communication entre le client Docker et le daemon.

Communication via une socket unix par défaut

Lorsque l'on suit par exemple la procédure d'Installation de Docker CE sur Ubuntu, le Docker daemon est géré par *systemd*. On peut facilement voir les options fournies au lancement du daemon:

```
$ ps aux | grep dockerd
root      5470  0.2  6.6 782564 66644 ?        Ssl  13:02   0:00
/usr/bin/dockerd -H fd://
```

Le résultat de cette commande nous dit simplement que le daemon écoute sur une socket mise en place par *systemd*.

```
$ sudo systemctl status docker
- docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset:
enabled)
   Active: active (running) since Thu 2019-01-17 13:02:19 UTC; 18min ago
     Docs: https://docs.docker.com
 Main PID: 5470 (dockerd)
    Tasks: 10
   CGroup: /system.slice/docker.service
           └─5470 /usr/bin/dockerd -H fd://
           ...
```

Si nous regardons d'un peu plus près la définition du *service.docker*, nous pouvons voir qu'il dépend de *docker.socket*.

```
$ cat /lib/systemd/system/docker.service
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
BindsTo=containerd.service
After=network-online.target firewalld.service
Wants=network-online.target
```

```
Requires=docker.socket

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues
still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd://
...
```

Le fichier de configuration de *docker.socket* montre que la socket mise en place par *systemd* est */var/run/docker.sock*

```
cat /lib/systemd/system/docker.socket
[Unit]
Description=Docker Socket for the API
PartOf=docker.service

[Socket]
ListenStream=/var/run/docker.sock
SocketMode=0660
SocketUser=root
SocketGroup=docker

[Install]
WantedBy=sockets.target
```

On voit donc ici que le daemon Docker écoute sur la socket */var/run/docker.sock*, c'est la socket par défaut. Celle-ci ne permet qu'une communication locale, c'est à dire enter le client *docker* et le daemon *dockerd* installés sur la machine. Le client *docker* communique sur cette socket par défaut.

Communication sécurisée via un réseau TCP

La commande ci-dessous montre l'exemple d'un daemon Docker exposé de façon sécurisée sur un réseau:

```
$ ps aux | grep dockerd
$ root      2601  0.4  7.0 354148 71688 ?        Sl   03:51   1:49 dockerd --
data-root /var/lib/docker -H unix:// --label provider=virtualbox -H
tcp://0.0.0.0:2376 --tlsverify --tlscacert=/var/lib/boot2docker/ca.pem --
tlskey=/var/lib/boot2docker/server-key.pem --
tlscert=/var/lib/boot2docker/server.pem --storage-driver overlay2 --pidfile
/var/run/docker.pid
```

Note: nous verrons dans un prochain chapitre l'utilitaire Docker Machine qui permet de créer des hôtes Docker sur différents cloud providers et qui configure automatiquement le daemon avec les différentes options utilisées ci-dessus.

Il y a 2 flags qui nous intéressent ici, ceux-ci permettant la communication entre un client Docker et le daemon:

- *-H unix://* fait référence à la socket unix que l'on a vu plus haut (le path n'est pas précisé car c'est */var/run/docker.sock* par défaut). Cette socket est utilisée par le client local pour communiquer avec le daemon
- *-H tcp://0.0.0.0:2376* permet d'exposer le daemon via le port *2376* sur n'importe quelle interface réseau de la machine. Les différents flags commençant par *--tls* permettent de sécuriser cette connection avec TLS et également de vérifier l'authenticité du client en s'assurant que le certificat TLS du client a bien été signé par l'authorité de certification du daemon.

Pour qu'un client puisse communiquer avec le daemon via la socket TCP il faut donc:

- que le port *2376* soit ouvert et accessible depuis l'extérieur
- d'avoir accès au certificat du daemon
- de créer un couple clé privée / clé publique pour le client et le signer avec le certificat du daemon

Il faut également que le client précise, sur la ligne de commande avec le flag *-H* ou via la variable d'environnement *DOCKER_HOST*, l'URL du daemon:

- en utilisant le flag *-H*:

```
$ docker -H IP:2376 info
```

- en utilisant la variable d'environnement *DOCKER_HOST*:

```
DOCKER_HOST=IP:2376
docker info
```

Note: il est possible, mais déconseillé, d'exposer le daemon via tcp sans sécuriser la communication. Dans ce cas, la commande de lancement aurait la forme suivante:

```
$ dockerd -H unix:// -H tcp://0.0.0.0:2375
```

Il y a différentes choses à noter ici:

- le flag *-H unix://* est obligatoire si l'on souhaite continuer à pouvoir communiquer avec le daemon en local. Si il n'est pas précisé, seule la communication via le réseau sera possible
- le port *2375* est utilisé par convention dans le cas d'une communication non sécurisée (port *2376* pour une communication sécurisée)

Communication via SSH

Le protocole SSH est largement utilisé et est souvent l'un des seuls protocoles autorisés par défaut pour accéder à une machine. La release *18.09* de Docker permet de comuniquer avec le daemon directement via SSH.

Nous supposons dans cet exemple que nous avons un machine virtuelle *node1* basée sur Ubuntu et sur laquelle Docker a été installé. Son IP est *192.168.33.10* et elle contient un utilisateur nommé *luc* qui peut se connecter en SSH soit avec son mot de passe soit avec la clé privée *~/.ssh/node1_private_key*.

Depuis Docker *18.09*, Il est possible de s'adresser au docker daemon avec la commande suivante:

```
$ docker -H ssh://luc@192.168.33.10 version
luc@192.168.33.10's password:
Client: Docker Engine - Community
 Version:           18.09.1
 API version:       1.39
 Go version:        go1.10.6
 Git commit:        4c52b90
 Built:             Wed Jan  9 19:33:12 2019
 OS/Arch:           darwin/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          18.09.1
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.10.6
  Git commit:       4c52b90
  Built:            Wed Jan  9 19:02:44 2019
  OS/Arch:          linux/amd64
  Experimental:     false
```

Il y a plusieurs choses à modifier ici de façon à simplifier la commande:

- ajouter la clé privée de l'utilisateur dans la liste des identités (`ssh-add ~/.ssh/node1node1_private_key`), cela permettra de ne pas avoir à renseigner le mot de passe de l'utilisateur à chaque commande.
- utiliser la variable d'environnement DOCKER_HOST (`export DOCKER_HOST="ssh://vagrant@192.168.33.10"`), cela permettra de ne pas avoir à utiliser le flag *-H* pour chaque commande.

Il est alors possible de lancer la commande suivante, elle retournera le même résultat que précédemment.

```
$ docker version
```

En résumé

Nous avons vu ici différentes façons pour le client docker de communiquer avec un hôte, local ou bien distant. La possibilité d'utiliser une connection via SSH, disponible depuis peu, permet de simplifier ces intéractions.