



# Projet d'interconnexion de systèmes autonomes

Groupe 2

Département Sciences du Numérique  
2020-2021

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Principaux choix</b>	<b>3</b>
2.1	Architecture du réseau . . . . .	3
2.2	Plateforme : Docker . . . . .	4
2.3	Distribution des tâches . . . . .	4
<b>3</b>	<b>Observations</b>	<b>5</b>
3.1	Compatibilité de l'architecture avec les contraintes imposées . . . . .	5
3.2	Premiers pings ... . . . .	6
3.3	Déroulement et tests de fonctionnement . . . . .	7
3.3.1	Serveur FTP : Saïd Ait-Faska . . . . .	7
3.3.2	Serveur Web, VoIP et <code>projet.sh</code> : Youssef Minyari . . . . .	8
3.3.3	Serveur VPN : Ayoub Najmeddine . . . . .	8
3.3.4	Serveur DNS : Hamza Zougari Belkhatat . . . . .	10
3.3.5	Routeur site principal, routeur entreprise et service DHCP : Mohammed M'hand . . . . .	11
3.3.6	Box, routeur particulier et service DHCP : Othmane mokrane . . . . .	11
3.3.7	Box client secondaire et service DHCP : Aymane Elktini . . . . .	11
3.3.8	Routeur Interconnexion et service DHCP : Juan David Granados . . . . .	11
<b>4</b>	<b>Remarques particulières</b>	<b>12</b>
<b>5</b>	<b>Difficulté principale</b>	<b>12</b>
<b>6</b>	<b>Conclusion</b>	<b>13</b>

## Table des figures

1	Hôte et images Docker en état de marche . . . . .	6
2	Transmission de packets réussie . . . . .	6

# 1 Introduction

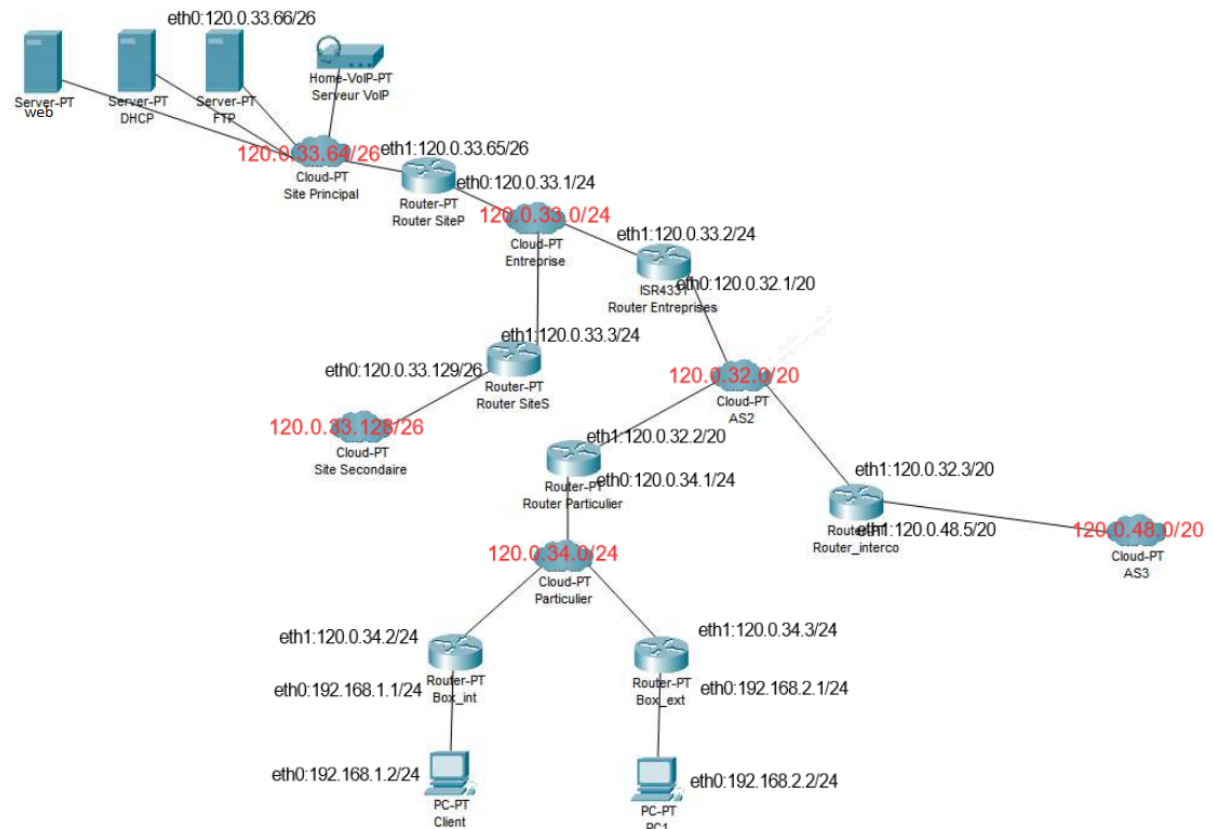
L'objectif de ce projet est de mettre en place un réseau d'entreprise multi-sites à travers la conception de **protocoles**, la **compréhension** de leur fonctionnement ainsi que leur **observation**. Il était ensuite question de réussir l'**interconnexion** avec d'autres AS (pour *autonomus system*).

Le travail imposait cependant un traitement des connaissances réseaux, tout comme un avancement de recherches au sujet des supports techniques et technologies IP à exploiter.

## 2 Principaux choix

### 2.1 Architecture du réseau

Avant d'envisager quelconque piste, il était tout d'abord question de définir une architecture convenable à un AS quelconque. Malgré plusieurs possibilités initialement abstraites et inaboutissantes, nous avons finalement consentis au modèle suivant :



Voici les principales composantes mises en places ci-dessus :

- **AS** : réseau central de l'architecture, à la politique de routage interne cohérente.
- **Entreprise** : réseau entreprise multi-sites, comportant un *site principal* et un *site secondaire* selon le cadre usuel d'un réseau d'entreprise aléatoire.
- **Particulier** : réseau du client faisant appel au service de l'entreprise.
- **Tunnel VPN Interco** : pour une encapsulation des données des protocoles réseau extérieurs (les autres AS).

- **Tunnel VPN interne vers entreprise** : même principe mais du côté particulier vers les sites de l'entreprise.
- **Serveur DHCP** : protocole réseau dont le rôle est d'assurer la configuration automatique des paramètres IP d'une station ou d'une machine, notamment en lui attribuant automatiquement une adresse IP et un masque de sous-réseau.
- **Serveur DNS** : service informatique distribué utilisé pour traduire les noms de domaine Internet en adresse IP ou autres enregistrements.
- **Serveur FTP** : permet de transférer des fichiers par Internet ou par le biais d'un réseau informatique local. Toute personne en ayant l'autorisation, peut télécharger et envoyer des fichiers sur un ordinateur distant faisant fonctionner un tel serveur.
- **VPN** : système permettant de créer un lien direct entre des ordinateurs distants, qui isole leurs échanges du reste du trafic se déroulant sur les autres réseaux.
- **Serveur VoIP** : permet de transmettre la voix sur des réseaux compatibles IP.
- **Serveur Web** : répond à des requêtes du World Wide Web sur un réseau public ou privé, en utilisant principalement le protocole HTTP.
- **Machine client Particulier** : pour tester l'adaptabilité du tunnel interne.
- **Machine client Entreprise** : permet le test du bon fonctionnement des différents services accordés au réseau.
- **Routeurs** : selon le service et le réseau concerné.
- **Pare-feu** : faire respecter la politique de sécurité du réseau, celle-ci définissant quels sont les types de communications autorisés sur ce réseau informatique. Il surveille et contrôle les applications et les flux de données.

## 2.2 Plateforme : Docker

L'une des étapes décisives du projet fût le choix d'un outil d'**empaquetage** de la database complète. Cela impliquait le suivi de formations particulières attribuées à chaque logiciel ou application, selon les dépendances des systèmes et la particularité des conteneurs isolés des serveurs. Le déploiement des unités normalisées nécessitait la recherche de configurations spécifiques selon un environnement exécutif adapté.

Cependant, la méthode standard pouvant assurer la majorité des applications opérationnelles, de la même manière qu'une *machine virtuelle*, ne pouvait être garantie par l'adoption d'une plateforme autre que **Docker**. Des *économies* considérables étaient assurées, ainsi que des migrations inter-systèmes aisées.

## 2.3 Distribution des tâches

Après la détermination de l'outil principal présenté en section précédente, il était question de partager efficacement les rôles, que se soit au niveau de la mise en œuvre globale ou d'une interprétation particulière à la visualisation interne. Des missions individuelles devaient être attribuées sans pour autant garder une singularité isolatrice, dans le sens où plusieurs composantes restent plus ou moins liées pour le fonctionnement général du réseau conçu.

Voici alors la distribution des missions (plus ou moins) individuelles :

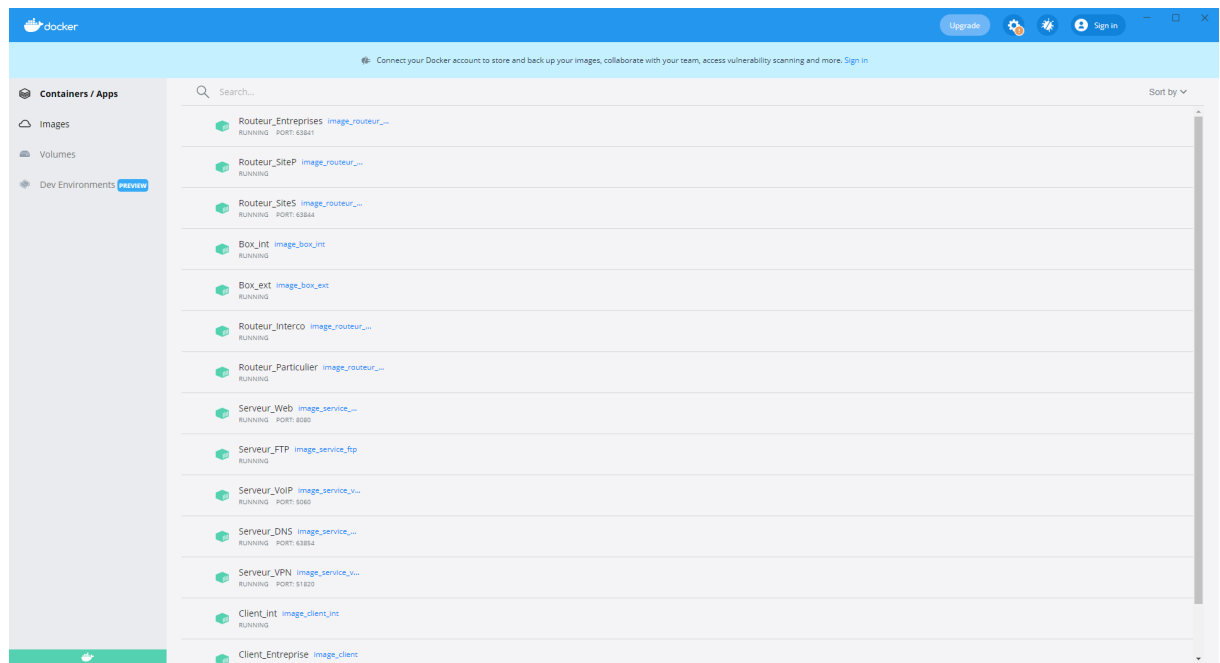
- **Youssef Minyari** : serveur VoIP + serveur Web + **projet.sh**.
- **Ayoub Najmeddine** : routeur + serveur VPN du site secondaire.
- **Said Aït-faska** : serveur FTP + pare-feu routeur site principal.
- **Hamza Zougari Belkhatat** : serveur DNS.
- **Juan David Granados** : routeur Interconnexion + service DHCP.

- **Mohammed M'hand** : routeur site principal + routeur entreprise + service DHCP.
- **Othmane mokrane** : box + routeur particulier + service DHCP.
- **Ayman Elktini** : box client secondaire + service DHCP.

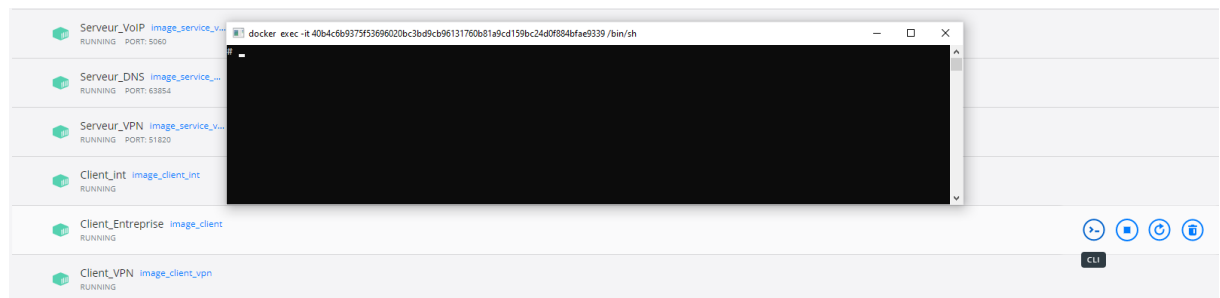
## 3 Observations

### 3.1 Compatibilité de l'architecture avec les contraintes imposées

Avant d'effectuer les tests de fonctionnement système, il était primordial de disposer d'**images Docker** convenables, fournissant ainsi un moyen pratique de regrouper des applications et des environnements de serveur préconfigurés, sous forme d'ensemble d'instructions créant un conteneur pouvant s'exécuter sur la plate-forme. Les recherches établies sur le sujet nous ont permis de déduire les modèles affichés dans la figure ci-contre :



Ensuite devait être constatée la bonne mise en place de ces images tout comme une manoeuvre adéquate. Un premier essai affichait par suite les résultat attendus (**environnement approprié et fenêtre de contrôle *bash* visible et exploitable**) :



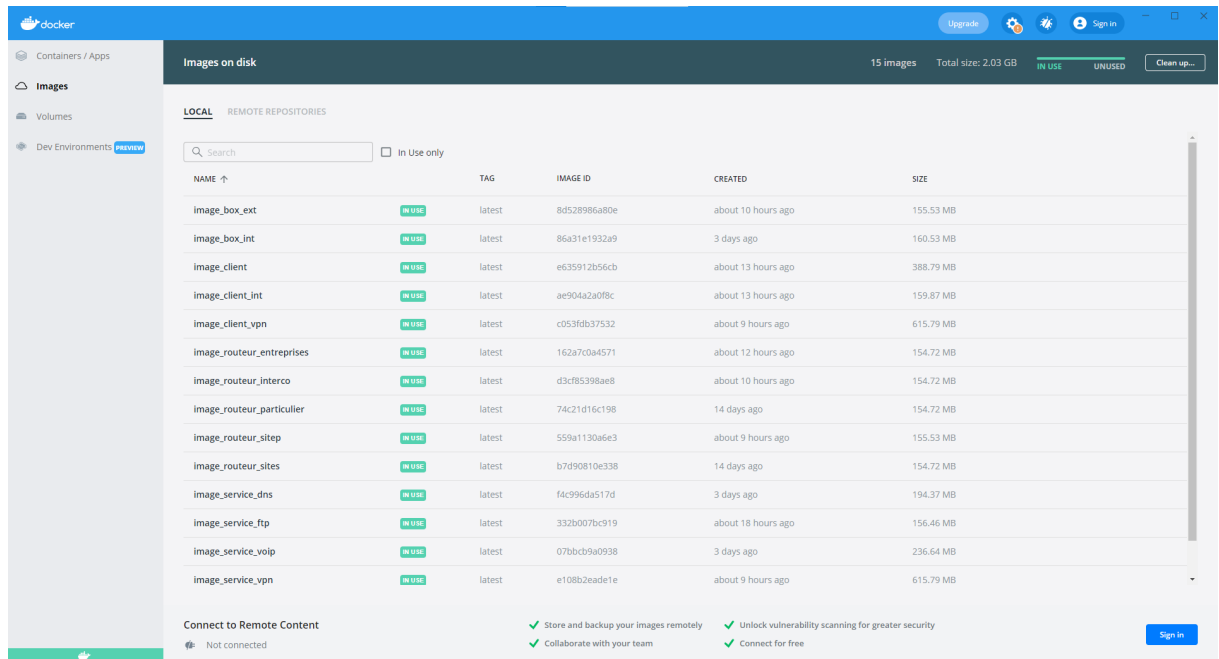


FIGURE 1 – Hôte et images Docker en état de marche

## 3.2 Premiers pings ...

Après instauration des équipements nécessaires, et avant toute démarche sérieuse vers le fond des choses, un premier test de transfert de ping s'avérerait bien utile pour évaluer la qualité de l'installation. Ceci fût alors un succès :

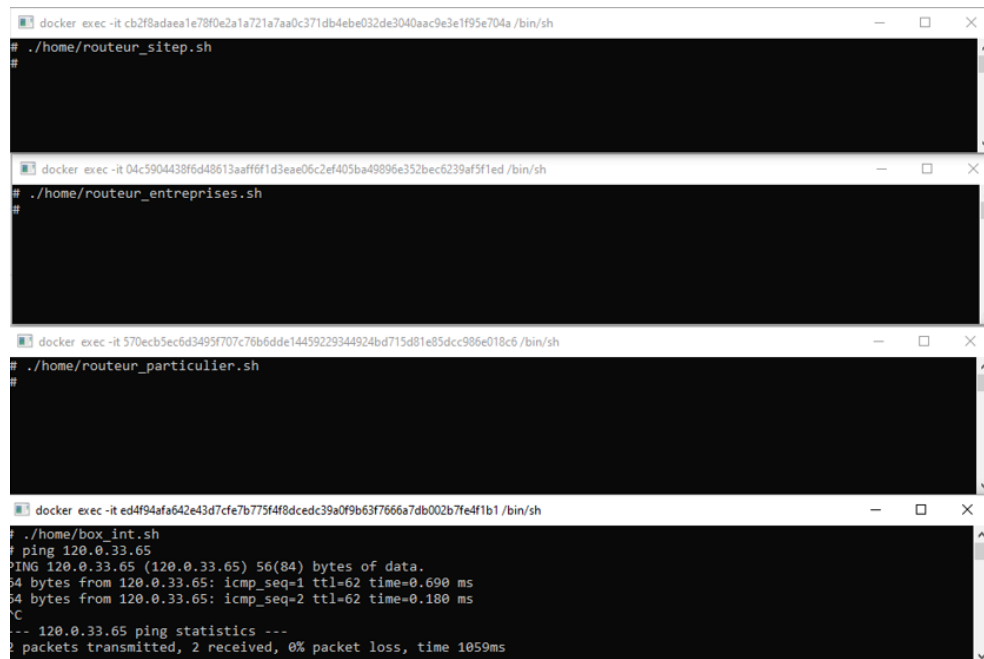


FIGURE 2 – Transmission de packets réussie

### 3.3 Déroulement et tests de fonctionnement

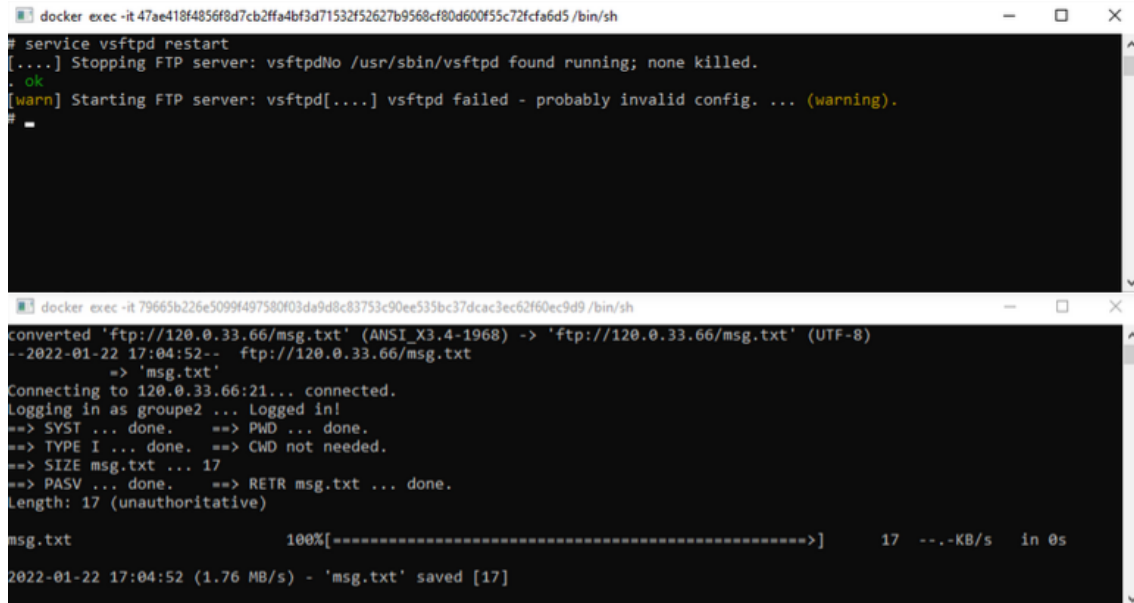
Dans cette section, il est question de présenter les principaux points d'avancement du projet, à travers un partage direct de l'expérience de chaque membre accompagné des explications nécessaires au pourquoi et comment des résultats. A noter que chacun s'est chargé des `fichiers.sh` propres à sa partie via des docker file, avec l'ajout indispensable de fichiers de configuration au sein des conteneurs lancés.

#### 3.3.1 Serveur FTP : Saïd Ait-Faska

Dans cette partie, j'ai rencontré plusieurs difficultés liées aux images devant être téléchargées dans Docker hub ; en effet, toute image trouvée n'était point adaptée au service FTP qu'on voulait implémenter : `proftpd`. A chaque fois qu'on trouvait une image contenant ce service, la configuration était pénible et très difficile à mettre en oeuvre, puisqu'en plus, il manquait toujours des fichiers de configuration dans ces mêmes images.

Pour pallier à cela, on a décidé d'implémenter une version alternative du FTP : `vsftpd`, ayant le même rôle que `proftpd`. Encore une fois, pour ce service, on a dû adopter des fichiers de configuration tout à fait à part (à l'instar de `vsftpd.cong` et autres) et les mettre dans le fichier du service.

**Pour la mise en test**, on a donc créé un répertoire nommée "groupe2" auquel on a attribué les droits d'exécution ainsi qu'un login et mot de passe associés, puis on a déposé un message dans ce répertoire ; le but était alors de **télécharger** ce message au niveau du **client**. Ce message est nommé `msg.txt` et contient un certain texte, par exemple : "BRAVO GROUPE2 VOTRE SERVICE FTP MARCHE BIEN!" Cette phrase test était cependant réalisable soit **avec une interface graphique**, et dans ce cas plus particulièrement avec **Filezilla**, soit **sur le terminal**. On a adhéré ainsi à la deuxième option car la manipulation des interfaces graphiques Docker s'avérait plutôt délicate.



```
docker exec -it 47ae418f4856f8d7cb2ffa4bf3d71532f52627b9568cf80d600f55c72fca6d5 /bin/sh
# service vsftpd restart
[....] Stopping FTP server: vsftpdNo /usr/sbin/vsftpd found running; none killed.
. ok
[warn] Starting FTP server: vsftpd[....] vsftpd failed - probably invalid config. ... (warning).
#

docker exec -it 79665b226e5099f497580f03da9d8c83753c90ee535bc37dcac3ec62f60ec9d9 /bin/sh
converted 'ftp://120.0.33.66/msg.txt' (ANSI_X3.4-1968) -> 'ftp://120.0.33.66/msg.txt' (UTF-8)
--2022-01-22 17:04:52-- ftp://120.0.33.66/msg.txt
=> 'msg.txt'
Connecting to 120.0.33.66:21... connected.
Logging in as groupe2 ... Logged in!
==> SYST ... done. ==> PWD ... done.
==> TYPE I ... done. ==> CWD not needed.
==> SIZE msg.txt ... 17
==> PASV ... done. ==> RETR msg.txt ... done.
Length: 17 (unauthoritative)

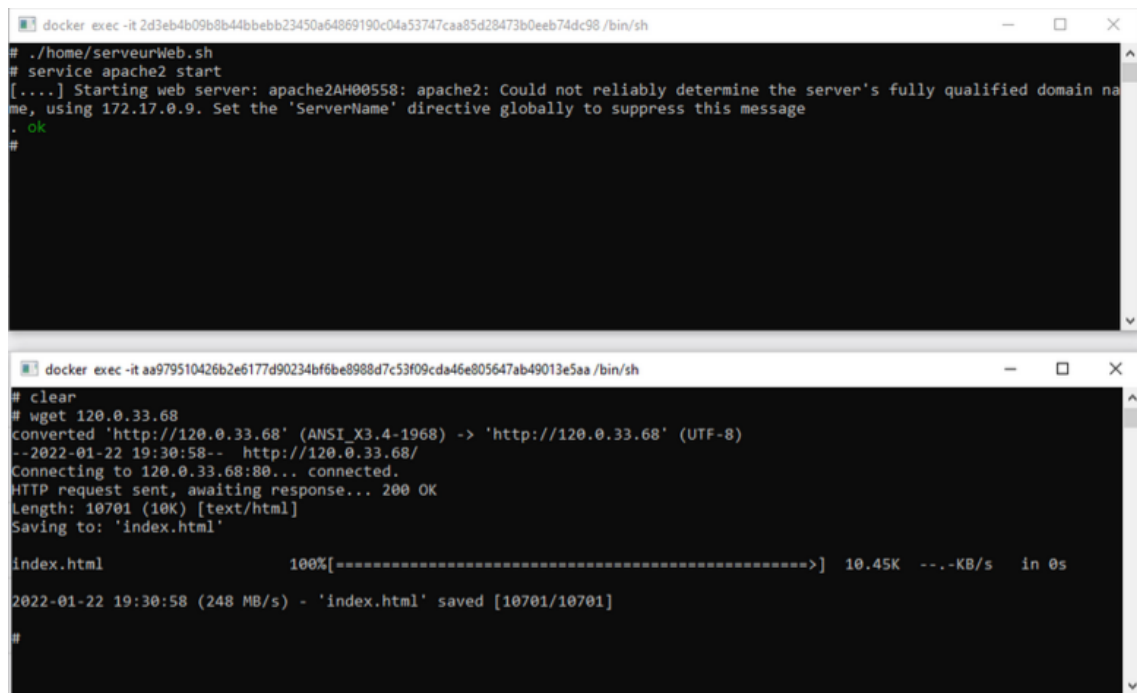
msg.txt          100%[=====]          17  --.-KB/s  in 0s

2022-01-22 17:04:52 (1.76 MB/s) - 'msg.txt' saved [17]
```

Notre service FTP marche correctement, et on a bien réussi à télécharger le message `msg.txt` chez le client.

### 3.3.2 Serveur Web, VoIP et projet.sh : Youssef Minyari

Pour ma part, le serveur Web était tout aussi simple à mettre en place : l'installation d'un logiciel dédié, **Apache 2** (jouant le rôle de serveur HTTP), a suffi et le test sur le client était un succès.



```
docker exec -it 2d3eb4b09b8b44bbbbb23450a64869190c04a53747caa85d28473b0eeb74dc98 /bin/sh
# ./home/serveurWeb.sh
# service apache2 start
[....] Starting web server: apache2AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.9. Set the 'ServerName' directive globally to suppress this message
. ok
#

docker exec -it aa979510426b2e6177d90234bf6be8988d7c53f09cda46e805647ab49013e5aa /bin/sh
# clear
# wget 120.0.33.68
converted 'http://120.0.33.68' (ANSI_X3.4-1968) -> 'http://120.0.33.68' (UTF-8)
--2022-01-22 19:30:58-- http://120.0.33.68/
Connecting to 120.0.33.68:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10701 (10K) [text/html]
Saving to: 'index.html'

index.html          100%[=====] 10.45K  --.-KB/s  in 0s
2022-01-22 19:30:58 (248 MB/s) - 'index.html' saved [10701/10701]
#
```

Néanmoins, en ce qui est du serveur VoIP, il s'agissait d'une **configuration par fichiers**. Ces derniers étaient correctement déposés, et deux utilisateurs associés ont été ajoutés pour compléter l'opération. Réaliser le test nécessitait ainsi l'exploitation de l'**interface graphique** d'un certain logiciel sous le nom de **EKIGA**, qui utilise les protocoles de communication standards et ouverts H.323 et SIP3, ce qui le rend compatible et interopérable avec les autres logiciels et appareils basés sur ces mêmes protocoles.

Je me suis aussi chargé de rédiger le fichier **projet.sh**, responsable de la mise en marche de la totalité de l'architecture du projet à travers les images préalablement introduites. En bref, ce même fichier peut être résumé en un "bouton de démarrage", sans lequel rien ne pourra être visualisé.

### 3.3.3 Serveur VPN : Ayoub Najmeddine

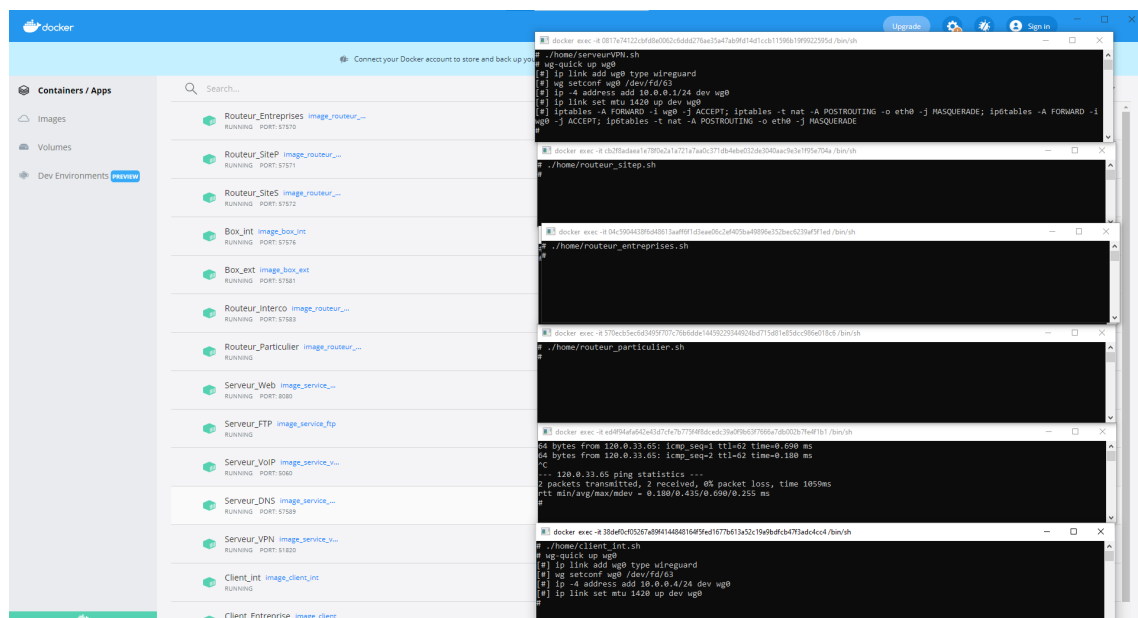
Il s'agissait ici d'une des parties les plus subtiles de tout le travail, autant au niveau des données pré-installées essentielles que les moyens de test les plus rationnels à exploiter.

Après des recherches approfondies, la décision était centrée sur l'exploitation **WireGuard VPN**, parmi tant de choix possibles, que ce soit pour des raisons techniques ou d'efficacité. Il s'avérait être en globalité le plus satisfaisant en terme d'usage dans nos conditions matérielles et numériques. En effet, WireGuard est un VPN simple (ce qui est pourtant une qualité privilégiée dans notre cas, surtout avec Docker dont on ignorait tout avant de commencer), rapide et offre un temps de ping beaucoup plus faible en comparaison à certains protocoles comme Open VPN que j'allais appliquer. Une connexion est établie par un échange de clés publiques entre serveur et

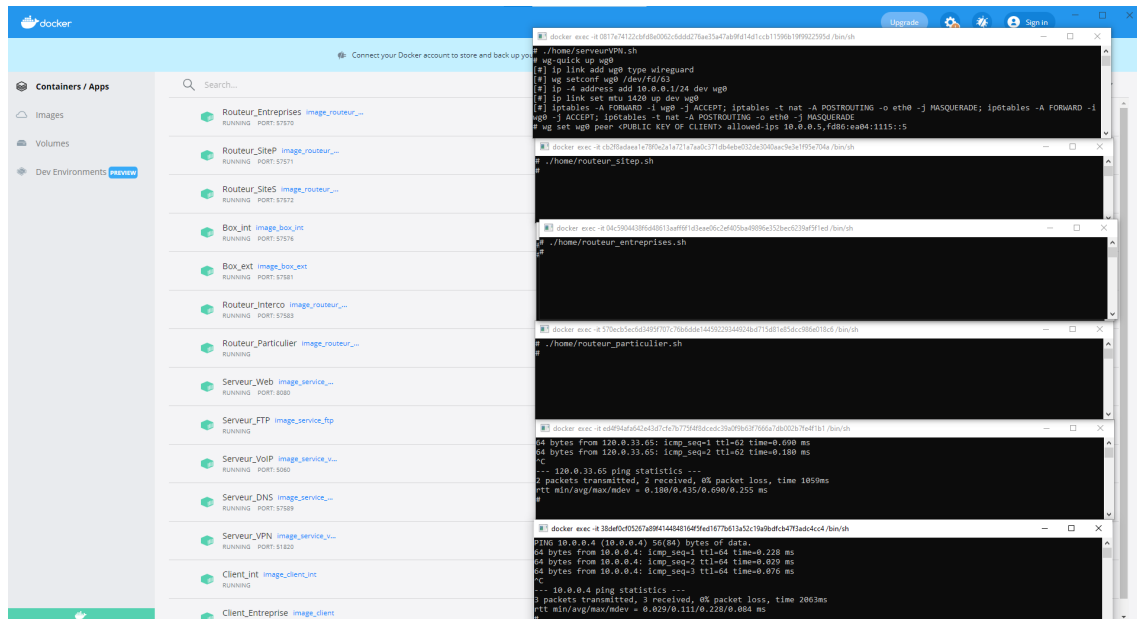


client tout comme les clés SSH et seul un client avec sa clé publique présente dans le fichier de configuration du serveur serait autorisé.

Tout d'abord, WireGuard est installé dans le client ainsi que le serveur, et toutes les configuration sont stockées dans un fichier à `/etc/wireguard/wg0.conf`. Il n'a pas besoin de s'appeler `wg0.conf`, il peut s'agir de `server.conf` ou `udp.conf`. J'ai ensuite défini l'interface du tunnel et spécifié la clé privée du serveur WireGuard générée. L'adresse définit les adresses IPv4 et IPv6 privées pour que le serveur WireGuard soit configuré derrière l'adresse IP publique du serveur Linux. ListenPort spécifie le port UDP que notre serveur VPN utiliserait pour écouter les connexions. Le transfert de compression est aussi nécessaire pour transférer le trafic des clients vers Internet. Le démarrage du serveur VPN WireGuard et activation de son exécution au redémarrage se fait grâce aux commandes `wg-quick up wg0` et `systemctl enable wg-quick@wg0`. Puis s'annonce la **configuration du client VPN WireGuard**, qui doit être fait sur une machine cliente locale avec Debian GNU/Linux, l'ajout du client WireGuard au serveur VPN sur le serveur Linux, et finalement la connexion au serveur VPN WireGuard à partir d'une machine locale avec la commande `sudo wg-quick up client`.



Arrivé à la partie test, j'ai vérifié la connexion avec la commande `wg` et en cinglant l'adresse IP de l'interface du serveur comme suit, ce qui a affiché le résultat attendu :

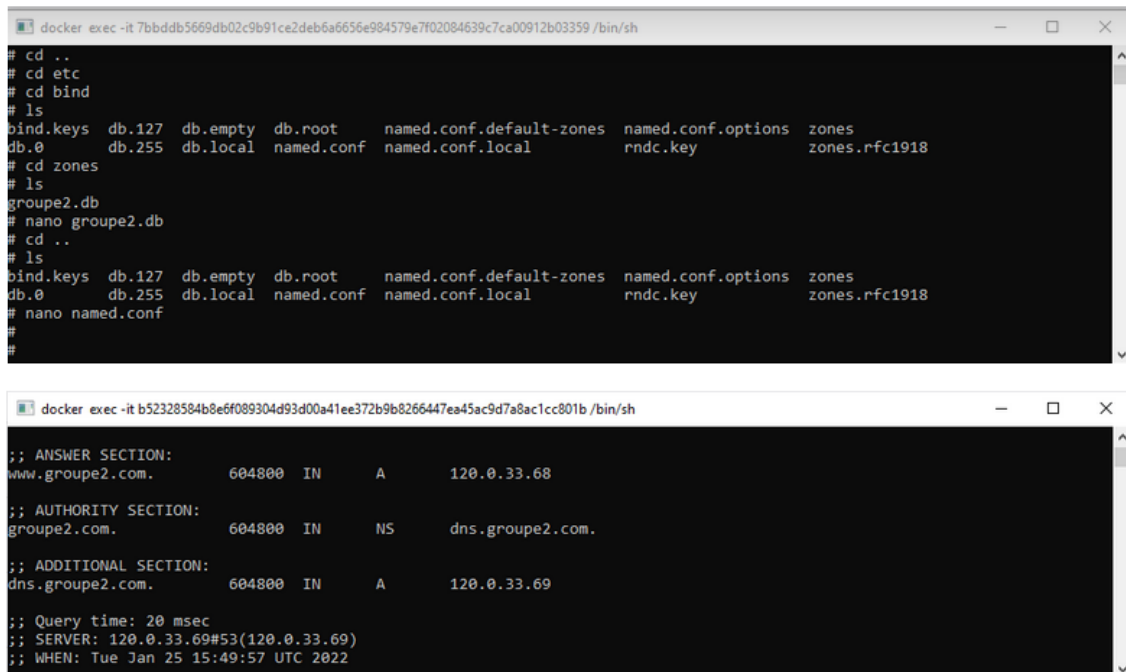


### 3.3.4 Serveur DNS : Hamza Zougari Belkhayat

La résolution de nom dans les environnements Linux est décrite dans le fichier `/etc/nsswitch.conf`. Par défaut, il a une entrée avec des fichiers dns, ce qui signifie qu'il vérifiera d'abord le fichier `/etc/hosts`, puis le serveur DNS.

Un domaine doit d'abord être réservé à l'adresse dédiée au serveur avec la commande `create -subnet=120.0.33.69 netmask 255.255.255.192 groupe2.net`. Il existe de nombreux serveurs de noms disponibles, mais un choix populaire est **BIND9**. C'est un serveur de noms **Open Source** et simple à configurer. Le premier fichier à configurer alors est le `named.conf.options`. Il décrit les configurations de base telles que les redirecteurs, l'interface d'écoute et le répertoire de cache. Ensuite, la zone appelée `groupe2.com` est créée et pointe vers un fichier appelé `/etc/bind/zones/db.groupe2.com`. Il est défini dans le fichier `named.conf`. Les changements nécessaires ont ainsi été effectués pour le réseau Docker et les domaines. En effet, le serveur DNS était initialement prévu comme réseau indépendant de l'entreprise, avant de l'introduire dans celle-ci pour un fonctionnement optimal. L'avantage du DNS est qu'on voit plus ou moins tout de suite si ça marche... ou pas. Premièrement, vous il est tout à fait possible de consulter l'activité DNS (blocages des nuisances y compris), et également les statistiques de Unbound avec docker logs unbound.

Pour la partie test, juste après activation du `bind9 daemon`, il est possible d'exécuter les deux hôtes en utilisant le conteneur `dns-server` comme serveur DNS. A l'intérieur du conteneur, on peut vérifier que l'hôte2 est accessible depuis l'hôte1, en utilisant le DNS : `exec -it host1 bash`. La sortie de la commande ping est la suivante :



The image contains two terminal window screenshots. The top window shows a series of commands and directory listings within a Docker container. The commands include `cd ..`, `cd etc`, `cd bind`, `ls`, `cd zones`, `ls`, `nano groupe2.db`, `cd ..`, and `nano named.conf`. The `ls` commands show the contents of the `/etc/bind` directory, listing files like `bind.keys`, `db.127`, `db.empty`, `db.root`, `named.conf.default-zones`, `named.conf.options`, `zones`, `db.0`, `db.255`, `db.local`, `named.conf`, `named.conf.local`, `rndc.key`, and `zones.rfc1918`. The bottom window shows the output of a DNS query using `dig`. The output includes the ANSWER SECTION, AUTHORITY SECTION, and ADDITIONAL SECTION, all showing records for `groupe2.com.` and `dns.groupe2.com.` with IP addresses `120.0.33.68` and `120.0.33.69`. It also shows query statistics like query time (20 msec) and server information.

```
# cd ..
# cd etc
# cd bind
# ls
bind.keys  db.127  db.empty  db.root  named.conf.default-zones  named.conf.options  zones
db.0       db.255  db.local  named.conf  named.conf.local        rndc.key            zones.rfc1918
# cd zones
# ls
groupe2.db
# nano groupe2.db
# cd ..
# ls
bind.keys  db.127  db.empty  db.root  named.conf.default-zones  named.conf.options  zones
db.0       db.255  db.local  named.conf  named.conf.local        rndc.key            zones.rfc1918
# nano named.conf
#
```

```
;; ANSWER SECTION:
www.groupe2.com.      604800  IN      A       120.0.33.68
;; AUTHORITY SECTION:
groupe2.com.          604800  IN      NS      dns.groupe2.com.
;; ADDITIONAL SECTION:
dns.groupe2.com.      604800  IN      A       120.0.33.69
;; Query time: 20 msec
;; SERVER: 120.0.33.69#53(120.0.33.69)
;; WHEN: Tue Jan 25 15:49:57 UTC 2022
```

### 3.3.5 Routeur site principal, routeur entreprise et service DHCP : Mohammed M’hand

Les conteneurs et les microservices sont de plus en plus utilisés pour le développement et le déploiement des applications. C’est ce qu’on appelle le développement ” cloud-native “. Dans ce contexte, Docker est devenue une solution massivement exploitée en entreprise.

L’image Docker utilisée convient à l’exécution d’un serveur DHCP pour notre réseau hôte Docker. Il utilise le serveur **DHCP ISC** qui est fourni avec la dernière distribution Ubuntu LTS auxquels sont ajoutés les fichiers de configuration nécessaires. DHCP ISC offre une solution open source complète pour la mise en œuvre de serveurs DHCP, d’agents de relais et de clients, prend en charge à la fois IPv4 et IPv6 et convient aux applications à haut volume et à haute fiabilité.

Un client DHCP est associé à chaque routeur (auquel une plage d’adresse est attribuée) ; il envoie des requêtes de configuration au serveur. La plupart des appareils et des systèmes d’exploitation ont déjà des clients DHCP inclus.

Les autres routeurs DHCP et box incluent dans l’architecture ont été configurées de la même façon, et sont tous fonctionnels. A chaque partie est associé une fenêtre de test correspondante.

### 3.3.6 Box, routeur particulier et service DHCP : Othmane mokrane

Même remarque que la précédente, avec des adresses différentes mais des résultats satisfaisants.

### 3.3.7 Box client secondaire et service DHCP : Aymane Elktini

Toujours la même remarque, puisqu’il s’agit d’un contexte similaire pour des zones différentes.

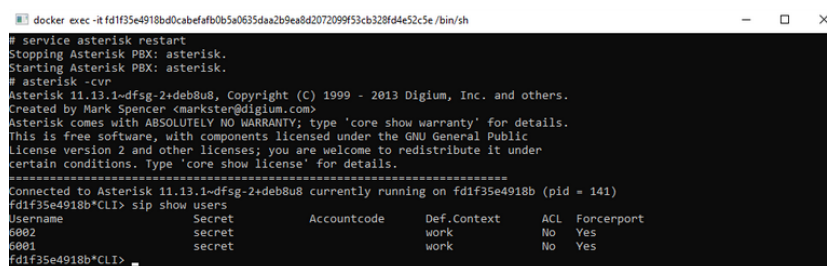
### 3.3.8 Routeur Interconnexion et service DHCP : Juan David Granados

Mis à part le routeur et service DHCP déjà évoqués, il s’agit de la partie ultime à ce qui paraît (puisque figure comme titre du projet). Cependant, une contrainte s’est imposée : notre groupe

ayant réalisé une architecture virtuelle contrairement à l'aplupart des autres, il était question de trouver un intermédiaire, ou soi-disant un moyen de "*convertir*", d'adapter les données classés en notre côté avec l'architecture, le mode de fonctionnement des unités extérieures. Malheureusement, ce morceau n'a pas pu être établi faute de temps et de moyens (en plus de la complexité du Docker et l'étendu de ses images).

## 4 Remarques particulières

- Pour le projet e général, on s'est servi de `debian:jessie-slim` afin de configurer à notre guise les routeurs et services de zéro, au lieu d'employer des images au contenu déjà prêt. Cela nous permettait de nous rapprocher mieux de la structure technique d'un composant, et d'éviter d'éventuelles erreurs d'adaptation à l'environnement étudié.
- La première versio d'FTP exploitée, `proftpd`, régis selon deux différents modes aléatoires, ne pouvant être choisis au lancement de l'image, d'où la nécessité de recourir à un accès libre fourni par `vsftpd`.
- En ce qui est de VoIP, deux utilisateurs été ajoutés pour les essais, avec activité correct d'`asterix` qui requiert l'application **EKIGA** présentée antérieurement :

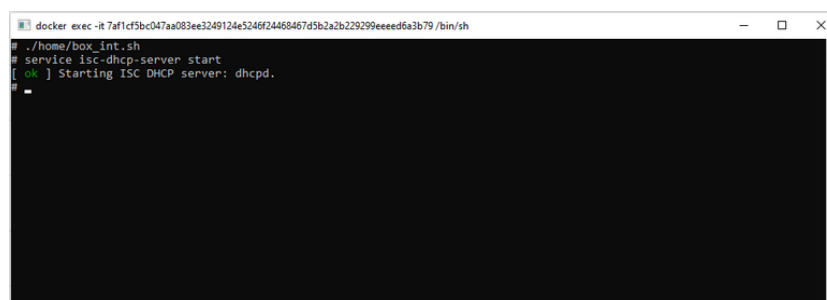


```

docker exec -it fd1f35e4918bd0cabefafb0b5a0635daa2b9ea8d2072099f53cb328fd4e52c5e /bin/sh
# service asterisk restart
Stopping Asterisk PBX: asterisk.
Starting Asterisk PBX: asterisk.
# asterisk -cwr
Asterisk 11.13.1~dfsg-2+deb8u8, Copyright (C) 1999 - 2013 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public
license version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=====
Connected to Asterisk 11.13.1~dfsg-2+deb8u8 currently running on fd1f35e4918b (pid = 141)
fd1f35e4918b*CLI> sip show users
Username      Secret      Accountcode  Def.Context  ACL  Forcerport
s002          secret      work         work         No   Yes
s003          secret      work         work         No   Yes
fd1f35e4918b*CLI>

```

- Le serveur DNS exige un attribution d'adresse FTP et Web devant être connues de la part du client.
- Voici un aperçu d'un test DHCP à succès :



```

docker exec -it 7af1cf5bc047aa083ee3249124e5246f24468467d5b2a2b229299eeedd6a3b79 /bin/sh
# ./home/box_int.sh
# service isc-dhcp-server start
[ ok ] Starting ISC DHCP server: dhcpd.
#

```

- **NB** : Seules les parties importantes et délicates sont développées dans ce rapport surtout pour la partie *déroulement*.

## 5 Difficulté principale

L'interconnexion avec les autres AS : on avait initialement prévu la mise en place d'un intermédiaire physique, un routeur représenté par une machine en salles internet, mais comme mentionné en partie dédiée, des contraintes de temps et surtout techniques ont empêché cette initiative, éventuellement ambitieuse mais complexe.

## 6 Conclusion

Ce projet s'est révélé très enrichissant dans la mesure où il a consisté en une approche concrète des architectures de réseaux autonomes et du métier d'ingénieur en général. En effet, la prise d'initiative, le respect des délais et le travail en équipe seront des aspects essentiels de notre futur métier. De plus, il nous a permis d'appliquer nos connaissances en réseaux et protocoles à un domaine pratique, qui se révèle aujourd'hui d'intérêt général au vu de l'augmentation des besoins des entreprises en outils et plateformes de conteneurisation.

Les principaux problèmes, que nous avons rencontrés, concernaient la collecte de données, souvent incompatibles tant sur les images que sur la structure globale. Ainsi, nous avons touché du doigt la difficulté d'extrapoler le jeu de données disponibles pour obtenir des résultats cohérentes sur l'ensemble du domaine étudié, ce que pourtant nous serons vraisemblablement amenés à faire dans notre futur métier.

Toutefois, si une telle étude devait être complétée, à plus ou moins longue échéance, par un bureau d'étude, Docker pourrait toujours bien figurer dans la liste du côté d'autres programmes adaptatifs.