

## PARTIE I

1)

Docker est une plate-forme logicielle qui permet de concevoir, tester et déployer des applications rapidement. Docker intègre les logiciels dans des conteneurs, qui rassemblent tous les éléments nécessaires à leur fonctionnement, dont les bibliothèques, les outils système, le code et l'environnement d'exécution. Avec Docker, on peut facilement déployer et tester des applications dans n'importe quel environnement.

Docker est construit sur une architecture client-serveur, ce qui signifie que les utilisateurs peuvent interagir avec le Docker via une interface en ligne de commande (CLI) ou une API. Les images Docker, qui sont des modèles de conteneurs, peuvent être téléchargées à partir du Docker Hub. Docker permet une plus grande efficacité en termes de développement et de déploiement, car il réduit les conflits entre les différentes bibliothèques et dépendances et permet un déploiement plus rapide.

2)

a)

1) On commence par mettre à jour les packages existants avec **sudo apt update**

2) Puis, On installe quelques paquets pré-requis qui vont permettre d'utiliser les paquets via HTTPS:

```
edgar@edgar-pc:~$ sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

3) Ensuite, on ajoute la clé GPG (un système de cryptographie utilisé pour vérifier l'authenticité et l'intégrité de fichiers) officielle de Docker :

```
edgar@edgar-pc:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -  
[sudo] password for edgar:  
OK
```

4) Puis, on ajoute le référentiel Docker stable à la liste des sources de paquets disponibles pour le système de gestion des paquets APT d'Ubuntu :

```
edgar@edgar-pc:~$ sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable"  
[sudo] password for edgar:  
Hit:1 http://fr.archive.ubuntu.com/ubuntu focal InRelease
```

5) On met à jour encore une fois avec **sudo apt update**

6) On vérifie notre installation pour voir si on installe bien le référentiel Docker (au lieu de référentiel Ubuntu par défaut)

```
edgar@edgar-pc:~$ apt-cache policy docker-ce  
docker-ce:  
  Installed: (none)  
  Candidate: 5:23.0.1-1~ubuntu.18.04~bionic  
  Version table:  
     5:23.0.1-1~ubuntu.18.04~bionic 500  
        500 https://download.docker.com/linux/ubuntu bionic/stable amd64 Packages  
     5:23.0.0-1~ubuntu.18.04~bionic 500
```

Donc cela indique que le paquet Docker CE n'est pas encore installé, mais la liste des versions disponibles pour l'installation inclut la version 5:23.0.1-1~ubuntu.18.04~bionic, qui est fournie par le référentiel Docker.

7) On installe Docker avec **sudo apt install docker-ce** et on vérifie si le docker est exécuté :

```
edgar@edgar-pc:~$ 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 Tue 2023-02-28 22:05:47 CET; 3min 26s ago
   TriggeredBy: ● docker.socket
```

Donc le Docker est bien installé .

b) On essaye de lancer avec **sudo docker run hello-world** , mais on obtient une erreur qui nous informe que l'image hello-world n'est pas installée, puis il télécharge et installe automatiquement.

```
edgar@edgar-pc:~$ sudo docker run hello-world
[sudo] password for edgar:
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:6e8b6f026e0b9c419ea0fd02d3905dd0952ad1fee67543f525c73a0a790fefb
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

On refait **sudo docker run hello-world** , donc on peut voir que l'image hello-world fonctionne.

```
edgar@edgar-pc:~$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

c) Il faut d'abord télécharger et installer l'image d'Ubuntu à partir du registre Docker :

```
edgar@edgar-pc:~$ sudo docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
677076032cca: Pull complete
Digest: sha256:9a0bdde4188b896a372804be2384015e90e3f84906b750c1a53539b585fbbe7f
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

Et on exécute ce conteneur avec **docker run -it ubuntu** :

```
edgar@edgar-pc:~$ sudo docker run -it ubuntu
root@1b49e4def0c9:/#
```

Cette commande exécute un nouveau conteneur Ubuntu en mode interactif (-it) et ouvre un shell dans le conteneur.

On peut vérifier la version d'Ubuntu avec **cat /etc/lsb-release** :

```
edgar@edgar-pc:~$ sudo docker run -it ubuntu
root@1b49e4def0c9:/# cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=22.04
DISTRIB_CODENAME=jammy
DISTRIB_DESCRIPTION="Ubuntu 22.04.1 LTS"
root@1b49e4def0c9:/#
```

d) On essaye de télécharger et installer le conteneur Windows :

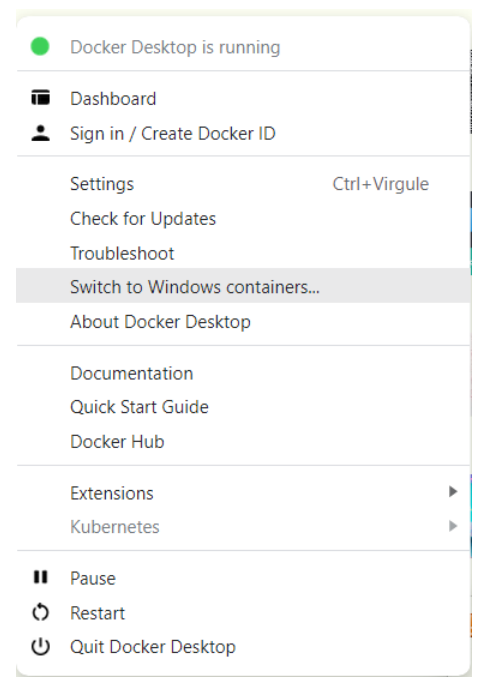
```
edgar@edgar-pc:~$ sudo docker pull mcr.microsoft.com/windows/servercore:ltsc2019[
sudo] password for edgar:
ltsc2019: Pulling from windows/servercore
no matching manifest for linux/amd64 in the manifest list entries
```

Mais on obtient un message d'erreur qui nous indique que le conteneur Windows n'est pas compatible avec l'architecture d'Ubuntu. C'est totalement normal, car les conteneurs Windows nécessitent un noyau Windows pour fonctionner, alors que Ubuntu utilise un noyau Linux.

Donc on doit refaire toutes les opérations sur Windows.

On télécharge le Docker Desktop depuis <https://docs.docker.com/desktop/install/windows-install/> et on l'installe (installation classique Windows).

Puis on change l'environnement de Docker Desktop pour qu'il travaille avec des images Windows.



Il faut aussi activer l'option qui permet d'utiliser les images Windows :

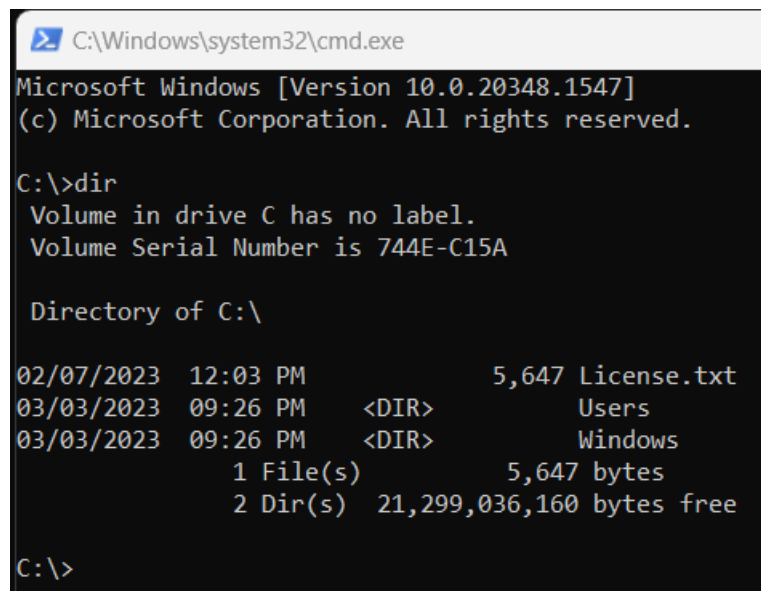
```
PS C:\WINDOWS\system32> Enable-WindowsOptionalFeature -Online -FeatureName $("Microsoft-Hyper-V", "Containers") -All
Voulez-vous redémarrer l'ordinateur pour terminer cette opération maintenant ?
[Y] Yes [N] No [?] Aide (la valeur par défaut est « Y ») : Y
```

On télécharge l'image de Windows avec **docker pull**  
**mcr.microsoft.com/windows/nanoserver:ltsc2022**

```
PS C:\WINDOWS\system32> docker pull mcr.microsoft.com/windows/nanoserver:ltsc2022
ltsc2022: Pulling from windows/nanoserver
546fcac75d6a: Pull complete
Digest: sha256:786a24be2bd1945bee9701f95a71d8573ace8641c112dc27206f826bef0229c1
Status: Downloaded newer image for mcr.microsoft.com/windows/nanoserver:ltsc2022
mcr.microsoft.com/windows/nanoserver:ltsc2022
```

On lance ce conteneur avec **docker run -it mcr.microsoft.com/windows/nanoserver:ltsc2022 cmd.exe**

Pour vérifier si le conteneur marche ou pas, on peut exécuter le « dir », on obtient la liste des répertoires :



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.20348.1547]
(c) Microsoft Corporation. All rights reserved.

C:\>dir
Volume in drive C has no label.
Volume Serial Number is 744E-C15A

Directory of C:\

02/07/2023  12:03 PM                5,647 License.txt
03/03/2023  09:26 PM             <DIR>         Users
03/03/2023  09:26 PM             <DIR>         Windows
               1 File(s)                5,647 bytes
               2 Dir(s)  21,299,036,160 bytes free

C:\>
```

e) On lance notre conteneur Ubuntu en indiquant qu'on veut utiliser le X-11 forwarding (pour affichage graphique) avec **sudo docker run -it --rm -e DISPLAY=\$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix ubuntu** et on installe gedit pour tester notre GUI.

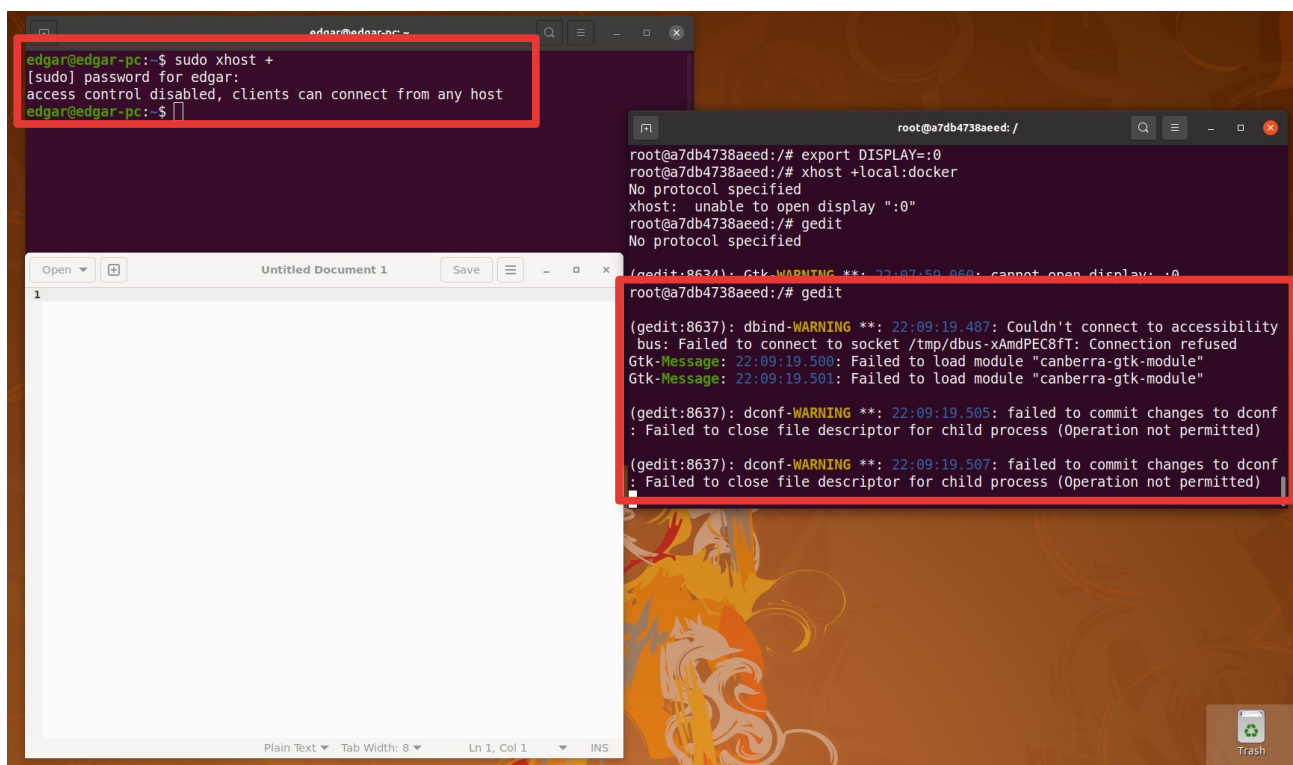
```
edgar@edgar-pc:~$ sudo docker run -it --rm -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix ubuntu
root@a7db4738aeed:/# apt-get install -y gedit
Reading package lists... Done
```

Puis on installe sur notre conteneur le xorg, qui va nous permettre de faire un x-11 forwarding :

```
sudo apt-get install xorg
```

```
root@a7db4738aeed:/# apt-get install xorg
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  apport apport-symptoms ca-certificates cpp cpp-11 desktop-file-utils dirmngr
```

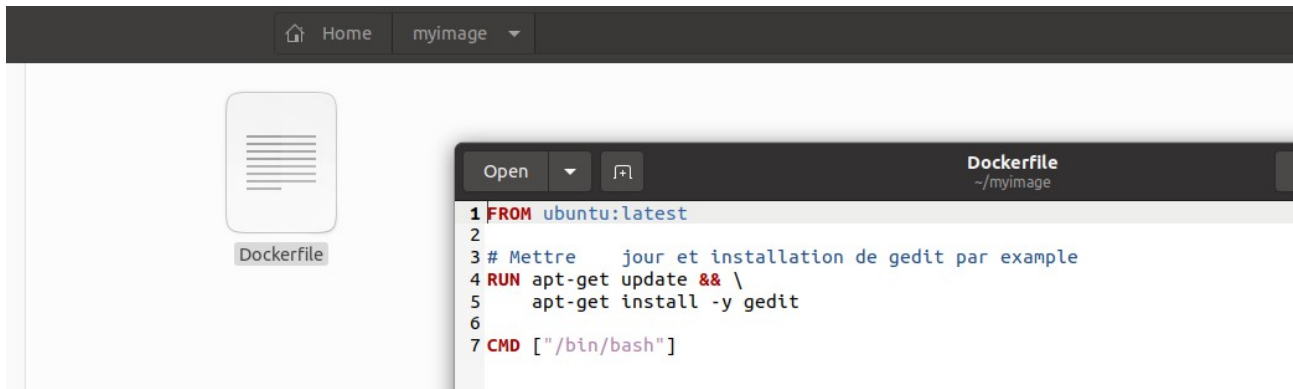
On essaye de lancer le gedit, mais cela ne marche pas, car sur notre machine(pas Docker) le x-11 forwarding est bloqué, donc on va désactiver le contrôle d'accès avec **sudo xhost +**



f) On commence par la création de la nouvelle image(l'originale + modifications).

On crée sur notre machine physique un répertoire appelle myimage et on crée dedans un fichier Dockerfile qui va contenir quelques instructions d'installation des paquets, qui vont être ajoutés à l'image originale.

```
edgar@edgar-pc:~$ mkdir myimage
edgar@edgar-pc:~$ cd myimage
edgar@edgar-pc:~/myimage$ touch Dockerfile
edgar@edgar-pc:~/myimage$ nano Dockerfile
edgar@edgar-pc:~/myimage$ docker build -t myimage:latest .
```



Donc cette modification va lancer une mise à jour et va installer un gedit. Il définit /bin/bash comme commande par défaut pour le conteneur.

```
edgar@edgar-pc:~/myimage$ sudo docker build -t myimage:latest .
[+] Building 938.0s (6/6) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 185B                                              0.0s
=> [internal] load .dockerignore                                                 0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest                 0.0s
=> [1/2] FROM docker.io/library/ubuntu:latest                                0.0s
=> [2/2] RUN apt-get update && apt-get install -y gedit                        935.8s
=> exporting to image                                                            2.2s
=> => exporting layers                                                            2.2s
=> => writing image sha256:0b2fe5fea8aa39e95fa5c85b88a20062e5166f6c09586      0.0s
=> => naming to docker.io/library/myimage:latest                             0.0s
```

On construit notre image avec **sudo docker build -t myimage:latest .**

L'image est bien créé :

```
edgar@edgar-pc:~/myimage$ sudo docker images
[sudo] password for edgar:
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
myimage             latest          0b2fe5fea8aa   20 minutes ago  679MB
ubuntu              latest          58db3edaf2be   5 weeks ago    77.8MB
docker/getting-started latest          3e4394f6b72f   2 months ago   47MB
hello-world         latest          feb5d9fea6a5   17 months ago  13.3kB
```



On execute le conteneur myimage et on peut voir la version de gedit :

```
edgar@edgar-pc:~$ sudo docker run -it myimage
[sudo] password for edgar:
root@c920c0feda94:/# gedit --version
gedit - Version 41.0
root@c920c0feda94:/#
```

Donc le gedit est installé (je n'ai pas activé le x-11 forwarding, car pour cette question ce n'est pas nécessaire, mais je montre que gedit est installé au moins), et on n'a pas un message de type bash: gedit: command not found . Donc cette image a pris en compte la code qu'on a mit dans Dockerfile.

Après l'inscription sur <https://hub.docker.com/> , on fait login :

```
edgar@edgar-pc:~$ sudo docker login
[sudo] password for edgar:
Login with your Docker ID to push and pull images from Docker Hub. If you don't
have a Docker ID, head over to https://hub.docker.com to create one.
Username: edg111777
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
edgar@edgar-pc:~$
```

Mais le Docker ne permet pas de transmettre des images qui n'ont pas de tag (p.e. versions).

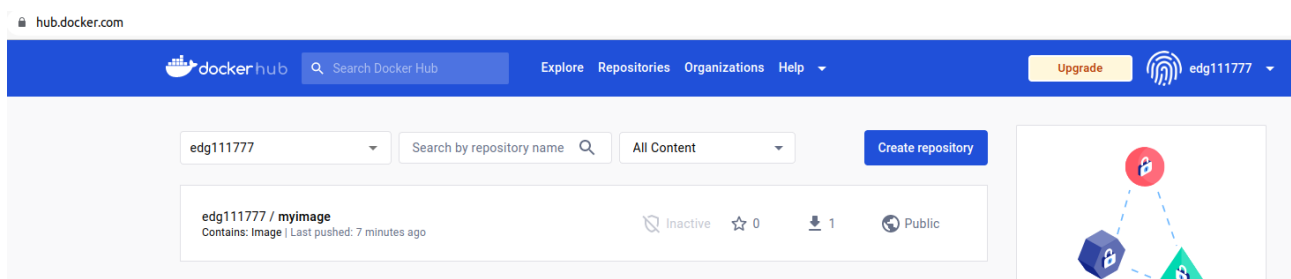
Donc on crée un tag pour notre image avec :

```
edgar@edgar-pc:~$ sudo docker tag 0b2fe5fea8aa edg111777/myimage:tag
0b2fe5fea8aa - ID de l'image qu'on veut transmettre envoyer.
```

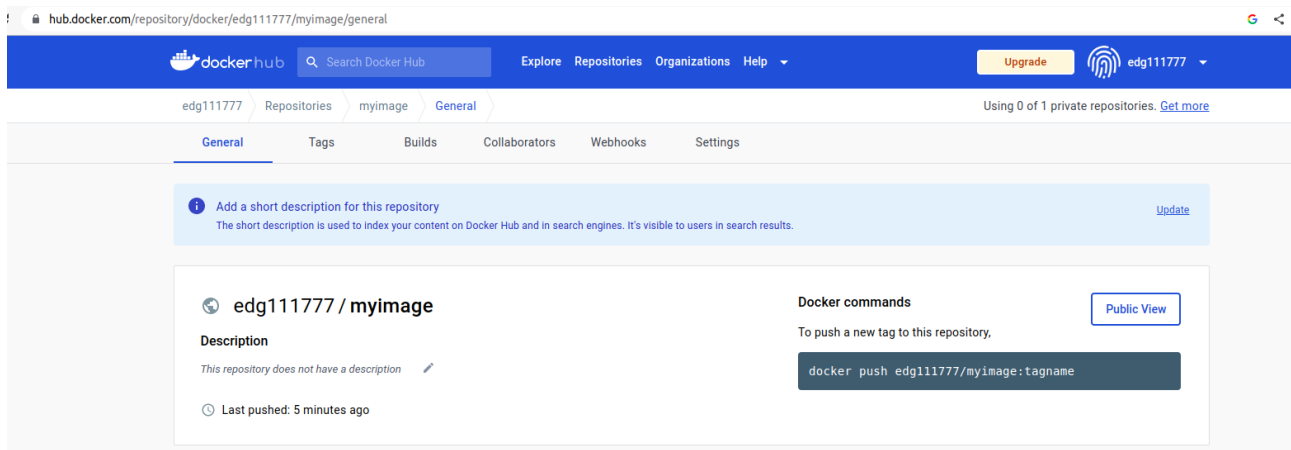
Avec **sudo docker push edg111777/myimage:tag**, on transmet notre image :

```
edgar@edgar-pc:~$ sudo docker push edg111777/myimage:tag
The push refers to repository [docker.io/edg111777/myimage]
6bc88cd12edc: Pushed
c5ff2d88f679: Mounted from library/ubuntu
tag: digest: sha256:36albe2a71944155de05c923a50d8e9788c7e763fe1a23ab7ce9b1173892
9062 size: 742
edgar@edgar-pc:~$
```

Sur le site de [hub.docker.com](https://hub.docker.com) :



Et



g) On peut commencer par exécuter la commande **sudo docker run -it myimage**, qui va créer un conteneur de myimage et va le lancer. Le docker va attribuer un nom à notre nouveau conteneur(lovig\_meninsky).On peut vérifier les conteneurs actifs avec **docker ps -a**.

```
edgar@edgar-pc:~$ sudo docker ps -a
[sudo] password for edgar:
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS       NAMES
cf754845b10b   myimage   "/bin/bash"             2 minutes ago Up 2 minutes                   loving_meninsky
```

Puis, avec **sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'** **lovig\_meninsky** (on peut utiliser l'id du conteneur aussi), on inspecte ce conteneur et on récupère le ip du conteneur.

```
edgar@edgar-pc:~$ sudo docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' loving_meninsky
172.17.0.2
edgar@edgar-pc:~$
```

L'IP est 172.17.0.2 car le conteneur a été démarré sur le réseau bridge par défaut de Docker. Le réseau bridge est un réseau interne privé pour les conteneurs Docker. Docker utilise par défaut un sous-réseau de 172.17.0.0/16 pour ce réseau bridge, et chaque conteneur qui se connecte à ce réseau bridge obtient une adresse IP dans ce sous-réseau. L'adresse 172.17.0.2 a été attribuée au conteneur loving\_meninsky car elle était disponible dans le sous-réseau lorsqu'il a été démarré.

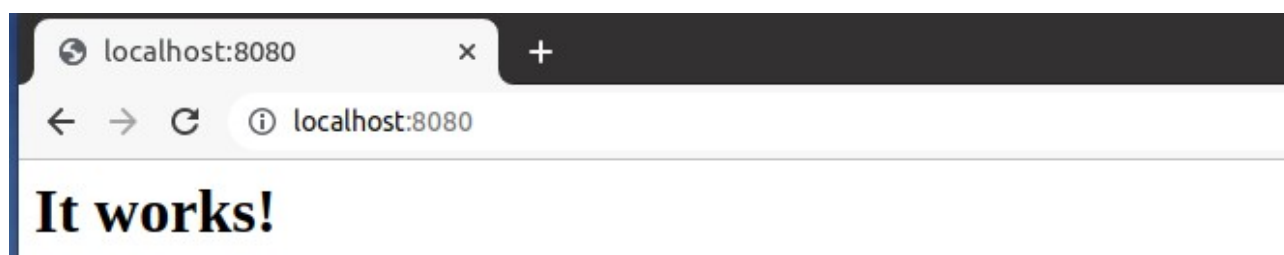


Avec la commande **sudo docker run -d -p 8080:80 httpd**(httpd est une image Docker officielle du serveur HTTP Apache qui est utilisée pour les pages web), on crée un nouveau conteneur Docker à partir de l'image httpd(cela n'existe pas localement, donc il va télécharger). Le port 80 du conteneur sera exposé sur le port 8080 de l'hôte (-p 8080:80).

```
edgar@edgar-pc:~$ sudo docker run -d -p 8080:80 httpd
[sudo] password for edgar:
Unable to find image 'httpd:latest' locally
latest: Pulling from library/httpd
3f9582a2cbe7: Pull complete
9423d69c3be7: Pull complete
d1f584c02b5d: Pull complete
8f73a485e312: Pull complete
b7697f8af320: Pull complete
Digest: sha256:83e99e7c437898cb564bbd3ceba7f1ea3f2d86e1cbd7a5324940086e59082f2b
Status: Downloaded newer image for httpd:latest
b189ec54c1ac913e8556002b04e38c6d60ece9e6faf5ad260684de9997c1568d
edgar@edgar-pc:~$ sudo docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b189ec54c1ac	httpd	"httpd-foreground"	5 seconds ago	Up 4 seconds	0.0.0.0:8080->80/tcp, :::8080->80/tcp	unruffled cannon
CT7548450100	myImage	"/bin/bash"	3 hours ago	Exited (0) 37 minutes ago		loving_meninsky

Donc on peut accéder au service HTTP sur le port 8080 de l'hôte.



## PARTIE II

1)

a) Les scanners de vulnérabilités pour conteneurs fonctionnent en analysant les images de conteneurs et les comparent à une base de données de vulnérabilités connues. Ils cherchent des vulnérabilités connues dans les bibliothèques, les configurations, les dépendances et les fichiers contenus dans l'image du conteneur. Le scanner de vulnérabilités va donc produire un rapport détaillé des vulnérabilités trouvées, qui vont nous permettre de corriger ces failles avant que les conteneurs ne soient déployés en production. Certains scanners de vulnérabilités fournissent des conseils et des recommandations pour améliorer la sécurité des conteneurs.

b)

- Analyse dynamique des conteneurs : Cette méthode analyse les conteneurs en cours d'exécution pour détecter les vulnérabilités en examinant les activités du conteneur, les fichiers système, les ports ouverts, etc.
- Analyse statique des images de conteneurs : Cette méthode analyse les images de conteneurs pour détecter les vulnérabilités connues en comparant les packages installés dans l'image avec une base de données de vulnérabilités connues
- Fuzz testing : Cette méthode consiste à générer des entrées de données invalides ou inattendues dans une application ou un système pour détecter les failles de sécurité.
- Signatures de comportement malveillant : Cette méthode analyse le comportement des conteneurs en cours d'exécution pour détecter les comportements malveillants tels que les tentatives de connexion SSH suspectes, l'accès aux fichiers système sensibles, etc.

2) On crée un compte sur <https://app.snyk.io> (j'ai associé à mon compte Docker)

Puis, on installe le snyk avec `npm install -g snyk`,

```
edgar@edgar-pc:~$ npm install -g snyk
/home/edgar/.npm/versions/node/v14.21.3/bin/snyk -> /home/edgar/.npm/versions/node/v14.21.3/lib/node_modules/snyk/bin/snyk
+ snyk@1.1111.0
added 1 package from 1 contributor in 1.008s
```

et on s'authentifie.

```
edgar@edgar-pc:~$ snyk auth
Now redirecting you to our auth page, go ahead and log in,
and once the auth is complete, return to this prompt and you'll
be ready to start using snyk.

If you can't wait use this url:
https://app.snyk.io/login?token=53658312-b4b0-43ae-95cd-d98049d1c99a&utm_medium=cli&utm_source=cli&utm_campaign=cli&os=linux&docker=false

Your account has been authenticated. Snyk is now ready to be used.
```

On peut parcourir les images presents sur [hub.docker.com](https://hub.docker.com), et choisir l'image appelé « alpin » .



Et on vérifie ces vulnérabilités avec la commande `snyk container test alpine:latest`

```
edgar@edgar-pc:~$ snyk container test alpine:latest

Testing alpine:latest...

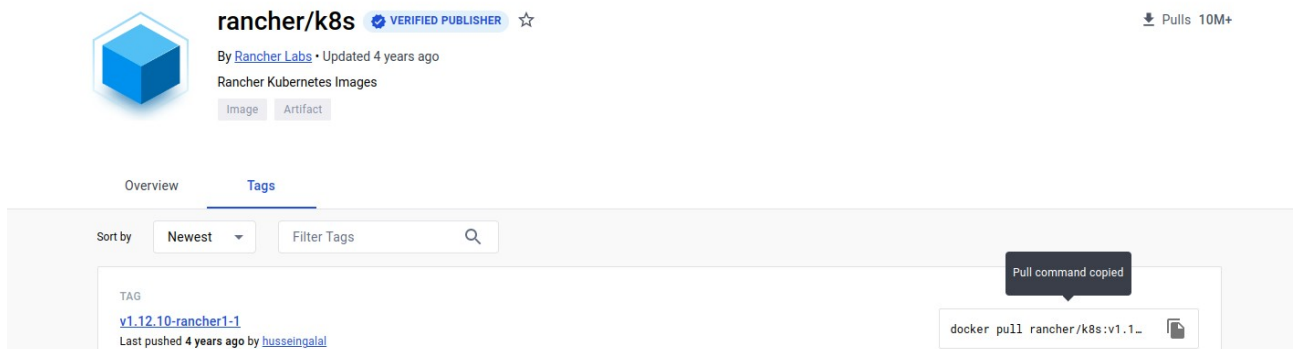
Organization:    edg111777
Package manager: apk
Project name:    docker-image|alpine
Docker image:    alpine:latest
Platform:        linux/amd64
Base image:      alpine:3.17.2
Licenses:        enabled

✓ Tested 15 dependencies for known issues, no vulnerable paths found.

According to our scan, you are currently using the most secure version of the selected base image
```

Le message indique que la version de l'image de base qu'on utilise est considérée comme sûre, c'est-à-dire qu'il n'y a pas de vulnérabilités connues dans cette version spécifique de l'image.

Mais comme on peut voir, le alpine est une image docker officielle, il est peu probable de trouver des vulnérabilités, donc on va choisir une image non-officielle, appelé « rancher/k8s:v1.12.10-rancher1-1 ».



Et on lance le test :

```
edgar@edgar-pc:~$ snyk container test rancher/k8s:v1.12.10-rancher1-1

Testing rancher/k8s:v1.12.10-rancher1-1...

x Low severity vulnerability found in zlib/zlib1g
  Description: Numeric Errors
```

On a plusieurs vulnérabilités mineures, moyennes et importants :

```
Organization:    edg111777
Package manager: deb
Project name:    docker-image|rancher/k8s
Docker image:    rancher/k8s:v1.12.10-rancher1-1
Platform:        linux/amd64
Licenses:        enabled

Tested 295 dependencies for known issues, found 751 issues.

Snyk wasn't able to auto detect the base image, use `--file` option to get base image remediation advice.
Example: $ snyk container test rancher/k8s:v1.12.10-rancher1-1 --file=path/to/Dockerfile

To remove this message in the future, please run `snyk config set disableSuggestions=true`
```

D'après l'analyse, on a testé 295 dépendances pour des problèmes connus et on a trouvé 751 problèmes. Cela indique que l'image qu'on a testé contient un certain nombre de vulnérabilités connues dans les dépendances incluses.

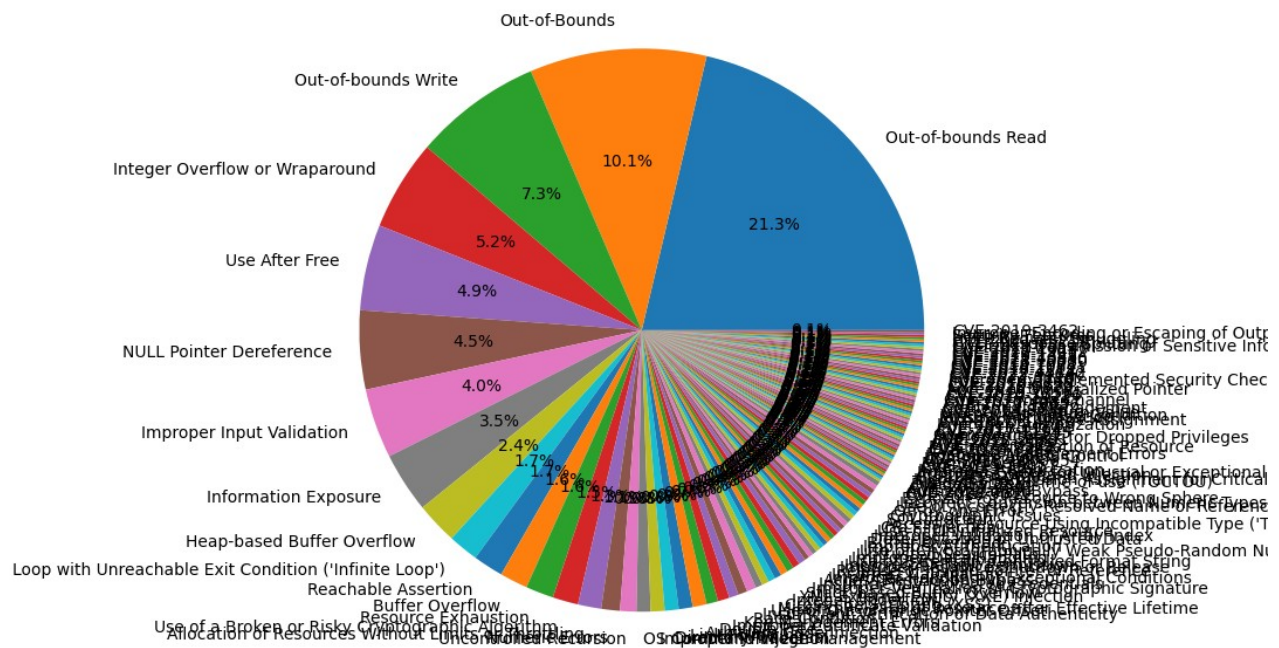
On fait un **snyk container test rancher/k8s:v1.12.10-rancher1-1 >>output.txt** pour stocker le résultat de test dans un fichier textuel.

On crée un script python(**description.py**) qui va récupérer toutes les descriptions des vulnérabilités depuis output.txt, il va les stocker dans un tableau, il va compter les occurrences pour chaque description et il va trier dans l'ordre décroissant :

```
description.py  x  output.txt  x  output2.t
1 import sys
2 from collections import Counter
3 import matplotlib.pyplot as plt
4
5
6 filename = sys.argv[1]
7
8 tableau = []
9
10
11 with open(filename, 'r') as file:
12     for line in file:
13
14         if 'Description' in line:
15             mot = line.split(':')[1].strip()
16             tableau.append(mot)
17
18 compteur_desc = Counter(tableau)
19
20 labels = [mot for mot, count in compteur_desc.items()]
21 sizes = [count for mot, count in compteur_desc.items()]
22
23 fig1, ax1 = plt.subplots()
24 ax1.pie(sizes, labels=labels, autopct='%1.1f%%')
25 ax1.axis('equal')
26
27 plt.show()
28
```

```
Out-of-bounds Read: 160
Out-of-Bounds: 76
Out-of-bounds Write: 55
Integer Overflow or Wraparound: 39
Use After Free: 37
NULL Pointer Dereference: 34
Improper Input Validation: 30
Information Exposure: 26
Heap-based Buffer Overflow: 18
```

Avec la bibliothèque matplotlib(pip install matplotlib), on peut afficher le diagramme circulaire pour les données.



Comme on peut voir ici, le plus grand nombre des vulnérabilités sont présentées par :

**Out-of-bounds Read** - Se produit généralement lorsque le programmeur accède à une partie de la mémoire qui ne lui appartient pas, ou qui n'est pas allouée pour la lecture pour un moment.

**Out-of-Bounds** - ou débordement de tampon, est une vulnérabilité de sécurité qui survient lorsque des données sont écrites ou lues au-delà des limites des tableaux ou des tampons alloués en mémoire.

**Integer Overflow or Wraparound** - une vulnérabilité qui se produit lorsqu'un entier dépasse sa capacité de stockage maximale et est renvoyé à la valeur minimale. Les attaquants peuvent utiliser cette vulnérabilité pour contourner les contrôles de sécurité, exécuter un code malveillant ou obtenir un accès non autorisé à un système informatique.

On a créé un script python, appelé **compteSynkResult.py**, qui va compter le nombre des vulnérabilités total, il va compter aussi les vulnérabilités avec des étiquettes low,medium,critical.

On le lance avec **python3 compteSynkResult.py output.txt**

```
filename = sys.argv[1]

tlow = 0
tmedium=0
thigh = 0
tcritical = 0
tableau=[]

with open(filename, 'r') as file:
    for line in file:
        if 'Low severity vulnerability' in line:
            tlow+=1
        if 'Medium severity vulnerability' in line:
            tmedium+=1
        if 'High severity vulnerability' in line:
            thigh+=1
        if 'Critical severity vulnerability' in line:
            tcritical+=1

tableau.append(tlow)
tableau.append(tmedium)
tableau.append(thigh)
tableau.append(tcritical)

compter_desc = Counter(tableau)
sorted_desc = sorted(compter_desc.items(), key=lambda x: x[1], reverse=True)

labels = ['Low','Medium','High','Critical']
values = [tableau[0],tableau[1],tableau[2],tableau[3]]

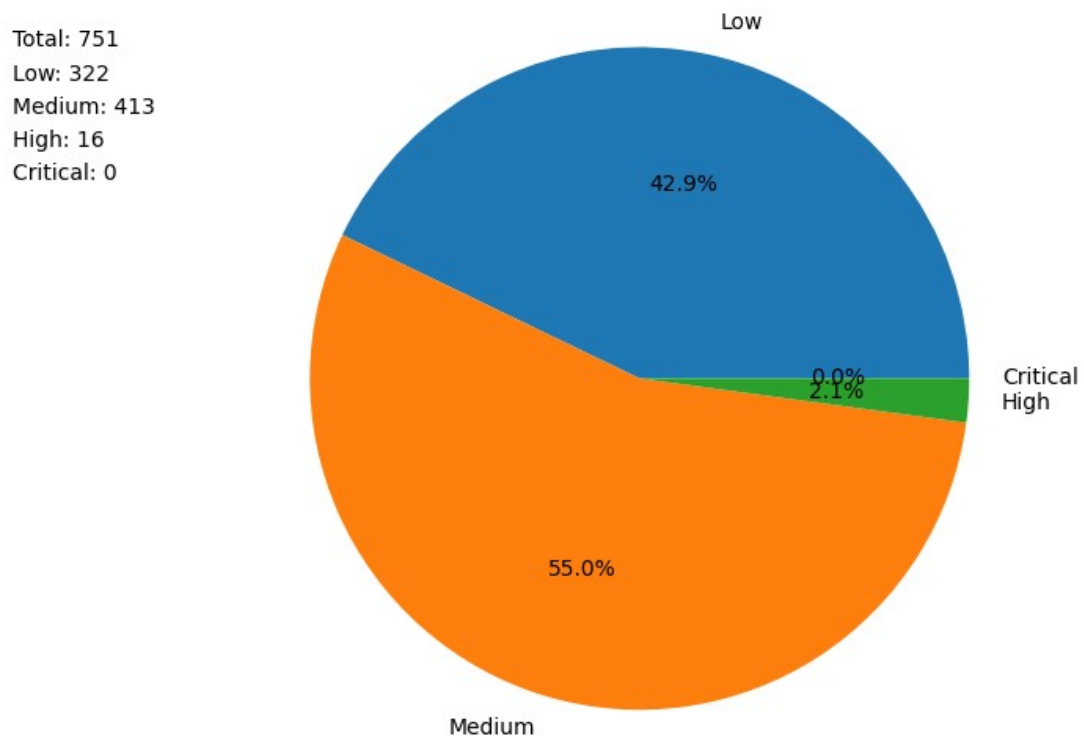
fig1, ax1 = plt.subplots()
ax1.pie(values, labels=labels, autopct='%1.1f%%')
ax1.axis('equal')

total = sum(values)
plt.text(-1.9, 1.0, f"Total: {total}")
plt.text(-1.9, 0.9, f"Low: {values[0]}")
plt.text(-1.9, 0.8, f"Medium: {values[1]}")
plt.text(-1.9, 0.7, f"High: {values[2]}")
plt.text(-1.9, 0.6, f"Critical: {values[3]}")

plt.show()
```



Les résultats sont présentés par le diagramme circulaire.

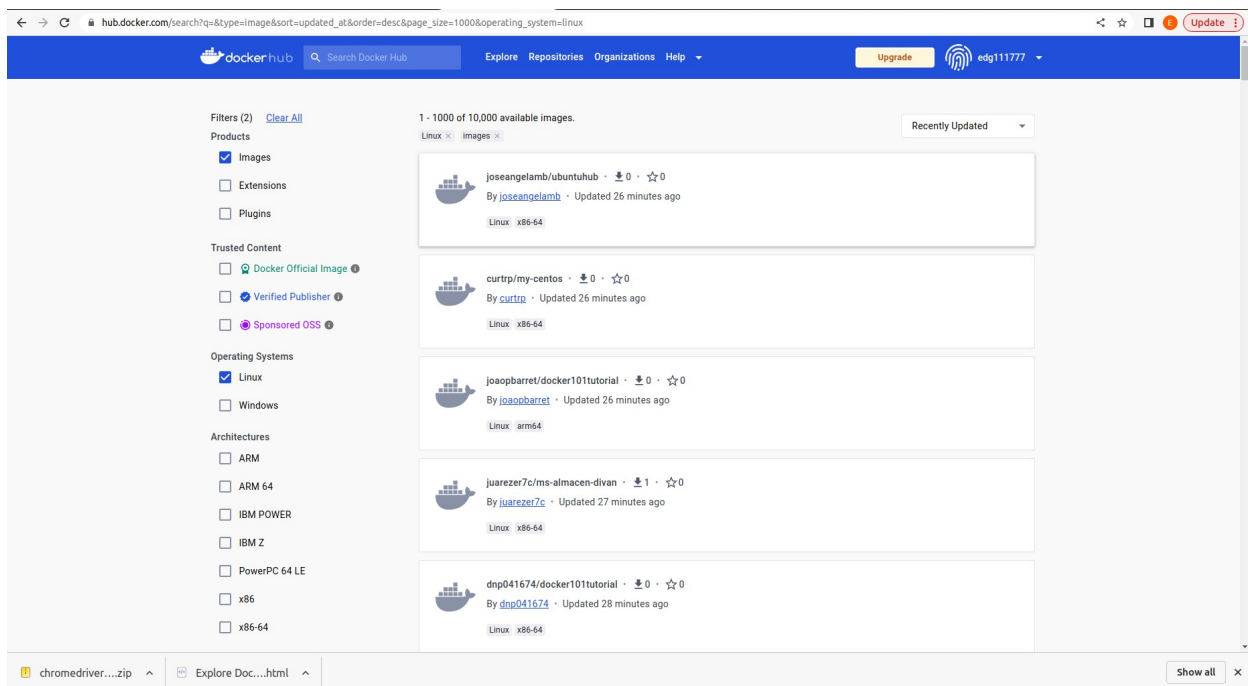


On peut déduire qu'on a 0 vulnérabilité critique, la majorité des vulnérabilités, 55 %, sont de niveau moyenne. Puis, des faibles vulnérabilités (42,9%), puis des vulnérabilités importantes (2,1).

Par contre, on a 0 vulnérabilité critique.

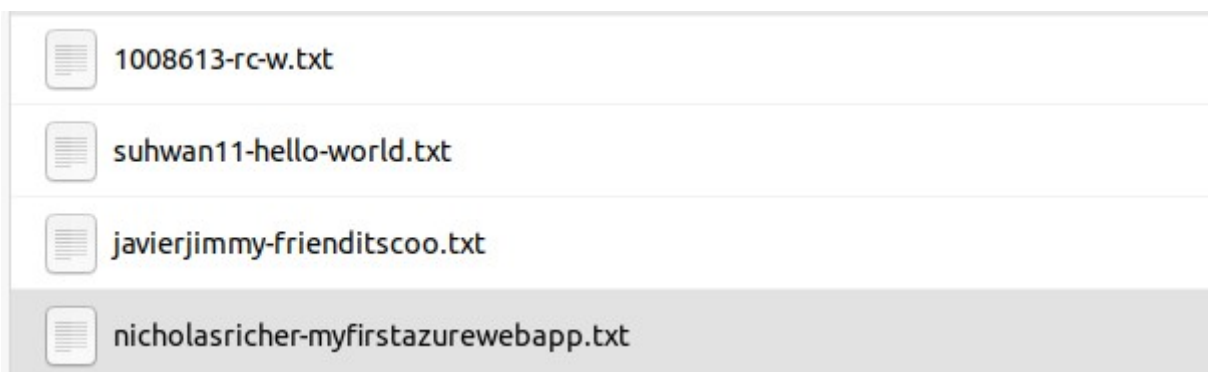
3) On a essayé d'analyser les images non-officielles (Linux), qui sont ajoutées récemment dans Docker Hub, et les images aléatoires officielles basées sur Linux (avec l'abonnement gratuit, je suis limité et je ne peux pas récupérer tous les noms des images à partir de leur API).

Contrairement aux images officielles, il n'y a pas de possibilité pour récupérer les noms des dernières images ajoutées avec les API, donc on a fait un scraping sur la page des dernières images ajoutées dans Docker Hub. Le script Python est appelé **scrape.py** et il utilise des technologies pour faire scrape des pages dynamiques.



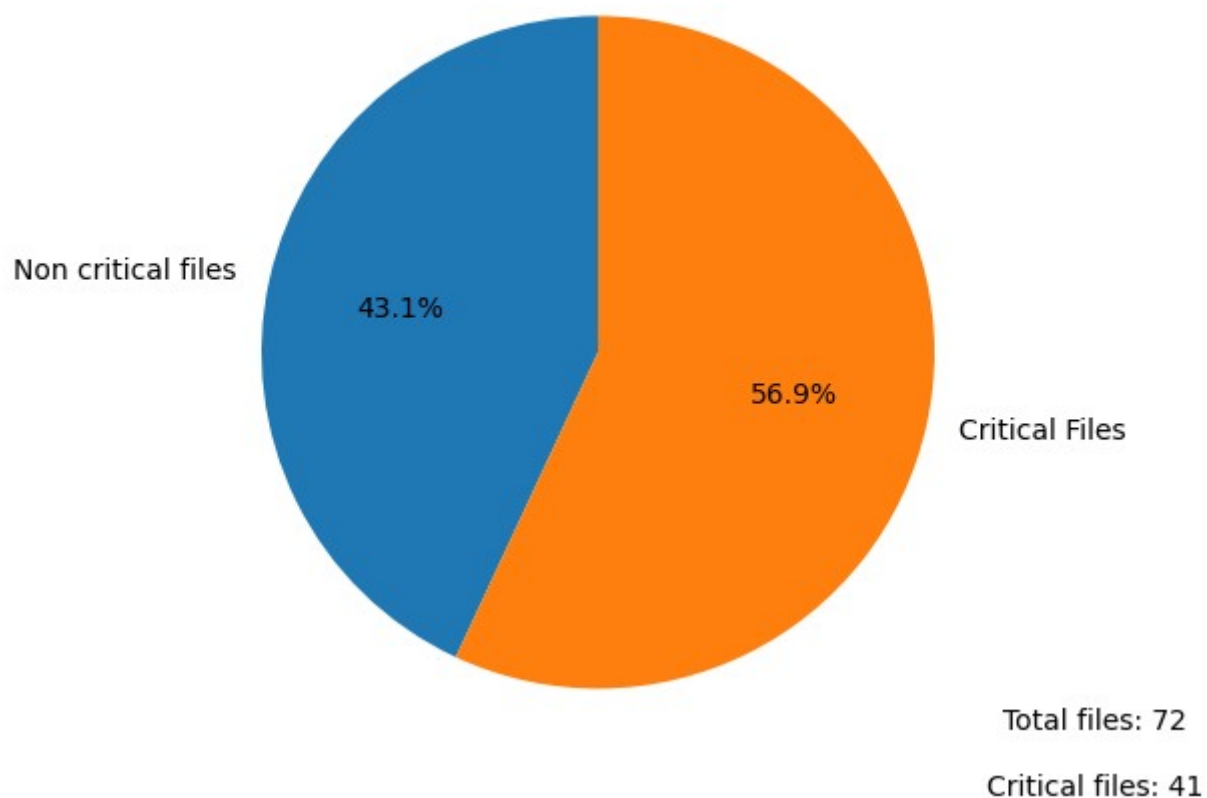
C'est la page qu'on a scrape.

Puis, on insère ces images dans un fichier appelé dockerImages.txt et on lance le scripte **analyse.py** qui récupère le fichier dockerImages.txt, et pour chaque ligne il fait la commande **snym container test {image\_name}**, les résultats sont gardées dans un répertoire appelé « results ».



Pour chaque image le script crée un fichier, où on garde le résultat de son test synk.

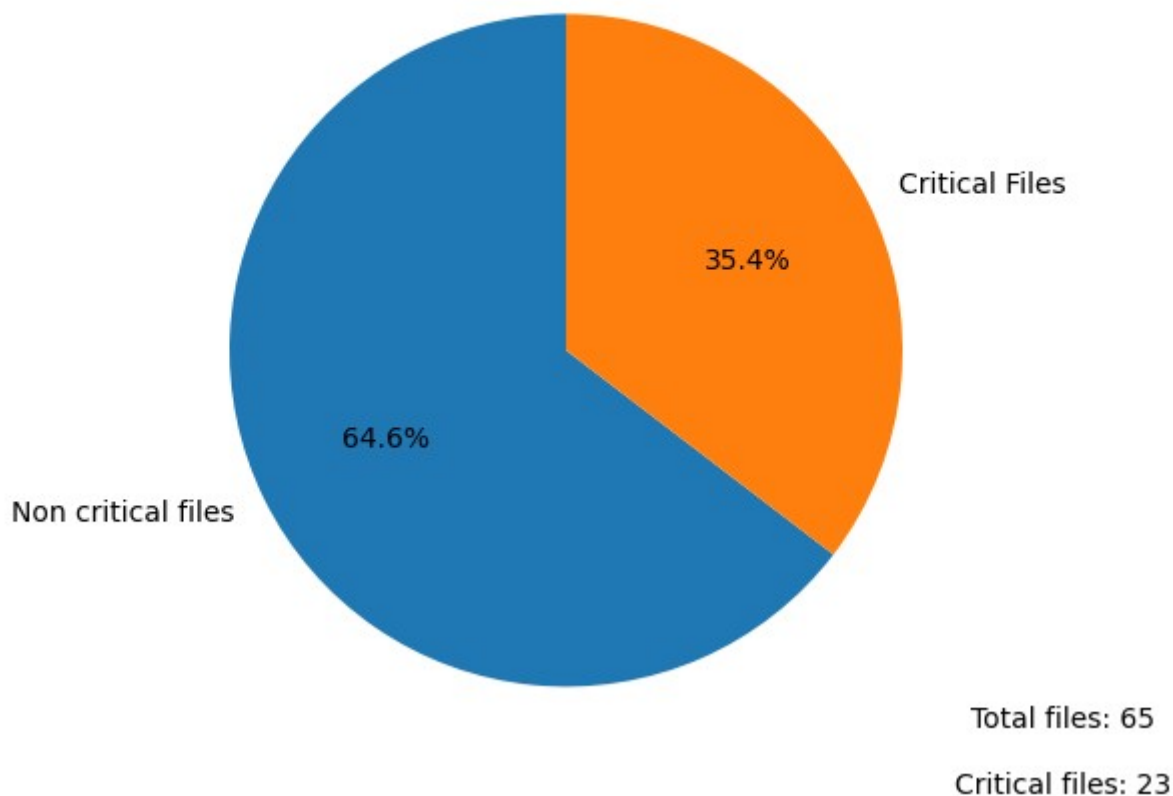
Puis, on lance le script `conclusion.py` pour montrer le pourcentage des erreurs critiques et non critiques.



Comme on peut voir dans ce diagramme, pour le total de 72 images, on a 41 qui ont des vulnérabilités critique. Donc 56,9 %, ce qui est énorme.

Après, on va faire exactement la même manipulation pour les images officielles, mais on utilisera le script `officialImagesAPI.py`, qui va récupérer les noms des images officielles depuis Docker Hub.

Avec `conclusion.py` on obtient ce résultat .



Donc même dans les images officielles on a eu des vulnérabilités critique. Pour 65 images on a 23 qui ont des failles critiques : 35,4 %.

Remarque : Dans l'analyse de l'image httpd, on a obtenu 2 vulnérabilités critiques :

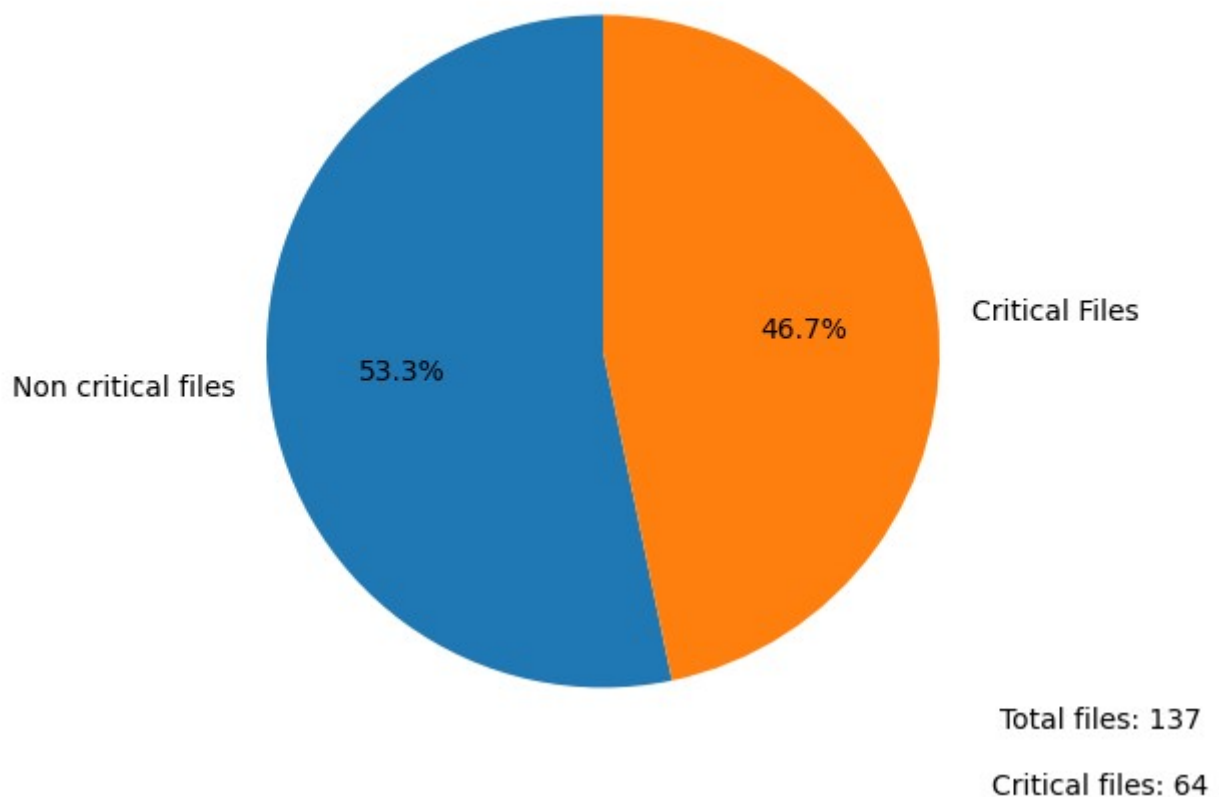
```
X Critical severity vulnerability found in curl/libcurl4
Description: Cleartext Transmission of Sensitive Information
Info: https://security.snyk.io/vuln/SNYK-DEBIAN11-CURL-3320493
Introduced through: curl/libcurl4@7.74.0-1.3+deb11u7
From: curl/libcurl4@7.74.0-1.3+deb11u7
Image layer: Introduced by your base image (httpd:2.4.56-bullseye)

X Critical severity vulnerability found in apr/libapr1
Description: Integer Overflow or Wraparound
```

- curl/libcurl4 & apr/libapr1 utilisées dans l'image httpd sont susceptibles d'être utilisés par des hackers pour compromettre la sécurité du système.

Et dans la question g de la partie I on a utilisé cette image.

On mélange les images officielles et non-officielles et on obtient ce résultat.



On obtient un pourcentage de 46,7% pour les vulnérabilités critiques.

On peut déduire que dans les images officielles on a moins de vulnérabilité critique que dans les non-officielles : 21,5 % de différentes.

Et en mélangeant les deux types, on obtient un résultat qui nous indique que presque la moitié(46,7%) des images de notre étude contiennent des failles importantes.

Mais les images non officielles sont dominantes dans Docker Hub, et comme on a vu dans l'analyse des images non-officielles, la majorité des images ont des vulnérabilités très importantes.